Electronic Notes in Theoretical Computer Science 85 No. 1 (2003)
URL: http://www.elsevier.nl/locate/entcs/volume85.html 19 pages

Lexically scoped distribution: what you see is what you get

António Ravara [a] Ana G. Matos [b] Vasco T. Vasconcelos [c]
Luís Lopes [d]

[a] *CLC and Dep. of Mathematics, IST, Technical University of Lisbon, Portugal.*

[b] *INRIA, Sophia-Antipolis, France.*

[c] *Dep. of Informatics, Faculty of Sciences, University of Lisbon, Portugal.*

[d] *Dep. of Computer Science, Faculty of Sciences, University of Porto, Portugal.*

Abstract

We define a lexically scoped, asynchronous and distributed π-calculus, with local communication and process migration. This calculus adopts the network-awareness principle for distributed programming and follows a simple model of distribution for mobile calculi: a lexical scope discipline combines static scoping with dynamic linking, associating channels to a fixed site throughout computation. This discipline provides for both remote invocation and process migration. A simple type system is a straightforward extension of that of the π-calculus, adapted to take into account the lexical scope of channels. An equivalence law captures the essence of this model: a process behavior depends on the channels it uses, not on where it runs.

1 Introduction

Current hardware developments in network technology, namely high-bandwidth, low-latency networks and wireless communication, has opened new prospects for mobile computation, while at the same time introducing new problems that need to be addressed at the software level. The fundamental problem stems from the lack of a formal background on which to assert the correctness of a given system specification. Thus, adequate theoretical modeling of distributed mobile systems is required to produce provably correct software specifications and to reason about distributed computations.

A natural and simple framework to study such systems is Dpi, the distributed π-calculus of Hennessy and Riely [4,3] that extends the π-calculus [6,7] by distributing processes over a (flat) network of *localities* — named sites where computation happens. Communication only takes place within localities (to avoid global synchronization), but processes may migrate between locations.

Lexically scoped programming. To motivate the importance of lexical scope in programming languages in general, consider the following function written in (a variant of) Pascal.

```
function f (): Integer;
  var x: Integer := 1;
  function g () : Integer;
    begin g := x end;
  begin f := x + g () end;
```

A *programmer* would write the body of function g knowing that x is global in g, and would develop function f keeping in mind that x is local to f, and that the x of g and that of f denote the same variable. Programmers have been doing this kind of reasoning for decades, both in imperative (Algol, Pascal) and in functional (ML, Haskell) languages. It is intuitive, and accumulated experience has shown the concept to be right. Furthermore, an unoptimized *compiler* would assign to variable x a memory location at the activation record for function f. To evaluate expressions in the body of a function, the generated code must read the value of x: if in f, it performs a local operation (reading from the data for the current activation); if in g, it first finds where f's activation is.

The good news is that we do not need to abandon these ideas when moving into a distributed setting, even in the presence of code migration. While in the above example the pertinent question is "which function does this variable belong to?", in a distributed setting we will ask "which locality does this channel belong to?". Under a lexical scope regime, the answer must be clear in the *syntax*. Therefore, we adhere to the *lexical scope in a distributed context* of Obliq [2], which is here understood as the discipline under which the locality of channels is fixed throughout computation and can be determined by straightforward code inspection. This principle must not be disturbed by computation, in particular by code mobility and dynamic linking. The paradigm allows us to *a priori* view channels as physical resources of sites, unlike Dpi where channels are network-wide identifiers that only *a posteriori* (via a type system) get located in sites.

Lexically scoped distributed π-calculus. *lsd*π is a variant of Dpi where channel location can be inferred from the syntax of networks. The lexical scope discipline is obtained by the following simple rule: *compound channels (such as* $a@s$*) belong to the site where they are explicitly located (*s*); while simple channels are (implicitly) located at the site where they occur.* To accommodate process mobility, (free) channels are renamed via a translation function that adapts the syntax of the migrated process to its new location. A form of *dynamic linking* (of a channel to a site) is obtained via a $(\nu\, a)$ binding, for the site that will allocate channel a will only be known at runtime.

In *lsd*π *what you see is what you get*, since the location of channels are clear in the program text, and the syntax plainly expresses the location of the resources used, even in presence of mobile code.

Dpi and *lsd*π. Consider a network where we declare a channel x within some site f, ask for the process x?()P to migrate to another site g, and in parallel launch a receptor x?()Q located at f. In an untyped version of Dpi, the network and the one obtained by a reduction step appear as follows.

```
f [new x                          new x@f
  go g. x?()P |                   g [x?()P] |
  x?()Q]                          f [x?()Q]
```

From the preceding discussion on the Pascal program, we are interested in asking "where does channel x belong to?" (or "where is x to be allocated?"). Analyzing the new x part, one would expect channel x to belong to site f, but at the level of the subnetworks for g and for f we cannot conclude that. The type system for Dpi identifies the receptor x at f with that at g, and, since x is bound at f, rejects the network.

In general, as we have said, by inspecting the name of a free channel within a site and name of the site, the physical location of a channel can be known at all times. This is achieved by allowing both simple (a) and compound ($a@s$) channels, which reflect two possible views: local (process-level), and global (network-level). In $lsd\pi$, the above left network is indeed typable, and reduces to:

```
new x@f
g [x@f?()P'] |
f [x?()Q]
```

where it is clear (in each of the three lines) that channel x still belongs to site f. Process P' (obtained from P by a translation on names) reflects the fact that P has migrated from f: a free simple channel y (implicitly located at f) becomes attached to its site, denoted y@f; all the remaining channels remain unchanged.

Explicitly located input/output processes (receptors x@f?(y)P and messages x@f!⟨v⟩), absent in every proposal of distributed process calculi to date, obviate the need for a migration primitive like go (written go s.P in [1] and in [4]). It suffices to attach to such processes the behavior "migrate toward the home site": a remote message or receptor at channel x@f migrate to f, in this way obtaining the required "local status". Hence, an input/output process can perform two types of reduction, depending on the format of the prefix: it either communicates locally, or it migrates to the site where it is located.

Contributions. The lexical scope discipline allows for a straightforward extension of the simple type system of the π-calculus, assigning the usual channel types to channels, and record types (maps from channels to channel types) to sites, guarantying the absence of arity mismatches in input/output.

From the standard notion of asynchronous bisimulation, this discipline yields a simple equivalence law, whereby a process behavior depends only on the names it uses (and not on where it runs).

Specifically, our contributions are: (1) a distributed asynchronous π-calculus that provides for local communication, remote invocation, and weak mobility, in a lexical scope regime; (2) a precise syntactic treatment of channels as resources of sites, which ensures that each channel belongs to a unique, lexically defined, site; (3) a simple type system revealing the site of each channel; and (4) a notion of equivalence, inducing a fundamental law.

