LIF

Laboratoire d'Informatique Fondamentale
de Marseille

Unité Mixte de Recherche 6166
CNRS - Université de Provence - Université de la Méditerranée

The receptive distributed π-Calculus

Roberto M. Amadio, Gérard Boudol, Cédric Lhoussaine

Rapport/Report 06-2002

31 Mai, 2002

The receptive distributed π-Calculus

Roberto M. Amadio (1), Gérard Boudol (2),
Cédric Lhoussaine (3)

Laboratoire d'Informatique Fondamentale

UMR 6166
CNRS - Université de Provence - Université de la Méditerranée

(1) Université de Provence, (2) INRIA-Sophia, (3) University of Sussex

amadio@cmi.univ-mrs.fr

Abstract/Résumé

We study an asynchronous distributed π-calculus, with constructs for localities and migration. We show that a static analysis ensures the receptiveness of channel names, which, together with a simple type system, guarantees the *message deliverability* property. This property states that any migrating message will find an appropriate receiver at its destination locality. We argue that this distributed, receptive calculus is still expressive enough while allowing for an effective type inference *à la* ML.

Nous étudions un π-calcul asynchrone réparti, muni de constructions de localités et de migration. Nous montrons qu'on peut garantir la réceptivité des canaux de communication grâce à une analyse statique très simple, et que ceci, combiné avec un sytème de types, garantit une propriété de *livrabilité des messages*. La propriété exprime que tout message migrant trouvera un récepteur disponible dans sa localité de destination. Nous montrons que ce π-calcul réparti et réceptif reste très expressif tout en permettant un mécanisme d'inférence de type à la ML.

Relecteurs/Reviewers: Silvano Dal Zilio, Denis Lugiez.

Notes: The first author is partly supported by IST Profundis. Some of the results presented here were announced in a preliminary form in a paper by the same authors appeared in the proceedings of the FST&TCS'99 Conference, Lecture Notes in Comp. Sci. 1738

1 Introduction

The *distributed* π-*calculus* Dπ introduced by Hennessy and Riely [9] is an enrichement of the π-calculus involving explicit notions of *locality* and *migration*, that is code movement from a location to another. In this model communication is purely *local*, so that messages to remote resources (*i.e.* receivers, or input processes, in π-calculus terminology) must be explicitly routed. This is, *e.g.*, similar to the Ambient calculus of Cardelli and Gordon [6] and different from the JOIN calculus [7, 8] or the π_{1l}-calculus [1] where messages transparently go through the network to reach their (unique) destination.

In this paper, we address a problem arising in a model with such a *local* communication: a message may stay in, or migrate to, a locality where no receiver will ever be available. Of course, in a non-distributed setting messages may already fail to be delivered for other reasons: the receiver of the message might terminate or it might wait for another event to occur. We will rule out this kind of failures by concentrating on a fragment of the π-calculus where receivers are persistent. Technically, we will rely on an elaboration over the π_1-calculus [1], called the *receptive* π-calculus or π_1^r-calculus [3, 4] in short[1], that guarantees that on every channel a unique input is always available up to some finite internal computation. The resulting fragment is still expressive as it allows a fully abstract encoding of the π_1-calculus[2].

Thanks to this hypothesis, we can concentrate on the deliverability failures introduced by *distribution and migration*. They may arise, *e.g.*, because a message addressed to a resource migrates to a location where the resource is not available or because the resource has migrated elsewhere. We are concerned with the design of a type system that guarantees that messages can be delivered. As usual, the problem is to find a compromise between expressivity (enough programs can be typed) and feasibility (type inference is practical).

Our inference system is naturally decomposed in two parts:

- A type system that checks the consistent use of the names as channels or locations (section 3). This system can be regarded as a variant of the *simple* type system of [9] which is tailored towards message deliverability and effective type inference.
- A system to derive judgments $I \Vdash P$, where I is a set of names and P is a process (section 4). If $I \Vdash P$ then we say that P is *well-formed*. This system checks that the resources in I are persistently available in P and it can be regarded as an elaboration over the system for the π_1-calculus taking into account distribution and migration.

Because of the dependencies among channel names, location names, and types the exact definition of these two systems is a non trivial task. For both the typing and the well-formation system we show the usual 'subject reduction property'. Then we prove that processes that are typed *and* well-formed have *receptive* (or persistent) channels. We derive as a corollary the *message deliverability* property: any message will find an appropriate receiver in its destination locality (section 5).

The last part of the paper (section 6) argues that the system we have designed is sufficiently *expressive* to type interesting examples of distributed programming. The style of programming which is adopted is one where resources are persistent and may always react in some way to requests.

The *type inference* problem is solved in the third author's PhD Thesis [13]. It is fairly obvious to produce an algorithm to check well-formedness. Inference for the type system can also be handled in a satisfying way. In particular, a principal type *à la* ML can be effectively computed for every typable process. This requires an elaboration over type inference techniques for records [10, 15, 20].

Related work A message that fails to be deliverable can be regarded as a kind of *local deadlock*. The issue of avoiding deadlocks by statically inferring some properties has already been addressed. For instance, in [5], a type system intended to prevent deadlocks is proposed for the blue calculus, where the types are formulae of Hennessy-Milner logic with recursion.

Kobayashi *et al.* [11, 18] also studied means to avoid deadlocks in the π-calculus. However, his approach is quite different from ours: he uses a sophisticated type system where types involve time tags and where typing contexts record an ordering on the usage of names, whereas we only use a very simple information – the set of receiver's names. Since we regard receivers as passive entities, providing resources for names, we are not seeking to avoid the situation where no message is sent to a receiver.

[1] The form of receptivity goes well beyond the 'uniform' or 'linear' one introduced in [16].

[2] In turn, there are fully abstract encodings of the asynchronous π-calculus in the join calculus [7] and of the latter in the π_1-calculus [2].

Hennessy and Riely [9] enrich the operational semantics so that every name is *tagged* with the capabilities it has acquired. They then show that in 'well-typed programs' it never happens that an agent tries to use a name in a way that goes beyond the capabilities it has acquired for it (a safety property). We do not repeat their result here and concentrate on the message deliverability property which requires that a requested service will eventually become available (a liveness property).

2 Distributed processes

We introduce a *subcalculus* of the distributed π-calculus of Hennessy and Riely [9].

2.1 Names

We assume a denumerable set $\mathcal{N}$ of *simple names* that is partitioned in two sets of channel names $\mathcal{N}_{ch} = \{a, b, \ldots\}$ and of location names $\mathcal{N}_{loc} = \{\ell, k, \ldots\}$. We shall actually use the names of $\mathcal{N}_{loc}$ also as *values*, which, in particular, may be compared for equality. We could obviously use more elaborate data types for values, but this is not really needed for the purposes of the paper. A *compound name* $a@\ell$ is a pair composed of a channel name a and a location name ℓ. Intuitively these names are needed to restrict the usage of the channel a to the location ℓ. We use u, v, $w, \ldots$ to denote simple or compound names, and a (possibly empty) vector of such names is written $\vec{u}$.

The operation $\_@\ell$ on simple or compound names is defined by

$$u@\ell = \begin{cases} a@\ell & \text{if } u = a \\ u & \text{if } u = a@\ell' \end{cases}$$

and extended to sets of names N, L in the obvious way: $N@L = \{\, u@\ell \mid u \in N \ \&\ \ell \in L \,\}$.

2.2 Types

The types for names are given as follows:

$$\begin{array}{llll} \tau,\ \sigma \ldots & ::= & \gamma \ \mid\ \psi \ \mid\ \gamma@\psi & \textit{types} \\ \gamma,\ \delta \ldots & ::= & \mathit{Ch}(\tau_1, \ldots, \tau_n) & \textit{channel types} \\ \psi,\ \phi \ldots & ::= & \{a_1 : \gamma_1, \ldots, a_n : \gamma_n\} & \textit{location types} \end{array}$$

Channel types $\gamma = \mathit{Ch}(\tau_1, \ldots, \tau_n)$ are usual in the π-calculus: a channel name has type γ if it can carry n names of type $\tau_1, \ldots, \tau_n$, respectively.

Location types $\psi = \{a_1 : \gamma_1, \ldots, a_n : \gamma_n\}$ were introduced in [9]. Here it is understood that the names $a_1, \ldots, a_n$ are all distinct, and that their order does not matter (cf. record types). Thus a location type is really a mapping from a finite set of channel names into the set of channel types. Intuitively, the 'fields' $a_1, \ldots, a_n$ correspond to the resources (channels) available in the location.

Compound types $\gamma@\psi$ correspond naturally to compound names $a@\ell$. These types can be regarded as a particular form of *existential* types, reading $\gamma@\psi$ as '$\exists c\, \{c : \gamma\} \uplus \psi$', where $\uplus$ is disjoint union. We denote by *loc* the empty location type and abbreviate $\gamma@\mathit{loc}$ with $\gamma@$. We also denote by $\psi \sqcup \phi$ the union of ψ and ϕ, which is only defined if ψ and ϕ assign the same types to the names they share.

2.3 Processes and systems

The syntax for (parametric) processes and systems is defined in figure 1. We reserve $U, V \ldots$ to denote any of these terms. Basically, we deal with an asynchronous, polyadic π-calculus, with some primitives for spatial distribution of processes and migration, organized as a two-level model: besides the usual notion of a *process* (or *thread*), performing inputs and outputs on communication channels, we consider *systems* that are networks of located processes. The operator $\mathsf{go}\,\ell.P$ expresses the *migration* of the process P to the location ℓ.

Processes and systems may contain three different forms of *name generation*: the generation of a channel name (νa), the generation of a channel name at a location $(\nu a@\ell)$, and the generation of a location name with a specified type $(\nu\ell : \psi)$. We define the subject $\mathsf{subj}(w)$ of a name generation as follows: $\mathsf{subj}(a) = \mathsf{subj}(a@\ell) = a$, $\mathsf{subj}(\ell : \psi) = \ell$. The subject of w is the only name occurring in w which is bound. Notice that the channel

$$
\begin{array}{rcll}
(\nu w) & ::= & (\nu u) \mid (\nu\ell:\psi) & \textit{name generation} \\
P\ldots & ::= & & \textit{processes (or threads)} \\
& \mid & \overline{a}(u_1,\ldots,u_n) & \quad\textit{message on channel } a \\
& \mid & a(u_1,\ldots,u_n).P & \quad\textit{input on channel } a \\
& \mid & (P \mid Q) & \quad\textit{parallel composition} \\
& \mid & [\ell=\ell']P,Q & \quad\textit{conditional branching} \\
& \mid & (\nu w)P & \quad\textit{name generation} \\
& \mid & \mathsf{go}\,\ell.P & \quad\textit{migration} \\
& \mid & T(u_1,\ldots,u_n) & \quad\textit{parametric process instanciation} \\
T & ::= & & \textit{parametric processes} \\
& & A & \quad\textit{identifier} \\
& \mid & (\mathsf{rec}\,A(u_1,\ldots,u_n).P) & \quad\textit{recursive parametric process} \\
S & ::= & & \textit{systems} \\
& \mid & [\ell :: P] & \quad\textit{located process} \\
& \mid & (S \mid S') & \quad\textit{parallel composition} \\
& \mid & (\nu w)S & \quad\textit{name generation}
\end{array}
$$

Figure 1: Syntax

names in $(\nu\ell:\psi)$ – that is, in ψ – are never bound. We also extend the $\_@\ell$ operation to generated names as follows:

$$w@\ell = \begin{cases} a@\ell & \text{if } w = a \\ w & \text{otherwise} \end{cases}$$

Processes rely on parametric recursive definitions rather than replication; this is instrumental to the expressivity of the typed model. Therefore we assume a denumerable set $\mathcal{P} = \{A, B, \ldots\}$ of parametric process identifiers disjoint from $\mathcal{N}$. We restrict our attention to terms that satisfy the following standard requirements:

(i) in an input $a(\vec{u}).P$, all the names occurring in $\vec{u}$ are assumed to be distinct, and different from a, and similarly in $(\mathsf{rec}\,A(\vec{u}).P)$.
(ii) recursion is *guarded*, that is in $(\mathsf{rec}\,A(\vec{u}).P)(\vec{v})$ all recursive calls to A in P occur under an input guard.

The notions of free and bound names are the usual ones; we denote by $\mathsf{fn}(U)$ and $\mathsf{bn}(U)$ the sets of free and bound names respectively occurring in the term U, and $\mathsf{nm}(U)$ denotes the union of these two sets (we shall reuse this notation, for instance for vectors of compound names). We shall say that U is *closed* if it does not contain any free process identifier (notice that a closed term may contain free names).

