Acta Informatica manuscript No.
(will be inserted by the editor)

The M^3 Paradigm

Types and Type Inference for Ambient and Process Mobility

Mario Coppo[⋆]**, Mariangiola Dezani-Ciancaglini** [⋆⋆]**, Elio Giovannetti**[⋆⋆⋆]

Dipartimento di Informatica – Università di Torino
corso Svizzera 185, 10149 Torino, Italy
e-mail: {coppo,dezani,elio}@di.unito.it

Received: February 4, 2004 22:19/ Revised version: date

Abstract. A new kind of ambient calculi is presented, where the open capability is replaced by direct mobility of *naked* processes, while the associated type systems are algorithmic in the sense that they directly provide type inference procedures. The calculus comes equipped with a labelled transition system in which types play a major role: this system allows us to show interesting algebraic laws. Types express, as usual, the communication, access and mobility properties of the modelled system; inferred types express the minimal constraints required for the system to well behave.

1 Introduction

In the last few years a new conceptual dimension of computing has acquired a prominent role and is looking for an adequate theoretical foundation: space and movement in space.

A huge amount of computational entities distributed worldwide, exchanging data, moving from one location to another, interacting with each other (either cooperatively or competitively), give rise to global computing activity. Computation has therefore to be abstractly described as something

Send offprint requests to: Elio Giovannetti

⋆ Partially supported by MURST Cofin'01 NAPOLI Project and by the EU within the FET - Global Computing initiative, project MIKADO IST-2001-32222.

⋆⋆ Partially supported by MURST Cofin'01 COMETA Project and by the EU within the FET - Global Computing initiative, project DART IST-2001-33477.

⋆⋆⋆ Partially supported by MURST Cofin'01 NAPOLI Project and by EU within the FET - Global Computing initiative, project DART IST-2001-33477.

that develops not only in time and in memory space, either sequentially (λ-calculus) or as a dynamic set of concurrent processes (π-calculus), but also in a wide geographical and administrative space (ambient calculi).

The calculus of Mobile Ambients (MA)[8], building upon the concurrency paradigm represented by the π-calculus [17,16], introduced the notion of an ambient as "a bounded place where (generally multithreaded) computation happens", which can contain nested subambients in a tree structure, and which can move in and out of other ambients, i.e., up and down the tree (thus rearranging the structure itself). Direct communication can only occur locally within each ambient (through a common anonymous channel); communication and interaction between different ambients has to be mediated by movement and by the dissolution of ambient boundaries.

Ambients are intended to model mobile agents and processes, messages or packets exchanged over the network, mobile devices, physical and virtual locations, administrative and security domains, etc. in a uniform way.

For this reason, in ambient calculi the distinction between processes and (possibly mobile) containers of processes is intentionally blurred. In MA there are implicitly two main forms of entities, which we will respectively call *naked processes* and *ambient-processes*. The former are unnamed lists of actions[1] $\mathsf{act}_1.\mathsf{act}_2 \ldots \mathsf{act}_m$ to be executed sequentially, generally in concurrency with other processes: they can perform communication and drive their containers through the spatial hierarchy, but cannot individually go from one ambient to another. The latter are named containers of concurrent processes $m[P_1 \mid P_2 \ldots \mid P_n]$: they can enter and exit other ambients, driven by their internal processes, but cannot directly perform communication.

Therefore, mobile processes must be represented by ambient-processes; communication between them is represented by the exchange of other ambient-processes of usually shorter life, which have their boundaries dissolved by an open action so as to expose their internal naked processes performing the input-output proper. Such capability of opening an ambient, however, has been perceived by many as potentially dangerous, since it could be used inadvertently or maliciously to open and thus destroy the individuality of a mobile agent.

Among the many proposed variations of MA handling this issue, the calculus of Safe Ambients [11,5] introduced the notion of coaction, by which – among other things – an ambient cannot be opened if it is not willing to.

In the calculus of Boxed Ambients [6], on the other hand, open is dropped altogether, and its absence is compensated by the possibility of direct communication between parent and children ambients.

[1] As a matter of fact, a sequence of actions may also end with an asynchronous output, or an ambient-process creation $m[P]$, or a forking into different parallel naked processes.

In the present work, we explore a slightly different approach, where we intend to keep the purely local character of communication so that no hidden costs are present in the input-output primitives. At the same time we also want to represent inter-ambient communication by pure input-output between naked processes, avoiding the more general opening mechanism.

We do this by recovering the idea, present in Distributed π-calculus (Dπ) [9], that a naked process may move directly from one location to another, without the need of being enclosed in an ambient. Mobile naked processes also seem to more closely represent *strong software mobility*, by which a procedure (or function, or method, or script, depending on the programming model) can – through a go instruction – suspend its execution on one machine and resume it exactly from the same point on another (generally remote) machine, though for the moment we do not explore modelling this kind of application, which would probably need additional constructs.

All ambient calculi come with type systems as essential components, since – like any formal description – they are intended as foundations for reasoning about program behaviours in the new global computing reality. In our proposal too the calculus is an elementary basis for a type discipline able to control communication as well as access and mobility properties. We have tried to abstract (or extract), from the many complex features that could be envisaged, a system that is non-trivial but simple enough to be easily readable and understandable.

The system is incremental in the sense that it can type components in incomplete environments, and is supplied with a type inference algorithm that determines the "minimal" requirements for accepting a component as well typed.

In spite of its simplicity, the (typed) calculus seems to possess sufficient expressive power, admitting natural encodings of two standard calculi of concurrency, π and Dπ.

As usual, our system M^3, of *Mobile processes and Mobile ambients with Mobility types*, consists of two main interconnected components: a calculus and a type system.

2 The Calculus

The structural syntax of the pre-terms of our calculus, shown in Figure 1, is the same as that of MA except for the absence of open and the presence of the new primitive to for lightweight process mobility. Also, synchronous output is allowed, of which the asynchronous version is a particular case. As in MA, we introduce types – for the sake of simplicity – already in the term syntax, namely in the input construct and in the restrictions w.r.t. ambient names and group names. The terms defined in Figure 1 are not exactly

$M,\ N,\ L$	$::=$		**messages**
		$m, n, \ldots x, y, \ldots$	ambient names, variables
		$\mathsf{in}\ M$	moves the containing ambient into ambient M
		$\mathsf{out}\ M$	moves the containing ambient out of ambient M
		$\mathsf{to}\ M$	goes out from its ambient into sibling ambient M
		$M.M'$	path
π	$::=$		**prefixes**
		$M\,.$	capability
		$\langle M \rangle$	synchronous output
		$(x{:}\,W)$	typed input
$P,\ Q,\ R$	$::=$		**processes**
		$\mathsf{0}$	null
		$\pi\, P$	prefixed
		$P \mid Q$	parallel composition
		$M[P]$	ambient
		$!\,\pi\, P$	guarded replication
		$(\nu n{:}\,\mathsf{amb}(g))P$	name restriction
		$(\nu\{\overrightarrow{\mathbf{g}{:}\mathbf{G}}\}_{(k)})P$	group restriction

where: W is a message type, g is a group name, $\nu\{\overrightarrow{\mathbf{g}{:}\mathbf{G}}\}_{(k)}$ is a concise notation for $\nu\{g_1{:}\,G_1, \ldots, g_k{:}\,G_k\}$, with $g_1, \ldots, g_k$ group names and $G_1, \ldots, G_k$ group types (see Fig. 4).

Fig. 1 Syntax

the terms of our calculus, since the type constraints are not yet taken into account. This will be done by the typing rules of Figure 5. The notions of reduction and structural equivalence do not rely upon the fact that terms are well-typed but obviously they are only meaningful for well-typed terms.

Since group types contain group names, it is crucial to restrict sets of group names. We denote by $\nu\{g_1{:}\,G_1, \ldots, g_k{:}\,G_k\}$ the simultaneous restriction of the group names $g_1, \ldots, g_k$ having respectively group types $G_1, \ldots, G_k$. Simultaneous restriction is needed since groups can have mutually dependent group types. We adopt the standard convention that action prefixing takes precedence over parallel composition: if act denotes a generic prefix, $\mathsf{act}.\alpha \mid \beta$ is read as $(\mathsf{act}.\alpha)|\beta$. We follow the traditional distinction between letters m, n, ... for *ambient names* and letters x, y, ... for input variables. Formally they are however in a single syntactic category, which we call *variables*; simply, ambient names are variables that either are free and do not occur in prefix or path prefix position (e.g., $x.P$, $x.M$), or are bound by the ν-binder (but not by the input binder). The letter ξ will be used to denote a generic variable that may be an ambient name or an input variable indifferently.

Also, the *Barendregt convention* [2] is assumed to hold on variables and on *group names* (see below) for any term or set of terms one is considering: all the bound variables (or respectively the bound group names) are distinct among themselves, and distinct from all the free variables (or re-

spectively the free group names). In this way all the problems connected with α-conversion and capturing of free names are avoided.

The operational semantics consists, as usual, of a reduction relation, along with a structural congruence which allows trivial syntactic restructuring of a term so that a reduction rule can next be applied.

Structural congruence (shown in Figure 2) is almost standard for the usual ambient constructors [7], but for the rule $(\nu n\colon g)n[\mathsf{0}] \equiv \mathsf{0}$ which is added to get a form of garbage collection in absence of the open primitive and for the rules to handle simultaneous group restriction. These new rules allow to permute, split and erase group restrictions under suitable conditions. Despite their awkward look they are essentially similar to the rules for names restriction. What complicates notations is the fact that mutually dependent group types must be handled contemporarily.

Observe that, owing to the Barendregt convention, we can omit to write several side conditions. Moreover the side conditions in certain rules are necessary only to control the moving of restriction from outside in. Barenderegt convention allows instead to move restriction from inside out without mentioning any side condition.

The congruence is almost the standard relation found in [7].

The reduction rules, shown in Figure 3, are the same as those for MA, with the obvious difference consisting in the synchronous output and the missing open, and with the new rule for the to action, similar to the go primitive of Dπ or to the "migrate" instructions for strong code mobility in software agents.

A lightweight process executing a $\mathsf{to}\, m$ action moves between sibling ambients: more precisely it goes from an ambient n, where it is initially located, to a (different) ambient of name m that is a sibling of n, thus crossing two boundaries in one step; the boundaries are however at the same level, so that – differently from moving upward or downward – the process does not change its nesting level.

Observe that the form of the rule, while entailing nondeterminism among different destinations of the same name, guarantees that the destination, though possibly having the same name as the source, must be a different ambient: so that a term of the form $m[\mathsf{to}\, m.P]$ cannot reduce to $m[P]$, with a jump from one to the very same location!

3 The Type System

The syntax definition of Figure 1 is not really complete without giving the rules for defining well-typed terms. Types firstly ensure that meaningless terms cannot be defined or be produced by computation.

equivalence:

$$P \equiv P \qquad P \equiv Q \implies Q \equiv P \qquad P \equiv Q,\ Q \equiv R \implies P \equiv R$$

congruence:

$$\begin{array}{ll} P \equiv Q \implies \pi\, P \equiv \pi\, Q & P \equiv Q \implies M[P] \equiv M[Q] \\ P \equiv Q \implies !\,\pi\, P \equiv !\,\pi\, Q & P \equiv Q \implies (\nu n\colon \mathsf{amb}(g))P \equiv (\nu n\colon \mathsf{amb}(g))Q \\ P \equiv Q \implies P \mid R \equiv Q \mid R & P \equiv Q \implies (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P \equiv (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})Q \end{array}$$

prefix associativity:

$$(M.M').P \equiv M.M'.P$$

parallel composition – associativity, commutativity, zero:

$$P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid \mathsf{0} \equiv P$$

replication:

$$!\,\pi\, P \equiv \pi\, P \mid !\,\pi\, P$$

restriction swapping and group restriction splitting :

$$(\nu n\colon \mathsf{amb}(g))(\nu m\colon \mathsf{amb}(g'))P \equiv (\nu m\colon \mathsf{amb}(g'))(\nu n\colon \mathsf{amb}(g))P$$

$$g_i \notin GN(G'_j) \& g'_j \notin GN(G_i)(1 \le i \le k)(1 \le j \le h)$$
$$\implies (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})(\nu\{\overrightarrow{\mathbf{g}':\mathbf{G}'}\}_{(h)})P \equiv (\nu\{\overrightarrow{\mathbf{g}':\mathbf{G}'}\}_{(h)})(\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P$$

$$(\nu n\colon \mathsf{amb}(g))(\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P \equiv (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})(\nu n\colon \mathsf{amb}(g))P$$

$$g_{k+j} \notin GN(G_i)(1 \le i \le k)(1 \le j \le h)$$
$$\implies (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k+h)})P \equiv (\nu\{g_1\colon G_1, \ldots, g_k\colon G_k\})(\nu\{g_{k+1}\colon G_{k+1}, \ldots, g_{k+h}\colon G_{k+h}\})P$$

scope extrusion:

$$\begin{array}{ll} (\nu n\colon \mathsf{amb}(g))P \mid Q \equiv (\nu n\colon \mathsf{amb}(g))(P \mid Q) & \text{if } n \notin AN(Q) \\ \xi[(\nu n\colon \mathsf{amb}(g))P] \equiv (\nu n\colon \mathsf{amb}(g))\xi[P] & \\ (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P \mid Q \equiv (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})(P \mid Q) & \text{if } \overrightarrow{g} \notin GN(Q) \\ \xi[(\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P] \equiv (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})\xi[P] & \end{array}$$

equivalence to zero:

$$(\nu n\colon \mathsf{amb}(g))\,\mathsf{0} \equiv \mathsf{0} \qquad (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})\,\mathsf{0} \equiv \mathsf{0} \qquad (\nu n\colon \mathsf{amb}(g))n[\mathsf{0}] \equiv \mathsf{0}$$

where $AN(Q)$ is the set of free ambient names in Q, $GN(Q)$ is the set of free group names in Q, and $GN(G)$ is the set of free group names in G.

Fig. 2 Structural congruence: general rules

In addition, we want use types for the control of access and mobility. As already observed in [7], expressing such properties in terms of single ambients leads to *dependent types*, for example in judgments of the form $n\colon CanEnter(\{m_1, \ldots, m_k\})$. An interesting proposal in this direction is [13]. Following the seminal work of [7], and [15] among others, we therefore adopt an approach based on ambient *groups*, which permits us to avoid direct dependence of types on values.