Related work. Dpi is an explicitly typed calculus. The processes accepted by the type system are similar in many respects to $lsd\pi$ processes. Our conjecture is that what Dpi achieves with a type system, what we guarantee with our semantics. Free names and substitution in Dpi are just like in π (hence the channels are global), but the type system is quite elaborate, being difficult to recognize what is a well-typed process. The semantics of $lsd\pi$ is more complex than that of π (to get channels as resources of sites), but the type system is fairly simple.

Klaim [8,9] is a formalism combining asynchronous higher-order process calculus with the Linda coordination paradigm. It supports process mobility and distribution and provides mechanisms for security. Although communication is not channel-based (else via distributed tuple spaces), in what respects the scoping discipline, $lsd\pi$ is similar to Klaim: both use static scoping to uniquely resolve identifiers at communication-time, and both support dynamic linking at migration-time.

Outline. The next two sections introduce the syntax and the operational semantics of the calculus. Section 4 presents the type system, and Section 5 the behavioral equivalence. Section 6 concludes the paper.

2 Syntax

Consider a countable set $\mathcal{C}$ of *simple channels* a, b, c, x, y, z, and a countable set $\mathcal{S}$ of *sites* s, r, t, such that the two sets are disjoint. Compound channels — pairs channel-site, like $a@s$ — form *located channels*, designating a channel a at site s, belonging to the set $\mathcal{C}@\mathcal{S} \stackrel{\text{def}}{=} \{a@s \mid a \in \mathcal{C} \wedge s \in \mathcal{S}\}$. Sets $a@\mathcal{S}$ and $\mathcal{C}@s$ are defined analogously as $\{a@s \mid s \in \mathcal{S}\}$ and $\{a@s \mid a \in \mathcal{C}\}$.

Let $u, v, \ldots$ stand for both simple and located channels, henceforth collectively called *channels*. Take $\tilde{x}$ as a sequence of pairwise distinct simple channels, and $\tilde{v}$ as a sequence of channels. Let $|\tilde{v}|$ denote the length of the sequence $\tilde{v}$, and let $\{\tilde{v}\}$ denote the set of the elements in the sequence $\tilde{v}$. Furthermore, let n, m stand for both sites and for channels, henceforth collectively called *names*, and belonging to the set $\mathcal{N} \stackrel{\text{def}}{=} \mathcal{C} \cup \mathcal{S} \cup \mathcal{C}@\mathcal{S}$. Finally, let g, h stand for both sites and compound channels, henceforth collectively called *global names*.

The grammars in Figure 1 define the languages of names, of processes, and of networks. *Receptors*, $u?(\tilde{x})P$, and *messages*, $u!\langle\tilde{v}\rangle$, are the basic processes in the calculus. A receptor is an input-guarded process. A message targets

$$
\begin{array}{rrcl}
\textit{Simple channels,} & a, b, c, x, y, z & \in & \mathcal{C} \qquad\qquad \textit{Sites,} \quad r, s, t \in \mathcal{S} \\
\textit{Channels,} & u, v & ::= & a \mid a@s \\
\textit{Globals,} & g, h & ::= & a@s \mid s \\
\textit{Names,} & n, m & ::= & a \mid a@s \mid s \\
\textit{Processes,} & P, Q & ::= & \mathbf{0} \mid (P \mid Q) \mid (\nu\, n)\, P \mid u!\langle \tilde{v} \rangle \\
 & & \mid & u?(\tilde{x})P \mid u?*(\tilde{x})P \\
\textit{Networks,} & N, M & ::= & \mathbf{0} \mid (N \mid M) \mid (\nu\, g)\, N \mid s[P] \\
\textit{Entities,} & X, Y, Z & ::= & P \mid N
\end{array}
$$

Fig. 1. Syntax of $lsd\pi$

a name u and carries a sequence of channels $\tilde{v}$. Prefixes of receptors and of messages, may occur as compound channels, but this is just a consequence of the local/remote form that all channels might exhibit. As to the objects of communication, notice that although parameters are simple channels only, arguments can be channels. We call this *global substitution*: channels are to be seen as identifiers, while parameters are "blind" to the structure of the value they receive.

The remaining constructors are fairly standard in name-passing process calculi: $u?*(\tilde{x})P$ denotes a persistent receptor; *inaction* $\mathbf{0}$, denotes the terminated process; $(P \mid Q)$ denotes the *parallel composition* of processes; and $(\nu\, n)\, P$ denotes the *restriction of the scope* of the name n to the process P.

The basic unit of *networks* is a process running at a given site, $s[P]$. Again, the remaining constructors are standard: *inaction* denotes the empty network; the parallel composition of networks, $(N \mid M)$, denotes a merge of networks; and $(\nu\, g)\, N$ represents the restriction of the scope of a global to a network. Name restriction is limited to globals at network level, for local channels make no sense outside the scope of a site in a lexical scope regime. Furthermore, if channels are to be understood as resources of sites, it is reasonable to limit the creation of local channels to within sites.

We will generally refer to processes and networks as *entities*; let $\mathcal{X}$ denote the set of all entities, ranged over by X, Y, Z. As usual in process calculi, for some $m \geq 0$, we abbreviate $(\nu\, n_1) \cdots (\nu\, n_m)\, P$ to $(\nu\, \widetilde{n})\, P$ and $(\nu\, g_1) \cdots (\nu\, g_m)\, N$ to $(\nu\, \tilde{g})\, N$, and consider that the operator 'ν' binds tighter than the operator '$|$'. Furthermore, in $(X \,|\, Y)$ we omit the parenthesis when the meaning is clear.

Finally, we do not allow the communication of site names. We believe this restriction does not result in any loss of expressive power.

X	$\mathsf{fn}(X)$	n	$\mathsf{subj}(n)$	$\mathsf{obj}(n)$
$\mathbf{0}$	$\{\}$	a	$\{a\}$	$\{a\}$
$(X \mid Y)$	$\mathsf{fn}(X) \cup \mathsf{fn}(Y)$	$a@s$	$\{a@s\}$	$\{a@s, s\}$
$(\nu\, n)\, X$	$(\mathsf{fn}(X) \cup \mathsf{obj}(n)) \setminus \mathsf{subj}(n)$	s	$\mathcal{C}@s \cup \{s\}$	$\{s\}$
$u?(\tilde{x})P$	$\mathsf{fn}(P) \setminus \{\tilde{x}\} \cup \mathsf{obj}(u)$			
$u!\langle\tilde{v}\rangle$	$\mathsf{obj}(u\tilde{v})$			
$s[P]$	$\mathsf{fn}(P)@s \cup \{s\}$			

Fig. 2. Subjects, objects and free names

3 Operational semantics

This section describes the reduction semantics of $lsd\pi$, starting from the definition of the free names, substitution, through structural congruence, ending in reduction.