As a running example for this paper, we shall use a simple formulation of the standard *remote procedure call* mechanism; the dual of this, that is *remote evaluation*, is trivially expressed in our language, by the primitive operation $\mathsf{go}\,\ell.P$. In this example, a client process Q, located at ℓ, wants to call a service named a, located at ℓ', with arguments $\vec{d}$, and waits for results, to be processed by a continuation agent P. Then the client creates a private return channel at its own location, and sends it, as the subject of a compound name indicating the return locality, together with the data $\vec{d}$. We write this as follows:

$$Q = (\nu r)(\,\mathsf{go}\,\ell'.\overline{a}(\vec{d}, r@\ell) \mid r(\vec{v}).P\,)$$

To describe the service – the remote procedure –, let us first recall the "replicated input", given by

$$a^*(\vec{u}).P \;=_{\text{def}}\; \mathsf{rec}\,A(a).a(\vec{u}).(P \mid A(a))$$

where we simply write $\mathsf{rec}\,A(\vec{u}).P$ for $(\mathsf{rec}\,A(\vec{u}).P)(\vec{u})$. Now assume that the service R computes a function $f(\vec{d})$ of the arguments and sends back the result to the caller; using an enriched syntax, we can write this simply as

$$R = a^*(\vec{x}, y@z).\mathsf{go}\,z.\overline{y}(f(\vec{x}))$$

One sees in particular how the compound name $y@z$ is decomposed into a locality, where to go, and a channel on which to send the result. In this example the migrating agents are just messages. We shall see more elaborate examples later, but since migrating messages are very often used, we introduce a specific notation for them:

$$\overline{a}@\ell(\vec{u}) =_{\text{def}} \mathsf{go}\,\ell.\overline{a}(\vec{u})$$

Then for instance the RPC example is $[\ell' :: R] \mid [\ell :: Q]$ where

$$\begin{array}{rcl} Q & = & (\nu r)(\,\overline{a}@\ell'(\vec{d}, r@\ell) \mid r(\vec{v}).P\,) \\ R & = & a^*(\vec{x}, y@z).\overline{y}@z(f(\vec{x})) \end{array}$$

2.4 Substitution

We use two kinds of substitutions, of names for names, and of parametric processes for identifiers. A name substitution is a mapping S from a finite subset $\mathsf{dom}(\mathsf{S})$ of $\mathcal{N}$ into $\mathcal{N}$. We denote by $\mathsf{im}(\mathsf{S})$ the image of S, that is the set $\{\, \mathsf{S}(a) \mid a \in \mathsf{dom}(\mathsf{S}) \,\}$. Applying the substitution to a term U is denoted $[\mathsf{S}]U$, but *this is only defined if no capture may arise*, that is if $\mathsf{im}(\mathsf{S}) \cap \mathsf{bn}(U) = \emptyset$. Moreover, whenever U is a system term, we require S to be *injective* for $[\mathsf{S}]U$ to be defined; the reason for this is that in the case of $U = (\nu\ell : \{a_1 : \gamma_1, \ldots, a_n : \gamma_n\})V$ the names $a_1, \ldots, a_n$ are free, and possibly in $\mathsf{dom}(\mathsf{S})$, but must be kept distinct after substitution. The definitions and technical results regarding substitutions are collected in the appendix A. If S is the mapping

$$a_1 \mapsto b_1, \ldots, a_k \mapsto b_k$$

then $[\mathsf{S}]U$ is also denoted $[b_1/a_1, \ldots, b_k/a_k]U$, or $[\vec{b}/\vec{a}]U$. We also use the notation $[v/u]U$ where u and v are possibly compound names; in this case, if $u = x@y$ then x is supposed to be distinct from y, and v must be a compound name $a@b$, and the substitution really is $[a/x, b/y]U$, whereas if u is a simple name, then v must be a simple name too. Regarding the substitution of parametric processes for indentifiers, we shall only use substitutions for a single name in process terms, written $[T/A]P$. Again, this is only defined if no capture may arise, that is $\mathsf{fn}(T) \cap \mathsf{bn}(P) = \emptyset$.

2.5 Structural equivalence

An *evaluation context* $\mathbf{E}$ is defined by the following grammar:

$$\mathbf{E} ::= [] \;\mid\; (\mathbf{E} \mid U) \;\mid\; (\nu w)\mathbf{E} \;\mid\; [\ell :: \mathbf{E}]\ .$$

Each evaluation contexts $\mathbf{E}$ contains exactly one hole $[]$ and filling the hole in a context by a term U results in a term denoted $\mathbf{E}[U]$.

We consider terms up to a *structural equivalence* relation $\equiv$ that identifies terms up α-conversion (see appendix A for the precise definition), associativity and commutativity of parallel composition, and satisfies the following axioms

$$\begin{array}{rclll} ((\nu w)U \mid V) & \equiv & (\nu w)(U \mid V) & \mathsf{subj}(w) \notin \mathsf{fn}(V) & \textit{scope migration (1)} \\ [\ell :: P \mid Q] & \equiv & [\ell :: P] \mid [\ell :: Q] & & \textit{routing} \\ [\ell :: (\nu w)P] & \equiv & (\nu w@\ell)[\ell :: P] & \ell \neq \mathsf{subj}(w) & \textit{scope migration (2)} \end{array}$$

and the following rule where $\mathbf{E}$ is any evaluation context

$$U \equiv V \quad\Rightarrow\quad \mathbf{E}[U] \equiv \mathbf{E}[V]\ .$$

Remark 1 *The routing axiom is mainly used, from left to right, to allow a migrating process to exit a locality, as in $[\ell :: \mathsf{go}\,\ell'.P \mid Q] \equiv [\ell :: \mathsf{go}\,\ell'.P] \mid [\ell :: Q]$ – we shall see that this evolves into $[\ell' :: P] \mid [\ell :: Q]$ –, and, from right to left, to allow a "packet" $[\ell' :: P]$ to join the destination locality. Up to this equivalence, any process P can be turned into a "canonical form", of the shape:*

$$P \equiv (\nu\overrightarrow{w})(R_1 \mid \cdots \mid R_n)$$

$$
\begin{array}{rcll}
(\overline{a}(\vec{v}) \mid a(\vec{u}).P) & \to & [\vec{v}/\vec{u}]P & \textit{communication} \\
[\ell' :: \mathsf{go}\,\ell.P] & \to & [\ell :: P] & \textit{movement} \\
(\mathsf{rec}\,A(\vec{u}).P)(\vec{v}) & \to & [(\mathsf{rec}\,A(\vec{u}).P)/A][\vec{v}/\vec{u}]P & \textit{unfolding} \\
[\ell = \ell]P, Q & \to & P & \textit{matching} \\
[\ell = \ell']P, Q & \to & Q \qquad \ell \neq \ell' & \textit{mismatching}
\end{array}
$$

$$
\frac{V \equiv U \ ,\ U \to U' \ ,\ U' \equiv V'}{V \to V'} \qquad\qquad \frac{U \to U'}{\mathbf{E}[U] \to \mathbf{E}[U']}
$$

Figure 2: Reduction rules

where the R_i's, that we may call "molecules", are either messages $\overline{a}(\vec{v})$, or receivers $a(\vec{u}).Q$, or branching processes $[a = b]Q_0, Q_1$, or else recursive or migrating processes $T(\vec{u})$ and $\mathsf{go}\,\ell.Q$. Similarly, any system S can be turned into a canonical form:

$$S \equiv (\nu\overrightarrow{w})([\ell_1 :: P_1] \mid \cdots \mid [\ell_k :: P_k])$$

where each P_1 is a parallel composition of molecules, and the ℓ_i's are pairwise distinct.

For instance, in our RPC example, we have:

$$[\ell' :: R] \mid [\ell :: Q] \equiv (\nu r@\ell)\big(\,[\ell' :: a^*(\vec{x}, y@z).\overline{y}@z(f(\vec{x}))] \mid [\ell :: \overline{a}@\ell'(\vec{d}, r@\ell) \mid r(\vec{v}).P]\,\big)$$

2.6 Reduction

The rules of reduction are given in figure 2. In the communication and unfolding rules we implicitly assume that the substitution is defined. This means for instance, in the case of a communication, that the vectors $\vec{u}$ and $\vec{v}$ have the same length, and the same kind (simple or compound) of components. Moreover, the names sent in a message must not be bound in the receiver (otherwise we can rename).

We shall denote by $U \twoheadrightarrow V$ the reduction relation obtained by forbidding the use of the communication rule in the definition of $\to$. It is easily checked that, thanks to the assumption that recursion is guarded, this notion of reduction is strongly normalizing, *i.e.*, there are only finite sequences of $\twoheadrightarrow$-reductions starting from any term. One can see for instance, using the axioms of structural equivalence, that

$$[\ell :: \mathsf{go}\,\ell'.P \mid Q] \mid [\ell' :: R] \twoheadrightarrow [\ell :: Q] \mid [\ell' :: P \mid R]$$

We could have taken this as the primitive rule of movement, thus avoiding the use of the "routing" law[3]. We should point out that *communication is only local*: typically, no reduction can arise from $[\ell :: \overline{a}(\vec{u})] \mid [\ell' :: a(\vec{v}).P]$ if $\ell \neq \ell'$, and one has to explicitly move and meet in order to communicate, like in $[\ell :: \mathsf{go}\,\ell'.\overline{a}(\vec{u})] \mid [\ell' :: a(\vec{v}).P]$. In other words, unlike in [1, 8] where messages can transparently go through domains to reach the corresponding receiver, we have the "go and communicate" semantics of the Dπ-calculus, which is also the one of the Ambient calculus [6]. Coming back to our RPC example, we have:

$$
\begin{array}{rcl}
[\ell' :: R] \mid [\ell :: Q] & \equiv & (\nu r@\ell)\big(\,[\ell' :: R] \mid [\ell :: \mathsf{go}\,\ell'.\overline{a}(\vec{d}, r@\ell)] \mid [\ell :: r(\vec{v}).P]\,\big) \\
& \twoheadrightarrow & (\nu r@\ell)\big(\,[\ell' :: R] \mid [\ell' :: \overline{a}(\vec{d}, r@\ell)] \mid [\ell :: r(\vec{v}).P]\,\big) \\
& \equiv & (\nu r@\ell)\big(\,[\ell' :: R \mid \overline{a}(\vec{d}, r@\ell)] \mid [\ell :: r(\vec{v}).P]\,\big)
\end{array}
$$

[3]Notice that the two formulations are not quite equivalent, since our simple rule allows to create a locality. We do not use this feature though, and we could have introduced an explicit notion of a "packet", but this would slightly complicate the formal developments.

Since $R \twoheadrightarrow a(\vec{x}, y@z).(\mathsf{go}\, z.\overline{y}(f(\vec{x})) \mid R)$, we then have:

$$\begin{array}{rcl}
[\ell' :: R] \mid [\ell :: Q] & \xrightarrow{*} & (\nu r@\ell)\big(\,[\ell' :: R \mid \mathsf{go}\,\ell.\overline{r}(f(\vec{d}))] \mid [\ell :: r(\vec{v}).P]\,\big) \\
& \equiv & (\nu r@\ell)\big(\,[\ell' :: R] \mid [\ell' :: \mathsf{go}\,\ell.\overline{r}(f(\vec{d}))] \mid [\ell :: r(\vec{v}).P]\,\big) \\
& \twoheadrightarrow & (\nu r@\ell)\big(\,[\ell' :: R] \mid [\ell :: \overline{r}(f(\vec{d}))] \mid [\ell :: r(\vec{v}).P]\,\big) \\
& \equiv & (\nu r@\ell)\big(\,[\ell' :: R] \mid [\ell :: \overline{r}(f(\vec{d})) \mid r(\vec{v}).P]\,\big) \\
& \rightarrow & (\nu r@\ell)\big(\,[\ell' :: R] \mid [\ell :: [f(\vec{d})/\vec{v}].P]\,\big)
\end{array}$$

Here we have assumed that the vectors $\vec{d}$ and $\vec{x}$ have the same length, and similarly for $f(\vec{d})$ and $\vec{v}$. This will be checked with the type system. This example demonstrates the use of compound names. However, in this example we could have used pairs of names r, ℓ (and y, z) instead of $r@\ell$ (resp. $y@z$). The difference will appear with the typing of processes: typically, a term like $a^*(\vec{x}, y@z).\overline{y}@z'(f(\vec{x}))$ will only be well-typed if $z' = z$.

3 A simple type system

The simple type system we propose differs in significant ways from the one in [9][section 4]. In a sense, it is more general because it includes parametric recursive definitions, in another sense, it is more restrictive because it has a much more limited form of subtyping as we look for an effective type inference algorithm. In our experience, these variations are error prone, and this is why we include a self-contained treatment of the subject-reduction property.

The type system presented in figure 3 deals with sequents of the form $\Gamma \vdash_\ell P$, for checking that the process P, placed at the current location ℓ, conforms to the typing assumption Γ, and $\Gamma \vdash S$ and $\Gamma \vdash_\ell T : \gamma$, respectively, for systems and for parametric processes. The type system also uses sequents $\Gamma \vdash^W_\ell \vec{u} : \vec{\tau}$ to type names *with weakening* of hypotheses and sequents $\Gamma \vdash_\ell \vec{u} : \vec{\tau}$ *without weakening*. The latter are used to type formal parameters in input and in recursive definitions. Here x stands for any name in $\mathcal{N} \cup \mathcal{P}$.

The typing context Γ is a mapping from a finite subset $\mathsf{dom}(\Gamma)$ of $\mathcal{N}_{loc} \cup \mathcal{P}$ into the set of types, subject to the following requirements:

(i) if $\ell \in \mathsf{dom}(\Gamma) \cap \mathcal{N}$ then $\Gamma(\ell)$ is a locality type ψ.
(ii) if $A \in \mathsf{dom}(\Gamma) \cap \mathcal{P}$ then $\Gamma(A)$ is a channel type.