As usual for ambients, there are three fundamental categories of types: ambient types, capability types, and process types, corresponding to the three main syntactic categories of terms. Since only ambient names and capabilities, but not processes, can be transmitted, message types – i.e.,

Basic reduction rules:

$$
\begin{array}{lrcl}
\text{(R-in)} & n[\,\mathsf{in}\,m\,.\,P \mid Q\,] \mid m[R] & \rightarrow & m[\,n[\,P \mid Q\,] \mid R\,] \\
\text{(R-out)} & m[\,n[\,\mathsf{out}\,m\,.\,P \mid Q\,] \mid R\,] & \rightarrow & n[\,P \mid Q\,] \mid m[R] \\
\text{(R-to)} & n[\mathsf{to}\,m\,.\,P \mid Q] \mid m[R] & \rightarrow & n[Q] \mid m[P \mid R] \\
\text{(R-comm)} & (x\colon W)P \mid \langle M\rangle\,.\,Q & \rightarrow & P\{x := M\} \mid Q
\end{array}
$$

Structural reduction rules:

$$
\begin{array}{lrcl}
\text{(R-in)} & P \rightarrow Q & \Rightarrow & P \mid R \rightarrow Q \mid R \\
\text{(R-amb)} & P \rightarrow Q & \Rightarrow & n[P] \rightarrow n[Q] \\
\text{(R-}\equiv\text{)} \quad P' \equiv P',\ P \rightarrow Q,\ Q \equiv Q' & & \Rightarrow & P' \rightarrow Q' \\
\text{(R-}\nu\text{)} & P \rightarrow Q & \Rightarrow & (\nu n\colon g)P \rightarrow (\nu n\colon g)Q \\
\text{(R-}\nu\text{-group)} & P \rightarrow Q & \Rightarrow & (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P \rightarrow (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})Q
\end{array}
$$

Fig. 3 Reduction

	$g, h, \ldots$	groups
	$\mathscr{I}, \mathscr{C}, \mathscr{E}, \ldots$	sets of groups; $\mathbb{G}$ is the universal set of groups
Amb	::= $\mathsf{amb}(g)$	ambient type: ambients of group g
Pro	::= $\mathsf{pr}(g)$	process type: processes that can stay in ambients of group g
Cap	::= $Pro_1 \rightarrow Pro_2$	capability type: capabilities that, prefixed to a process of type Pro_1, turn it into a process of type Pro_2
W	::=	message type
	Amb	ambient type
	Cap	capability type
T	::=	communication type
	shh	no communication
	W	communication of messages of type W
G	::= $\mathsf{gr}(\mathscr{I}, \mathscr{C}, \mathscr{E}, T)$ where $\mathscr{C} \subseteq \mathscr{I}$	group type

Fig. 4 Types

the ones explicitly attached to input variables – can only be ambient or capability types.

Syntactically, groups are merely names $g, h, \ldots$ occurring as basic components of other types. Formally, they may be considered atomic types, which represent sets of ambients sharing some common features.

There is a subtle difference w.r.t. [7] and [15]. In those systems, ambient types are of the schematic form $\mathsf{amb}(g, \mathcal{B})$, where $\mathcal{B}$ is the expression of some behavioral properties concerning mobility and communication. In our proposal the property $\mathcal{B}$ is instead (the content of) the type of the group. The typing judgment $m\colon\mathsf{amb}(g, \mathcal{B})$ becomes, in our system, $m\colon\mathsf{amb}(g),\ g\colon\mathcal{B}$.

The first form is more general, allowing different ambient types for the same group. In our approach, on the other hand, a group represents a set of

ambients guaranteed to share common mobility and access properties (and communication behaviour), as specified by the group's type.

The only component of an ambient type $\mathsf{amb}(g)$ or a process type $\mathsf{proc}(g)$ is a group name g, whose type[2] G describes – in terms of other group names (possibly including the very group g typed by G) – the properties of all the ambients and processes of that group. In a sense, groups and group types work as indirections between types and values, so as to avoid that types directly "point to" (i.e., depend on) values.

As is standard, the connection between ambients and processes is given by the fact that processes of type $\mathsf{pr}(g)$ can run safely only within ambients of type $\mathsf{amb}(g)$.

The form of capability types is a relative novelty: they are (very particular) sorts of function types from processes to processes, corresponding to the fact that, from a syntactic point of view, a prefix turns a process into another process; $\mathsf{proc}(g_1) \to \mathsf{proc}(g_2)$ is the type of a capability that, prefixed to a process of type $\mathsf{proc}(g_1)$, transforms it into a process of type $\mathsf{proc}(g_2)$; or, viewed at runtime, a capability that, when exercised by a process of type $\mathsf{proc}(g_2)$, of course located in an ambient of type $\mathsf{amb}(g_2)$, leaves a continuation of type $\mathsf{proc}(g_1)$, located in an ambient of type $\mathsf{amb}(g_1)$.

This form bears some non-superficial resemblance to that of [1], where a capability type is a type context which, when filled with a process type, yields another process type.

Notational Remark: we shall simply write g both for $\mathsf{amb}(g)$ and for $\mathsf{pr}(g)$, the distinction always being clear from the context. As a consequence, capability types will be written in the concise form $g_1 \to g_2$.

Besides, we may use the abbreviations *g-ambients* and *g-processes* respectively for *the ambients of group g* and *the processes of group g*.

The communication type is completely standard: it is either the atomic type shh, denoting absence of communication, or a message type, which in turn may be an ambient type or a capability type. Note that the type shh, typical of ambient systems, is not the type of empty messages (which can be used for synchronization), but the one denoting the very absence of input-output.

Finally, *group types* (ranged over by G) consist of four components and are of the form $\mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, T)$, where $\mathscr{S}, \mathscr{C}$ and $\mathscr{E}$ are sets of group names, and T is a communication type. If g is a group of type $\mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, T)$, then the intuitive meanings of the type's components are the following:

– $\mathscr{S}$ is the set of ambient groups where the ambients of group g can stay;

[2] Observe that a group type is the type of a type. Thus, following a rather standard terminology, it is a *kind*; moreover, since it contains group names, it might be considered a "kind dependent on types". However, this double level is used, as is clear, only in a very limited and ad hoc way, with no real stratification; it is therefore justified not to use the expression *group kind*, but simply stick to *group type*.

- $\mathscr{C}$ is the set of ambient groups that g-ambients can cross, i.e., those that they may be driven into or out of, respectively, by in or out actions; clearly, it must be $\mathscr{C} \subseteq \mathscr{S}$ (and $\mathscr{C}$ is empty if the ambients of group g are immobile);
- $\mathscr{E}$ is the set of ambients that (lightweight) g-processes can "enter": more precisely, those to which a g-process may send its continuation by means of a to action (it is empty if lightweight g-processes are immobile);
- T is the (fixed) communication type (or topics of conversation) within g-ambients.

The information of $\mathscr{S}$ and $\mathscr{C}$ component of a group type was considered also in [15].

If $G = \mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, T)$ is a group type, we write $\mathscr{S}(G)$, $\mathscr{C}(G)$, $\mathscr{E}(G)$, $T(G)$ respectively to denote the components $\mathscr{S}$, $\mathscr{C}$, $\mathscr{E}$, T of G.

An *environment* Ξ consists of two components: a *group environment* Γ and a *variable (and ambient) environment* Δ, as defined by the following syntax:

$$\Xi ::= \Gamma; \Delta \qquad \Gamma ::= \varnothing \mid \Gamma,\ g{:}G \qquad \Delta ::= \varnothing \mid \Delta,\ \xi{:}W$$

where ξ is a variable or an ambient name.

The *domain* of an environment is $Dom(\Xi) = Dom(\Gamma) \cup Dom(\Delta)$, where:

$$\begin{gathered} Dom(\varnothing) = \varnothing \\ Dom(\Gamma,\ g{:}G) = Dom(\Gamma) \cup \{g\} \\ Dom(\Delta,\ \xi{:}W) = Dom(\Delta) \cup \{\xi\} \end{gathered}$$

$GN(G)$ denotes the set of all group names occurring in a group type G, and $GN(\Xi)$ denotes the set of all group names occurring in Ξ, not only in $Dom(\Gamma)$ but also in the components of the types in Ξ. Environments are considered as sets of statements, therefore modulo permutations.

A variable environment Δ is *well-formed* if for each $\xi \in Dom(\Delta)$ there is exactly one type associated to it in Δ, i.e., there cannot exist $\xi{:}W_1,\ \xi{:}W_2 \in \Delta$ with W_1 different from W_2. We assume that all variable environments are well-formed.

Analogously, a group environment Γ is *well-formed* if for each $g \in Dom(\Gamma)$ there is exactly one group type G associated to it in Γ. Of course, only well-formed group environments are allowed in a typing judgement, but we will see that (potentially) non-well-formed group environments are used by the type inference procedure.

Environments are seen as sets, modulo permutations and duplication. We use the standard notation $\Delta,\ \xi{:}W$ to denote a variable environment containing a statement $\xi{:}W$, assuming that $\xi \notin Dom(\Delta)$. Moreover we

intend that Δ_1, Δ_2 represent the variable environment defined by the set union of Δ_1 and Δ_2 (i.e. by eliminating duplicates). We adopt a similar convention for group environments and environments.

The formal definition of the type assignment rules is shown in Figure 5.

$$\frac{\xi : T \in \Delta}{\Gamma;\, \Delta \vdash \xi : T}\ (\textsc{Var}) \qquad \frac{g : G \in \Gamma}{\Gamma;\, \Delta \vdash g : G}\ (\textsc{Group}) \qquad \frac{}{\Xi \vdash 0 : g}\ (\textsc{Null})$$

$$\frac{\Xi \vdash g_2 : G_2 \quad \Xi \vdash M : g_1 \quad g_1 \in \mathscr{C}(G_2)}{\Xi \vdash \mathsf{in}\, M : g_2 \to g_2}\ (\textsc{In})$$

$$\frac{\Xi \vdash g_1 : G_1 \quad \Xi \vdash g_2 : G_2 \quad \Xi \vdash M : g_1 \quad g_1 \in \mathscr{C}(G_2) \quad \mathscr{S}(G_1) \subseteq \mathscr{S}(G_2)}{\Xi \vdash \mathsf{out}\, M : g_2 \to g_2}\ (\textsc{Out})$$

$$\frac{\Xi \vdash g_2 : G_2 \quad \Xi \vdash M : g_1 \quad g_1 \in \mathscr{E}(G_2)}{\Xi \vdash \mathsf{to}\, M : g_1 \to g_2}\ (\textsc{To})$$

$$\frac{\Xi \vdash M : g_3 \to g_2 \quad \Xi \vdash N : g_1 \to g_3}{\Xi \vdash M.N : g_1 \to g_2}\ (\textsc{Path})$$

$$\frac{\Xi \vdash M : g_1 \to g_2 \quad \Xi \vdash P : g_1}{\Xi \vdash M.P : g_2}\ (\textsc{Prefix-Cap})$$

$$\frac{\Gamma;\, \Delta, x : W \vdash P : g}{\Gamma;\, \Delta \vdash (x : W)P : g}\ (\textsc{Input})$$

$$\frac{\Xi \vdash P : g \quad \Xi \vdash M : W \quad \Xi \vdash g : \mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, W)}{\Xi \vdash \langle M \rangle P : g}\ (\textsc{Output})$$

$$\frac{\Xi \vdash P : g \quad \Xi \vdash M : g \quad \Xi \vdash g : G \quad g' \in \mathscr{S}(G)}{\Xi \vdash M[P] : g'}\ (\textsc{Amb})$$

$$\frac{\Xi \vdash P : g \qquad \Xi \vdash Q : g}{\Xi \vdash P \,|\, Q : g}\ (\textsc{Par}) \qquad \frac{\Xi \vdash \pi\, P : g}{\Xi \vdash !\, \pi\, P : g}\ (\textsc{Repl})$$

$$\frac{\Gamma;\, \Delta, m : g' \vdash P : g}{\Gamma;\, \Delta \vdash (\nu m : g')P : g}\ (\textsc{AmbRes})$$

$$\frac{\Gamma,\, g_1 : G_1, \ldots, g_k : G_k;\, \Delta \vdash P : g \quad g_i \notin GN(\Gamma;\, \Delta) \quad g_i \neq g \quad (1 \leq i \leq k)}{\Gamma;\, \Delta \vdash (\nu\{g_1 : G_1, \ldots, g_k : G_k\})P : g}\ (\textsc{GrpRes})$$

Fig. 5 Typing rules

The system's fundamental rule (AMB) is quite standard: it requires that in a term $m[P]$ the ambient m and its content P be of the same group, while the process $m[P]$, being a completely passive object, unable both to communicate and to move other ambients, may in turn stay in any ambient of any group g' (i.e., it may be of any group g'), *provided* its "membrane" m, of type g, has permission from the specification G to stay in a g'-ambient.

Since a process executing an action $\mathsf{to}\, m$ goes from its ambient (in)to an ambient m, the rule (TO) states that the action $\mathsf{to}\, m$, if performed by a process of group g_2 (in a g_2-ambient), leaves as continuation a process of group g_1, if g_1 is the group of m and moreover is one of the groups to which g_2-*processes* are allowed to go (i.e., to send their continuations) by a to.

The rules (IN) and (OUT) state that a process exercising an $\mathsf{in}/\mathsf{out}\ m$ capability does not change its group g_2 since it does not change its enclosing g_2-ambient, which must however have permission to cross the g_1-ambient m; in the case of (OUT), moreover, the g_2-ambient – being driven out of m – becomes a sibling of m, and must therefore have permission to stay where m stays (i.e., the condition $\mathscr{S}(G_1) \subseteq \mathscr{S}(G_2)$). The analogous side condition in the rule (IN), ensuring that the moving g_2-ambient has the permission to stay inside the g_1-ambient m ($g_1 \in \mathscr{S}(G_2)$) is subsumed by the condition $\mathscr{C}(G) \subseteq \mathscr{S}(G)$ on group types.

The rules (PATH) and (PREFIX-CAP) are as expected from the informal definitions of process and capability types: kinds, respectively, of function composition and function application. The other rules are standard: in the group restriction the set of group names $g_1, \ldots, g_k$ that are abstracted from the environment (i.e., moved from the l.h.s. to the r.h.s. of the turnstile) cannot contain the group g of the restricted term.

The type assignment system is clearly syntax-directed and therefore a Generation Lemma trivially holds.

Lemma 1 (Generation Lemma).

1. *If* $\Xi \vdash \mathsf{in}\ M : g_2 \to g_2$ *then* $\Xi \vdash g_2 : G_2$, $\Xi \vdash M : g_1$, *and* $g_1 \in \mathscr{C}(G_2)$ *for a unique* g_1.
2. *If* $\Xi \vdash \mathsf{out}\ M : g_2 \to g_2$ *then* $\Xi \vdash g_1 : G_1$, $\Xi \vdash g_2 : G_2$, $\Xi \vdash M : g_1$, $g_1 \in \mathscr{C}(G_2)$, *and* $\mathscr{S}(G_1) \subseteq \mathscr{S}(G_2)$ *for a unique* g_1.
3. *If* $\Xi \vdash \mathsf{to}\, M : g_1 \to g_2$ *then* $\Xi \vdash g_2 : G_2$, $\Xi \vdash M : g_1$, *and* $g_1 \in \mathscr{E}(G_2)$.
4. *If* $\Xi \vdash M.N : g_1 \to g_2$ *then* $\Xi \vdash M : g_3 \to g_2$, *and* $\Xi \vdash N : g_1 \to g_3$ *for a unique* g_3.
5. *If* $\Xi \vdash M.P : g_2$ *then* $\Xi \vdash M : g_1 \to g_2$, *and* $\Xi \vdash P : g_1$ *for a unique* g_1.
6. *If* $\Gamma; \Delta \vdash (x : W)P : g$ *then* $\Gamma; \Delta, x : W \vdash P : g$.
7. *If* $\Xi \vdash \langle M \rangle P : g$ *then* $\Xi \vdash P : g$, $\Xi \vdash M : W$, *and* $\Xi \vdash g : \mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, W)$ *for a unique* $\mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, W)$.