Free names in networks and in processes. The definition of free names establishes the basic principles of the lexical scope discipline. Presence of compound channels inherently introduces binding subtleties, and to implement a lexical scope regime on names, we cannot rely on the definition of free names used in Dpi.

It becomes useful to distinguish two central concepts regarding name bindings: *subjects* of a name n, $\mathsf{subj}(n)$, those that are caught by a restriction on n; and *objects* of a name n, $\mathsf{obj}(n)$, those which can catch n by restriction. In calculi with simple names only we find that $\mathsf{subj}(n) = \{n\} = \mathsf{obj}(n)$. Here, the definition must be refined.

The rules in the right table in Figure 2 inductively define the sets of *subjects* and of *objects of names*, extended to sets and to sequences of names in the expected way. Subjects and objects are just two sides of the same binding relation. We should be able to say interchangeably that m is bound by n, or that m is a subject of n, that is n is an object of m. In fact, $m \in \mathsf{subj}(n) \Leftrightarrow n \in \mathsf{obj}(m)$. This property allows us to say n *binds* m, $n \hookrightarrow m$, if $m \in \mathsf{subj}(n)$, or equivalently, if $n \in \mathsf{obj}(m)$.

The rules in the left table in Figure 2 inductively define $\mathsf{fn}(X)$, the set of *free names* in processes and in networks, considering $n@s$ to be n if $n = g$, and assuming that this operator extends to sequences and to sets of names. The case of the persistent receptor is, in many ways, analogous to the simple receptor. Hereafter, we omit the treatment of this case when it is analogous to that of the simple receptor.

Substitution. Channels and sites occupy different positions in structured names, so substitution is not a total function on names. We are mainly interested in its application to the operations performed within the calculus,

restricting our attention to three kinds of substitutions of m by n: *name replacement*, for *change of bound names*, if either $m, n \in \mathcal{C}, m, n \in \mathcal{S}$, or $m, n \in \mathcal{C}@s$ for some s; *name instantiation*, for *communication*, if $m \in \mathcal{C}$ and $n \in (\mathcal{C} \cup \mathcal{C}@\mathcal{S})$; *name translation*, for *migration*, if $m \in \mathcal{C}$ and $n \in m@\mathcal{S}$.

Name substitution must be defined in such a way that when applied to processes or to networks, the correspondence between their binders and bound names does not change. However, some binder properties which are true in calculi like λ or π — as $\mathsf{subj}(n) = \{n\} = \mathsf{obj}(n)$ — no longer stand in this setting, so one must generalize the old intuitions: (1) *A binder on a name n binds, in a process within its scope, the free occurrences of that name n.* But $a@s$ is bound in $(\nu\, s)\, a@s!\langle\rangle$, so we say more generally that the set of names bound by n is $\mathsf{subj}(n)$. (2) *If name n is changed by a substitution $\{m_2/m_1\}$ then $n = m_1$ and the result is m_2.* But s is free in $a@s!\langle\rangle$, so we say more generally that n is changed by $\{m_2/m_1\}$ if it is a subject of m_1, and the change affects components of n, namely elements of $\mathsf{obj}(n)$.

Substitution may be defined for $lsd\pi$ in the usual way if these generalizations are taken into account. Following Hindley and Seldin [5], a *substitution on names and entities* is a partial function $\mathcal{X}\{\mathcal{N}/\mathcal{N}\} \mapsto \mathcal{X}$, inductively defined by the rules in Figure 3.

The case $s[P]\{b@s/a@s\}$ incorporates the fact that while outsiders see free channels as being uniformly remote, *locally* they might take both the compound or the simple form. Therefore, when crossing the site boundary, this substitution must be unfolded in two. Let $\{\tilde{v}/\tilde{x}\}$ denote a set of simultaneous substitutions, defined as usual.

Structural congruence. We define a static binary relation on processes and networks — *structural congruence* — as the least congruence relation containing the rules in Figure 4.

Rule S-ALPHA uses an alpha-congruence relation, denoted $\equiv_\alpha$, defined in the usual way from the definition of substitution.

The rules in the top of Figure 4 (down to SP-GC are adapted from the standard rules of the π-calculus [6,7]. Differences reside mainly in the side conditions, resulting directly from the fact that name creation introduces free names, which are themselves subject to bindings. Rule SN-GC garbage collects inactive sites and channel restrictions. The same is true for SP-GC, which is intended for processes. Rule S-RESO controls the capture of sites in compound channel creations by site restrictions. However, two channel name creations may always commute.

The rules S-ROUT and S-SCOS) are inspired on those proposed by Hennessy and Riely for Dpi [4], but are adapted to a lexical scope regime: Rule SN-ROUT describes how processes within a site may be split or aggregated. Rule SN-SCOS depicts how a site boundary affects the view over a name restriction: outsiders always see channel creations as remote, so their compound and simple forms are indistinguishable (up to name capture) to them, while local viewers differentiate those two forms of local channel creation.

$$(n)\{m_2/m_1\} \stackrel{\text{def}}{=} \begin{cases} m_2 & \text{if } n = m_1 \\ a@m_2 & \text{if } n = a@m_1 \\ n & \text{if } m_1 \not\rightarrow n \end{cases}$$

$$\mathbf{0}\{m_2/m_1\} \stackrel{\text{def}}{=} \mathbf{0}$$

$$(X \mid Y)\{m_2/m_1\} \stackrel{\text{def}}{=} X\{m_2/m_1\} \mid Y\{m_2/m_1\}$$

$$((\nu\, n)\, X)\{m_2/m_1\} \stackrel{\text{def}}{=} \begin{cases} (\nu\, n)\, X & \text{if (1)} \\ (\nu\, n\{m_2/m_1\})X\{m_2/m_1\} & \text{if } \neg(1) \wedge (2) \\ (\nu\, n')X\{n'/n\}\{m_2/m_1\} & \text{if } \neg(1) \wedge \neg(2) \wedge (3) \end{cases}$$

$$(u?(\tilde{x})P)\{m_2/m_1\} \stackrel{\text{def}}{=} \begin{cases} u\{m_2/m_1\}?(\tilde{x})P & \text{if (4)} \\ u\{m_2/m_1\}?(\tilde{x})P\{m_2/m_1\} & \text{if } \neg(4) \wedge (5) \\ u\{m_2/m_1\}?(\tilde{x}')P\{y/x_i\}\{n/m\} & \text{if } \neg(4) \wedge \neg(5) \wedge (6) \end{cases}$$

$$(u!\langle v_1 \ldots v_n\rangle)\{m_2/m_1\} \stackrel{\text{def}}{=} u\{m_2/m_1\}!\langle v_1\{m_2/m_1\} \ldots v_n\{m_2/m_1\}\rangle$$

$$(s[P])\{m_2/m_1\} \stackrel{\text{def}}{=} \begin{cases} s[P\{b/a\}\{b@s/a@s\}] & \text{if } m_1 = a@s \wedge m_2 = b@s \\ s\{m_2/m_1\}[P\{m_2/m_1\}] & \text{otherwise} \end{cases}$$