We will write a typing context as a list of typing assumptions:

$$\Gamma = \ldots, \ell : \{a_1 : \gamma_1, \ldots, a_n : \gamma_n\}, \ldots, A : Ch(\vec{\tau}), \ldots$$

In the type system we make use of a partial operation of union Δ, Γ of typing contexts, defined as follows:

$$(\Delta, \Gamma)(x) = \begin{cases} \Delta(x) & \text{if } x \in \mathsf{dom}(\Delta) - \mathsf{dom}(\Gamma) \text{ or } \Delta(x) = \Gamma(x) \\ \phi \sqcup \psi & \text{if } \Delta(x) = \phi \ \&\ \Gamma(x) = \psi \\ \Gamma(x) & \text{if } x \in \mathsf{dom}(\Gamma) - \mathsf{dom}(\Delta) \end{cases}$$

We make the usual assumption that bound names are renamed so that no collision with other bound or free names arises. We comment the rules of the system:

1. In the rules for names without weakening the conclusion determines the context. Namely, if $\Gamma \vdash_\ell \vec{u} : \vec{\tau}$ and $\Delta \vdash_\ell \vec{u} : \vec{\tau}$ then $\Gamma = \Delta$.
2. The rule for output, involves a form of subtyping for localities, due to the use of weakening: for instance, the judgement
$$\ell : \{b : \gamma_0, c : \gamma_1\}, \ \ell' : \{a : Ch(loc)\} \vdash_{\ell'} \overline{a}(\ell)$$
is valid, even though the type of ℓ given by the context is more generous than the one carried by the channel a. This form of subtyping is not allowed in input, since the received names cannot occur in the typing context.
3. The only point to note about typing parallel composition is that the component must agree on the type of names they use, since they share the same context.

RULES FOR NAMES:

$$\frac{}{\ell : \{a : \gamma\} \vdash_\ell a : \gamma} \qquad \frac{}{\ell : \{a : \gamma\} \uplus \psi \vdash_\ell a@\ell : \gamma@\psi}$$

$$\frac{}{\ell' : \psi \vdash_\ell \ell' : \psi} \qquad \frac{\Gamma \vdash_\ell \vec{u} : \vec{\tau} \quad \Delta \vdash_\ell \vec{v} : \vec{\sigma}}{\Gamma, \Delta \vdash_\ell \vec{u} : \vec{\tau}, \vec{v} : \vec{\sigma}}$$

RULES FOR NAMES WITH WEAKENING:

$$\frac{}{\Gamma, \ell : \{a : \gamma\} \vdash^W_\ell a : \gamma} \qquad \frac{}{\Gamma, \ell : \{a : \gamma\} \uplus \psi \vdash^W_{\ell'} a@\ell : \gamma@\psi}$$

$$\frac{}{\Gamma, \ell' : \psi \vdash^W_\ell \ell' : \psi} \qquad \frac{\Gamma \vdash^W_\ell \vec{u} : \vec{\tau} \quad \Gamma \vdash^W_\ell \vec{v} : \vec{\sigma}}{\Gamma \vdash^W_\ell \vec{u} : \vec{\tau}, \vec{v} : \vec{\sigma}}$$

RULES FOR PROCESSES:

$$\frac{\Gamma \vdash^W_\ell \vec{u} : \vec{\tau}}{\ell : \{a : Ch(\vec{\tau})\}, \Gamma \vdash_\ell \overline{a}\vec{u}} \qquad \frac{\Gamma\,,\, \Delta \vdash_\ell P \quad \Gamma \vdash_\ell \vec{u} : \vec{\tau}}{\ell : \{a : Ch(\vec{\tau})\}, \Delta \vdash_\ell a(\vec{u}).P}$$

$$\frac{\Gamma \vdash_\ell P, \Gamma \vdash_\ell Q}{\Gamma \vdash_\ell (P \mid Q)} \qquad \frac{\Gamma \vdash_\ell P \quad \Gamma \vdash_\ell Q}{\ell, \ell' : loc, \Gamma \vdash_\ell [\ell = \ell']P, Q}$$

$$\frac{\ell : \{a : \gamma\}, \Gamma \vdash_\ell P}{\Gamma \vdash_\ell (\nu a)P} \qquad \frac{\ell' : \{a : \gamma\}, \Gamma \vdash_\ell P}{\Gamma \vdash_\ell (\nu a@\ell')P}$$

$$\frac{\ell' : \psi, \Gamma \vdash_\ell P}{\Gamma \vdash_\ell (\nu \ell' : \psi)P} \qquad \frac{\Gamma \vdash_{\ell'} P}{\Gamma \vdash_\ell \mathsf{go}\, \ell'.P}$$

$$\frac{}{A : \gamma, \Gamma \vdash_\ell A : \gamma} \qquad \frac{\Delta \vdash_\ell T : Ch(\vec{\tau}) \quad \Gamma \vdash_\ell \vec{u} : \vec{\tau}}{\Delta\,,\, \Gamma \vdash_\ell T(\vec{u})}$$

$$\frac{A : Ch(\vec{\tau}), \Gamma, \Delta \vdash_\ell P \quad \Gamma \vdash_\ell \vec{u} : \vec{\tau}}{\Delta \vdash_\ell (\mathsf{rec}\, A(\vec{u}).P) : Ch(\vec{\tau})} \qquad \frac{\Gamma \vdash_\ell Q \quad P =_\alpha Q}{\Gamma \vdash_\ell P}$$

RULES FOR SYSTEMS:

$$\frac{\Gamma \vdash_\ell P}{\Gamma \vdash [\ell :: P]} \qquad \frac{\Gamma \vdash S \quad \Gamma \vdash S'}{\Gamma \vdash (S \mid S')}$$

$$\frac{\ell : \{a : \gamma\}, \Gamma \vdash S}{\Gamma \vdash (\nu a@\ell)S} \qquad \frac{\ell : \psi, \Gamma \vdash S}{\Gamma \vdash (\nu \ell : \psi)S} \qquad \frac{\Gamma \vdash_\ell S' \quad S =_\alpha S'}{\Gamma \vdash_\ell S}$$

Figure 3: Type system for names, processes, and systems

4. In the rules for conditional branching, we see that we can compare only names with an empty location type. This condition can be relaxed.
5. There are three cases for typing a name generation $(\nu w)P$ and the rules follow the same pattern as the ones for typing names.
6. To type a migrating process $\mathsf{go}\,\ell.P$, one must be able to type P at locality ℓ, while the resulting current locality is immaterial.

Let us see how to type the client of the RPC example, assuming that the data and results are values, that is, in our simplified approach, names of type *loc*, and assuming that the continuation P is typable in the context $\overrightarrow{v : loc}, \Gamma$ at the location ℓ of the client. Let $\psi = \{a : Ch(\overrightarrow{loc}, Ch(\overrightarrow{loc})^{@})\}$ and $\phi = \{r : Ch(\overrightarrow{loc})\}$, and $\Delta = \ell' : \psi\,,\, \ell : \phi\,,\, \overrightarrow{d : loc}\,,\, \Gamma$ – assuming that this is a legal context. Then we have:

$$
\dfrac{
\dfrac{
\dfrac{
\dfrac{
\dfrac{}{\ell : \phi\,,\, \overrightarrow{d : loc} \vdash^{W}_{\ell'} \overrightarrow{d : loc}\,,\, r@\ell : Ch(\overrightarrow{loc})^{@}}
}{\ell' : \psi\,,\, \ell : \phi\,,\, \overrightarrow{d : loc} \vdash_{\ell'} \overline{a}(\vec{d}, r@\ell)}
}{\ell' : \psi\,,\, \ell : \phi\,,\, \overrightarrow{d : loc} \vdash_{\ell} \mathsf{go}\,\ell'.\overline{a}(\vec{d}, r@\ell)}
}{\Delta \vdash_{\ell} \mathsf{go}\,\ell'.\overline{a}(\vec{d}, r@\ell)}
\qquad
\dfrac{
\dfrac{
\dfrac{\vdots}{\overrightarrow{v : loc}\,,\, \Gamma \vdash_{\ell} P}
\qquad
\dfrac{}{\overrightarrow{v : loc} \vdash_{\ell} \overrightarrow{v : loc}}
}{\ell : \phi\,,\, \Gamma \vdash_{\ell} r(\vec{v}).P}
}{\Delta \vdash_{\ell} r(\vec{v}).P}
}{
\dfrac{\Delta \vdash_{\ell} (\mathsf{go}\,\ell'.\overline{a}(\vec{d}, r@\ell) \mid r(\vec{v}).P)}{
\dfrac{\ell' : \psi\,,\, \overrightarrow{d : loc}\,,\, \Gamma \vdash_{\ell} (\nu r)(\mathsf{go}\,\ell'.\overline{a}(\vec{d}, r@\ell) \mid r(\vec{v}).P)}{\ell' : \psi\,,\, \overrightarrow{d : loc}\,,\, \Gamma \vdash [\ell :: (\nu r)(\mathsf{go}\,\ell'.\overline{a}(\vec{d}, r@\ell) \mid r(\vec{v}).P)]}
}
}
$$

The RPC example illustrates a situation where a request is moving towards a service. One may wonder what happens for a process migrating out of its current location, where some ressource is available. For instance, to type $(\nu a)(a(\vec{u}).R \mid \mathsf{go}\,\ell.P \mid Q)$ at the current locality ℓ' requires, by virtue of the rules for scoping, input, parallel composition and migration, to type P at ℓ with a typing context containing $\ell' : \{a : Ch(\vec{\tau})\}$ as the only assumption about a. This means that in P the name a can only be used at ℓ', that is, if $\ell' \neq \ell$, either in a compound name $a@\ell'$ or inside a guard $\mathsf{go}\,\ell'$. Similarly, in the case of a moving "server", like in $(\nu a)(\mathsf{go}\,\ell.a(\vec{u}).R \mid P)$, one can see that the process P must use the name a at ℓ, again in a compound name (unless ℓ is the current locality), or inside a migration construct $\mathsf{go}\,\ell$.

To prove that typing is preserved by reduction, we need some auxiliary results, mainly relating substitution and typing, which are stated and proved in appendix A. Let us denote by $P \sqsubseteq_{\mathcal{T}} Q$ the fact that Q has more typings than P. Formally, the relation $\sqsubseteq_{\mathcal{T}}$ is defined by

$$
\begin{array}{rcl}
P \sqsubseteq_{\mathcal{T}} Q & \Leftrightarrow & \forall\Gamma\,\forall\ell\,(\Gamma \vdash_{\ell} P \;\Rightarrow\; \Gamma \vdash_{\ell} Q) \\
T \sqsubseteq_{\mathcal{T}} T' & \Leftrightarrow & \forall\Gamma\,\forall\ell\,\forall\tau\,(\Gamma \vdash_{\ell} T : \tau \;\Rightarrow\; \Gamma \vdash_{\ell} T' : \tau) \\
S \sqsubseteq_{\mathcal{T}} S' & \Leftrightarrow & \forall\Gamma\,(\Gamma \vdash S \;\Rightarrow\; \Gamma \vdash S')
\end{array}
$$

The following lemma follows from the observation that the typing of a term only depends on the typing of its components.

Lemma 2 *The relation $\sqsubseteq_{\mathcal{T}}$ is a precongruence, that is a preorder compatible with the syntactic constructs*

We denote by $=_{\mathcal{T}}$ the associated equivalence relation, that is $U =_{\mathcal{T}} V$ if and only if $U \sqsubseteq_{\mathcal{T}} V$ and $V \sqsubseteq_{\mathcal{T}} U$. This is obviously a congruence. The first step towards "subject reduction" is to show that typing is preserved by structural equivalence.

Proposition 3 *The relation $\sqsubseteq_{\mathcal{T}}$ contains the structural equivalence relation $\equiv$.*

Proof Hint. By the lemma 2, the relation $=_{\mathcal{T}}$ is a congruence containing α-conversion, and therefore it is enough to show that for each axiom $U \equiv V$ we have $U \sqsubseteq_{\mathcal{T}} V$ and $V \sqsubseteq_{\mathcal{T}} U$. The case analysis is given in appendix B.1. ❏

We can then state the announced result.

$$\frac{}{\Vdash \overline{a}\vec{u}} \qquad \frac{a \Vdash P}{a \Vdash a(\vec{u}).P}$$

$$\frac{I \Vdash U \quad I' \Vdash V \quad I \cap I' = \emptyset}{I, I' \Vdash (U \mid V)} \qquad \frac{I \Vdash P \ , \ I \Vdash Q}{I \Vdash [\ell = \ell']P, Q}$$

$$\frac{u, I \Vdash U}{I \Vdash (\nu u)U} \qquad \frac{\mathsf{dom}(\psi)@\ell \, , \, I \Vdash U}{I \Vdash (\nu \ell : \psi)U}$$

$$\frac{I \Vdash P \quad \{\, a \mid a, a@\ell \in I \,\} = \emptyset}{I@\ell \Vdash \mathsf{go}\,\ell.P} \qquad \frac{I \Vdash P \quad \{\, a \mid a, a@\ell \in I \,\} = \emptyset}{I@\ell \Vdash [\ell :: P]}$$

$$\frac{}{\Vdash A} \qquad \frac{a \Vdash P}{\Vdash (\mathsf{rec}\, A(a, \vec{u}).P)} \qquad \frac{\Vdash T}{a \Vdash T(a, \vec{u})}$$

Figure 4: Well-formed terms

Theorem 4 (preservation of typing) $U \to V \ \Rightarrow\ U \sqsubseteq_{\mathcal{T}} V$.