8. *If* $\Xi \vdash M[P] : g$ *then* $\Xi \vdash P : g'$, $\Xi \vdash M : g'$, $\Xi \vdash g' : G$, *and* $g \in \mathscr{S}(G)$ *for a unique* g', G.
9. *If* $\Xi \vdash P \mid Q : g$ *then* $\Xi \vdash P : g$, *and* $\Xi \vdash Q : g$.
10. *If* $\Xi \vdash ! \, \pi \, P : g$ *then* $\Xi \vdash \pi \, P : g$.
11. *If* $\Gamma; \Delta \vdash (\nu m : g')P : g$ *then* $\Gamma;\ \Delta, m : g' \vdash P : g$.
12. *If* $\Gamma, \Delta \vdash (\nu\{g_1 : G_1, \ldots, g_k : G_k\})P : g$ *then* $\Gamma,\ g_1 : G_1, \ldots, g_k : G_k;\ \Delta \vdash P : g$, $g_i \notin GN(\Gamma;\ \Delta)$ *and* $g_i \neq g$ $(1 \leq i \leq k)$.

Note this lemma implies that for each typing judgement $\Xi \vdash P : g$ there is a unique deduction of it.

The usual property of subject reduction holds, which guarantees the soundness of the system by ensuring that typing is preserved by computation. Notice that we do not need to expand environments as [7], since we allow variable environments to contain group names in types also when there is no group type associated to them in the pairing group envionment (provided this is compatible with the assignment rules), i.e., we allow $\Gamma;\ \Delta$ with $\xi : g \in \Delta$ even if $g \notin Dom(\Gamma)$.

Theorem 1 (Subject reduction). *Let* $\Xi \vdash P : g$ *. Then*

1. $P \equiv Q$ *implies* $\Xi \vdash Q : g$.
2. $P \ \rightarrow \ Q$ *implies* $\Xi \vdash Q : g$

Proof. The proof is standard, by induction on the derivations of $P \equiv Q$ and $P \ \rightarrow \ Q$ using the Generation Lemma. We only consider rule (R-to):

$$n[\mathsf{to}\, m \,.\, P \mid Q] \mid m[R] \ \rightarrow \ n[Q] \mid m[P \mid R].$$

If $\Xi \vdash n[\mathsf{to}\, m \,.\, P \mid Q] \mid m[R] : g$ then by Lemma 1(9) $\Xi \vdash n[\mathsf{to}\, m \,.\, P \mid Q] : g$ and $\Gamma \vdash m[R] : g$. By 1(8) we must have $\Xi \vdash \mathsf{to}\, m \,.\, P \mid Q : g_n$, $\Xi \vdash n : g_n$, $\Gamma \vdash g_n : G_n$, $g \in \mathscr{S}(G_n)$, $\Xi \vdash R : g_m$, $\Xi \vdash m : g_m$, $\Gamma \vdash g_m : G_m$ and $g \in \mathscr{S}(G_m)$, for some g_n, G_n, g_m, G_m. From $\Gamma \vdash \mathsf{to}\, m \,.\, P \mid Q : g_n$ by Lemma 1(9) we have $\Xi \vdash Q : g_n$ and $\Xi \vdash \mathsf{to}\, m \,.\, P : g_n$, which imply by Lemma 1(5) and (3) $\Xi \vdash P : g_m$.
Rule $(\textsc{Amb})$ applied to $\Xi \vdash Q : g_n$, $\Xi \vdash n : g_n$, $\Xi \vdash g_n : G_n$ gives $\Xi \vdash n[Q] : g$ being $g \in \mathscr{S}(G_n)$. Rule $(\textsc{Par})$ applied to $\Xi \vdash P : g_m$, $\Xi \vdash R : g_m$ gives $\Xi \vdash P \mid R : g_m$. Since $g \in \mathscr{S}(G_m)$ we can deduce $\Xi \vdash m[P \mid R] : g$ using rule $(\textsc{Amb})$. We conclude $\Xi \vdash n[Q] \mid m[P \mid R] : g$ from $\Xi \vdash n[Q] : g$ and $\Xi \vdash m[P \mid R] : g$ by rule $(\textsc{Amb})$.

4 Behavioural Semantics

Being M3 a typed calculus we have to take into account the restrictions on movements given by the environments. So we cannot use the original notion of observable of Mobile Ambients [8]:

$$P\downarrow_n \quad \triangleq \quad P \equiv (\nu\tilde{m})(n[P'] \mid Q) \qquad n \notin \tilde{m}$$

where all names of ambients at top level are observable. We require for an ambient name in order to be observable that one can build at least one ambient allowed to cross it and/or to send it a process. Similar conditions are assumed for instance for safe ambients [12], where this control is given by the presence of suitable coactions. In our case this depends on the existence in the environment of at least one group with the necessary rights.

Note also that group restriction cannot affect in any way the computational properties of processes. In fact a group name g can be restricted only if in the variable environment there are neither variables nor ambient names whose type mention g. This meas that these names have already been abstracted or restricted and then in both cases are no more visible. So in this section we will consider only processes without occurrences of group restrictions.

Therefore we take the following definition of *observability of an ambient name n in an environment Ξ*. Let $\Xi = \Delta,\, n : g';\, \Gamma$:

$$\begin{array}{l} P\downarrow_n^{\Xi,g} \triangleq\ P \equiv (\widetilde{\nu m{:}g})(n[P'] \mid Q) \ \text{ and } \ \Xi \vdash P : g \\ \qquad\qquad \text{and there exists } \ g'' : G'' \in \Gamma \text{ such that } g' \in \mathscr{C}(G'') \cup \mathscr{E}(G'') \end{array}$$

We will use $P\Downarrow_n^{\Xi,g}$ as short for $P \ \rightarrow^*\downarrow_n^{\Xi,g}$.

Being observability relative to environments and groups we define also barbed bisimulation and barbed congruence [18], [11] by taking into account environments and group types. More precisely environments and groups are used both to type the processes to be compared with the enclosing contexts and to test observability.

Let $\mathcal{R}^{\Xi,g}$ denote a relation between processes such that $P\mathcal{R}^{\Xi,g}\,Q$ implies that $\Xi \vdash P : g$ and $\Xi \vdash Q : g$. We call it a Ξ, g*-relation.*

Definition 1. *(i) An Ξ, g-relation is reduction closed if $P\mathcal{R}^{\Xi,g}\,Q$ and $P \ \rightarrow P'$ imply the existence of some Q' such that $Q \ \rightarrow^* Q'$ and $P'\mathcal{R}^{\Xi,g}\,Q'$*
(ii)An Ξ, g-relation is barb preserving with respect to if $P\mathcal{R}^{\Xi,g}\,Q$ and $P\downarrow_n^{\Xi,g}$ imply $Q\Downarrow_n^{\Xi,g}$.
*(iii) The $\Xi,\ g$-*barbed bisimulation*, noted $\cong_b^{\Xi,g}$, is the largest Ξ, g-equivalence relation over closed processes which is reduction closed and barb preserving.*

To introduce the notion of barbed congruence we need the notion of *context* $C[\ ]$, defined as usual as a process with an hole in it. As we remarked before behavioral properties are transparent to group restrictions. Then it is not restrictive to consider in the following only contexts without occurrences of group restrictions. We say that a context $C[\,]$ *agrees* with an

environment Ξ and a group g, notation $C[\,]^{\Xi,g}$ if there is an environment Ξ' and a deduction $\mathcal{D}$ of $\Xi' \vdash C[0] : g'$ for some type g' such that in $\mathcal{D}$ there is a statement $\Xi \vdash 0 : g$ corresponding to the occurrence of 0 in the hole. Since we assume that there are no group restrictions this imply $\Xi' \subseteq \Xi$. In this case we say that $\Xi',\ g'$ are *compatible* with $C[\,]^{\Xi,g}$.

Definition 2 (Barbed Bisimulation). *(i) We say that a relation $\mathcal{R}^{\Xi,g}$ is* contextual *if $P\mathcal{R}^{\Xi,g}\,Q$ implies that $C[P]^{\Xi,g}\mathcal{R}^{\Xi',g'}\,C[Q]^{\Xi,g}$ for all contexts $C[\,]^{\Xi,g}$ and all Ξ', g' compatible with $C[\,]^{\Xi,g}$.*
(ii) barbed congruence $\cong^{\Xi,g}$ *is the maximal contextual $\Xi,\ g$-equivalence relation which is a barbed bisimulation when restricted to closed processes.*

As usual to study barbed congruences we introduce a labelled transition system. Our labelled transitions have the shape:

$$P \xrightarrow{\ \alpha\ }_{\Xi,g} O$$

where

- $\Xi \vdash P : g$
- the *label* α encodes the minimal contribution by the surrounding context needed by the process to complete the transition;
- the *outcome* O can be either a *concretion*, i.e. a partial derivative which needs a contribution from the surrounding context to be completed, or a process.

Table 1 defines labels and concretions. In $(\boldsymbol{\nu}\widetilde{p:g})\langle P\rangle Q$ the process P represents the moving ambient and the process Q represents the remaining system not affected by the movement. In $(\boldsymbol{\nu}\widetilde{p:g})\langle M\rangle P$ the message M represents the information transmitted and the process P represents the remaining system not affected by the output. In both cases $\widetilde{p:g}$ is the set of shared private names.

Tables 2, 3, and 4 give the labelled transition system. The rules are standard [11], [14], [4], but for rules (CO-IN) and (CO-TO) which require the environment Ξ gives visibility to the name n as in the definition of $\Downarrow_n^{\Xi}$.

To avoid to write boring side conditions on transition rules we convene that in a rule of the shape

$$\frac{P \xrightarrow{\ \alpha\ }_{\Xi,g} O}{P' \xrightarrow{\ \alpha'\ }_{\Xi,g'} O'}$$

we implicitly assume $\Xi \vdash P : g$ and $\Xi \vdash P' : g'$.

In writing rules we will use the following standard conventions:

- if O is the concretion $(\boldsymbol{\nu}\widetilde{p:g})\langle P\rangle Q$, then:

- $(\boldsymbol{\nu} r : g')O = (\boldsymbol{\nu}\widetilde{p:g})\langle P\rangle(\boldsymbol{\nu} r : g')Q$, if $r \notin \mathrm{fn}(P)$, and $(\boldsymbol{\nu} r : g')O = (\boldsymbol{\nu} r : g', \widetilde{p:g})\langle P\rangle Q$ otherwise.
- $O \mid R = (\boldsymbol{\nu}\widetilde{p:g})\langle P\rangle(Q \mid R)$.

– if O is the concretion $(\boldsymbol{\nu}\widetilde{p:g})\langle M\rangle P$, then:
- $(\boldsymbol{\nu} r : g')O$ is $(\boldsymbol{\nu}\widetilde{p:g})\langle M\rangle((\boldsymbol{\nu} r : g')P)$, if $r \notin \mathrm{fn}(M)$, and $(\boldsymbol{\nu} r : g', \widetilde{p:g})\langle M\rangle Q$ otherwise.
- $O \mid R = (\boldsymbol{\nu}\widetilde{p:g})\langle M\rangle(P \mid R)$.

Moreover we assume that ****

Labels	α	$::= \tau \mid \mathsf{in}\ n \mid \mathsf{out}\ n \mid \mathsf{to}\ n \mid [\mathsf{in}\ n] \mid [\mathsf{out}\ n] \mid [\mathsf{to}\ n] \mid \overline{\mathsf{in}\ n} \mid \overline{\mathsf{to}\ n} \mid \langle M\rangle \mid \langle -\rangle$
Concretions	K	$::= (\boldsymbol{\nu}\tilde{m})\langle P\rangle Q \mid (\boldsymbol{\nu}\tilde{m})\langle M\rangle P$
Outcomes	O	$::= P \mid K$

Table 1 Labels, concretions and outcomes

It is standard to check that labelled transitions agree with reductions when we restrict to well-typed processes in the current environment.

Theorem 2. *Let $\Xi \vdash P : g$ for some g.*

(CAP-IN-OUT)
$$\frac{M \in \{\mathsf{in}\ n, \mathsf{out}\ n\}}{M.P \xrightarrow{M}_{\Xi,g} P}$$

(CAP-TO)
$$\frac{}{\mathsf{to}\ n\,.\,P \xrightarrow{\mathsf{to}\ n}_{\Xi,g} \langle P\rangle \mathsf{0}}$$

(PATH)
$$\frac{M_1.(M_2.P) \xrightarrow{\alpha}_{\Xi,g} O}{(M_1\,.\,M_2)\,.\,P \xrightarrow{\alpha}_{\Xi,g} O}$$

(IN-OUT)
$$\frac{P \xrightarrow{M}_{\Xi,g'} P' \quad M \in \{\mathsf{in}\ n, \mathsf{out}\ n\}}{m[P] \xrightarrow{[M]}_{\Xi,g} \langle m[P']\rangle \mathsf{0}}$$

(TO)
$$\frac{P \xrightarrow{\mathsf{to}\ n}_{\Xi,g'} (\boldsymbol{\nu}\widetilde{p:g})\langle P_1\rangle P_2}{m[P] \xrightarrow{[\mathsf{to}\ n]}_{\Xi,g} (\boldsymbol{\nu}\widetilde{p:g})\langle P_1\rangle m[P_2]}$$

(CO-IN)
$$\frac{\Xi = \Gamma, g' : G; \Delta, n : g \quad \& \quad g \in \mathscr{C}(G)}{n[P] \xrightarrow{\overline{\mathsf{in}\ n}}_{\Xi,g'} \langle P\rangle \mathsf{0}}$$

(CO-TO)
$$\frac{\Xi = \Gamma, g' : G; \Delta, n : g \quad \& \quad g \in \mathscr{E}(G)}{n[P] \xrightarrow{\overline{\mathsf{to}\ n}}_{\Xi,g'} \langle P\rangle \mathsf{0}}$$

(INPUT)
$$\frac{}{(x{:}W)P \xrightarrow{\langle M\rangle}_{\Xi,g} P\{x := M\}}$$

(OUTPUT)
$$\frac{}{\langle M\rangle.P \xrightarrow{\langle -\rangle}_{\Xi,g} \langle M\rangle P}$$

Table 2 Commitments: Visible transitions respecting Ξ

(τ-ENTER)

$$P \xrightarrow{[M]}_{\Xi,g} (\boldsymbol{\nu}\widetilde{p:g})\langle P_1\rangle P_2 \quad Q \xrightarrow{\overline{M}}_{\Xi,g} (\boldsymbol{\nu}\widetilde{q:g'})\langle Q_1\rangle Q_2 \quad M \in \{\mathsf{in}\ n, \mathsf{to}\ n\}$$

$$P \mid Q \xrightarrow{\tau}_{\Xi,g} (\boldsymbol{\nu}\widetilde{p:g}, \widetilde{q:g'})(n[Q_1 \mid P_1] \mid P_2 \mid Q_2)$$