1. $n \hookrightarrow m_1$
2. $n \not\rightarrow m_2 \vee m_1 \notin \mathsf{fn}(X)$
3. n' is fresh
4. $\exists_{x_i \in \{\tilde{x}\}}.x_i \hookrightarrow m_1$
5. $(\forall_{x_i \in \{\tilde{x}\}}.x_i \not\rightarrow m_2) \vee m_1 \notin \mathsf{fn}(P)$
6. y is fresh and $\exists_{x_i \in \{\tilde{x}\}}.\tilde{x}' = \tilde{x}\{y/x_i\}$

Fig. 3. Substitution on names, processes and networks

The rules SN-MIGO, SN-MIGI and SN-RMIGI are specific to a lexical scope regime of channels. They make use of the fact that a channel which is explicitly located in the site where it occurs is in fact a local channel, and so it might as well take its simple format. This law is only needed at communication time [1], so it regards only channels which appear in the top-level of a process.

Name translation. When migrating a process P previously running in r, the simple channels of P (which were once local) will become remote references. Therefore, they must be located accordingly — they become explicitly located at r — while all other channels remain unchanged.

[1] See rule RP-COMM in Figure 5.

$$
\begin{array}{lrll}
\text{[S-ALPHA]} & X \equiv Y & & \text{if } X \equiv_\alpha Y \\
\text{[S-ASSO]} & ((X \mid Y) \mid Z) \equiv (X \mid (Y \mid Z)) & & \\
\text{[S-COMM]} & (X \mid Y) \equiv (Y \mid X) & & \\
\text{[S-NEUT]} & (X \mid \mathbf{0}) \equiv X & & \\
\text{[S-SCOP]} & ((\nu\, n)\, X) \mid Y \equiv (\nu\, n)\, (X \mid Y) & & \text{if } \mathsf{subj}(n) \cap \mathsf{fn}(Y) = \emptyset \\
\text{[S-RESO]} & (\nu\, n)\, (\nu\, m)\, X \equiv (\nu\, m)\, (\nu\, n)\, X & & \text{if } n \not\rightarrow m \text{ and } m \not\rightarrow n \\
\text{[SP-GC]} & (\nu\, \widetilde{n})\, \mathbf{0} \equiv \mathbf{0} & & \text{if } \mathsf{fn}((\nu\, \widetilde{n})\, \mathbf{0}) = \emptyset \\
\text{[SN-GC]} & (\nu\, \widetilde{n})\, s[\mathbf{0}] \equiv \mathbf{0} & & \text{if } \mathsf{fn}((\nu\, \widetilde{n})\, s[\mathbf{0}]) = \emptyset \\
\text{[SN-ROUT]} & (s[P] \mid s[Q]) \equiv s[(P \mid Q)] & & \\
\text{[SN-SCOS}_1\text{]} & (\nu\, g)\, s[P] \equiv s[(\nu\, g)\, P] & & \text{if } g \not\rightarrow s \wedge g \notin \mathcal{C}@s \\
\text{[SN-SCOS}_2\text{]} & (\nu\, a@s)\, s[P] \equiv s[(\nu\, a@s)\, P] & & \text{if } a \notin \mathsf{fn}(P) \\
\text{[SN-SCOS}_3\text{]} & (\nu\, a@s)\, s[P] \equiv s[(\nu\, a)\, P] & & \text{if } a@s \notin \mathsf{fn}(P) \\
\text{[SN-MIGO]} & s[a@s!\langle \tilde{v} \rangle] \equiv s[a!\langle \tilde{v} \rangle] & & \\
\text{[SN-MIGI]} & s[a@s?(\tilde{x})P] \equiv s[a?(\tilde{x})P] & & \\
\text{[SN-MIGR]} & s[a@s?*(\tilde{x})P] \equiv s[a?*(\tilde{x})P] & &
\end{array}
$$

Fig. 4. Structural congruence on processes and networks

It should be clear that some compound channels will become local after migration. Their simplification to simple channels is only necessary at communication time, and is dealt with by the already mentioned structural congruence rules SN-MIGO, SN-MIGI and SN-RMIGI.

Reduction. The rules in Figure 5 inductively define the *reduction relation* on processes and networks.

Axiom RP-COMM, standard in the π-calculus [6,7], only applies to simple communication channels, i.e., local ones. Whichever the site the process occurs in, we know that it has allocated the communication channel. This is why we can say communication is local. Rule RP-RCOMM reveals that a replicated input is a persistent resource.

The axioms RN-MIGO and RN-MIGI, to which we add RN-RMIGI, are those proposed by Vasconcelos *et al.* for DiTyCO [13], and only make sense in a lexical scope setting. Whenever an output/input process is prefixed over a remote channel, we know it requires a remote resource to perform its communication. Therefore, in order to obtain the status of "local", the code simply

$$\text{[RP-COMM]} \quad a?(\tilde{x})P \mid a!\langle\tilde{v}\rangle \to P\{\tilde{v}/\tilde{x}\}$$

$$\text{[RP-RCOMM]} \quad a?*(\tilde{x})P \mid a!\langle\tilde{v}\rangle \to a?*(\tilde{x})P \mid P\{\tilde{v}/\tilde{x}\}$$

$$\text{[RN-MIGO]} \quad r[a@s!\langle\tilde{v}\rangle] \to s[(a@s!\langle\tilde{v}\rangle)\sigma_r] \qquad r \neq s$$

$$\text{[RN-MIGI]} \quad r[a@s?(\tilde{x})P] \to s[(a@s?(\tilde{x})P)\sigma_r] \quad r \neq s$$

$$\text{[RN-RMIGI]} \quad r[a@s?*(\tilde{x})P] \to s[(a@s?*(\tilde{x})P)\sigma_r] \quad r \neq s$$

$$\text{[RN-SITE]} \quad \frac{P \to Q}{s[P] \to s[Q]}$$

$$\text{[R-CONT]} \quad \frac{X \to Y}{E[X] \to E[Y]}$$

$$\text{[R-STR]} \quad \frac{X \equiv X' \quad X' \to Y' \quad Y' \equiv Y}{X \to Y}$$

$$E \quad ::= \quad [] \quad | \quad (E \mid X) \quad | \quad (\nu\, n)\, E$$

$$P\sigma_r \stackrel{\text{def}}{=} P\{a_1@r/a_1, \ldots, a_n@r/a_n\} \text{ where } \mathsf{fn}(P) \cap \mathcal{C} = \{a_1, \ldots, a_n\}$$

Fig. 5. Reduction rules

migrates to the site the communication channel belongs to (which is explicit in the name of the channel). A name translation is then performed upon the code by means of σ_r, so as to reflect the new site where it is running: channels that previously were simple (thus implicitly located) are now composed with the site from where the code came from.