Proof Hint. By induction on the definition of $U \to V$. Since $\sqsubseteq_{\mathcal{T}}$ is a precongruence containing the structural equivalence relation, it is enough to consider the axioms of reduction. The case analysis is given in appendix B.2. ❏

4 Interfaces and well-formed processes

The type system does not guarantee message deliverability, consider, *e.g.*, $(\nu a)\overline{a}()$. Then we introduce in figure 4 an inference system for checking 'well-formedness' of terms. This system relies on sequents $I \Vdash U$, where U is a term and I is a finite set of names to be thought as the *interface* of U. Intuitively, if $I \Vdash U$ and $u \in I$ then U can persistently perform input on u. As usual, we represent a set $I = \{u_1, \ldots, u_n\}$ as the list $u_1, \ldots, u_n$ of its elements, and we write $I \cup I'$ as I, I'. We comment the system and observe some of its basic properties.

1. If one forgets about distribution then the system is a refinement of the one for the π_1-calculus [1] that achieves receptivity by forcing a process not to contain nested inputs on different channels. Thus in the rule for input the interface of the continuation P is assumed to be $\{a\}$. Observe also that the only way to introduce a name a in the interface is to introduce a process identifier applied to a, that is $A(a, \vec{u})$, which will be involved in a guarded recursion, where the input guarding the recursive call must be on the same name a – thanks to the rule for input.
2. Two other conditions that are helpful in proving receptivity are that: (i) a received channel name cannot be used as an input channel (a standard condition in the join-calculus [7] and the 'local' π-calculus [14]), and (ii) there is a receiver for every generated channel.
3. The system also forces the unicity of the receiver, *i.e.*, two parallel components of a process are not allowed to receive on a same name. This condition can be relaxed without compromising receptivity by removing the condition $I \cap I' = \emptyset$ in the rule for parallel composition and, in the distributed case, by removing the side condition in the rules for $\mathsf{go}\,\ell.P$ and $[\ell :: P]$. Nevertheless, we require the uniqueness of the receivers because it seems a sensible assumption both from a specification and an implementation point of view (*cf.* object-oriented models, and [7]). Moreover, this property is useful in proving full abstraction results.
4. Turning to the distributed aspects, we note that when generating a location name $(\nu \ell : \psi)$ with $\psi \equiv \{a_1 : \gamma_1, \ldots, a_n : \gamma_n\}$ we require a receiver to be provided at ℓ for each name $a_1, \ldots, a_n$. Thus the location type ψ describes the set of resources available to an agent at the location ℓ.

5. We require resources in a location to be persistent. For instance, a term like $a(\vec{u}).(P \mid \mathsf{go}\,\ell.a(\vec{v}).Q)$ is not acceptable, because the subterm $(P \mid \mathsf{go}\,\ell.a(\vec{v}).Q)$ should have interface $\{a\}$, while it is only acceptable with an interface containing $a@\ell$. This means that servers are not so easily moved, which is quite natural, given the properties we are looking for. Indeed, we do not want the receiver to disappear going elsewhere after having received a message. We shall see in section 6.4 how to manage migrating resources while respecting this constraint.

Let us consider our running RPC example. The client process, that is $Q = (\nu r)(\,\mathsf{go}\,\ell'.\overline{a}(\vec{d}, r@\ell) \mid r(\vec{v}).P\,)$ is only well-formed if r is in the interface of P. This may seem somewhat unnatural, since r is a return channel which, intuitively, is supposed to receive just one answer from the called procedure. Indeed, it would be better to incorporate the treatment of *linear channels* as in [12]; we do not do this here because this would complicate the type system (see a discussion on this point in [4]). There is actually a standard way to turn a process like the continuation P, that is not supposed to be receptive on r, into a receptive process, which is to use the *input once* construct of [1], given by

$$a(\vec{u}){:}P \ =_{\mathrm{def}}\ a(\vec{u}).(P \mid T_a)$$

where T_a is the process that repeatedly receives messages on a and ignores them:

$$T_a \ =_{\mathrm{def}}\ \mathsf{rec}\,A(a).a(\vec{u}).A(a)$$

A more realistic program would rather send back an exception in case further messages are received on the return channel (unless, as we said, this channel has been typed as linear). It is easy to see that this agent is well-formed, with interface $\{a\}$, and therefore $a(\vec{u}){:}P$ has the same typing as $a(\vec{u}).P$, and has interface $\{a\}$ if $\Vdash P$. Then we can turn the RPC client into a well-formed process – provided that the continuation P has an empty interface – by replacing the ordinary input with an input once:

$$Q = (\nu r)(\,\overline{a}@\ell'(\vec{d}, r@\ell) \mid r(\vec{v}){:}P\,)$$

which has the same typing as before. Regarding the remote procedure, one can see that the replicated input $a^*(\vec{u}).P$ has the same typing and well-formedness as the input once construct – that is the same typing as input, and $a \Vdash a^*(\vec{u}).P$ if $\Vdash P$.

Now we prove a subject reduction property for well-formedness. The following is easily seen.

Remark 5 *If $I \Vdash U$ is provable then $\mathsf{nm}(I) \subseteq \mathsf{fn}(U)$.*

Again we need to establish a property relating well-formedness and substitution. In the following lemma we use a notation $\mathsf{S}(I \Vdash U)$ similar to the one we used regarding typing. The proof goes by a simple induction on the inference of $I \Vdash U$.

Lemma 6 *Let $I \Vdash U$ and S be a substitution such that $[\mathsf{S}]U$ is defined, and S is injective on I. Then $\mathsf{S}(I \Vdash U)$, and if $I \Vdash [\mathsf{S}]U$ then $\mathsf{S}^{-1}(I) \Vdash U$. If $I \Vdash P$ and $\Vdash T$, and $\mathsf{fn}(T) \cap \mathsf{bn}(P) = \emptyset$ then $I \Vdash [T/A]P$.*

The relation $\sqsubseteq_{\mathcal{R}}$ given by

$$U \sqsubseteq_{\mathcal{R}} V \ \Leftrightarrow\ \forall I\,(I \Vdash U \ \Rightarrow\ I \Vdash V)$$

is clearly a precongruence. Let $=_{\mathcal{R}}$ be the associated equivalence. Then by an easy induction on the definition of $=_\alpha$, we derive:

Corollary 7 *The relation $=_{\mathcal{R}}$ contains the relation $=_\alpha$ of α-conversion.*

In the proof of subject reduction we rely on the following property.

Lemma 8 *The relation $\sqsubseteq_{\mathcal{R}}$ contains the structural equivalence relation $\equiv$.*

PROOF. Since $\sqsubseteq_{\mathcal{R}}$ is a precongruence containing the relation of α-conversion, it is enough to show that for each axiom $U \equiv V$ we have $U \sqsubseteq_{\mathcal{R}} V$ and $V \sqsubseteq_{\mathcal{R}} U$. This is trivial for the associativity and commutativity of parallel composition, and for the "routing" axiom $[\ell :: P \mid Q] \equiv [\ell :: P] \mid [\ell :: Q]$.

Let $I \Vdash ((\nu w)U \mid V)$ with $\mathsf{subj}(w) \notin \mathsf{fn}(V)$. By the corollary 7 we may assume that $\mathsf{subj}(w)$ does not occur in I. Let us examine the case where $w = u$. We have

$$\dfrac{\dfrac{\dfrac{\vdots}{u\,,\, I_0 \Vdash U}}{I_0 \Vdash (\nu u)U} \qquad \dfrac{\vdots}{I_1 \Vdash V}}{I_0\,,\, I_1 \Vdash ((\nu u)U \mid V)}\; I_0 \cap I_1 = \emptyset$$

where $I_0, I_1 = I$. By the remark 5 we have $u \notin I_1$, therefore $I \Vdash (\nu u)(U \mid V)$. The other cases are similar, as well as the symmetric case where the hypothesis is $I \Vdash (\nu w)(U \mid V)$. Let us now examine the case where $I \Vdash [\ell :: (\nu u)P]$. We have

$$\dfrac{\dfrac{\dfrac{\vdots}{u\,,\, I' \Vdash P}}{I' \Vdash (\nu u)P}}{I \Vdash [\ell :: (\nu u)P]}\; \{\, a \mid a, a@\ell \in I' \,\} = \emptyset$$

where $\mathsf{subj}(u) \notin \mathsf{nm}(I')$ and $I = I'@\ell$. Then we also have $\{\, a \mid a, a@\ell \in u, I' \,\} = \emptyset$, and therefore the following is a valid inference:

$$\dfrac{\dfrac{\dfrac{\vdots}{u\,,\, I' \Vdash P}}{u@\ell\,,\, I'@\ell \Vdash [\ell :: P]}}{I \Vdash (\nu u@\ell)[\ell :: P]}$$

The symmetric case, where the hypothesis is $I \Vdash (\nu u@\ell)[\ell :: P]$, is easy. ❏

Proposition 9 (preservation of well-formedness) $U \to V \;\Rightarrow\; U \sqsubseteq_{\mathcal{R}} V$.

Proof. By induction on the definition of $U \to V$. It is enough to consider the axioms of reduction. The cases of $[\ell' :: \mathsf{go}\,\ell.P] \to [\ell :: P]$, $[\ell = \ell]P, Q \to P$ and $[\ell = \ell']P, Q \to Q$ are trivial. For the case of $(\overline{a}(\vec{v}) \mid a(\vec{u}).P) \to [\vec{v}/\vec{u}]P$, we have

$$\dfrac{\dfrac{}{\Vdash \overline{a}(\vec{v})} \qquad \dfrac{\dfrac{\vdots}{a \Vdash P}}{a \Vdash a(\vec{u}).P}}{a \Vdash \overline{a}(\vec{v}) \mid a(\vec{u}).P}$$

with $a \notin \mathsf{nm}(\vec{u})$. Then by the lemma 6 we also have $a \Vdash [\vec{v}/\vec{u}]P$. For the unfolding case, that is $T(\vec{v}) \to [T/A][\vec{v}/\vec{u}]P$ where $T = (\mathsf{rec}\, A(\vec{u}).P)$, we have

$$\dfrac{\dfrac{\dfrac{\vdots}{a' \Vdash P}}{\Vdash (\mathsf{rec}\, A(a', \vec{u'}).P)}}{a \Vdash (\mathsf{rec}\, A(a', \vec{u'}).P)(a, \vec{v'})}$$

where $\vec{u} = a', \vec{u'}$ and $\vec{v} = a, \vec{v'}$, and we conclude again by the lemma 6. ❏

5 Receptivity and message deliverability

The receptivity property states that if a name is in the interface of a (well-formed) process, then this process offers a resource (a receiver) for that name. This resource may not be immediately available, but one shows that some normalizing computation will reveal it. To express this formally, let us define the predicates $U \downarrow u$ and $U \not\downarrow u$ inductively as follows – recall that $U \twoheadrightarrow V$ is reduction without communication:

$$\frac{}{a(\vec{u}).P \downarrow a} \qquad \frac{U \downarrow u}{(U \mid V) \downarrow u} \qquad \frac{V \downarrow u}{(U \mid V) \downarrow u} \qquad \frac{U \downarrow u}{(\nu w)U \downarrow u}\ \mathsf{nm}(u) \cap \mathsf{subj}(w) = \emptyset$$

$$\frac{S \downarrow a@\ell}{(\nu \ell' : \psi)S \downarrow a@\ell}\ a \neq \ell' \neq \ell \qquad \frac{P \not\downarrow u}{\mathsf{go}\,\ell.P \downarrow u@\ell} \qquad \frac{P \downarrow u}{[\ell :: P] \downarrow u@\ell} \qquad \frac{U \overset{*}{\twoheadrightarrow} V\ ,\ V \downarrow u}{U \not\downarrow u}$$

The only non-standard clause in this definition is the one for $\mathsf{go}\,\ell.P$: although this is a process that does not reduce, we must anticipate the fact that after having migrated, P will offer some input. It is easy to see that if $U \downarrow u$ and $V \equiv U$ then $V \downarrow u$, and that

$$\begin{array}{rcll} P \downarrow a & \Leftrightarrow & P \equiv (\nu \vec{w})(a(\vec{u}).R \mid Q) & a \notin \mathsf{subj}(\vec{w}) \\ S \downarrow a@\ell & \Leftrightarrow & S \equiv (\nu \vec{w})([\ell' :: P] \mid S') & P \downarrow v,\ v@\ell' = a@\ell \text{ and } a, \ell \notin \mathsf{subj}(\vec{w}) \end{array}$$

The following lemma states that, after a preliminary phase where some migration can occur, all the receivers that are exhibited in a system are immediately available.