(τ-EXIT)

$$\frac{P \xrightarrow{[\mathsf{out}\ n]}_{\Xi,g'} (\boldsymbol{\nu}\widetilde{p:g})\langle P_1\rangle P_2}{n[P] \xrightarrow{\tau}_{\Xi,g} (\boldsymbol{\nu}\widetilde{p:g})(P_1 \mid n[P_2])}$$

(τ-EXCHANGE)

$$\frac{P \xrightarrow{\langle M\rangle}_{\Xi,g} P_1 \quad Q \xrightarrow{\langle -\rangle}_{\Xi,g} (\boldsymbol{\nu}\,\widetilde{q:g'})\langle M\rangle Q_1}{P \mid Q \xrightarrow{\tau}_{\Xi,g} (\nu\,\widetilde{q:g'})(P_1 \mid Q_1)}$$

Table 3 Commitments: τ transitions

(PAR)

$$\frac{P \xrightarrow{\alpha}_{\Xi,g} O}{P \mid Q \xrightarrow{\alpha}_{\Xi,g} O \mid Q}$$

(RES)

$$\frac{P \xrightarrow{\alpha}_{\Delta,\boldsymbol{\nu}g;\Gamma,g} O \quad n \notin \mathrm{fn}(\alpha) =}{(\boldsymbol{\nu}n)P \xrightarrow{\alpha}_{\Delta;\Gamma,g} (\boldsymbol{\nu}n)O}$$

(τ-AMB)

$$\frac{P \xrightarrow{\tau}_{\Xi,g'} gP'}{n[P] \xrightarrow{\tau}_{\Xi,g} n[P']}$$

(REPL)

$$\frac{\pi\,.\,P \xrightarrow{\alpha}_{\Xi,g} O}{!M\,.\,P \xrightarrow{\alpha}_{\Xi,g} !\pi\,.\,P \mid O}$$

Table 4 Commitments: Structural transitions

1. *If* $P \xrightarrow{\tau}_{\Xi,g} Q$ *then* $P \rightarrow Q$.
2. *If* $P \rightarrow Q$ *then* $P \xrightarrow{\tau}_{\Xi,g} Q'$ *for some* $Q' \equiv Q$.

Comparing the definition of observability with rules (CO-IN) and (CO-TO) one can easily argue that a name is observable iff at least one of the two actions can be performed.

Proposition 1. *$P \downarrow_n^{\Xi,g}$ if and only if $P \xrightarrow{\alpha}_{\Xi,g} (\boldsymbol{\nu}\widetilde{m:g})\langle Q\rangle R$ where $\alpha \in \{\overline{\mathsf{in}}\ n, \overline{\mathsf{to}}\ n\}$ for some Q, R.*

In order to get a labelled transition comparable with our barbed congruence we follow [14], [4] in defining higher-order transitions allowing

(HO OUTPUT)

$$\frac{P \xrightarrow{\langle - \rangle}_{\Xi,g} (\boldsymbol{\nu} \widetilde{p:g})\langle M \rangle P' \quad \Xi = \Gamma, g{:}G;\, \Delta \quad \Gamma, g{:}G;\, \Delta, \widetilde{p:g}, x{:}T(G) \vdash Q : g}{P \xrightarrow{\langle - \rangle Q}_{\Xi,g} \widetilde{p:g}(P' \mid Q\{x := M\})}$$

(HO IN/TO/CO-IN-TO)

$$\frac{P \xrightarrow{M}_{\Xi,g} (\boldsymbol{\nu} \widetilde{p:g})\langle P_1 \rangle P_2 \quad M \in \{[\mathsf{in}\ n], [\mathsf{to}\ n], n[\,]\} \quad \Xi = \Gamma;\, \Delta, n{:}g' \quad \Gamma;\, \Delta, n{:}g', \widetilde{p:g} \vdash Q : g'}{P \xrightarrow{MQ}_{\Xi,g} (\boldsymbol{\nu} \tilde{p})(n[P_1 \mid Q] \mid P_2)}$$

(HO OUT)

$$\frac{P \xrightarrow{[\mathsf{out}\ n]}_{\Xi,g} (\boldsymbol{\nu} \tilde{p})\langle P_1 \rangle P_2 \quad \Xi = \Gamma;\, \Delta, n{:}g' \quad \Gamma;\, \Delta, n{:}g', \widetilde{p:g} \vdash Q : g'}{P \xrightarrow{[\mathsf{out}\ n]Q}_{\Xi,g} (\boldsymbol{\nu} \tilde{p})(P_1 \mid n[P_2 \mid Q])}$$

Table 5 Commitments: Higher-Order transitions

us to get rid of the transitions whose outcome is a concretion rather than a process. In these transitions we allow labels of the shape αQ and then transitions of the form

$$P \xrightarrow{\alpha Q}_{\Xi,g} P'$$

where Q, which is assumed to be well typed from Ξ with the proper group type, is the minimal process which must be in the enclosing context in order to fire the action α. We denote by Λ be the set of all labels and for $\lambda \in \Lambda$ and convene that:

i) $\overset{\lambda}{\Longrightarrow}_{\Xi}$ denotes $\xrightarrow{\tau}_{\Xi,g}{}^{*} \xrightarrow{\lambda}_{\Xi,g} \xrightarrow{\tau}_{\Xi,g}{}^{*}$;

ii) $\overset{\hat{\lambda}}{\Longrightarrow}_{\Xi}$ denotes $\xrightarrow{\tau}_{\Xi,g}{}^{*}$ (also noted $\Longrightarrow_{\Xi,g}$) if $\lambda = \tau$ and $\overset{\lambda}{\Longrightarrow}_{\Xi}$ otherwise.

Having only labelled transitions from processes to processes the standard definition of labelled bisimulation adapt smoothly to our calculus.

Definition 3 (Labelled bisimilarity with respect to environments). *(i) A Ξ, g-symmetric relation over closed processes is a Ξ, g-*labelled bisimulation *if $P \mathcal{R}^{\Xi,g} Q$ and $P \xrightarrow{\lambda}_{\Xi} P'$ imply that there exists Q' such that $Q \overset{\hat{\lambda}}{\Longrightarrow}_{\Xi} Q'$ and $P' \mathcal{R}^{\Xi,g} Q'$.*
*(ii) Two closed processes P and Q are Ξ, g-*labelled bisimilar*, written $P \approx_{\mathrm{c}}^{\Xi,g} Q$, if $P \mathcal{R}^{\Xi,g} Q$ for some Ξ, g-labelled bisimulation.*

Definition 4 (Full bisimilarity with respect to environments). *Two processes P and Q are Ξ, g-*full bisimilar*, $P \approx^{\Xi,g} Q$, if $P\sigma \approx_c^{\Xi,g} Q\sigma$ for every closing substitution σ.*

Full bisimilarity is a congruence: the proof follows the scheme of [14], [4]. Moreover full bisimilarity is sound but not complete w.r.t. the reduction barbed congruence.

Theorem 3 (Soundness of full bisimilarity). *If $P \approx^{\Xi,g} Q$ then $P \cong^{\Xi,g} Q$.*

SONO QUI

Proof. It is enough to show that $\approx_c^{\Xi}$ is reduction closed and barb preserving, up to $\equiv$. Assume that $P \rightarrow P'$. By Theorem 2 $P \stackrel{\tau}{\Longrightarrow}_{\Xi} \equiv P'$. Since $P \approx_c^{\Xi} Q$, there exits Q' such that $Q \rightarrow^* Q'$ and $P' \equiv \approx_c^{\Xi} \equiv Q'$. Thus also $P' \equiv \approx_c^{\Xi} \equiv Q'$. Now assume $P \approx_c^{\Xi} Q$. If $P \downarrow_n$ then, by Proposition 1, and rule (HO CO-IN), $P \stackrel{\alpha R}{\longrightarrow}_{\Xi,g} S$ where $\alpha \in \{\overline{\mathsf{in}\ n}, \overline{\mathsf{to}\ n}\}$, for some R, S. Since $P \approx_c^{\Xi} Q$ we know that $Q \stackrel{\alpha R}{\longrightarrow}_{\Xi,g} S'$ for some $S' \approx_c^{\Xi} S$, from which $Q \Downarrow_n$, as desired.

The failure of completeness is due to the fact that contexts are insensible to repeated entering and exiting. This phenomena is called *stuttering* in [11] and it is typical of movements which do not consume co-capabilities as happen in our calculus. Let us recall the example of [11]. If $P + Q$ denotes the non deterministic choice (which can be encoded for example as $(\boldsymbol{\nu} a)(a[(x : W)\mathsf{out}\ n\,.\,\mathsf{to}\ n\,.\,P \mid (x : W)\mathsf{out}\ n\,.\,\mathsf{to}\ n\,.\,Q \mid \langle m \rangle])$, where n is the actual ambient and x is fresh), then no context can distinguish between the processes:

$$\mathsf{in}\ a\,.\,\mathsf{out}\ a\,.\,\mathsf{in}\ a\,.\,\mathsf{0}$$
$$\mathsf{in}\ a\,.\,\mathsf{out}\ a\,.\,\mathsf{in}\ a\,.\,\mathsf{0} + \mathsf{in}\ a\,.\,\mathsf{0}$$

We end this section by discussing some properties of processes and ambients in a fixed environment which allow us to prove interesting algebraic laws. These laws are also useful to discuss the examples of section **??**. di qui in poi manca la prosa ed e' un tentativo!

Definition 5. *1. A process P is an Ξ-*motor *iff $P \stackrel{\lambda}{\Longrightarrow}_{\Xi} \stackrel{\mathsf{in}\ n}{\longrightarrow}_{\Xi} O$ or $P \stackrel{\lambda}{\Longrightarrow}_{\Xi} \stackrel{\mathsf{out}\ n}{\longrightarrow}_{\Xi} O$ for some λ, n, O;*

*2. A process P is an Ξ-*sender *iff $P \stackrel{\lambda}{\Longrightarrow}_{\Xi} \stackrel{\mathsf{to}\ n}{\longrightarrow}_{\Xi} O$ for some λ, n, O;*

*3. A process P is an Ξ-*communicator *iff $P \stackrel{\lambda}{\Longrightarrow}_{\Xi} \stackrel{\langle M \rangle}{\longrightarrow}_{\Xi} O$ or $P \stackrel{\lambda}{\Longrightarrow}_{\Xi} \stackrel{\langle - \rangle}{\longrightarrow}_{\Xi} O$ for some λ, M, O.*

Lemma 2. *Let $\Xi = \Gamma, g\colon G; \Delta$.*

1. *If* $\Xi \vdash P\colon g$ *and* $\mathscr{C}(G) = \varnothing$ *then* P *is not an* Ξ*-motor;*
2. *If* $\Xi \vdash P\colon g$ *and* $\mathscr{E}(G) = \varnothing$ *then* P *is not an* Ξ*-sender;*
3. *If* $\Xi \vdash P\colon g$ *and* $T(G) = \mathsf{shh}$ *then* P *is not an* Ξ*-communicator.*

Definition 6. 1. *An ambient* n *is* Ξ-traversable *iff* $n[P] \xrightarrow{\overline{\mathsf{in}\ n}}_{\Xi} O$ *for some* P, O*;*
2. *An ambient* n *is* Ξ-mobile *iff* $n[P] \xrightarrow{\mathsf{in}\ m}_{\Xi} O$ *or* $n[P] \xrightarrow{\mathsf{out}\ m}_{\Xi} O$ *for some* P, m, O*;*
3. *An ambient* n *is an* Ξ-receiver *iff* $n[P] \xrightarrow{\overline{\mathsf{to}\ n}}_{\Xi} O$ *for some* P, O*.*

Lemma 3. *Let* $\Xi = \Gamma; \Delta, n\colon g$*.*

1. *If* $g \notin \bigcup_{g'\colon G \in \Gamma} \mathscr{C}(G)$ *then* n *is not* Ξ*-traversable;*
2. *If* $\Gamma = \Gamma', g\colon G$ *and* $\mathscr{C}(G) = \varnothing$ *then* n *is not* Ξ*-mobile;*
3. *If* $g \notin \bigcup_{g'\colon G \in \Gamma} \mathscr{E}(G)$ *then* n *is not an* Ξ*-receiver.*

Theorem 4. 1. *If* n *is not an* Ξ*-receiver,* Q *is not an* Ξ*-motor and* m *is not* Ξ*-mobile then:*

$$(\nu m\colon g)(n[\mathsf{in}\ m\,.\,P \mid Q] \mid m[R]) \cong^{\Xi} (\nu m\colon g)(m[n[P \mid Q] \mid R])$$

2. *If* n *is not an* Ξ*-receiver and* Q *is not an* Ξ*-motor then:*

$$m[n[\mathsf{out}\ m\,.\,P \mid Q] \mid R] \cong^{\Xi} n[P \mid Q] \mid m[R]$$

3. *If* m, n *are not* Ξ*-mobile then:*

$$(\nu m\colon g)(n[\mathsf{to}\ m\,.\,P \mid Q] \mid m[R]) \cong^{\Xi} (\nu m\colon g)(n[Q] \mid m[P \mid R])$$

4. *If* n *is not an* Ξ*-receiver and* R *is not an* Ξ*-communicator then:*

$$n[(x\colon W)\,.\,P \mid \langle M\rangle\,.\,Q \mid R] \cong^{\Xi} n[P\{x := M\} \mid Q \mid R]$$

5. *If* n *is not an* Ξ*-receiver and it is not* Ξ*-traversable then:*

$$n[P] \cong^{\Xi} (\nu n\colon g)(n[P])$$

6. *If* Q *is not an* Ξ*-sender and it is not* Ξ*-motor then:*

$$(\nu n\colon g)(n[P] \mid m[Q]) \cong^{\Xi} (\nu n\colon g)(n[P]) \mid (\nu n\colon g)(m[Q])$$

Proof. (1) Use the bisimilarity $\{(\nu m\colon g)(n[\mathsf{in}\ m\,.\,P \mid Q] \mid m[R]) \mid S, (\nu m\colon g)(m[n[P \mid Q] \mid R]) \mid S\} \cup \{I\}$ where I is the identity. E' necessaria o no la S?

5 Examples

In this section we test the expressiveness of our calculus, by showing first how to model some common protocols considered in the literature (such as a firewall, and a defence against Trojan-horse attacks), and then how to encode other process mobility primitives (up and down) and other well-known calculi for modelling concurrent (π-calculus) and distributed (Dπ) systems.