It is important to notice that no new free name is created during reduction.

Proposition 3.1 (Reduction preserves free names) *If $X \to Y$, then* $\mathsf{fn}(X) \supseteq \mathsf{fn}(Y)$.

Dynamic linking. A channel which is created under an input prefix cannot be used until that input has been "consumed". In particular, the site where local channels are to be created may result from some computation [2]. We support this view with an example that shows how to create local channels "anywhere" in the network by using dynamic linking. Consider a server with address a at site s providing some application, while requiring some resources (say a new channel b). It is possible to define these resources as being local to

[2] Notice though, that once the local channel is created, the site it is associated to will remain fixed.

$$\text{TS-Chl}\ \Gamma \vdash a@s{:}\Gamma(s)(a) \qquad \text{TS-Chs}\ \Gamma \vdash a{:}\Gamma(\hbar)(a) \qquad \text{TS-Uni}\ \frac{\Gamma \vdash \widetilde{v_1{:}\gamma_1} \quad \Gamma \vdash \widetilde{v_2{:}\gamma_2}}{\Gamma \vdash v_1\widetilde{v_2{:}\gamma_1\gamma_2}}$$

$$\text{TP-Out}\ \frac{\Gamma \vdash \widetilde{v{:}\gamma} \quad \Gamma \vdash u{:}Ch(\widetilde{\gamma})}{\Gamma \vdash u!\langle \tilde{v} \rangle} \qquad \text{TP-Inp}\ \frac{\Gamma \vdash \widetilde{x{:}\gamma} \quad \Gamma \vdash u{:}Ch(\widetilde{\gamma}) \quad \Gamma \vdash P}{\Gamma \setminus \tilde{x}@\hbar \vdash u?(\tilde{x})P}$$

$$\text{TP-Ress}\ \frac{\Gamma \vdash P}{\Gamma \setminus a@\hbar \vdash (\nu\, a)\, P} \qquad \text{T-Resl}\ \frac{\Gamma \vdash X}{\Gamma \setminus a@s \vdash (\nu\, a@s)\, X}$$

$$\text{T-Resn}\ \frac{\Gamma \vdash X}{\Gamma \setminus s \vdash (\nu\, s)\, X} \qquad \text{T-Par}\ \frac{\Gamma \vdash X \quad \Gamma \vdash Y}{\Gamma \vdash (X \mid Y)}$$

$$\text{TN-Net}\ \frac{\Gamma \vdash P}{\Gamma\{s/\hbar\} \vdash s[P]} \qquad \text{T-Nil}\quad \Gamma \vdash \mathbf{0}$$

Fig. 6. Typing names, processes and networks

the site that downloads the application:

$$s[a?(x)x?()((\nu\, b)\, P)] \mid r[a@s!\langle c \rangle \mid c!\langle\rangle)] \mid t[a@s!\langle c \rangle \mid c!\langle\rangle)]\,.$$

In fact, the above process can reduce both to $r[(\nu\, b)\, P]$ or to $t[(\nu\, b)\, P]$ (if $x \notin \mathsf{fn}(P)$). By maintaining both simple and located forms of channels, we are able to express the creation of a local name (the above b) in a site which will be determined only at run time.

4 The type system

This section presents a type system for *lsd*π. The system is a straightforward extension of that for the simply typed π-calculus [12] and it is a simplified form of the one in Amadio *et al.* for $d\pi_1^r$ [1], adapted to deal with the lexical scope of channels. The main result of this section ensures the preservation of typability by reduction, a property usually know as *subject reduction*. From there it is simple to prove a type safety corollary, ensuring the absence of arity mismatch in communication.

Types for channels are those of the simply typed π-calculus, $Ch(\gamma_1, \ldots, \gamma_n)$, describing a channel capable of carrying a series of channels of types $\gamma_1, \ldots, \gamma_n$. Types for sites only capture the types of its free channels: if a_1 to a_n of types γ_1 to γ_n are the free channels of site s, then we assign to site s the type $\{a_1{:}\gamma_1, \ldots, a_n{:}\gamma_n\}$.

The types of $lsd\pi$ are a subset of those of $d\pi_1^r$: simply remove the located type — a form of existential type — unnecessary in this setting due to name instantiation. The type system of $lsd\pi$ is syntax-oriented, though it considers a special site — $\hbar$, for "here" — which designates the current site when typing the occurrences of simple channels in processes. Due to the motto '*simple channels are local*, "here" corresponds always to the same site. The "site variable" is instantiated at the network level, or disposed of if the simple channels is bound. At the end of the typing procedure, all channels are explicitly associated to one site.

Definition 4.1 (Types) *Consider the finite sets* $\{s_1, \ldots, s_n\} \subseteq \mathcal{S} \uplus \{\hbar\}$ *and* $\{a_1, \ldots, a_n\} \subseteq \mathcal{C}$ *, with* $n \geq 0$ *and where the designated elements of each sets are pairwise distinct.*

$$\begin{array}{rrcl} \textit{Channel types,} & \gamma & ::= & \mathrm{Ch}(\gamma_1, \ldots, \gamma_n) \\ \textit{Site types,} & \varphi & ::= & \{a_1{:}\gamma_1, \ldots, a_n{:}\gamma_n\} \\ \textit{Typings,} & \Gamma & ::= & \{s_1{:}\varphi_1, \ldots, s_n{:}\varphi_n\} \end{array}$$

A site type is a map from channels into channel types; a typing is a map from sites into site types. If Γ is a typing, then $\Gamma \setminus s$ removes s from the domain of Γ; $\Gamma \setminus a@s$ removes a from the domain of $\Gamma(s)$; and $\Gamma\{s/t\}$ replaces s by t in the domain of Γ, provided that channels common to t and to s have the same types. The *union of typing assumptions* $\Gamma + \Delta$ is defined pointwise, if for all $s \in \mathsf{dom}(\Gamma)$ and for all $a \in \Gamma(s) \cap \Delta(s)$ we have $\Gamma(s)(a) = \Delta(s)(a)$, as:

$$(\Gamma + \Delta)(s) \stackrel{\text{def}}{=} \begin{cases} \Gamma(s), & \text{if } s \in \mathsf{dom}(\Gamma) \setminus \mathsf{dom}(\Delta) \text{ or } \Gamma(s) = \Delta(s), \\ \Gamma(s) \cup \Delta(s), & \text{if } s \in \mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta) \text{ and } \Gamma(s)(a) = \Delta(s)(a), \\ \Delta(s), & \text{if } s \in \mathsf{dom}(\Delta) \setminus \mathsf{dom}(\Gamma). \end{cases}$$

$$\Gamma\{s/t\} \stackrel{\text{def}}{=} \Gamma \setminus t + \{s{:}\Gamma(t)\}$$

The rules in Figure 6 inductively define the type system of $lsd\pi$. It uses two kinds of judgments: $\Gamma \vdash \widetilde{v{:}\gamma}$ asserts that Γ types (simple and located) channels $\tilde{v}$ with types $\widetilde{\gamma}$, according to the typing assumption Γ; $\Gamma \vdash X$ says that process or network X conforms to the typing assumption Γ.