Lemma 10 *If $P \downarrow a@\ell$ then there exists P' such that $P' \downarrow a$ and $[\ell' :: P] \overset{*}{\twoheadrightarrow} (\nu \vec{w})([\ell :: P'] \mid S)$ for some S, with $a \notin \mathsf{subj}(\vec{w})$.*

Proof. By induction on the definition of $P \downarrow a@\ell$. If $P = \mathsf{go}\,\ell''.Q$ with $Q \not\downarrow u$ and $u@\ell'' = a@\ell$, we have $[\ell' :: P] \twoheadrightarrow [\ell'' :: Q]$, and by definition there exists P' such that $Q \overset{*}{\twoheadrightarrow} P'$ and $P' \downarrow u$. Then we have $[\ell' :: P] \overset{*}{\twoheadrightarrow} [\ell'' :: P']$, and there are two cases: if $u = a$ and $\ell'' = \ell$, then we are done; if $u = a@\ell$ then we use the induction hypothesis for $[\ell'' :: P']$. If $P = (Q \mid R)$ with, for instance, $Q \downarrow a@\ell$ then $[\ell' :: P] \equiv [\ell' :: Q] \mid [\ell' :: R]$, and we use the induction hypothesis. If $P = (\nu w)Q$ with $Q \downarrow a@\ell$ and $a \notin \mathsf{subj}(w)$ we have $[\ell' :: P] \equiv (\nu w@\ell')[\ell' :: Q]$, and we use the induction hypothesis. □

Proposition 11 (receptivity) *Let U be a well-formed, typable term. Then:*
(i) *if $I \Vdash U$ then $u \in I \Leftrightarrow U \not\downarrow u$,*
(ii) *if $U \not\downarrow u$ and $U \rightarrow V$ then $V \not\downarrow u$.*

Proof.
(i)($\Rightarrow$) We prove a more general statement, namely that $u, I \Vdash U \ \Rightarrow [\mathsf{S}]U \not\downarrow \mathsf{S}(u)$ for any substitution S such that $[\mathsf{S}]U$ is defined. We proceed by induction on U. Most of the cases are trivial (using the axioms of structural equivalence); let us only examine the ones where some reduction is needed (notice that in the case of $\mathsf{go}\,\ell.P$ we see why a non standard rule for $\downarrow u$ is needed). If U is $[\ell = \ell']Q, R$, then the inference of $u, I \Vdash U$ has the form

$$\dfrac{\dfrac{\vdots}{u\,,\, I \Vdash Q} \qquad \dfrac{\vdots}{u\,,\, I \Vdash R}}{u\,,\, I \Vdash [\ell = \ell']Q, R}$$

then one has $[\mathsf{S}]([\ell = \ell']Q, R) = [\mathsf{S}(\ell) = \mathsf{S}(\ell')][\mathsf{S}]Q, [\mathsf{S}]R$, and therefore $[\mathsf{S}]([\ell = \ell']Q, R) \rightarrow [\mathsf{S}]Q$ or $[\mathsf{S}]([\ell = \ell']Q, R) \rightarrow [\mathsf{S}]R$, and one uses the induction hypothesis. If $U = T(a, \vec{u})$ and $I = \emptyset$, we have $T = (\mathsf{rec}\,A(b, \vec{v}).Q)$ since U is guarded, and $\vec{v}$ and $\vec{u}$ have the same length since U is typable. Then the inference $a \Vdash U$ has the form

$$\dfrac{\dfrac{\dfrac{\vdots}{b \Vdash Q}}{\Vdash (\mathsf{rec}\,A(b, \vec{v}).Q)}}{a \Vdash (\mathsf{rec}\,A(b, \vec{v}).Q)(a, \vec{u})}$$

We have $[\mathsf{S}]((\mathsf{rec}\,A(b,\vec{v}).Q)(a,\vec{u})) = T'(c,\vec{w})$ with $T' = (\mathsf{rec}\,A(b,\vec{v}).[\mathsf{S}']Q)$ where $\mathsf{S}' = \mathsf{S}\!\restriction \mathsf{dom}(\mathsf{S}) - \{b\} \cup \mathsf{nm}(\vec{v})$, and $(c,\vec{w}) = [\mathsf{S}](a,\vec{u})$, and therefore

$$[\mathsf{S}]((\mathsf{rec}\,A(b,\vec{v}).Q)(a,\vec{u})) \to [T'/A][c,\vec{w}/b,\vec{v}][\mathsf{S}']Q$$

Clearly Q is guarded and typable, and therefore by induction hypothesis $[c,\vec{w}/b,\vec{v}][\mathsf{S}']Q \not\downarrow c$, hence also $[\mathsf{S}]((\mathsf{rec}\,A(b,\vec{v}).Q)(a,\vec{u})) \not\downarrow \mathsf{S}(a)$ by the lemma 14.

(i)($\Leftarrow$) we prove that $I \Vdash U$ and $U \downarrow u$ or $U \not\downarrow u$ implies $u \in I$, by induction on the definition of the predicates $U \downarrow u$ and $U \not\downarrow u$. The only case that deserves some consideration is $U \not\downarrow u$ with $U \overset{*}{\twoheadrightarrow} V$ and $V \downarrow u$. By the proposition 9 we have $I \Vdash V$ (and V is typable by the proposition 4), and by induction hypothesis this implies $u \in I$.

(ii) assume that $I \Vdash U$. Then by the previous point $u \in I$. By the propositions 4 and 9, V is typable and $I \Vdash V$, which, by (i)($\Rightarrow$), implies $V \not\downarrow u$. ❑

An immediate consequence of this proposition is that we statically know the locality of each receiver in a term: for a public name, it is indicated in the interface – the locality of a receiver on a is the current one if $a \in I$, and is ℓ if $a@\ell \in I$ (as we shall see with the "distributed object" example below, the same channel may have several receivers, though in different locations). For a subterm $(\nu u)P$ of a well-formed process, we know that u is in the interface of P. Note however that in $(\nu a@\ell)P$, the location name ℓ is free, and may be received as a parameter from a message, see our last example of a "migrating cell" in the next section.

This result suggests the denomination "*distributed (asynchronous) receptive π-calculus, with unique receivers*", in short the $\mathrm{D}\pi_1^r$-calculus, for the set of well-formed closed processes, which is closed by reduction. Similarly we call π_1^r, that is "*the receptive π_1-calculus*", the sub-calculus where we do not use any locality based feature. Another immediate consequence of receptivity is:

Corollary 12 *If $a(\vec{u}).P$ is well-formed and typable, then $P \not\downarrow a$.*

This property says that a receiver is *persistent*, in the sense that it will still be available, under the same name, whatever messages are sent to it. Therefore we may call such a receiver a "server", that always accept requests (i.e. messages), or a "resource". We now turn to the issue of message delivery, showing that, if a message is sent (at some locality) on a channel of a known scope in a well-formed and typed process, then the process offers a receiver for this message.

Theorem 13 (message deliverability) *Let S be a well-formed and typable term, with $I \Vdash S$. If $S \overset{*}{\to} (\nu\vec{w})([\ell :: \overline{a}(\vec{v})] \mid S_0)$ with $a@\ell{\in}I$ or $a{\in}\mathsf{subj}(\vec{w})$, then there exist R and S_1 such that $S_0 \overset{*}{\twoheadrightarrow} (\nu\overrightarrow{w'})([\ell :: a(\vec{u}).R] \mid S_1)$ with $a \notin \mathsf{subj}(\overrightarrow{w'})$.*

Proof. By the proposition 9, we have $I \Vdash (\nu\vec{w})([\ell :: \overline{a}(\vec{v})] \mid S_0)$ with $a@\ell \in I$ or $a \in \mathsf{subj}(\vec{w})$. We first show by induction on the length of $\vec{w}$ that this implies $S_0 \not\downarrow a@\ell$. If this length is 0 we have $I \Vdash S'$ and $a@\ell \in I$, and we conclude using the proposition 11. Otherwise $\vec{w} = w, \overrightarrow{w_0}$. Let us assume that $a = \mathsf{subj}(w)$ and $a \notin \mathsf{subj}(\overrightarrow{w_0})$ (otherwise $a@\ell \in I$ or $a \in \mathsf{subj}(\overrightarrow{w_0})$ and we simply use the induction hypothesis). Since S is typed, say with $\Gamma \vdash S$, we also have $\Gamma \vdash (\nu\vec{w})([\ell :: \overline{a}(\vec{v})] \mid S_0)$ by the theorem 4. We have $w = a@\ell'$, and

$$\cfrac{\cfrac{\vdots}{\ell' : \{a : \gamma\}\,,\, \Gamma \vdash (\nu\overrightarrow{w_0})([\ell :: \overline{a}(\vec{v})] \mid S_0)}}{\Gamma \vdash (\nu a@\ell')(\nu\overrightarrow{w_0})([\ell :: \overline{a}(\vec{v})] \mid S_0)}$$

with a not in Γ. Since a is free in $(\nu\overrightarrow{w_0})([\ell :: \overline{a}(\vec{v})] \mid S_0)$, and the typing of $[\ell :: \overline{a}(\vec{v})]$ has the form

$$\cfrac{\cfrac{\cfrac{\vdots}{\Delta' \vdash_\ell^W \vec{v} : \vec{\tau}}}{\ell : \{a : Ch(\vec{\tau})\}, \Delta' \vdash_\ell \overline{a}\vec{v}}}{\Delta \vdash [\ell :: \overline{a}\vec{v}]}$$

with $\ell : \{a : Ch(\vec{\tau})\}, \Delta' = \Delta$, the context $\ell' : \{a : \gamma\}, \Gamma$ must contain the assumption $\ell : \{a : Ch(\vec{\tau})\}$. Since a do not occur in Γ, we have $\ell' = \ell$, and $\gamma = Ch(\vec{\tau})$. Then we have

$$\dfrac{\dfrac{\vdots}{a@\ell\,,\, I \Vdash (\nu\overrightarrow{w_0})([\ell :: \overline{a}(\vec{v})] \mid S_0)}}{I \Vdash (\nu a@\ell)(\nu\overrightarrow{w_0})([\ell :: \overline{a}(\vec{v})] \mid S_0)}$$

therefore $S_0 \not\Downarrow a@\ell$ by induction hypothesis.

Now by definition $S_0 \overset{*}{\twoheadrightarrow} S_1$ for some S_1 such that $S_1 \downarrow a@\ell$, that is $S_1 \equiv (\nu\overrightarrow{w_1})([\ell' :: P] \mid S_2)$ with $P \downarrow v$, $v@\ell' = a@\ell$ and $a, \ell \notin \mathsf{subj}(\overrightarrow{w_1})$. Then we have either $P \downarrow a$ and $\ell' = \ell$, or $P \downarrow a@\ell$. In the first case we have

$$S_1 \equiv (\nu\overrightarrow{w_1})([\ell :: (\nu\overrightarrow{w_2})(a(\vec{u}).R \mid Q)] \mid S_2) \equiv (\nu\overrightarrow{w_3})([\ell :: a(\vec{u}).R] \mid ([\ell :: Q] \mid S_2))$$

with $a \notin \mathsf{subj}(\overrightarrow{w_1}, \overrightarrow{w_2}, \overrightarrow{w_3})$, concluding the proof in that case. If $P \downarrow a@\ell$ then by the lemma 10 there exists P' such that $P' \downarrow a$ and $[\ell' :: P] \overset{*}{\twoheadrightarrow} (\nu\overrightarrow{w_2})([\ell :: P'] \mid S_3)$ for some S_3, with $a \notin \mathsf{subj}(\overrightarrow{w_2})$, and we conclude as in the previous case. □

The proof of this theorem emphasizes the need of our type system. For instance, although the term $(\nu a@\ell')([\ell :: \overline{a}] \mid [\ell' :: Id_a])$ – where $Id_a = (\mathsf{rec}\, A(a).a.(\overline{a} \mid A(a)))(a)$ which is well formed with interface $\{a\}$ – is well-formed with an empty interface, it clearly does not satisfy the message deliverability. Indeed, this term is not typable because it uses the name a at ℓ, whereas it should (regarding the restriction) only use it at ℓ'. A similar example is $(\nu b@\ell)[\ell :: \overline{a}b \mid a(x).(\mathsf{go}\,\ell'.\overline{x} \mid Id_a) \mid Id_b]$, which is well-formed, but not typable because the channel a and its argument x cannot be typed at the same locality.

6 Receptive distributed programming

In this section we give some examples to illustrate the style of "programming" that one must adopt to conform to the receptive discipline. For the non-distributed case we refer to [4], where the reader may find standard examples of synchronisation (buffers, mutual exclusion) in the receptive style. Here we focus on examples involving spatial distribution and migration. In the following examples we use the standard notation $\mathsf{if}\ k = k'\ \mathsf{then}\ P\ \mathsf{else}\ Q$ – and more generally $\mathsf{if}\ C\ \mathsf{then}\ P\ \mathsf{else}\ Q$ – for $[k = k']P, Q$. We denote by $\overline{a}(\vec{v}, \_, \vec{v'})$ the term $(\nu b)(\overline{a}(\vec{v}, b, \vec{v'}) \mid T_b)$, or $(\nu b : loc)\overline{a}(\vec{v}, b, \vec{v'})$, depending on the type of a. We recall that we write a recursive process $(\mathsf{rec}\, A(\vec{u}).P)(\vec{u})$ that does not introduce new parameters simply as $\mathsf{rec}\, A(\vec{u}).P$.