5.1 Protocols

Firewall

A system protected by a firewall can be viewed as an ambient *fw* that supplies the incoming $agent$ with a "password" (represented by its very name) which allows the process P to enter it. We define:

$$\text{AG} = \mathit{ag}[(x : g_{\mathit{ag}} \to g_{\mathit{fw}})\langle M\rangle x.P].$$
$$\text{FW} = (\nu\, \mathit{fw} : g_{\mathit{fw}})\mathit{fw}[\mathsf{to}\ \mathit{ag}.\langle \mathsf{in}\ \mathit{fw}\rangle \mid (y : W)Q]$$

where W is a suitable message type, M has type W, x does not occur in P and y does not occur in Q. The only role of the communication of the message M is that of blocking process Q until process P has entered the firewall.

The system agent-plus-firewall is represented by the top-level process $\text{AG} \mid \text{FW}$. It can be typed with the group g_0 by assuming $agent : g_{\mathit{ag}}$ and $fw : g_{\mathit{fw}}$, where the groups are typed as follows:

$$g_{\mathit{ag}} : \mathsf{gr}(\{g_0\}, \varnothing, \{g_{\mathit{fw}}\}, g_{\mathit{ag}} \to g_{\mathit{fw}})$$
$$g_{\mathit{fw}} : \mathsf{gr}(\{g_0\}, \varnothing, \{g_{\mathit{ag}}\}, \mathsf{shh})$$

Using Theorem 4 one can show that

$$(\nu\, \mathit{ag} : g_{\mathit{ag}})\text{AG} \mid \text{FW} \cong^{\Xi} (\nu\, \mathit{fw} : g_{\mathit{fw}})\mathit{fw}[P \mid Q]$$

where Ξ contains the group types above together with the premises necessary to type P, Q and M.

This example shows how the to capability allows to cross firewalls in a simple way.

The Trojan Horse Attack

In this example, we show how our type system can detect a Trojan horse attack. Ulysses is naturally encoded as a mobile ambient-process that enters an ambient *horse* containing an in $troy$ action, and then goes out of it into

Troy to destroy Priam's palace. The initial situation is represented by the term:

$$ulysses[\mathsf{in}\ horse.\mathsf{out}\ horse.\mathsf{to}\ \textit{palace}.\mathrm{DESTROY}] \mid horse[\mathsf{in}\ troy] \mid troy[\textit{palace}[\mathrm{P}]]$$

If ambients $ulysses, horse, \ldots$ belong respectively to groups $g_{ulysses}, g_{horse}, \ldots$, the whole $myth$ic process can be typed by a group g_{myth} w.r.t. an environment which contains the following assumptions:

$g_{ulysses} : \mathsf{gr}(\{g_{myth}, g_{troy}, g_{horse}\}, \{g_{horse}\}, \{g_{palace}\}, \mathsf{shh})$
$g_{horse} : \mathsf{gr}(\{g_{myth}, g_{troy}\}, \{g_{troy}\}, \varnothing, \mathsf{shh})$
$g_{troy} : \mathsf{gr}(\{g_{myth}\}, \varnothing, \varnothing, \mathsf{shh})$
$g_{palace} : \mathsf{gr}(\{g_{troy}\}, \varnothing, \varnothing, \mathsf{shh})$

from which it is clear that ambients of group $g_{ulysses}$ must have permission to stay within ambients of group g_{troy} and to send processes to ambients of group g_{palace} in order to make the myth well-typed.

5.2 Encoding Process Calculi

Encoding other process mobility actions

The main choice in designing a calculus with mobile (lightweight) processes is the one of the mobility primitives for them. We have chosen to introduce, for the moment, only one primitive, to,[3] since already present, though in a context of immobile locations, in a well established concurrent calculus such as Dπ [9]. Also, this primitive might be argued to naturally model the elementary instruction by which an agent moves from one location to another at the same level.

A natural alternative, or a natural extension, would be a thread mobility analogous to that for ambients, i.e., capabilities to go one step up or down the tree hierarchy, by exiting or entering an ambient.

Consider for example the two primitives up and down, with the following reduction rules:

$$\text{(R-down)} \qquad \mathsf{down}\ m\,.\,P \mid m[R] \ \rightarrow\ m[\,P \mid R\,]$$
$$\text{(R-up)} \qquad m[\,p[\,\mathsf{up}\ m\,.\,P \mid Q\,] \mid R\,] \ \rightarrow\ m[\,P \mid p[Q\,] \mid R]$$

where also the up takes as argument the destination ambient, instead of the source (as the analogous out does). It is interesting that both can be encoded in the to-language, though only as actions in process prefixes and not as capabilities transmissible in a message.

[3] the pun was initially unintended.

Such encodings are carried out by means of auxiliary ambients, with an interplay of ambient and process mobility. The action down can be defined as:

$$[\![\mathsf{down}\ m\,.\,P]\!] = (\nu g_n : G_n)(\nu n : g_n)n[\mathsf{to}\ m\,.\,P]$$

where $G_n = \mathsf{gr}(\{g\}, \varnothing, \{g_m\}, \mathsf{shh})$, g is the type of the whole process and g_m is the group of m. This encoding is more permissive, from the typing point of view, than the natural typing rule associated to down, which is:

$$\frac{\Gamma \vdash g : G \quad \Gamma \vdash m : g_m \quad \Gamma \vdash P : g_m \quad g_m \in \mathscr{E}(G)}{\Gamma \vdash \mathsf{down}\ m.P : g} \qquad (\mathsf{down}\ m)$$

In fact, the term $n[\mathsf{to}\ m\,.\,P]$ may be given a type g by a derived rule with the same premisses as the rule down $m.P$, but without the condition that $g_m \in \mathscr{E}(G)$. Of course, such assumption on the group type of g may always be made.

In the case of up, we can only define the encoding of an action up^p where p is the name of the ambient wherefrom the process comes, needed to simulate the action with the basic to primitive:

$$[\![\mathsf{up}\ ^p m\,.\,P]\!] = (\nu g_n : G_n)(\nu n : g_n)n[\mathsf{out}\ p\,.\,\mathsf{out}\ m\,.\,\mathsf{to}\ m\,.\,P]$$

where $g_p\colon G_p$ and $g_m\colon G_m$ are respectively the groups of p and m, obviously $g_m \in \mathscr{S}(G_p)$, and $G_n = \mathsf{gr}(\{g_p\} \cup \mathscr{S}(G_p) \cup \mathscr{S}(G_m), \{g_p, g_m\}, \{g_m\}, \mathsf{shh})$. This definition correctly simulates the reduction rule of the up action, as the following reduction sequence shows:

$$\begin{array}{ll} & m[p[(\nu g_n : G_n)(\nu n : g_n)n[\mathsf{out}\ p\,.\,\mathsf{out}\ m\,.\,\mathsf{to}\ m\,.\,P] \mid Q] \mid R] \\ \rightarrow & (\nu g_n : G_n)(\nu n : g_n)(n[\mathsf{to}\ m\,.\,P] \mid m[p[Q] \mid R]) \\ \rightarrow & (\nu g_n : G_n)(\nu n : g_n)n[\,] \mid m[P \mid p[Q] \mid R] \equiv m[P \mid p[Q] \mid R] \end{array}$$

Observe that if there is another ambient named m in parallel with the one which is the subject of the definition, the to m action may take the process into the wrong m, i.e., the following reduction is possible:

$$\begin{array}{ll} & m[p[(\nu g_n : G_n)(\nu n : g_n)n[\mathsf{out}\ p\,.\,\mathsf{out}\ m\,.\,\mathsf{to}\ m\,.\,P] \mid Q] \mid R] \mid m[R'] \\ \rightarrow^* & m[p[Q] \mid R] \mid m[P \mid R'].\end{array}$$

Thus the encoding may nondeterministically allow, besides the effect of the original up action, also a completely different evolution (one might say, following the terminology of [11], that it suffers from a grave interference).

The encoding is consistent w.r.t. typing, in the same sense as the down action considered above.

Encoding the π-calculus

A standard expressiveness test for the Ambient Calculus and its variants is the encoding of communication on named channels via local anonymous communication within ambients. We consider a core fragment of the typed monadic synchronous π-calculus, given by the following grammar (we use letters a–d for channel names and x–z for variables):

$$P \ ::= \ c(x:T).P \ \mid \ c\langle a\rangle.P \ \mid \ (\nu c:T)P \ \mid \ P \,|\, Q \ \mid \ !P \ \mid \ \mathsf{0}$$

T ranges over a type linear hierarchy:

$$T \ ::= \ Ch() \ \mid \ Ch(T)$$

The reduction relation, which will be denoted by $\rightarrow_\pi$, is defined by one basic rule:

$$c(x:T).P \,|\, c\langle a\rangle.Q \rightarrow_\pi P\{x := a\} \,|\, Q$$

and by structural reduction rules similar to those of Figure 3 for our calculus, except (R-amb) and (R-ν-group).

The type system, defined by the typing rules of Figure 6, derives judgements of the form $\Lambda \vdash P$, where Λ is a set of judgements of the form $c : Ch(T)$. The informal meaning of $\Lambda \vdash P$ is that P is a well-typed process w.r.t. the environment Λ, i.e., communication is well-typed in P w.r.t. assumptions Λ.

$$\frac{\Lambda \vdash c : Ch(T) \quad \Lambda, x:T \vdash P}{\Lambda \vdash c(x:T)P} \qquad \frac{\Lambda \vdash c : Ch(T) \quad \Lambda \vdash a : T \quad \Lambda \vdash P}{\Lambda \vdash c\langle a\rangle P}$$

$$\frac{\Lambda, c:T \vdash P}{\Lambda \vdash (\nu c:T)P} \qquad \frac{\Lambda \vdash P \quad \Lambda \vdash Q}{\Lambda \vdash P \,|\, Q} \qquad \frac{\Lambda \vdash P}{\Lambda \vdash !P} \qquad \frac{}{\Lambda \vdash \mathsf{0}}$$

Fig. 6 Type system for π-calculus

The basic idea of the encoding consists, as usual, in representing each channel as an ambient: processes prefixed with a communication action on a channel c are encoded as mobile processes that first go (in)to the ambient $c[\,]$ where they communicate, and then go back to where they belong. Since, however, a to action can only move a process between sibling ambients, the introduction is needed, in parallel with the channel-ambients, of an ambient p containing the encoding proper $[\![P]\!]$ of the top-level π-calculus term P.

The infinite sequence of π-calculus types $Ch(), Ch(Ch()), \ldots, Ch^n(), \ldots$ is encoded as an infinite sequence of group names $g_0, g_1, \ldots, g_{n-1}, \ldots$

along with the sequence of their respective group types $G_0, G_1, \ldots, G_{n-1}, \ldots$:

$$\begin{array}{lll} [\![Ch()]\!] = g_0 & \text{with} & g_0 : G_0 = \mathsf{gr}(\{g, g_p\}, \{g_p\}, \{g_p\}, \mathsf{shh}) \\ [\![Ch(T)]\!] = g_{j+1} \text{ if } [\![T]\!] = g_j & \text{with} & g_{j+1} : G_{j+1} = \mathsf{gr}(\{g, g_p\}, \{g_p\}, \{g_p\}, g_j) \end{array}$$

where g_p is the group of the above-mentioned ambient p, while g is the group of the top-level M^3 ambient-processes, i.e., both the process $p[[\![P]\!]]$ and the channel-processes $a[\ldots]$.

Let P be a π-process, the set $\{c_1, \ldots, c_h\}$ and the integer k be such that:

- $\{c_1, \ldots, c_h\}$ contains the set $\mathsf{fn}(P)$ of the free channel names occurring in P;
- k is greater or equal to the maximum nesting of $Ch()$ in types occurring in P.

The global encoding of P, denoted by $\mathcal{C}([\![P]\!], c_1, \ldots, c_h, k)$, is then the process of group g given by:

$$\mathcal{C}([\![P]\!], c_1, \ldots, c_h, k) = p[[\![P]\!]] \,|\, c_1[\,] \,|\, \ldots \,|\, c_h[\,]$$

where $[\![P]\!]$ is defined in Figure 7. Of course, this requires that the (finite) set of free channel names occurring in the term be known in advance. We can, however, always assume to be working on closed terms.

$$\begin{array}{ll} [\![c(x:T)P]\!] & = \mathsf{to}\ c.(x : [\![T]\!])\mathsf{to}\ p.[\![P]\!] \\ [\![c\langle a\rangle P]\!] & = \mathsf{to}\ c.\langle a\rangle\mathsf{to}\ p.[\![P]\!] \\ [\![(\nu c : T)P]\!] & = (\nu c : [\![T]\!])(c[\mathsf{out}\ p] \,|\, [\![P]\!]) \\ [\![P \,|\, Q]\!] & = [\![P]\!] \,|\, [\![Q]\!] \\ [\![!P]\!] & = ![\![P]\!] \\ [\![0]\!] & = 0 \end{array}$$

Fig. 7 Encoding of π-calculus

If one defines the translation of a type environment $[\![\Lambda]\!]$ as the set of assumptions $\{c : [\![T]\!] \mid c : T \in \Lambda\}$, the Theorem 5 below states that the translation respects types. Also, the translation is correct in the sense expressed by Theorem 6.

In the following, Π denotes the group environment

$$\Pi = \{g_0 : G_0, \ldots, g_k : G_k, g_p : G_p, p : g_p\}$$

where $G_p = \mathsf{gr}(\{g\}, \varnothing, \{g_i \mid 0 \leq i \leq k\}, \mathsf{shh})$.

Theorem 5. *Let $\Lambda \vdash P$, $\{c_1, \ldots, c_h\} \supseteq \mathsf{fn}(P)$, and k is greater or equal to the maximum nesting of $Ch()$ in types occurring in P. Then, for any group type G: $\Pi, g : G;\ [\![\Lambda]\!] \vdash \mathcal{C}([\![P]\!], c_1, \ldots, c_h, k) : g$.*

Proof. By induction on the structure of the process one can show that, for every well-typed π-process P, the judgement $\Phi \vdash [\![P]\!] : g_p$ is always derivable by taking as Φ the following environment:

$$\Phi = \Pi, g : G\,;\, [\![\Lambda]\!], c_1 : [\![T_1]\!], \ldots, c_h : [\![T_h]\!]$$

As an example, let us consider $P \equiv c(x : T)P'$. We have the following typing derivation for $[\![P]\!]$ (let $c : g_c$ be the type assumption for the name c in Φ):

$$\dfrac{\dfrac{\dfrac{\Phi, x : [\![T]\!] \vdash \mathsf{to}\ p : g_p \to g_c \quad \Phi, x : [\![T]\!] \vdash [\![P']\!] : g_p}{\Phi, x : [\![T]\!] \vdash \mathsf{to}\ p.[\![P']\!] : g_c}}{\Phi \vdash (x : [\![T]\!])\mathsf{to}\ p.[\![P']\!] : g_c} \qquad \Phi \vdash \mathsf{to}\ c : g_c \to g_p}{\Phi \vdash \mathsf{to}\ c.(x : [\![T]\!])\mathsf{to}\ p.[\![P']\!] : g_p}$$

The statement of the theorem easily follows, considering the type judgements $\Phi \vdash c_i[\,] : g$, $\Phi \vdash p[[\![P]\!]] : g$ and then using type derivation rules (PAR), (AMBRES), and (GRPRES).