The special site $\hbar$ plays a very important role: it allows delaying decisions when typing processes. If in a process a channel with subject a occurs both in a simple and in a compound form, we assume the simple one located "*here*" (at $\hbar$), and the compound one located at s. Dpi type systems ([1,4]) commit too soon, assuming the same site. For instance, the following judgments hold.

$$\emptyset \vdash_s (\nu\, a@s)\, a!\langle\rangle \qquad \text{and} \qquad \emptyset \vdash_s (\nu\, a)\, a@s!\langle\rangle$$

These judgments, however, should not hold in our type system, since, contrary to what happens in Dpi, the binders do not capture the channel subject of the process. Therefore, the valid judgments are:

$$\{\hbar{:}\{a{:}Ch()\}\} \vdash (\nu\, a@s)\, a!\langle\rangle \qquad \text{and} \qquad \{s{:}\{a{:}Ch()\}\} \vdash (\nu\, a)\, a@s!\langle\rangle$$

This distinction is also useful when deciding if a process should be rejected: is $a@s?()P \mid a!\langle v\rangle$ a communication error? it depends on the site where the process is running and on the channels being the same. Thus, a process like $(\nu\, a@s)\,(a@s?()P \mid a!\langle v\rangle)$ is well behaved, and our type system accepts it [3]. In short, the special site $\hbar$ is the key for a lexically scoped type system.

The type system enjoys subject-reduction, a result which depends on the preservation of typability by appropriate substitutions.

Lemma 4.2 (Substitution lemma) *Let Υ be a sequence of name replacements or name instantiations. Then, $\Gamma \vdash X$ implies $\Gamma\Upsilon \vdash X\Upsilon$.*

Proof. By induction on the derivation of $\Gamma \vdash X$. It is useful to prove that $\Gamma \vdash \widetilde{v{:}\gamma}$ implies $\Gamma\Upsilon \vdash \widetilde{v\Upsilon{:}\gamma}$ by using induction on the length of Υ and $\tilde{v}$, and a case analysis on names. □

We show now that the type system enjoys subject-reduction. Recall that we omit the treatment of the replicated input, as it is very similar to the case of the simple input.

Theorem 4.3 (Subject reduction) *If $\Gamma \vdash X$ and $X \to Y$, then $\Gamma \vdash Y$.*

Proof. The proof consists of inductions on the derivations of the reduction step. As usual, we use a lemma stating that alpha and structural congruence also preserve typability. The base cases in the derivation of $P \to Q$ are the process axioms, for which we use Lemma 4.2 and typing rules T-PAR, TP-INPS, TP-OUTS and TP-RINPS. The base cases for the derivation of $X \to Y$ are the migration axioms. We use the result for processes, Lemma 4.2, the definition of the translation function, as well as typing rules TN-NET, TP-INPL, TP-OUTL and TP-RINP. The cases of the induction steps are straightforward. □

Type safety follows.

Theorem 4.4 (Type Safety) *If $N \equiv s[a!\langle\tilde{v}\rangle \mid a?(\tilde{x})P \mid M]$ and $\Gamma \vdash N$, then $|\tilde{v}| = |\tilde{x}|$, and similarly for the replicated input.*

5 Behavioral equivalence

The aim of this section is the behavioral characterization of a lexically scoped approach to distributed mobile calculi. We present a standard (early) labeled transition system, and based on it we define a standard asynchronous

[3] Dpi type systems reject it since the channel is the same.

α	$\mathsf{fn}(\alpha)$	$\mathsf{bn}(\alpha)$
τ	$\emptyset$	$\emptyset$
$u?\langle\tilde{v}\rangle$	$\{u, \tilde{v}\}$	$\emptyset$
$(\nu\,\widetilde{n})\,u!\langle\tilde{v}\rangle$	$\{u\} \cup \{\tilde{v}\} \setminus \mathsf{subj}(\widetilde{n})$	$\mathsf{subj}(\widetilde{n})$

Fig. 7. Free and bound names in actions

bisimulation. We then prove an important law of lexically scoped distributed settings: a process behavior depends on the names it uses, not on where it runs. Using it, we show how to simulate a general go primitive, and a code optimization for process migration.

Labeled transition system. Consider two kinds of actions: those of processes, which are naturally like π-process actions; and those of networks, obtained from processes' actions by explicitly locating all its channels. The following grammar defines the set of *actions*, considering $\forall n \in \{\widetilde{n}\}.n \not\rightarrow u$.

$$\alpha ::= \tau \;\mid\; u?\langle\tilde{v}\rangle \;\mid\; (\nu\,\widetilde{n})\,u!\langle\tilde{v}\rangle$$

The subject of an action, $\mathsf{subj}(\alpha)$, is a partial function that uses the homonym notion over names, defined in Figure 2:

$$\mathsf{subj}(u?\langle\tilde{v}\rangle) = \mathsf{subj}((\nu\,\widetilde{n})\,u!\langle\tilde{v}\rangle) \stackrel{\text{def}}{=} \mathsf{subj}(u)\,.$$

Extending this notion to sequences of names in the expected way, the rules in the Figure 7 inductively define the sets of *free names*, $\mathsf{fn}(\alpha)$, and of *bound names*, $\mathsf{bn}(\alpha)$, in a action α. Let $\mathsf{names}(\alpha) \stackrel{\text{def}}{=} \mathsf{fn}(\alpha) \cup \mathsf{bn}(\alpha)$.

The channel translation function σ_s used in the migration axioms of the reduction relation is also used to locate all channels of an action.

$$\alpha\sigma_s \stackrel{\text{def}}{=} \begin{cases} \tau, & \text{if } \alpha = \tau\,; \\ u@s?\langle\tilde{v}@s\rangle, & \text{if } \alpha = u?\langle\tilde{v}\rangle\,; \\ (\nu\,\widetilde{n}@s)\,u@s!\langle\tilde{v}@s\rangle, & \text{if } \alpha = (\nu\,\widetilde{n})\,u!\langle\tilde{v}\rangle\,. \end{cases}$$

The rules in the Figure 8, together with rules symmetric to R-PAR and RP-COM, inductively define an *early labeled transition system* for $lsd\pi$. The system is standard, apart from rule RN-SITE that makes the location of all channels explicit for the network level.