6.1 Forwarders

We begin with a very simple example of process that repeatedly receives messages on some channel and forwards their content to another channel at some locality ℓ – recall that $\overline{x}@y() = \mathsf{go}\, y.\overline{x}()$:

$$Fwd(a, b@\ell) = a^*(\vec{x}).\overline{b}@\ell(\vec{x})$$

This is typable at any current locality ℓ' which holds the channel a, assuming that the x's are values, that is of type loc, as follows:

$$\ell' : \{a : Ch(\overrightarrow{loc})\}\,,\, \ell : \{b : Ch(\overrightarrow{loc})\} \vdash_{\ell'} Fwd(a, b@\ell)$$

The forwarder is also clearly well-formed, with interface $\{a\}$. Such a process is similar to the "link" of [14].

6.2 An object server

Suppose we have written a generic "object" $obj(b, \vec{d})$, with name b – that is, $b \Vdash obj(b, \vec{d})$ – and state parameters $\vec{d}$, and that we would like to create at some location ℓ_0 a server for that kind of objects, which will deliver instances of it on requests to a name s – this is similar to *code on demand*. Then we write this as follows, assuming that the parameters $\vec{d}$ are values:

$$s^*(c@\ell, \vec{d}).\mathsf{go}\,\ell.(\nu b)(\overline{c}(b) \mid obj(b, \vec{d}))$$

A client at site ℓ_1 that wants to acquire its own instance of the object, and use it in the process P will be written:

$$(\nu c)(\,\overline{s}@\ell_0(c@\ell_1, \vec{d}) \;\mid\; c(b){:}P\,)$$

We let the reader check that, if we can type $obj(b, \vec{d})$ with type ζ for b, then these terms are typable with $\ell_0 : \{s : Ch(Ch(\zeta)@, \overrightarrow{loc})\}$.

6.3 A distributed object

In this example we imagine an "object" which is located both at ℓ_0 and ℓ_1 (with the same name), and has a shared, possibly distributed state. More specifically, we program a distributed "button", named a. "Pushing the button" at ℓ_i means sending a message to the channel a at ℓ_i. Pushing twice the button at either ℓ_0 or ℓ_1 results in the emission of a message c at ℓ. In particular, pushing once at ℓ_0 and once at ℓ_1 will produce an emission. This condition obviously forces some cooperation between the two locations. We seek a solution that will allow emission even when one of the two locations stops working. This condition rules out centralized solutions where emissions are always decided at one location. On the other hand, some degree of asymmetry seems desirable to rule out a circular situation where each location tries to "borrow a push" from the other location in order to produce an emission. In the presented solution, the button at location ℓ_0 is allowed to borrow from the one at ℓ_1, but not the other way round. The distributed button relies on a private value b – to borrow a push:

$$S = (\nu b : loc)\big([\ell_0 :: P_0] \mid [\ell_1 :: P_1]\big)$$

where

$$\begin{array}{rcl} P_0 & = & \mathsf{rec}\, A(a).a(i).\big(\overline{a}@\ell_1(b) \mid a(i).(A(a) \mid \overline{c}@\ell())\big) \\ P_1 & = & \mathsf{rec}\, A'(a).a(i).\mathsf{if}\ i = b\ \mathsf{then}\ (A'(a) \mid \overline{a}(b)) \\ & & \qquad\qquad \mathsf{else}\ a(i).\mathsf{if}\ i = b\ \mathsf{then}\ (A'(a) \mid \overline{a}@\ell_0(b)) \\ & & \qquad\qquad\qquad\qquad \mathsf{else}\ (A'(a) \mid \overline{c}@\ell()) \end{array}$$

It is easy to see that these terms are well-formed, with interface $\{a\}$, and therefore $a@\ell_0, a@\ell_1 \Vdash P$, and typable with $\ell_0, \ell_1 : \{a : Ch(loc)\}$. The message $\overline{a}@\ell_1(b)$ is interpreted by P_1 as a request to borrow a push, while the message $\overline{a}@\ell_0(b)$ is interpreted by P_0 as a push lent by P_1. This solution could be refined. For instance, the processes may want to check that the request to borrow received by the process at ℓ_1 corresponds to the *current* request to borrow from the process at ℓ_0 (if it is not the case then the offer to lend is declined). Similar, but more complicated techniques could be applied for the programming of a distributed channel manager or, more generally, of a distributed memory.

Notice that to fix the scope of the name of such a distributed object, we should extend the syntax to allow $(\nu a@L)P$, where L is a non-empty finite set of (location) names. It is then straightforward to modify the type system and the system for well-formedness to deal with this extension.

6.4 Migrating cells

In this examples, we adopt a notation similar to the one for "objects" of TyCO [19], that is we write

$$a\{k_1 = (\vec{u}_1)P_1, \ldots, k_n = (\vec{u}_n)P_n, Q\}$$

for the process

$$\begin{array}{l} a(k, \vec{u}_1, \ldots, \vec{u}_n).\ [k = k_1]\, P_1, \\ \qquad\qquad \vdots \\ \qquad\qquad [k = k_n]\, P_n, Q \end{array}$$

where the names k_i are assumed to be values (of type loc). This is well-formed if $a \Vdash P_i$ for all i, and also $a \Vdash Q$. Such an agent behaves as an "object", or more accurately as an "actor" of name a, offering "methods" $k_1, \ldots, k_n$, and possibly changing its behaviour – but not its identity – after having received a message. The messages for such an agent may be written $a \triangleleft k_i(\vec{u}_i)$, which abbreviates $\overline{a}(k_i, \_, \ldots, \vec{u}_i, \ldots, \_)$. Admittedly, in a more language oriented version of the $\mathrm{D}\pi_1^{\tau}$-calculus it would be preferable to adopt as primitive the notations (and typing) of TyCO; we just use them for the sake of illustration here. In many cases these objects have a simple recursive behaviour, given as follows:

$$\begin{array}{l} \mathsf{rec}\, A(a, \vec{v}).a(k, \vec{u}_1, \ldots, \vec{u}_n).\ [k = k_1]\, (P_1 \mid A(a, \vec{v})), \\ \qquad\qquad\qquad \vdots \\ \qquad\qquad\qquad [k = k_n]\, (P_n \mid A(a, \vec{v})), A(a, \vec{v}) \end{array}$$

where $\Vdash P_i$ for all i. Such an agent reacts uniformly to the "method calls", by performing the corresponding process P_i and returning itself.

A *cell* is a process that stores some value (of type *loc*), that can be read or updated. To make this example more interesting, we suppose that the cell not only responds to requests for reading and writing, but also to an instruction to *migrate* to some given location. The name c of the cell has type $\gamma = Ch(loc, Ch(loc), loc, Ch(), loc)$, where the first argument is the key for reading, writing and migrating, the second and fourth arguments are return channels, for readers and writers respectively, the third argument is the content of the cell, and the last is the destination locality in case of migration. To maintain some consistency, the cell moved at a new location bears a new name, which is used to forward messages arriving at the location it originates from. Notice that if we keep the same name our example is not well-formed. The migrating cell $MigCell(c, x, \ell)$ has parameters x and ℓ, which are its current value and location; it is the following term:

$$\begin{aligned} MigCell(c, x, \ell) = \mathsf{rec}\, A(c, x, \ell).c\{\ &\mathsf{read} = (y)(\overline{y}(x) \mid A(c, x, \ell))\,, \\ &\mathsf{write} = (x', z)(\overline{z}() \mid A(c, x', \ell))\,, \\ &\mathsf{migr} = M\,,\ A(c, x, \ell)\} \end{aligned}$$

where

$$\begin{aligned} M &= (\ell')(\nu c'@\ell')\big(\mathsf{go}\,\ell'.A(c', x, \ell') \mid Proxy\big) \\ Proxy &= c^*(k, y, x, z, t).\mathsf{go}\,\ell'.(\nu y')\big(\,Fwd(y', y@\ell) \mid \\ &\qquad (\nu z')\big(\,Fwd(z', z@\ell) \mid \\ &\qquad\qquad \overline{c'}(k, y', x, z', t)\,\big)\big) \end{aligned}$$

where, as we have seen, the "forwarder" is

$$Fwd(a, b@\ell) = a^*(\vec{x}).\overline{b}@\ell(\vec{x})$$

In case a read operation, that is $\overline{c} \triangleleft \mathsf{read}(y)$, has to be processed, the cell returns its current value x on the channel y to the reader. In case of a write operation $\overline{c} \triangleleft \mathsf{write}(x', z)$, the cell is updated with the new value x', and an acknowledgement z is sent back to the writer. Notice that readers and writers are always supposed to be local processes, or more precisely processes at the same locality as the cell. Finally a migration message $\overline{c} \triangleleft \mathsf{migr}(\ell')$ causes the cell to be *moved* at the destination locality ℓ', with a new name c', its current content x and a new current location ℓ', while a "proxy" managing the calls to c is set up at the locality that the cell is leaving.

The reader can check that $MigCell(c, x, \ell)$ is both well-formed, with interface $\{c\}$, and typable, with c of type γ at the current location ℓ, and x of type *loc*. In this version, the remote cell can only be accessed by messages emitted in the location it originated from. We could also make the new cell available at its new location, on a name a received as argument, that is changing M into:

$$M = (a@\ell')(\nu c'@\ell')\big(\mathsf{go}\,\ell'.(\overline{a}(c') \mid A(c', x, \ell')) \mid Proxy\big)$$

Then a process using this cell and wishing to migrate together with this part of its state would typically execute:

$$Q = (\nu a@\ell')\big(\overline{c} \triangleleft \mathsf{migr}(a@\ell') \mid \mathsf{go}\,\ell'.a(c'){:}P\big)$$

It is easy to see that we have the following behaviour:

$$[\ell :: MigCell(c, x, \ell) \mid Q \mid R] \xrightarrow{*} (\nu c'@\ell')\big([\ell :: Proxy \mid R] \mid [\ell' :: MigCell(c', x, \ell') \mid P \mid (\nu a)\,T_a]\big)$$

However, to type this new version, we need recursive types. These are introduced by adding type variables and recursion, as follows:

$$\tau ::= \ \cdots \ \mid \ t \ \mid \ \mu t.\tau$$

These are used in the type system as usual, that is, identifying a recursive type with the (infinite, regular) tree it determines (notice that there is also an implicit notion of equality of location types, that may be regarded as record types). Now if we let

$$\delta = \mu t.Ch(loc, Ch(loc), loc, Ch(), Ch(t)@)$$

we can type the new migrating cell with c of type δ.

Our migrating cell – say, the second version, with recursive types –, is only one possible kind of "state mobility". In [17] Sekiguchi and Yonezawa identify several other "mobility types". Our previous example is of what they call "carry type", where a migrating user of the cell carries it with him, while leaving access to the cell by means of a proxy. The "copy type", where the cell is duplicated, with no relation between the two copies, is easily written, by just changing the code of the migration method, as follows:

$$M_{copy} = (a@\ell')\big(\mathsf{go}\,\ell'.(\nu c')(\overline{a}(c') \mid A(c',x,\ell')) \mid A(c,x,\ell)\big)$$

Similarly, a migrating cell of "resident type" – the "dual" of "carry type" –, which stays in its location but may be accessed remotely via a proxy, is written:

$$M_{resident} = (a@\ell')\big(\mathsf{go}\,\ell'.(\nu c')(\overline{a}(c') \mid \mathit{Proxy}') \mid A(c,x,\ell)\big)$$

where Proxy' is the "dual" of Proxy, accepting requests on c' at ℓ' and sending them back to c at ℓ. There are two other mobility types, called "proper" and "takeaway" in [17], where the link of the migrating (resp. immobile) user with the cell is lost, that do not seem to be expressible in our calculus, where any request may be received. However, we may use the "fake receiver" T_a to simulate the corresponding migrating cells:

$$\begin{array}{rcl} M_{proper} & = & (a@\ell')\big(\mathsf{go}\,\ell'.(\nu c')(\overline{a}(c') \mid T_{c'}) \mid A(c,x,\ell)\big) \\ M_{takeaway} & = & (a@\ell')\big(\mathsf{go}\,\ell'.(\nu c')(\overline{a}(c') \mid A(c',x,\ell')) \mid T_c\big) \end{array}$$

A more realistic implementation would raise exceptions, rather than just throw away messages. In this particular example, we see that the discipline of receptive programming imposes that we cannot move a resource without providing the locality it is leaving with a replacement behaviour. Then we cannot actually (re)move a resource: what we may do is dynamically change its behaviour, and create new remote resources.

This example suggests that we could use our distributed π-calculus as a low-level "assembly" language, in which to encode higher-level migrating agents that move with part of their state. We could also use our calculus to encode more abstract migration schemes, like for instance "transparent routing", which is assumed in the Join-calculus [8]. More specifically, we may assume that we have a set of "global names", whose locations are known to the implementation by means of a mapping $\mathcal{L}$. Then we would like to program with these names without having to explicitly move the corresponding messages to their destination, writing simply $\overline{a}(\vec{u})$ where we actually mean $\overline{a}@?(\vec{u})$. Then the "implementation" of this transparent routing should be clear: given the mapping $\mathcal{L}$ assigning localities to "global names", we would translate an input as $[\![a(\vec{b}).P]\!] = a(\overrightarrow{b@\ell_b})[\![P]\!]\mathcal{L}[\overrightarrow{b \mapsto \ell_b}]$ – assuming that all the received names are global ones –, and similarly for the output $[\![\overline{a}(\vec{b})]\!] = a@\mathcal{L}(a)(\overrightarrow{b@\mathcal{L}(b)})$.