Theorem 6. *Let P be a term of the π-calculus such that $\{c_1, \ldots, c_h\} \supseteq \mathsf{fn}(P)$, and k is greater or equal to the maximum nesting of $Ch()$ in types occurring in P.*
(i) If $P \to_\pi Q$ then $\mathcal{C}([\![P]\!], c_1, \ldots, c_h, k) \to^* \mathcal{C}([\![Q]\!], c_1, \ldots, c_h, k)$.
(ii) If $\mathcal{C}([\![P]\!], c_1, \ldots, c_h, k) \to^* \mathcal{C}(Q, c_1, \ldots, c_h, k)$, and $Q = [\![R]\!]$ for some π-calculus process R, then $P \to_\pi^* R$.

Proof. (i) Let us consider, as interesting case, the one where P is of the form $c(x : T).P' \mid c\langle a\rangle.P'' \mid R$, which can reduce to $Q \equiv P'\{x := a\} \mid P'' \mid R$. By definition of $[\![\cdot]\!]$, we have that $[\![P]\!]$ is $\mathsf{to}\ c.(x : [\![T]\!]).\mathsf{to}\ p.[\![P']\!] \mid \mathsf{to}\ c.\langle a\rangle.\mathsf{to}\ p.[\![P'']\!] \mid [\![R]\!]$. Of course, c is a free variable of P, and in the term $\mathcal{C}([\![P]\!], c_1, \ldots, c_h, k)$ there is by hypothesis an ambient $c[\,]$ which runs in parallel with the process $[\![P]\!]$. We have therefore the following reduction (we only show the relevant subprocesses):

$$\begin{aligned} & p[\mathsf{to}\ c.(x : [\![T]\!]).\mathsf{to}\ p.[\![P']\!] \mid \mathsf{to}\ c.\langle a\rangle.\mathsf{to}\ p.[\![P'']\!] \mid c[\,] \\ & \to^* p[\,] \mid c[(x : [\![T]\!]).\mathsf{to}\ p.[\![P']\!] \mid \langle a\rangle.\mathsf{to}\ p.[\![P'']\!]] \\ & \to\ \ p[\,] \mid c[\mathsf{to}\ p.[\![P']\!]\{x := a\} \mid \mathsf{to}\ p.[\![P'']\!]] \\ & \to^* p[[\![P']\!]\{x := a\} \mid [\![P'']\!]] \mid c[\,] \end{aligned}$$

whereas $[\![Q]\!]$ is $[\![P'\{x := a\}]\!] \mid [\![P'']\!] \mid [\![R]\!]$.

An easy induction on the definition of the translation concludes the proof by showing that $[\![P\{x := a\}]\!] = [\![P]\!]\{x := a\}$.

(ii) By the definition of the encoding, $\mathcal{C}([\![P]\!], c_1, \ldots, c_h, k)$ consists of a parallel composition of processes running inside the ambient $p[\,]$. In this

case the only possible move the system can perform is a to action. Consider a subprocess of the form $\mathsf{to}\, c.(x : [\![T]\!]).\mathsf{to}\, p.[\![Q']\!]$: this is the translation of the π-calculus process $Q \equiv (x : T).Q'$, hence $[\![P]\!] \equiv [\![Q]\!] \mid [\![U]\!]$ for some process U. The $\mathsf{to}\, c$ action can always be performed, if $c = c_i$, for some i, or if we are in the scope of a restriction of the c name. Once the $\mathsf{to}\, c$ action is performed, the process will reduce to a process in the form $\mathcal{C}([\![R]\!], c_1, \ldots, c_h, k)$ only if the channel c is cleared. To obtain this the other two actions in the prefix must be fired. This happens only if the input action is consumed inside the ambient c, hence if there is a process $[\![U]\!] \equiv \mathsf{to}\, c.\langle a\rangle.\mathsf{to}\, p.[\![Q'']\!] \mid [\![U']\!]$ that translates a π-calculus process $c\langle a\rangle.Q'' \mid U'$. To conclude the proof, it suffices to observe that after clearing channel c the process $\mathcal{C}([\![P]\!], c_1, \ldots, c_h, k)$ reduces to $\mathcal{C}([\![Q']\!]\{x := a\} \mid [\![Q'']\!] \mid [\![U']\!], c_1, \ldots, c_h, k)$. Lastly observe that
$P = (x : T).Q' \mid \langle a\rangle.Q'' \mid U'. \rightarrow_\pi Q'\{x := a\} \mid Q" \mid U'$ and $[\![Q'\{x := a\}]\!] = [\![Q']\!]\{x := a\}$.

Encoding the Dπ-calculus

We refer to the version of Dπ presented in [9], but assuming a much simpler type system. The basic syntactic categories are shown in Figure 8, where the set of values V consists of channel and location names.

Thread		*System*	
$p, q, r =$	stop	$P, Q, R =$	0
	$u?(X : T).p$		$P \mid Q$
	$u!\langle V\rangle.p$		$l[p]$
	go $l.p$		$(\nu_l u : \mathsf{ch}(T))P$
	$p \mid q$		
	$*P$		
	$(\nu u : \mathsf{ch}(T))p$		

Fig. 8 Syntax of Dπ-calculus

We assume in this presentation a very basic type system similar to that of π, aimed only at preventing communication errors. The main limitation is that channels with the same name in different locations must transmit values of the same type. We assume the following syntax for types:

$$\mathsf{loc} \mid \mathsf{ch}(T)$$

The reduction semantics of the Dπ-calculus, that we will denote with $\rightarrow_D$, has the following reduction rules:

$$l[u?(X : T).p] \mid l[u!\langle V\rangle.q] \rightarrow_D l[p\{X := V\}] \mid l[q]$$
$$l[\mathsf{go}\, l'.p] \rightarrow_D l'[p]$$

Operational semantics, as usual, is given together with rules that define structural equivalence. Besides standard ones, it is worth mentioning the following rules, peculiar to Dπ, which state that two locations with the same name are the same location (differently from our calculus), and that we can enlarge the scope of a channel restriction outside a location, provided that we keep information about the location:

$$l[p \mid q] \equiv l[p] \mid l[q]$$
$$l[(\nu u)p] \equiv (\nu_l u)l[p]$$

The typing rules we consider are similar to those of π-calculus (see Figure 9).

$$\frac{\Gamma \vdash u : \mathsf{ch}(T) \quad \Gamma, x : T \vdash P}{\Gamma \vdash u?(x:T).p} \qquad \frac{\Gamma \vdash u : \mathsf{ch}(T) \quad \Gamma \vdash V : T \quad \Gamma \vdash p}{\Gamma \vdash u!\langle V \rangle.p}$$

$$\frac{\Gamma, u : T \vdash p}{\Gamma \vdash (\nu u : T)p} \qquad \frac{\Gamma \vdash p \quad \Gamma \vdash q}{\Gamma \vdash p \mid q} \qquad \frac{\Gamma \vdash p}{\Gamma \vdash *p} \qquad \frac{}{\Gamma \vdash \mathsf{stop}}$$

$$\frac{\Gamma, u : T \vdash P}{\Gamma \vdash (\nu_l u : T)P} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \frac{\Gamma \vdash p}{\Gamma \vdash l[p]} \qquad \frac{}{\Gamma \vdash 0}$$

Fig. 9 Type system for Dπ-calculus

Like in the π-calculus encoding, we assume that we know in advance the (finite) set of locations (and of free channel names for each location, written $l^{u_1,\ldots,u_n}$) required to run the process. The set of free channel names used within a location is statically determinable, as shown by Dπ type systems. We also remark that in our version of Dπ channel names are absolute and not relative to locations. This implies that channels with the same name must communicate values of the same type also if they are in different locations.

The basic idea of the translation is the following. Each location l is represented by an ambient $l[\,]$ that contains as subambients the encodings of channels used in communications local to l. Moreover, each location l contains an internal ambient $self$ that contains a process $!\langle l \rangle$ offering as output the location name. This is used to allow processes to perform communications on channels located elsewhere with respect to their original site. In particular, a system P using at most locations l_1 (with at most local free channels $u_1^1, \ldots, u_1^{k_1}, k_1 \geq 0$), $\ldots$, l_n (with at most local free channels $u_1^n, \ldots, u_{k_n}^n, k_n \geq 0$) will be represented by:

$$[\![P]\!] \mid l_1[self[!\langle l_1 \rangle] \mid u_1^1[\,] \mid \ldots u_{k_1}^1[\,]] \mid \ldots \mid l_n[self[!\langle l_n \rangle] \mid u_1^n[\,] \mid \ldots u_{k_n}^n[\,]]$$

We will use for this term the notation $\mathcal{D}([\![M]\!], l_1^{u_1^1,\ldots,u_{k_1}^1}, \ldots, l_n^{u_1^n,\ldots,u_{k_n}^n}, k)$, where k is an integer greater or equal to the maximum nesting of $\mathsf{ch}()$ in types occurring in P.

Let now be g_{loc} the group of locations and g_D the group of the whole system. We define the following group and group types:

$$g_0 = g_{loc} : G_{loc} = \mathsf{gr}(\{g_D\}, \varnothing, \{g_{loc}\}, \mathsf{shh})$$
$$g_{i+1} : G_{i+1} = \mathsf{gr}(\{g_{loc}, g_D\}, \{g_{loc}\}, \{g_{aux}\}, \{g_i\})$$

Our types are encoded in the following way:

$$[\![\mathsf{loc}]\!] = g_{loc}$$
$$[\![\mathsf{ch}(T)]\!] = g_{i+1} \text{ where } [\![T]\!] = g_i$$

Assume the following groups and group types:

$$g_{aux} : G_{aux} = \mathsf{gr}(\{g_{loc}, g_D\}, \{g_{loc}\}, \{g_{self}, g_{loc}\}, \mathsf{shh})$$
$$g_{self} : G_{self} = \mathsf{gr}(\{g_{loc}\}, \varnothing, \{g_0, \ldots, g_k\}, g_{loc})$$

Moreover assume $l_i : g_{loc}$ (for all location names that occur in the current process), $self : g_{self}$, $u : g_{i+1}$ where $g_i = [\![T]\!]$ and $\mathsf{ch}(T)$ is the type of communications on channel u (for all channel names that occur in the current process).
The encoding of Dπ-calculus terms is given in Figure 10.

THREADS :

$[\![\mathsf{stop}]\!] = \mathsf{0}$
$[\![p \mid q]\!] = [\![p]\!] \mid [\![q]\!]$
$[\![\mathsf{go}\ l.\, p]\!] = \mathsf{to}\ l.[\![p]\!]$
$[\![u?(X{:}T).p]\!] = (\nu n{:}\, g_{aux})n[\mathsf{to}\ self.(x{:}\, g_{loc}).\mathsf{to}\ u.(X{:}\, [\![T]\!]).\mathsf{to}\ n.\mathsf{out}\ x.\mathsf{to}\ x.[\![p]\!]]$
$[\![u!\langle V\rangle.p]\!] = (\nu\, n{:}\, g_{aux})n[\mathsf{to}\ self.(x{:}\, g_{loc}).\mathsf{to}\ u.\langle [\![V]\!]\rangle.\mathsf{to}\ n.\mathsf{out}\ x.\mathsf{to}\ x.[\![p]\!]]$
$[\![(\nu e{:}\mathsf{ch}(T))p]\!] = (\nu e{:}\, g_{i+1})(e[\,] \mid [\![p]\!])$ where $[\![T]\!] = g_i$
$[\![(*\, p)]\!] = !\, [\![p]\!]$

SYSTEM :

$[\![\mathsf{0}]\!] = \mathsf{0}$
$[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$
$[\![l[p]]\!] = (\nu\, n{:}\, g_{aux})n[\mathsf{to}\ l.[\![p]\!]]$
$[\![(\nu_l e{:}\mathsf{ch}(T))P]\!] = (\nu e{:}\, g_{i+1})(e[\mathsf{in}\ l] \mid [\![P]\!])$ where $[\![T]\!] = g_i$

Fig. 10 Encoding of Dπ-calculus

The correctness of the translation, in the same sense as for the π-calculus, is ensured by analogous theorems, where $\rightarrow_D$ denotes reduction in the Dπ-

calculus, and the translation of a type environment $[\![\Gamma]\!]$ is the set of assumptions $\{u : [\![T]\!] \mid u : T \in \Gamma\}$. Moreover let Π_D be the environment:

$$g_{loc} : G_{loc}, g_1 : G_1, \ldots, g_k : G_k, g_{aux} : G_{aux}$$
$$g_{self} : G_{self}, l_1 : g_{loc}, \ldots, l_h : g_{loc}, self : g_{self}$$

Theorem 7. *Let P be a term of the Dπ-calculus, such that the set of locations is a subset of $\{l_1, \ldots, l_h\}$, and for each location l_i the set of free channel names located at l_i is a subset of $\{u_i^1, \ldots, u_i^{k_i}\}$ ($\overline{u}_i$ for short). Moreover let j be an integer greater or equal to the maximum nesting of $\mathsf{ch}()$ in types occurring in P. If $\Gamma \vdash P$ then, for any group type G we have: $\Pi_D, [\![\Gamma]\!], g_D : G \vdash \mathcal{D}([\![P]\!], l_1^{\overline{u}_1}, \ldots, l_h^{\overline{u}_h}, j) : g_D$.*

Theorem 8. *Let P be a term of the Dπ-calculus, such that the set of locations is a subset of $\{l_1, \ldots, l_h\}$, and for each location l_i the set of free channel names located at l_i is a subset of $\{u_i^1, \ldots, u_i^{k_i}\}$ ($\overline{u}_i$ for short). Moreover let j be an integer greater or equal to the maximum nesting of $\mathsf{ch}()$ in types occurring in P. We have:*
(i) If $P \rightarrow_D Q$ then $\mathcal{D}([\![P]\!], l_1^{\overline{u}_1}, \ldots, l_h^{\overline{u}_h}, j) \rightarrow^* \mathcal{D}([\![Q]\!], l_1^{\overline{u}_1}, \ldots, l_h^{\overline{u}_h}, j)$;
(ii) If $\mathcal{D}([\![P]\!], l_1^{\overline{u}_1}, \ldots, l_h^{\overline{u}_h}, j) \rightarrow^* \mathcal{D}(Q, l_1^{\overline{u}_1}, \ldots, l_h^{\overline{u}_h}, j)$, and $Q = [\![R]\!]$, for some Dπ process R, then $P \rightarrow_D^* Q$.

6 Type Inference

In this section, we present a type inference algorithm for our type assignment system. A type inference algorithm is a desirable feature in distributed systems, because it allows a type analysis to be performed even when only incomplete type assumptions about a process are available.

Given a raw process R, i.e., a well-formed process in which all type annotations have been erased, our type inference algorithm computes an environment Ξ and a well typed version P of R (obtained by assigning types to the the bound names occurring in R) such that $\Xi \vdash P : g$ for some group g. Moreover $\Xi,\ P,\ g$ are the "most general" ones in the sense that all other typings $\Xi',\ P',\ g'$ (that can be given to processes P' obtained from R by introducing type annotations) can be derived from $\Xi,\ P,\ g$ via substitution, weakening and a kind of subtyping.