Asynchronous bisimulation. We adapt the standard definition of an asynchronous bisimulation to $lsd\pi$ networks (cf. Sangiorgi and Walker [11]).

Definition 5.1 (Asynchronous bisimulation) *A symmetric binary relation $\mathcal{R}$ on networks is a* strong bisimulation*, or simply a bisimulation if, whenever $N\mathcal{R}M$:*

$$\text{[LRP-OUT]} \quad a!\langle\tilde{v}\rangle \xrightarrow{a!\langle\tilde{v}\rangle} \mathbf{0}$$

$$\text{[LRP-INP]} \quad a?(\tilde{x})P \xrightarrow{a?\langle\tilde{v}\rangle} P\{\tilde{v}/\tilde{x}\}$$

$$\text{[LRN-MIGO]} \quad r[a@s!\langle\tilde{v}\rangle] \xrightarrow{\tau} s[(a@s!\langle\tilde{v}\rangle)\sigma_r] \qquad r \neq s$$

$$\text{[LRN-MIGI]} \quad r[a@s?(\tilde{x})P] \xrightarrow{\tau} s[(a@s?(\tilde{x})P)\sigma_r] \qquad r \neq s$$

$$\text{[LRN-RMIGI]} \quad r[a@s?*(\tilde{x})P] \xrightarrow{\tau} s[(a@s?*(\tilde{x})P)\sigma_r] \qquad r \neq s$$

$$\text{[LRP-RCOMM]} \quad \frac{a?(\tilde{x})P \xrightarrow{\alpha} Q}{a?*(\tilde{x})P \xrightarrow{\alpha} a?*(\tilde{x})P \mid Q}$$

$$\text{[LRP-COMM]} \quad \frac{P \xrightarrow{(\nu\,\widetilde{n})\,a!\langle\tilde{v}\rangle} P' \quad Q \xrightarrow{a?\langle\tilde{v}\rangle} Q'}{P \mid Q \xrightarrow{\tau} (\nu\,\widetilde{n})\,(P' \mid Q')} \quad \mathsf{subj}(\widetilde{n}) \cap \mathsf{fn}(Q) = \emptyset$$

$$\text{[LR-RES]} \quad \frac{X \xrightarrow{\alpha} X'}{(\nu\,n)\,X \xrightarrow{\alpha} (\nu\,n)\,X'} \qquad \mathsf{subj}(n) \notin \mathsf{names}(\alpha)$$

$$\text{[LR-EXT]} \quad \frac{X \xrightarrow{\alpha} X'}{(\nu\,n)\,X \xrightarrow{(\nu\,n)\,\alpha} X'} \qquad \mathsf{subj}(n) \in \mathsf{fn}(\alpha) \setminus \mathsf{subj}(\alpha)$$

$$\text{[LR-PAR]} \quad \frac{X \xrightarrow{\alpha} X'}{X \mid Y \xrightarrow{\alpha} X' \mid Y} \qquad \mathsf{bn}(\alpha) \cap \mathsf{fn}(Y) = \emptyset$$

$$\text{[LR-ALPHA]} \quad \frac{X \equiv_\alpha X' \quad X' \xrightarrow{\alpha} Y' \quad Y' \equiv_\alpha Y}{X \xrightarrow{\alpha} Y} \qquad \text{[LRN-SITE]} \quad \frac{P \xrightarrow{\alpha} Q}{s[P] \xrightarrow{\alpha\sigma_s} s[Q]}$$

Fig. 8. Early labeled transition system of *lsdπ*

(i) $N \xrightarrow{\tau} N'$ *implies* $\exists M'$ $(M \xrightarrow{\tau} M'$ *and* $N'\mathcal{R}M')$.

(ii) $N \xrightarrow{((\nu\,\widetilde{n})\,u!\langle\tilde{v}\rangle)\sigma_s} N'$ *with* $\mathsf{subj}(\widetilde{n}\sigma_s) \cap \mathsf{fn}(M) = \emptyset$ *implies* $\exists M'$ $(M \xrightarrow{((\nu\,\widetilde{n})\,u!\langle\tilde{v}\rangle)\sigma_s} M' \wedge N'\mathcal{R}M')$.

(iii) $N \xrightarrow{(u?\langle\tilde{v}\rangle)\sigma_s} N'$ *implies* $\exists M'$ $((M \xrightarrow{(u?\langle\tilde{v}\rangle)\sigma_s} M'$ *and* $N'\mathcal{R}M') \vee (M \xrightarrow{\tau} M' \wedge N'\mathcal{R}(M \mid s[u!\langle\tilde{v}\rangle]))$.

We denote by $\sim$ *the largest bisimulation. Two networks* N *and* M *are* strongly bisimilar*, if there is a strong bisimulation* $\mathcal{R}$ *such that* $N\mathcal{R}M$.

The following proposition is a basic property of a bisimilarity relation.

Proposition 5.2 *Strong bisimilarity contains the structural congruence relation and is a congruence relation.*

The lexical scoping law. The main result of this section is an equivalence law capturing the lexical scope nature of *lsdπ*: a process behaves similarly when running at site s or at site r, as long as simple channels are instantiated

according to the environment they were meant for: that is the purpose of the translation function σ_s.

We say that P *is absolute* if $\mathsf{fn}(P) \cap \mathcal{C} = \emptyset$, i.e., if P has only located channels. A simple channel is *relative* to the site where it appears. Note that $P\sigma_s$ is an absolute process, irrespective of s. The following result says that the name translation that occurs at migration time does not change the behavior of the network, for it merely instantiates the simple local channels, which are relative to the site they appear in.

Proposition 5.3 $s[P] \sim s[P\sigma_s]$.

This law lies at the basis of several properties of *lsd*π networks. To show some of those properties, we introduce a weak form of bisimulation. Let a weak transition be $\stackrel{\alpha}{\Rightarrow} \stackrel{\text{abv}}{=} \stackrel{\tau}{\rightarrow}^* \stackrel{\alpha}{\rightarrow} \stackrel{\tau}{\rightarrow}^*$, if $\alpha \neq \tau$, and let $\stackrel{\tau}{\Rightarrow} \stackrel{\text{abv}}{=} \stackrel{\tau}{\rightarrow}^*$. Replacing everywhere in the Definition 5.1, strong transitions with weak transitions, one gets the notion of *weak bisimulation*. The largest weak bisimulation is weak bisimilarity, denoted by $\approx$, which is also a congruence.

The behavior of an absolute process depends on the names it uses, not on where it runs.

Theorem 5.4 *If P is absolute, then $s[P] \approx r[P]$.*

Migration does not change the behavior of the process; it simply advances computation.