7 Conclusion

In the area of sequential programming, one is usually not concerned with receptivity, since the "resources" – what can be found in the heap or the memory – are assumed to be always available, although with a possibly mutable value. However this is only true in high-level languages, where no explicit manipulation of the memory is allowed. In low-level languages, like C, with a "free" instruction, one is potentially confronted with run-time errors where a program is trying to access to a non-existent resource. In this paper we have taken the view that, even for a low-level model like the π-calculus, it is a good discipline to "program" with persistent resources, which may always react in some way to requests – whether the reaction is the expected one is a matter of semantics, but the programmer of a service for instance is compelled to code some reaction in any state of this service.

References

[1] R. Amadio, *An asynchronous model of locality, failure, and process mobility*, In Proc. COORDINATION'97, Springer Lect. Notes in Comp. Sci. 1282 (1997). Extended version appeared as Res. Report INRIA 3109.

[2] R. Amadio, *On modeling mobility*, Journal of Theoretical Computer Science Vol. 240, No. 1 (2000) 147-176.

[3] R. Amadio, G. Boudol, and C. Lhoussaine, *The distributed receptive π-calculus*, Technical report, INRIA Research Report 4080, November 2000.

[4] R. Amadio, G. Boudol, and C. Lhoussaine, *On message deliverability and non-uniform receptivity*, submitted for publication, February 2002.

[5] G. Boudol, *Typing the use of resources in a concurrent calculus*, Proc. ASIAN 97, Springer Lect. Notes in Comp. Sci. 1345 (1997) 239-253.

[6] L. Cardelli and A. Gordon, *Mobile ambients*, In Proc. FoSSaCS, ETAPS 98, Springer Lect. Notes in Comp. Sci. 1378 (1998) 140-155.

[7] C. Fournet and G. Gonthier, *The reflexive CHAM and the join-calculus*, In Proc. ACM Principles of Prog. Lang. (1996) 372-385.

[8] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy, *A calculus of mobile agents*, In Proc. CONCUR'96, Springer Lect. Notes in Comp. Sci. 1119 (1996) 406-421.

[9] M. Hennessy and J. Riely, *Resource access control in systems of mobile agents*, Information and Computation, Vol. 173 (2002) 82-120. First appeared as TR 98/02, University of Sussex.

[10] L. Jategaonkar and J. Mitchell *Type inference with extended pattern matching and subtypes*, Fundamenta Informaticae Vol. 19 (1993) 127-166.

[11] N. Kobayashi, *A partially deadlock-free typed process calculus*, ACM ToPLaS Vol. 20 No 2 (1998) 436-482.

[12] N. Kobayashi, B. Pierce and D. Turner *Linearity and the π-calculus* ACM ToPLaS Vol. 21, No. 5 (1999) 914-947.

[13] C. Lhoussaine *Réceptivité, Mobilité et π-Calcul*. PhD Thesis, Univ. Aix-Marseille I, 2002 (Forthcoming).

[14] M. Merro and D. Sangiorgi, *On asynchrony in name-passing calculi*, In Proc. ICALP'98, Springer Lect. Notes in Comp. Sci. 1443 (1998) 856-867.

[15] D. Remy, *Type inference for records in a natural extension of* ML, in Theoretical aspects of object-oriented programming. Types, semantics, and langugage design. C. Gunter, J. Mitchell (eds). Academic Press, 1993.

[16] D. Sangiorgi, *The name discipline of uniform receptiveness*, In Proc. ICALP'97, Springer Lect. Notes in Comp. Sci. 1256 (1997) 303-313.

[17] T. Sekiguchi and A. Yonezawa, *A calculus with code mobility* FMOODS'97, Chapman & Hall (1997).

[18] E. Sumii and N. Kobayashi, *A generalized deadlock-free process calculus*, In Proc. HLCL'98, Electronic Notes in Comp. Sci. vol. 16-3 (1998).

[19] V. Vasconcelos, *Typed concurrent objects* In Proc. ECOOP'94, Springer Lect. Notes in Comp. Sci. 821 (1994) 100-117.

[20] M. Wand, *Complete type inference for simple objects*, in Proc. IEEE-LICS, 1987.

A Substitutions

We recall that a name substitution is a mapping S from a finite subset $\mathsf{dom}(\mathsf{S})$ of $\mathcal{N}$ into $\mathcal{N}$, and that the result of applying S to a term U is denoted $[\mathsf{S}]U$, which is only defined if $\mathsf{im}(\mathsf{S}) \cap \mathsf{bn}(U) = \emptyset$. Since we only deal with capture-free substitutions, $[\mathsf{S}]U$ is trivially defined; let us just give the cases of the binding constructs, where by convention $\mathsf{S}(a) = a$ if $a \notin \mathsf{dom}(\mathsf{S})$:

$$\begin{array}{rcll}
[\mathsf{S}]a(\vec{u}).P & = & \mathsf{S}(a)(\vec{u}).[\mathsf{S}']P & \mathsf{S}' = \mathsf{S}\lceil(\mathsf{dom}(\mathsf{S}) - \mathsf{nm}(\vec{u}))\\
[\mathsf{S}](\nu w)P & = & \left\{\begin{array}{ll} (\nu a)[\mathsf{S}']P & \text{if } w = a\\ (\nu a@\mathsf{S}(\ell))[\mathsf{S}']P & \text{if } w = a@\ell\\ (\nu \ell : \mathsf{S}(\psi))[\mathsf{S}']P & \text{if } w = \ell : \psi \end{array}\right\} & \text{where } \mathsf{S}' = \mathsf{S}\lceil(\mathsf{dom}(\mathsf{S}) - \mathsf{subj}(w))\\
[\mathsf{S}](\mathsf{rec}\, A(\vec{u}).P) & = & (\mathsf{rec}\, A(\vec{u}).[\mathsf{S}']P) & \mathsf{S}' = \mathsf{S}\lceil(\mathsf{dom}(\mathsf{S}) - \mathsf{nm}(\vec{u}))\\
[\mathsf{S}](\nu \ell : \psi)S & = & (\nu \ell : \mathsf{S}(\psi))[\mathsf{S}']S & \mathsf{S}' = \mathsf{S}\lceil(\mathsf{dom}(\mathsf{S}) - \{\ell\})
\end{array}$$

where $\mathsf{S}(\psi)$ is defined in the obvious way. Similarly, the substitution of parametric processes for identifiers $[T/A]P$, with $\mathsf{fn}(T) \cap \mathsf{bn}(P) = \emptyset$, is essentially given by

$$[T/A](\mathsf{rec}\, B(\vec{u}).P) = \left\{\begin{array}{ll} (\mathsf{rec}\, B(\vec{u}).[T/A]P) & \text{if } A \neq B\\ (\mathsf{rec}\, B(\vec{u}).P) & \text{otherwise}\end{array}\right.$$

The relation $=_\alpha$ of *α-conversion* is the least congruence satisfying the following axioms:

$$\begin{array}{rcll} a(\vec{u}).P & =_\alpha & a(\vec{u'}).[\vec{u'}/\vec{u}]P & \mathsf{nm}(\vec{u'}) \cap (\mathsf{nm}(P) \cup \{a\}) = \emptyset \\ (\nu a)U & =_\alpha & (\nu a')[a'/a]U & a' \notin \mathsf{nm}(U) \\ (\nu a@\ell)U & =_\alpha & (\nu a'@\ell)[a'/a]U & a' \notin \mathsf{nm}(U) \\ (\nu \ell : \psi)U & =_\alpha & (\nu \ell' : \psi)[\ell'/\ell]U & \ell' \notin \mathsf{nm}(U) \\ (\mathsf{rec}\, A(\vec{u}).P) & =_\alpha & (\mathsf{rec}\, A'(\vec{u'}).[A'/A][\vec{u'}/\vec{u}]P) & A' \notin \mathsf{nm}(P),\ \mathsf{nm}(\vec{u'}) \cap \mathsf{nm}(P) = \emptyset \end{array}$$

Notice that, due to our implicit requirements, in the cases of $a(\vec{u}).P$ and $(\mathsf{rec}\, A(\vec{u}).P)$ the substitution $(\vec{u'}/\vec{u})$ has to be injective. The following result states that reduction is "compatible" with (term) substitution:

Lemma 14 *If $P \to P'$ then there exist Q and Q' such that $Q =_\alpha P$ and $Q' =_\alpha P'$, and $[T/A]Q \to [T/A]Q'$.*

PROOF. By induction on the inference of the reduction, straightforward. ❑

Now we establish the technical results needed to prove the subject reduction property. First we show that we can restrict the typing context to names that actually occur free in the typed term. In the following lemma, when we write $x \notin (\Gamma \vdash_\ell P)$ – where $x \in \mathcal{N} \cup \mathcal{P}$ – we mean that x does not occur in Γ (either in the domain or in the types assigned by this context), nor in P, and that $x \neq \ell$.

Lemma 15 (Strengthening)
(i) *If $\ell' : \psi, \Gamma \vdash_\ell^W u : \tau$ is provable and $\ell' \notin \mathsf{nm}(u) \cup \{\ell\}$, then $\Gamma \vdash_\ell^W u : \tau$ is provable*;
(ii) *if $x : \tau\,, \Gamma \vdash_\ell P$ is provable and $x \notin \mathsf{fn}(P) \cup \{\ell\}$ then $\Gamma \vdash_\ell P$ is provable*;
(iii) *if $x : \tau\,, \Gamma \vdash_\ell T : \sigma$ is provable and $x \notin \mathsf{fn}(T) \cup \{\ell\}$ then $\Gamma \vdash_\ell T : \sigma$ is provable*;
(iv) *if $x : \tau\,, \Gamma \vdash S$ is provable and $x \notin \mathsf{fn}(S)$ then $\Gamma \vdash S$*;
(v) *If $\ell' : \{a : \gamma\}, \Gamma \vdash_\ell^W u : \tau$ is provable and $a \notin \mathsf{nm}(u) \cup \mathsf{nm}(\tau)$, then $\Gamma \vdash_\ell^W u : \tau$ is provable*;
(vi) *if $\ell' : \{a : \gamma\}\,, \Gamma \vdash_\ell P$ is provable and $a \notin (\Gamma \vdash_\ell P)$ then $\Gamma \vdash_\ell P$ is provable*;
(vii) *if $\ell' : \{a : \gamma\}\,, \Gamma \vdash_\ell T : \sigma$ is provable and $a \notin (\Gamma \vdash_\ell T : \sigma)$ then $\Gamma \vdash_\ell T : \sigma$ is provable*;
(viii) *if $\ell' : \{a : \gamma\}\,, \Gamma \vdash S$ is provable and $a \notin (\Gamma \vdash S)$ then $\Gamma \vdash S$ is provable.*

PROOF. By induction on the inference of the sequent, using the convention we made about the rules for input, restriction and recursion. Notice that if $x : \tau\,, \Gamma \vdash_\ell u : \sigma$ then $x \in \mathsf{nm}(u) \cup \{\ell\}$, and similarly if $\ell : \{a : \gamma\}\,, \Gamma \vdash_\ell u : \sigma$ then $a \in \mathsf{nm}(u) \cup \mathsf{nm}(\sigma)$. ❑

The following lemma is the dual of the previous one. It shows that we can weaken a typing context.

Lemma 16 (Weakening) *Let Γ and Δ be typing contexts such that Γ, Δ is also a typing context, then*
(i) *if $\Gamma \vdash_\ell^W u : \tau$ is provable, then $\Gamma, \Delta \vdash_\ell^W u : \tau$*;
(ii) *if $\Gamma \vdash_\ell P$ is provable, then $\Gamma, \Delta \vdash_\ell P$ is provable*;
(iii) *if $\Gamma \vdash_\ell T : \gamma$ is provable, then $\Gamma, \Delta \vdash_\ell T : \gamma$ is provable*;
(iv) *if $\Gamma \vdash S$ is provable, then $\Gamma, \Delta \vdash S$ is provable.*

PROOF. By induction on the inference of the sequent. We just consider the case $P = a(\vec{u}).Q$ assuming that $\mathsf{nm}(\vec{u}) \cap \mathsf{nm}(\Delta) \neq \emptyset$. Let $\vec{v}$ and R such that $\mathsf{nm}(\vec{v}) \cap (\mathsf{nm}(\Delta) \cup \mathsf{nm}(\Gamma)) = \emptyset$ $(*)$ and $a(\vec{u}).Q =_\alpha a(\vec{v}).R$. By the α-conversion rule, $\Gamma \vdash_\ell a(\vec{v}).R$ is provable, and then $\Gamma = \ell : \{a : Ch(\vec{\tau})\}, \Gamma'$ and $\Gamma', \Delta' \vdash_\ell R$ with $\Delta' \vdash_\ell \vec{v} : \vec{\tau}$. From the fact that Δ, Γ is legal and $(*)$, we can easily verify that Γ', Δ, Δ' is legal. Then, by induction hypothesis, we have $\Gamma', \Delta, \Delta' \vdash_\ell R$ is provable and we conclude by the typing rule for the input and α-conversion. ❑

For the next lemma, we use a notion of *height* of a proof of a sequent, which should be clear – viewing a proof as a tree: an axiom has height 0, and each rule increases the maximum of the height of its premises by 1. In the statement of the lemma, we denote by $\mathsf{S}(\Gamma \vdash_\ell P)$ – and similarly for the other kinds of sequents – the sequent obtained by applying the substitution S to P, but also to the context, if this results in a legal context, and to the current location; that is, $\mathsf{S}(\Gamma \vdash_\ell P) = \mathsf{S}(\Gamma) \vdash_{\mathsf{S}(\ell)} [\mathsf{S}]P$, where $\mathsf{S}(\Gamma)$ is defined in the obvious way. Similarly, when we consider sequents $\Gamma \vdash_\ell^{(W)} \overrightarrow{u : \tau}$ or $\Gamma \vdash_\ell A : Ch(\vec{\tau})$, we have to apply the substitution to the types $\vec{\tau}$, and we must be careful that this results in legal types.