Note that our notion of most general typing, although in some sense classical (see for instance [10]), is formally different from that of [19] where "principal" typing is defined via a partial order between typings independently from terms. The reason is that [19] deals with type inference systems, in which types are assigned to initially untyped terms. In this case no type commitment is written in terms and all typing information for a term is contained in the environment and in the final type. This makes sensible

to compare typing intended as pairs environment-type, without reference to the terms to which they are assigned. In our system instead types are also written inside terms, and this information cannot in general be retrieved from the environment and the final type. So to have the complete type information about a row term the term itself must be considered. Aside from these technicalities, however, the two definitions represent essentially the same concept.

Type Variables, Substitutions and Environment Operations

We first introduce some technical tools that will be useful in defining the inference procedure and its properties. The algorithm deals with type variables, substitutions, unifiers, and merging of type environments; moreover, a partial order relation on group types and type environments is needed for the derivation of a principal typing property.

Let a *raw* process R be a well-formed process in which all type annotations have been erased. In a raw process, in particular, group restrictions are missing, name restrictions are the form $(\nu n)R$, and inputs are of the form $(x)R$. If P is a process, its raw form is the raw process $|P|$ obtained from P by erasing all group restrictions and all type annotations.

In order to describe and prove properties of the inference algorithm, it is necessary to generalize the previously introduced syntactic categories for types by extending the syntax of communication types T by a set $\mathcal{V}_T$ of communication-type variables (denoted by t). Let's call *extended* types the resulting set of types.

On the other hand in the inference algorithm only environments with no occurrences of shh type (*hush-less* environments) are generated. Type shh will be introduced only in the final step of the algorithm. In the inference process the role of type shh will be played by communication type variables with only one occurrence in the inference judgements.

The new syntax of extended types, where most of the syntactic sugar has been eliminated, is shown in Figure 11.

As usual, in computing types and type environments the algorithm is driven by the syntax of the process; it has therefore to put together distinct environments whenever the process has more than one subprocess. A fresh group name is assigned to each name, but when different environments are put together groups must be equated. This is achieved by means of substitutions. In the context of this paper a substitution maps group names to group names, and communication-type variables to communication types (where types are extended with variables). Let $\mathbb{T}$ be the set of communication types, extended with type variables, as in Figure 11.

	$g, h, \ldots$	groups (ambient/process types)
	$\mathscr{S}, \mathscr{C}, \mathscr{E}, \ldots$	sets of groups; $\mathbb{G}$ is the universal set of groups
$\mathbb{W}$ ::=		message type
	g	ambient type
	$g_1 \to g_2$	capability type
$\mathbb{T}$ ::=		communication type
	t	communication-type variable
	$\mathbb{W}$	communication of messages of type W
G ::=	$\mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, T)$	group type

Fig. 11 Types for inference
[types-for-infer-fig]

Definition 7. **[sub]** *An* inference substitution *(substitution for short) is a finite mapping in* $(\mathbb{G} \to \mathbb{G}) \cup (\mathcal{V}_T \to \mathbb{T})$. *Let* φ_i *range over* $\mathbb{G} \cup \mathcal{V}_T$ *and* A_i *over* $\mathbb{T} \cup \mathbb{G}$. *A substitution* σ *can be represented as an expression* $[\varphi_1 := A_1, \ldots, \varphi_n := A_n]$, *where* $i \neq j$ *implies* $\varphi_i \neq \varphi_j$. *As usual, we assume* $\sigma(\varphi) = \begin{cases} A_j & \text{if } \varphi = \varphi_j \text{ for some } 1 \leq j \leq n \\ \varphi & \text{otherwise.} \end{cases}$

The application of a substitution to a variable environment (denoted by $\sigma(\Delta)$) is defined in the standard way. Well formed variable environments are closed under substitution. Applying substitutions to group environment we must take into account that in a statement $g : G$ group names occur also in the subjects (left hand side) of the statements. Taken a well-formed group environment Γ and an arbitrary substitution σ, $\sigma(\Gamma)$ could be non-well-formed. For instance if $g_1 : G_1, g_2 : G_2 \in \Gamma$ we could have $\sigma(g_1) = \sigma(g_2)$ but $\sigma(G_1) \neq \sigma(G_2)$. Consequently type inference is not, in general, closed under substitution. However it is easy to see that if $\Gamma;\ \Delta \vdash P : g$ and $\sigma(\Gamma)$, $\sigma(P)$ are well-formed then $\sigma(\Gamma);\ \sigma(\Delta) \vdash \sigma(P) : \sigma(g)$.

When two different environments are put together, it may be necessary to *unify* different message types and group names. This is impossible if they are an ambient type and a capability type: in such case we get a *failure*. Therefore, in the definitions below, operations on types and environments may yield an *undefined* result denoting failure. A failure is raised whenever none of the cases considered in definitions can be applied. Failure propagates, and an operation produces an *undefined* result whenever some steps in its evaluation produces *undefined*. To simplify notation, we assume failure propagation as understood and we avoid indicating it explicitly in the definitions.

Definition 8. **[unifier]** *We denote by* $\phi(\{(A_i, A'_i) | 1 \leq i \leq n\})$ *the* three-sorted most general unifier *of the set of equations* $\{A_i = A'_i \mid 1 \leq i \leq n\}$, *if it exists, which is a substitution according to Definition 7. We will simply call* $\phi(\{(A_i, A'_i) | 1 \leq i \leq n\})$ *the* unifier *of* $\{(A_i, A'_i) | 1 \leq i \leq n\}$.

Let $\Delta,\ \Delta'$ be two variable environments. We denote with (Δ, Δ') the set

$$\{(W, W')\ \mid\ x : W \in \Delta \text{ and } x : W' \in \Delta'\}.$$

Let's write for short $\phi(\Delta, \Delta')$ instead of $\phi((\Delta, \Delta'))$. Note that $\phi(\Delta, \Delta')$ is the most general substitution σ (if it exists) such that $\sigma(\Delta, \Delta')$ is well-formed.

As already observed, the application of a substitution σ to a group environment Γ gives an environment that is not, in general, well-formed, because σ can map two distinct group names of $Dom(\Gamma)$ to the same group name. In the sequel, we show how to recover a well-formed group environment after the application of substitutions. This task is performed in two steps:

1. by unifying the communication types in different type assumptions for the same group name (*completion-unification*, Definition 9);
2. by merging (by componentwise set union) group types having the same communication type (*compression*, Definition 10).

Completion-unification and compression are also useful to recover a well-formed environment when the algorithm needs to merge two environments, for example in typing a parallel composition.

We say that a group environment Γ is *consistent* if for all $g : G_1,\ g : G_2 \in \Gamma$ we have $T(G_1) = T(G_2)$. This condition does not imply $G_1 = G_2$, so consistent environments are in general not well-formed.

Definition 9. **[completion]** *Let Γ be an arbitrary group environment. The* completion-unifier *of Γ, denoted $\Sigma[\Gamma]$, is the substitution defined in the following way:*

1. *if Γ is consistent $\Sigma[\Gamma]$ is the empty substitution.*
2. *Otherwise let in $\Sigma[\sigma(\Gamma)] \circ \sigma$.*

The environment $\Sigma[\Gamma](\Gamma)$ is called the completion-unification *of Γ.*

The completion procedure always terminates, either with a failure or with a finite result consisting of a substitution $\Sigma[\Gamma]$. This is obvious, since the number of distinct group names decreases at each iteration and the number of group names in Γ is finite.

The environment $\Sigma[\Gamma](\Gamma)$ is consistent but it is not, in general, well-formed. The basic properties of completion are formalized by the following.

Lemma 4. **[completion prop]** *Let Γ be an arbitrary group environment. If $\Sigma[\Gamma]$ is defined then:*

(i) $\Sigma[\Gamma](\Gamma)$ is consistent;

(ii) for all σ such that $\sigma(\Gamma)$ is consistent, there is a substitution σ' such that $\sigma = \sigma' \circ \Sigma[\Gamma]$.

Otherwise there is no substitution σ such that $\sigma(\Gamma)$ is consistent.

Proof. Assume that to get $\Sigma[\Gamma]$ step 2. of Definition 9 is applied n times. The proof is by induction on n. If $n = 0$ the lemma is trivially true. Otherwise let

$$S = \{(T, T') \mid \exists g \,.\, g\colon \mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, T), g\colon \mathsf{gr}(\mathscr{S}', \mathscr{C}', \mathscr{E}', T') \in \Gamma \text{ such that } T \neq T'\}.$$

Note that since $\sigma'(\Gamma)$ is consistent we must have $\sigma'(T) = \sigma'(T')$ for all pairs $(T, T') \in S$. But now being σ the most general unifier of S we must have $\sigma' = \sigma'' \circ \sigma$ for some substitution σ''. So we have that $\sigma'(\Gamma) = \sigma''(\sigma(\Gamma))$. Now observe that $\Sigma[\sigma(\Gamma)]$ is defined in $n - 1$ steps and apply induction hypothesis.

Let $G_1 = \mathsf{gr}(\mathscr{S}_1, \mathscr{C}_1, \mathscr{E}_1, T)$ and $G_2 = \mathsf{gr}(\mathscr{S}_2, \mathscr{C}_2, \mathscr{E}_2, T)$. Define

$$G_1 \cup G_2 = \mathsf{gr}(\mathscr{S}_1 \cup \mathscr{S}_2, \mathscr{C}_1 \cup \mathscr{C}_2, \mathscr{E}_1 \cup \mathscr{E}_2, T)$$

It is easy to see that $\cup$ is associative an commutative.

Definition 10. **[compression]** *The* compression $\uplus(\Gamma)$ *of a consistent group environment Γ is the well-formed group environment defined as:*

$$\uplus(\Gamma) \triangleq \{g : \bigcup_{i \in I_g} G_i \mid g \in Dom(\Gamma),\ I_g = \{i \mid g : G_i \in \Gamma\}\}.$$

Using completion-unification and compression, we define an operation $\diamondsuit$ which transforms (if it is possible) an arbitrary group environment into a well-formed one.

Definition 11. **[uplus]** *Let Γ be an arbitrary group environment. Define $\diamondsuit(\Gamma) = \uplus(\Sigma[\Gamma](\Gamma))$.*

In order to discuss properties of this operation (and of the algorithm), we introduce a partial order on types and environments.

Definition 12. *The relation $\leq$ on group types is defined by:*

$$\mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, T) \leq \mathsf{gr}(\mathscr{S}', \mathscr{C}', \mathscr{E}', T')$$

if $\mathscr{S} \subseteq \mathscr{S}'$, $\mathscr{C} \subseteq \mathscr{C}'$, $\mathscr{E} \subseteq \mathscr{E}'$ and either $T = \mathsf{shh}$ or $T = T'$.

This order is extended monotonically to arbitrary environments by adding set inclusion:

$$\begin{array}{rl} \Gamma \leq \Gamma' & \text{if } \forall\, (g{:}G) \in \Gamma \,.\, \exists\, (g{:}G') \in \Gamma' \,.\, G \leq G' \\ \Delta \leq \Delta' & \text{if } \forall (\xi{:}W) \in \Delta \,.\, \exists\, (\xi{:}W') \in \Delta' \,.\, W \leq W' \\ \Gamma; \Delta \leq \Gamma'; \Delta' & \text{if } \Gamma \leq \Gamma' \text{ and } \Delta \leq \Delta' \end{array}$$

The meaning of $\Gamma \leq \Gamma'$ is that Γ' is more permissive than Γ on $Dom(\Gamma)$ but can contain statements which are not present in Γ. The definitions of $\Delta \leq \Delta'$ and $\Xi \leq \Xi'$ are similar. However a process P that is well typed with respect to Γ is not necessarily well typed with respect to Γ', owing to the condition in rule (OUT) which forces some inclusion conditions on the $\mathscr{S}$ components of groups.

Lemma 5. **[plusunion prop]** *Let Γ be an arbitrary group environment. Then*

(i) $\diamondsuit(\Gamma)$ (if defined) is a well-formed environment such that $\Sigma[\Gamma](\Gamma) \leq \diamondsuit(\Gamma)$.
(ii) For all substitutions σ and well-formed environments Γ' such that $\sigma(\Gamma) \leq \Gamma'$, there is a substitution σ' such that $\sigma'(\diamondsuit(\Gamma)) \leq \Gamma'$.

Proof. (i) Consistency follows immediately by Lemma 4 (i) so by Definition 10 we get well-formedness. Moreover note that $g : G \in \Sigma[\Gamma](\Gamma)$ implies $g : G' \in \diamondsuit(\Gamma)$ for some G' such that $G \leq G'$.
(ii) First notice that by definition of "$\leq$" if Γ_2 is well-formed and $\Gamma_1 \leq \Gamma_2$ then Γ_1 is well-formed too. Therefore $\sigma(\Gamma)$ is well-formed, and hence consistent. By Lemma 4 (ii) we have that there exists σ' such that $\sigma(\Gamma) = \sigma' \circ \Sigma[\Gamma](\Gamma)$. Now observe that $\sigma' \circ \Sigma[\Gamma](\Gamma) \leq \Gamma'$ implies that for each $g : G \in \Sigma[\Gamma](\Gamma)$ there is $\sigma'(g') : G' \in \Gamma'$ such that $\sigma'(G) \leq G'$. Let $I_g = \{G \mid g : G \in \Sigma[\Gamma](\Gamma)\}$. We have immediately that $\sigma'(\bigcup_{i \in I_g} G_i) \leq G'$ and this implies $\sigma'(\diamondsuit(\Gamma)) \leq \Gamma'$.

$$\varnothing; \{\xi : g_1 \to g_2\} \vdash_{\mathrm{I}} \xi : g_1 \to g_2 \quad \text{(I-Name)} \quad \text{where } \xi \text{ is a variable or an ambient name}$$

$$\{g_2 : \mathsf{gr}(\varnothing, \varnothing, \{g_1\}, t_2); \{\xi : g_1\}\} \vdash_{\mathrm{I}} \mathsf{to}\ \xi : g_1 \to g_2 \quad \text{(I-to)}$$

$$\{g_2 : \mathsf{gr}(\{g_1\}, \{g_1\}, \varnothing, t_2); \{\xi : g_1\}\} \vdash_{\mathrm{I}} \mathsf{in}\ \xi : g_2 \to g_2 \quad \text{(I-in)}$$

$$\{g_2 : \mathsf{gr}(\{g_1^*\}, \{g_1\}, \varnothing, t_2); \{\xi : g_1\}\} \vdash_{\mathrm{I}} \mathsf{out}\ \xi : g_2 \to g_2 \quad \text{(I-out)}$$

$$\frac{\Gamma; \Delta \vdash_{\mathrm{I}} M : W \qquad \Gamma'; \Delta' \vdash_{\mathrm{I}} N : W'}{\diamondsuit(\sigma(\Gamma, \Gamma')); \sigma'(\Delta, \Delta') \vdash_{\mathrm{I}} M.N : \sigma'(g_3 \to g_2)} \quad \text{(I-Path)}$$

where $\sigma' = \Sigma[\sigma(\Gamma, \Gamma')] \circ \sigma$ and $\sigma = \phi(\{(W, g_1 \to g_2), (W', g_3 \to g_1)\} \cup (\Delta, \Delta'))$

Fig. 12 Type reconstruction for messages
[tim]

Type Inference Algorithm

We give now an algorithm for reconstructing type information from a row process in the most general way. The algorithm is defined through a set of ”natural semantics” rules []. A reasonable judgement produced by such system would be of the form

$$R \Rightarrow E, P, g$$

where R is a raw process, P is a typed version of R (i.e. $|P| = R$) and $E,\ g$ are such that $E \vdash P : g$ is the “most general” typing judgement for R in the sense that that any other typing $\Xi' \vdash Q : g'$ for a process Q such that $|Q| = R$ can be obtained from it by substitution and $\leq$. To make this algorithm more readable we use instead a judgements of the form

$$\Xi \vdash_{\mathtt{I}} P : g$$

which is intended to represent the most general typing for the raw process $|P|$. Note that this could be expresses in the former form by writing $|P| \Rightarrow E, P, g$.