Corollary 5.5 $s[P] \approx r[P\sigma_s]$

Assume a new production $\mathsf{go}\, r.P$ in the grammar of processes, which migrates a process P from site s to site r, according to the axiom.

$$s[\mathsf{go}\, r.P] \stackrel{\tau}{\rightarrow} r[P\sigma_s]$$

The results below are easy to establish using the previous laws.

Proposition 5.6

(i) $s[\mathsf{go}\, r.P] \approx s[(\nu\, a@r)\,(a@r!\langle\rangle \mid a@r?()P)]$, *where* $a@r \notin \mathsf{fn}(P)$.

(ii) $s[a@r?(\tilde{x})P] \approx s[(\nu\, c)\,(a@r?(\tilde{x})c!\langle\tilde{x}\rangle \mid c?(\tilde{y})P)]$, *where* $|\tilde{x}|=|\tilde{y}|$ *and* $c \notin \mathsf{fn}(P)$.

These last properties are examples of possible applications of the basic law in proving the correctness of encodings or of code optimizations.

6 Conclusions and future work

This paper presents *lsd*π, a calculus for distribution and mobility which proposes a lexical scope discipline for channels as an alternative to the established approach that assumes all channels as global. We have contributed with the theoretical treatment, stemming from the general lexical scope principles. It was not a trivial work, since basic aspects like binder properties, ranging from the definition of free names to its coherence with the operational semantics,

had to be reconsidered and redesigned. The result of this task is presented in a technical report [10], but here we have simplified the technicalities significantly: the withdrawal of syntactic restrictions and the pruning of the name translation associated to the migration primitive introduced in [13].

We have shown that the premise under which channels are the resources of sites is tightly weaved in the semantics of the language, and hope to have given elucidative examples and intuitions. Finally, a further evidence of the potentialities of the calculus is an equivalence law stating that a process behavior depends on the names it uses, not on where it runs. We foresee a wide range of applications of the law, but herein show only two of them: on comparing the expressive power of distributed calculi; and on designing code optimizations.

We find three main areas of future work to elaborate on the foundations presented here.

(i) To carry on work on the comparison between a lexical scope approach and an approach that considers channels as global network resources. The closest calculus to $lsd\pi$ is $d\pi_1^r$ (an asynchronous and receptive version of Dpi), but still several differences in the design choices of the calculi lead to technical difficulties in obtaining an absolute comparison. Therefore, we aim at comparing the *consequences* of adopting one scheme or another, on two of which we have ongoing work.

Sites as first class citizens. We have confronted ourselves with the question *Why pass sites?*, for it seems that lexical scope obviates the need of introducing such feature — instead of revealing the name of a site, giving in fact access to all of its resources (current and future), one can send the subset of its channels, to whom one intends to give access (like Dpi does via typing). We believe this choice does not result in serious loss of expressiveness, and is a research topic to pursue.

Migration mechanisms. While Dpi uses an explicit migration primitive, in $lsd\pi$ migration is triggered by the necessity of using a remote channel. We have presented a simulation of the migration primitive in $lsd\pi$, although the other direction has not yet been studied.

(ii) We believe lexically scoped distribution can be further shown worthwhile in various crucial subjects like implementation, security, and programming design. In particular, we envisage to control unrestricted migration (cf. [4]), and are currently working on a type system for doing that. We intend to investigate the security advantages of not passing sites, for we expect this choice to allow us to obtain granularity in the control of the information disclosed by a site at the language level.

(iii) At a time where effort is being put in the establishment of a common core programming model for global computation, we do not seek to present a complete calculus that would satisfy all the desirable requirements for such a challenging goal. Instead, we aim at a simple calculus which con-

centrates on the study of the name restriction discipline, arguing that lexically scoped distribution is a good option, since it allows to treat channels as the resources of locations. The work in this paper can constitute a foundation for generalizations and variations of the language, paving the way for further research making use of this discipline.

Acknowledgement

This work was partially supported by the EU FEDER and by the Portuguese Fundação para a Ciência e a Tecnologia (via CLC, the project MIMO, POSI/-CHS/39789/2001, and scholarships POCTI/SFRH/BPD/6782/2001 and PO-SI/SFRH/BD/7100/20∅1). The EU IST proactive initiative FET-Global Computing (via projects Mikado, IST–2001–32222, and Profundis, IST–2001–3310-0) was another source of finantial support.

We thank Gérard Boudol, Ilaria Castellani, Matthew Hennessy and Francisco Martins, as well as the anonymous referees, for their comments.

References

[1] Roberto M. Amadio, Gérard Boudol, and Cédric Lhoussaine. The receptive distributed π-calculus. Rapport de Recherche 4080, INRIA Sophia-Antipolis, 2000.

[2] Luca Cardelli. A language with distributed scope. In ACM, editor, *POPL'95: 22nd Annual ACM Symposium on Principles of Programming Languages (San Francisco, CA, U.S.A.)*, pages 286–297. ACM Press, 1995.

[3] Matthew Hennessy, Massimo Merro, and Julian Rathke. Towards a behavioural theory of access and mobility control in distributed systems. In Andrew Gordon, editor, *FOSSACS'03: 6th International Conference on Foundations of Software Science and Computation Structures (Warsaw, Poland)*, volume 2620 of *Lecture Notes in Computer Science*, pages 282–298. Springer-Verlag, 2003. Available as Technical Report COGS 2002:1, University of Sussex, U. K., 2002.

[4] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Journal of Information and Computation*, 173:82–120, 2002.

[5] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-Calculus*. Cambridge University Press, 1986.

[6] Robin Milner. The polyadic π-calculus: A tutorial. In *Logic and Algebra of Specification*, volume 94 of *Series F*. Springer-Verlag, 1993. Available as Technical Report ECS-LFCS-91-180, University of Edinburgh, U. K., 1991.

[7] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, 1992. Available as Technical Reports ECS-LFCS-89-85 and ECS-LFCS-89-86, University of Edinburgh, U. K., 1989.

[8] Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese. KLAIM: A kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.

[9] Rocco De Nicola, GianLuigi Ferrari, Rosario Pugliese, and Betti Veneri. Types for access control. *Theoretical Computer Science*, 240(1):215–254, 2000.

[10] António Ravara, Ana G. Matos, Vasco T. Vasconcelos, and Luís Lopes. A lexically scoped distributed π-calculus. DI/FCUL TR 02–4, Department of Computer Science, University of Lisbon, 2002.

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

[12] Vasco T. Vasconcelos and Kohei Honda. Principal typing schemes in a polyadic π-calculus. In Eike Best, editor, *Proceedings of CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 524–538. Springer-Verlag, 1993.

[13] Vasco T. Vasconcelos, Luís Lopes, and Fernando Silva. Distribution and mobility with lexical scoping in process calculi. In *HLCL'98*, volume 16 (3) of *Electronic Notes in Theoretical Computer Science*. Elsevier Science Publishers, 1998.