Lemma 17 (Substitution) (i) *if* $\Gamma \vdash_\ell^{(W)} \overrightarrow{u:\tau}$ *is provable with a proof of heigth* h*, then so is* $\mathsf{S}(\Gamma \vdash_\ell^{(W)} \overrightarrow{u:\tau})$ *for any substitution* S *such that* $\mathsf{S}(\Gamma)$ *is a typing context;*

(ii) *if* $\Gamma \vdash_\ell P$ *(resp.* $\Gamma \vdash_\ell T : Ch(\vec{\tau})$*) is provable with a proof of heigth* h*, then so is* $\mathsf{S}(\Gamma \vdash_\ell P)$ *(resp.* $\mathsf{S}(\Gamma \vdash_\ell T : Ch(\vec{\tau}))$*) for any substitution* S *such that* $[\mathsf{S}]P$ *(resp.* $[\mathsf{S}]T$*) is defined and* $\mathsf{S}(\Gamma)$ *is a typing context (and* $\mathsf{S}(\vec{\tau})$ *are valid types);*

(iii) *if* $\Gamma \vdash S$ *is provable with a proof of heigth* h*, then so is* $\mathsf{S}(\Gamma \vdash S)$*, for any substitution* S *such that* $[\mathsf{S}]S$ *is defined and* $\mathsf{S}(\Gamma)$ *is a typing context;*

(iv) *if* $\Delta \vdash_\ell T : Ch(\vec{\tau})$ *and* $A : Ch(\vec{\tau})\,,\,\Gamma \vdash_\ell P$ *(resp.* $A : Ch(\vec{\tau})\,,\,\Gamma \vdash_\ell T' : \sigma$*, resp.* $A : Ch(\vec{\tau})\,,\,\Gamma \vdash S$*) are provable, and* A *does not occur in* Γ*, then* $\Delta\,,\,\Gamma \vdash_\ell [T/A]P$ *(resp.* $\Delta\,,\,\Gamma \vdash_\ell [T/A]T' : \sigma$*, resp.* $\Delta\,,\,\Gamma \vdash [T/A]S$*) is provable, provided that no free name of* T *is bound in* P *(resp.* T'*,* S*) and that* Γ, Δ *is a typing contexte.*

Proof.
(i) by induction on h, easy.

(ii) by induction on h, and by case on the last rule used to infer the sequent. If P is $a(\vec{u}).R$ and $\Gamma = \ell : \{a : Ch(\vec{\tau})\}\,,\,\Delta$ with $\ell : \{a : Ch(\vec{\tau})\}\,,\,\Theta\,,\,\Delta \vdash_\ell R$ where $\Theta \vdash_\ell \overrightarrow{u:\tau}$, then by induction hypothesis (and the convention $\mathsf{im}(\mathsf{S}) \cap \mathsf{nm}(\vec{u}) = \emptyset$) $\mathsf{S}'(\ell : \{a : Ch(\vec{\tau})\}\,,\,\Theta\,,\,\Delta \vdash_\ell R)$ is provable, where $\mathsf{S}' = \mathsf{S}\restriction \mathsf{dom}(\mathsf{S}) - \mathsf{nm}(\vec{u})$. Since $\mathsf{S}'(\Theta) \vdash_{\ell'} \overrightarrow{u : \mathsf{S}'(\tau)}$ where $\ell' = \mathsf{S}(\ell)$, we may use the rule for typing the input construct to conclude. The case of $P = (\mathsf{rec}\, A(\vec{u}).Q)$ is similar. All the other cases are very easy. The proof of (iii) is similar.

(iv) by induction on the inference of $A : Ch(\vec{\tau})\,,\,\Gamma \vdash_\ell U$ or $A : Ch(\vec{\tau})\,,\,\Gamma \vdash_\ell T' : \sigma$. If this sequent is an axiom $A : Ch(\vec{\tau}), \Gamma \vdash_\ell A : Ch(\vec{\tau})$, by the lemma 16 and the hypothesis, we have $\Delta, \Gamma \vdash_\ell T : Ch(\vec{\tau})$. The other cases are equally easy. □

B Proofs

B.1 Proof of proposition 3

By the lemmas 2 the relation $=_\mathcal{T}$ is a congruence containing α-conversion, and therefore it is enough to show that for each axiom $U \equiv V$ we have $U \sqsubseteq_\mathcal{T} V$ and $V \sqsubseteq_\mathcal{T} U$. This is trivial for the associativity and commutativity of parallel composition and for the "routing" axiom $[\ell :: P \mid Q] \equiv [\ell :: P] \mid [\ell :: Q]$.

Let $\Gamma \vdash_\ell ((\nu w)P \mid Q)$ with $\mathsf{subj}(w) \notin \mathsf{fn}(Q)$. Since $=_\alpha \subseteq =_\mathcal{T}$, we may assume that the inference does not use the rule of α-conversion, that is $\mathsf{subj}(w)$ does not occur in Γ. The proof of this sequent has the following structure:

$$\dfrac{\dfrac{\dfrac{\vdots}{x : \tau, \Gamma \vdash_\ell P}}{\Gamma \vdash_\ell (\nu w)P} \qquad \dfrac{\vdots}{\Gamma \vdash_\ell Q}}{\Gamma \vdash_\ell ((\nu w)P \mid Q)}$$

where $x : \tau$ is respectively, $\ell : \{a : \gamma\}$ if $w = a$, $\ell' : \{a : \gamma\}$ if $w = a@\ell'$ and $\ell' : \psi$ if $w = \ell' : \psi$. Since $\mathsf{subj}(w) \notin \mathsf{nm}(\Gamma)$, the context $x : \tau, \Gamma$ is a typing context and, by the lemma 16, the sequent $x : \Gamma \vdash_\ell Q$ is provable. We finaly deduce the inference of $\Gamma \vdash_\ell (\nu w)(P \mid Q)$ by application of the rules for parallel composition and restriction.

Conversely, a proof of $\Gamma \vdash_\ell (\nu w)(P \mid Q)$ has the following form:

$$\dfrac{\dfrac{\dfrac{\vdots}{x : \tau, \Gamma \vdash_\ell P} \qquad \dfrac{\vdots}{x : \tau, \Gamma \vdash_\ell Q}}{x : \tau, \Gamma \vdash_\ell P \mid Q}}{\Gamma \vdash_\ell (\nu w)(P \mid Q)}$$

If $x = \ell' = \mathsf{subj}(w)$, by the lemma 15(ii), $\Gamma \vdash_\ell Q$ is provable. If $w = a@\ell'$ or $w = a$, by the lemma 15(vi), $\Gamma \vdash_\ell Q$ is still provable. Moreover, $\Gamma \vdash_\ell (\nu w)P$ is provable, then so is $\Gamma \vdash_\ell ((\nu w)P \mid Q)$ by the rule of parallel composition. The proof is similar for the case of $((\nu w)S \mid S') \equiv (\nu w)(S \mid S')$.

For $[\ell :: (\nu w)P] \equiv (\nu w@\ell)[\ell :: P]$, let us assume that $\Gamma \vdash [\ell :: (\nu w)P]$. The proof of this sequent has the following structure:

$$\dfrac{\dfrac{\dfrac{\vdots}{x : \tau, \Gamma \vdash_\ell P}}{\Gamma \vdash_\ell (\nu w)P}}{\Gamma \vdash [\ell :: (\nu w)P]}$$

where $x : \tau$ is, respectively $\ell : \{a : \gamma\}$ if $w = a$, or $\ell' : \{a : \gamma\}$ if $w = a@\ell'$, or $\ell' : \psi$ if $w = \ell' : \psi$. In all these cases, the following inference is valid:

$$\dfrac{\dfrac{\dfrac{\vdots}{x : \tau, \Gamma \vdash_\ell P}}{x : \tau, \Gamma \vdash [\ell :: P]}}{\Gamma \vdash (\nu w@\ell)[\ell :: P]}$$

For the symmetric case where $\Gamma \vdash (\nu w@\ell)[\ell :: P]$ the proof is similar. $\square$

B.2 Proof of theorem 4

We begin with a usefull remark relating the typing rules for names with and without weakening.

Remark 18 *If $\Gamma \vdash_\ell^W \vec{u} : \vec{\tau}$, then there exist Δ and Θ such that $\Gamma = \Delta, \Theta$ and $\Delta \vdash_\ell \vec{u} : \vec{\tau}$. Moreover, if $\Gamma \vdash_\ell \vec{u} : \vec{\tau}$ then $\Gamma \vdash_\ell^W \vec{u} : \vec{\tau}$.*

By induction on the definition of $U \to V$. Since $\sqsubseteq_{\mathcal{T}}$ is a precongruence containing the structural equivalence relation, it is enough to consider the axioms of reduction. The cases of $[\ell' :: \mathsf{go}\, \ell.P] \to [\ell :: P]$, $[\ell = \ell]P, Q \to P$ and $[\ell = \ell']P, Q \to Q$ are trivial.

For the case of the communication rule $(\overline{a}(\vec{v}) \mid a(\vec{u}).P) \to [\vec{v}/\vec{u}]P$, let Γ be a context such that $\Gamma \vdash_\ell (\overline{a}(\vec{v}) \mid a(\vec{u}).P)$. Then the proof of this sequent has the following structure:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Theta \vdash_\ell^W \vec{v} : \vec{\tau}}}{\Gamma \vdash_\ell \overline{a}(\vec{u})} \qquad \dfrac{\dfrac{\vdots}{\Theta, \Delta \vdash_\ell P} \quad \dfrac{\vdots}{\Delta \vdash_\ell \vec{u} : \vec{\tau}}}{\Gamma \vdash_\ell a(\vec{u}).P}}{\Gamma \vdash_\ell \overline{a}(\vec{v}) \mid a(\vec{u}).P}$$

where we also assume that we do not have to use the rule of α-conversion. From the typing rule for the input, we have $\Gamma = \ell : \{a : Ch(\vec{\tau})\}, \Theta$. Since no name of $\mathsf{nm}(\vec{u})$ occurs in $\vec{\tau}$, $[\vec{v}/\vec{u}]\Delta$ is a typing context and, by the substitution lemma 17 (i), $[\vec{v}/\vec{u}]\Delta \vdash_\ell \vec{v} : \vec{\tau}$ is provable. By the remark 18, we have $\Theta = \Theta_1, \Theta_2$ with $\Theta_1 \vdash_\ell \vec{v} : \vec{\tau}$ then, by the first elementary property given about the typing rules in section 3, we have $\Theta_1 = [\vec{v}/\vec{u}]\Delta$. Since $\mathsf{nm}(\vec{u}) \cap \mathsf{nm}(\Gamma) = \emptyset$, we have $[\vec{v}/\vec{u}](\Theta, \Delta) = (\Theta, [\vec{v}/\vec{u}]\Delta) = \Theta$ which is typing context. Therefore, by the substitution lemma 17(ii), we conclude that $\Theta \vdash_\ell [\vec{v}/\vec{u}]P$.

Now let us consider the unfolding, that is the case of $(\mathsf{rec}\, A(\vec{u}).P)(\vec{v}) \to [\mathsf{rec}\, A(\vec{u}).P/A][\vec{v}/\vec{u}]P$, and assume that the sequent $\Gamma \vdash_\ell (\mathsf{rec}\, A(\vec{u}).P)(\vec{v})$ is provable. The proof of this sequent has the following structure:

$$\dfrac{\dfrac{\dfrac{\vdots}{A : Ch(\vec{\tau}), \Gamma, \Delta \vdash_\ell P} \quad \dfrac{\vdots}{\Delta \vdash_\ell \vec{u} : \vec{\tau}}}{\Gamma \vdash_\ell (\mathsf{rec}\, A(\vec{u}).P)(\vec{v}) : Ch(\vec{\tau})} \qquad \dfrac{\vdots}{\Gamma \vdash_\ell^W \vec{v} : \vec{\tau}}}{\Gamma \vdash_\ell (\mathsf{rec}\, A(\vec{u}).P)(\vec{v})}$$

As in the previous case, the sequent $A : Ch(\vec{\tau})\,, \Gamma \vdash_\ell [\vec{v}/\vec{u}]P$ is provable. As before, we may assume that there is no name which is both free and bound in P, so that the substitution $[(\mathsf{rec}\, A(\vec{u}).P)/A][\vec{v}/\vec{u}]P$ is defined. By the substitution lemma 17(iv), we conclude that $\Gamma \vdash_\ell [(\mathsf{rec}\, A(\vec{u}).P)/A][\vec{v}/\vec{u}]P$. $\square$