Figure 12 gives the inference rules for messages. Figure 13 those for processes. In all the type inference rules, group names and type variables that appear in the conclusion and do not appear in the premises are fresh. In rule (I-out) (Figure 12) some occurrences of group names in the generated environment are distinguished marking then with an asterix (*) for later use. The marked occurrences are preserved by substitution. So if an occurrence $g*$ is in the domain of a substitution replacing g by g' the occurrence is changed in $g'*$[4]. Marking is completely transparent to all operations introduced in this section.

In all the rules with two premises (i.e., rules (I-Path), (I-Prefix-Cap), (I-Output) and (I-Par)), the algorithm merges the two environments of the premises using completion-unification and compression. In rules (I-Path), (I-Prefix-Cap) and (I-Par) two groups are identified. More precisely: in the rule (I-Path) the output group of N is identified with the input group of M; in rule (I-Prefix-Cap) the input group of M is identified with the group of P; in rule (I-Par) the groups of P and Q are identified.

In the rule (I-Output), the algorithm identifies the communication types of P and M using the unifier defined in Definition 8. The same kind of unification is performed by the rule (I-Input). In the (I-Amb) rule the ambient name ξ can only be obtained by using rule (I-Name). The type of the resulting process is a fresh group name g'. The group g' is added to the set of the ambient groups where g ambients can stay.

[4] Note that in inference substitutions a group name can be replaced only by another group name.

$$\{g : \mathsf{gr}(\varnothing,\varnothing,\varnothing,t)\};\varnothing \vdash_{\mathrm{I}} 0 : g \quad \text{(I-Null)}$$

$$\frac{\Gamma;\Delta \vdash_{\mathrm{I}} M : W \qquad \Gamma';\Delta' \vdash_{\mathrm{I}} P : g_1}{\diamondsuit(\sigma(\Gamma,\Gamma'));\ \sigma'(\Delta,\Delta') \vdash_{\mathrm{I}} M.\sigma'(P) : \sigma'(g_2)}\ \text{(I-Prefix-Cap)}$$
where $\sigma' = \Sigma[\sigma(\Gamma,\Gamma')]\circ\sigma$ and $\sigma = \phi(\{(W, g_1 \to g_2)\} \cup (\Delta,\Delta'))$

$$\frac{\Gamma, g : G;\ \Delta, x : W \vdash_{\mathrm{I}} P : g}{\diamondsuit(\sigma(\Gamma, g : G));\ \sigma'(\Delta) \vdash_{\mathrm{I}} (x : \sigma'(W))\,\sigma'(P) : \sigma'(g)}\ \text{(I-Input)}$$
where $\sigma' = \Sigma[\sigma(\Gamma, g : G)]\circ\sigma$ and $\sigma = \phi(W, T(G))$

$$\frac{\Gamma;\Delta \vdash_{\mathrm{I}} P : g \quad g : G \in \Gamma \quad x \notin Dom(\Delta)}{\Gamma;\Delta \vdash_{\mathrm{I}} (x : T(G))P : g}\ \text{(I-Input-Fresh)}$$

$$\frac{\Gamma, g : G;\ \Delta \vdash_{\mathrm{I}} P : g \qquad \Gamma';\Delta' \vdash_{\mathrm{I}} M : W}{\diamondsuit(\sigma(\Gamma, g : G, \Gamma')) :\ \sigma'(\Delta;\Delta') \vdash_{\mathrm{I}} \langle M\rangle\sigma'(P) : \sigma'(g)}\ \text{(I-Output)}$$
where $\sigma' = \Sigma[\sigma(\Gamma, g : G, \Gamma')]\circ\sigma$ and $\sigma = \phi((\Delta,\Delta') \cup \{W, T(G)\})$

$$\frac{\Gamma;\Delta \vdash_{\mathrm{I}} P : g_1 \quad \Gamma';\Delta' \vdash_{\mathrm{I}} Q : g_2}{\diamondsuit(\sigma(\Gamma,\Gamma'));\sigma'(\Delta,\Delta') \vdash_{\mathrm{I}} \sigma'(P \mid Q) : \sigma'(g_2)}\ \text{(I-Par)}$$
where $\sigma' = \Sigma[\sigma(\Gamma,\Gamma')]\circ\sigma$ and $\sigma = \phi((\Delta,\Delta') \cup \{(g_1, g_2)\})$

$$\frac{\Gamma, g : \mathsf{gr}(\mathscr{S},\mathscr{C},\mathscr{E},T);\ \Delta \vdash_{\mathrm{I}} P : g}{\sigma'(\Gamma');\ \sigma'(\Delta, \xi : g) \vdash_{\mathrm{I}} \xi[\sigma'(P)] : g'}\ \text{(I-Amb)}$$
where $\sigma' = \Sigma[\Gamma']\circ\sigma$, $\Gamma' = \sigma(\Gamma,\ g : \mathsf{gr}(\mathscr{S} \cup \{g'\},\mathscr{C},\mathscr{E},T))$
and $\sigma = \phi(\{\xi : g\},\Delta)$

$$\frac{\Gamma;\ \Delta, n : g' \vdash_{\mathrm{I}} P : g}{\Gamma;\ \Delta \vdash_{\mathrm{I}} (\nu n : g')P : g}\ \text{(I-Res)} \qquad \frac{\Gamma;\ \Delta \vdash_{\mathrm{I}} P : g \quad n \notin Dom(\Delta)}{\Gamma;\ \Delta \vdash_{\mathrm{I}} (\nu n : g')P : g}\ \text{(I-Res-Fresh)}$$

$$\frac{\Gamma;\Delta \vdash_{\mathrm{I}} P : g}{\Gamma;\Delta \vdash_{\mathrm{I}} !\pi\, P : g}\ \text{(I-Repl)}$$

Fig. 13 Type reconstruction for raw processes
[tip]

Soundness and Completeness

In this subsection, we give the soundness and completeness proofs of the inference algorithm. To this aim, we introduce restricted versions of our type assignment system.

As first remark notice that the group restrictions can always be moved from a typed term to the environment preserving the typing. This can be formalised as follows:

Lemma 6. **[restriction prop]** *If $\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)}$ are all the group restrictions occurring in P and $\overline{P}$ is obtained from P by removing them, then:*

$$\Xi \vdash P : g \quad \textit{iff} \quad \Xi, \{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)} \vdash \overline{P} : g.$$

Proof. Let $C[\ ]$ be a context not containing group restrictions. It is easy to show using Lemma 1 by induction on $C[\ ]$ that:

$$\Xi \vdash C[(\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P] : g \quad \text{iff} \quad \Xi, \{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)} \vdash C[P] : g.$$

The lemma then follows.

Owing to the above Lemma the type inference process can ignore group restrictions.

Let $\vdash_{\text{-}\nu\text{g}}$ denote the the type assignment system for the variant of our language in which there are no group restrictions (i.e. rule (GRPRES)). Moreover, let $\vdash_{\text{-}\nu\text{g}}{}^{+\mathcal{V}}$ denote type assignment for the variant of $\vdash_{\text{-}\nu\text{g}}$ in which also type variables are allowed to occur in communication types. The inference rules in Figure 5 are not affected by the presence of variables. Obviously $\vdash_{\text{-}\nu\text{g}}{}^{+\mathcal{V}}$ is an extension of $\vdash_{\text{-}\nu\text{g}}$ in the sense that each statement valid in $\vdash_{\text{-}\nu\text{g}}$ is also valid in $\vdash_{\text{-}\nu\text{g}}{}^{+\mathcal{V}}$.

Finally, let $\vdash_{\text{-}\nu\text{g}-\mathscr{S}}^{+\mathcal{V}}$ denote the type inference system obtained from $\vdash_{\text{-}\nu\text{g}}{}^{+\mathcal{V}}$ by ignoring the condition on inclusion of $\mathscr{S}$ components in rule (OUT). It is easy to see by induction on deduction that $\vdash_{\text{-}\nu\text{g}-\mathscr{S}}^{+\mathcal{V}}$ is closed under substitution and it is preserved when the current environment is replaced by a well-formed and bigger (w.r.t. $\leq$) environment.

Notational convention: in the following Π denote generically a process R or a message M and τ either a process type g or a capability type $g_1 \to g_2$, the meaning being clear from the context.

Lemma 7. **[subleqclosure]**

(i) Let $\Gamma; \Delta \vdash_{\text{-}\nu g-\mathscr{S}}^{+\mathcal{V}} \Pi : \tau$ and σ be a substitution such that $\sigma(\Gamma)$ is well-formed. Then $\sigma(\Gamma; \Delta) \vdash_{\text{-}\nu g-\mathscr{S}}^{+\mathcal{V}} \sigma(\Pi) : \sigma(\tau)$.

(ii) Let $\Xi \vdash_{\text{-}\nu g-\mathscr{S}}^{+\mathcal{V}} \Pi : \tau$ and Ξ' be a well formed environment such that $\Xi \leq \Xi'$. Then $\Xi' \vdash_{\text{-}\nu g-\mathscr{S}}^{+\mathcal{V}} \Pi : \tau$.

It is now easy to see that the generation Lemma 1 holds for deductions in $\vdash_{\text{-}\nu\text{g}-\mathscr{S}}^{+\mathcal{V}}$ too. This implies that there is a unique deduction for each valid statement $\Xi \vdash_{\text{-}\nu\text{g}-\mathscr{S}}^{+\mathcal{V}} P : g$.

The main lemma for the soundness proof is the following.

Lemma 8. **[soundness main]** *If* $\Xi \vdash_{\mathtt{I}} \Pi : \tau$ *then* $\Xi \vdash^{+\mathcal{V}}_{\text{-}\nu g\_\mathscr{S}} \Pi : \tau$.

Proof. The proof is by induction on the type inference deduction (system $\vdash_{\mathtt{I}}$). Notice that in all rules the environments in the conclusions are well-formed by Lemma 5 (i). Moreover Lemma 7 (i) and (ii) assures us that we can apply the substitutions in the conclusions also to the corresponding premises and that we can weakening the environments in the premises.
As an example we show the case of rule (I-Prefix-Cap). We have:

$$\frac{\Gamma_1; \Delta_1 \vdash_{\mathtt{I}} M : W \qquad \Gamma_2; \Delta_2 \vdash_{\mathtt{I}} P : g_1}{\diamondsuit(\sigma(\Gamma_1, \Gamma_2)); \sigma'(\Delta_1, \Delta_2) \vdash_{\mathtt{I}} M.\sigma'(P) : \sigma'(g_2)}$$

where $\sigma' = \Sigma[\sigma(\Gamma_1, \Gamma_2)] \circ \sigma$ and $\sigma = \phi(\{(W, g_1 \to g_2)\} \cup (\Delta_1, \Delta_2))$.
By induction we have $\Gamma_1; \Delta_1 \vdash^{+\mathcal{V}}_{\text{-}\nu g\_\mathscr{S}} M : W$ and $\Gamma_2; \Delta_2 \vdash^{+\mathcal{V}}_{\text{-}\nu g\_\mathscr{S}} P : g_1$.
By construction we have that $\sigma'(\Delta_1, \Delta_2)$ is consistent and $\sigma'(\Delta_i) \leq \sigma'(\Delta_1, \Delta_2)$ for $i = 1, 2$.
By Lemma 5 $\diamondsuit(\sigma(\Gamma, \Gamma')) = \uplus(\sigma'(\Gamma, \Gamma'))$ is well-formed and, then also $\uplus(\sigma'(\Gamma_i))$, for $i = 1, 2$, are well-formed and such that $\uplus(\sigma'(\Gamma_i)) \leq \diamondsuit(\sigma(\Gamma_1, \Gamma_2))$.
Moreover $\sigma'(W) = g \to g'$ and $\sigma'(g_1) = g,\ \sigma'(g_2) = g'$ for some groups $g,\ g'$.
By Lemma 7 (i) and (ii) we have $\diamondsuit(\sigma(\Gamma_1, \Gamma_2)); \sigma'(\Delta_1, \Delta_2) \vdash^{+\mathcal{V}}_{\text{-}\nu g\_\mathscr{S}} M : g \to g'$ and $\diamondsuit(\sigma(\Gamma_1, \Gamma_2)); \sigma'(\Delta_1, \Delta_2) \vdash^{+\mathcal{V}}_{\text{-}\nu g\_\mathscr{S}} P : g$ and the proof follows by rule (PREFIX-CAP) .

Note that, by Lemma 8 and the generation Lemma 1, if $\Xi \vdash_{\mathtt{I}} \Pi : \tau$ there is a unique deduction of $\Xi \vdash^{+\mathcal{V}}_{\text{-}\nu g\_\mathscr{S}} \Pi : \tau$. This can indeed be obtained form the deduction of $\Xi \vdash_{\mathtt{I}} \Pi : \tau$ by applying backward in the deduction tree the substitutions generated in $\vdash_{\mathtt{I}}$ and using $\leq$ to match environments. We can keep trace of the starred group names in doing this: in particular in the union of $\mathscr{S}$ fields in the compression operation a group name is starred if it is starred in at least of the $\mathscr{S}_i$. It will turn out that in the resulting deduction in $\vdash^{+\mathcal{V}}_{\text{-}\nu g\_\mathscr{S}}$ the starred group names are all and only the occurrences of g_1 in $\mathscr{S}(G_2)$ such that there is a statement $\mathsf{out}\ \xi : g_2 \to g_2$ where $\xi : g_1,\ g_2 : G_2 \in \Xi$.

We define the following closure operation.

Definition 13. **[closure]** *The* closure *of a group environment* Γ *(written* $\triangle(\Gamma)$*), where* Γ *may contain occurrences of the marker* $*$*, is the environment computed as follows:*

repeat
if for some $g^* \in \mathscr{S}(G)$, $g : G' \in \Gamma$, *and* $\mathscr{S}(G') \not\subseteq \mathscr{S}(G)$ **then** *replace* $\mathscr{S}(G)$ *by* $\mathscr{S}(G) \cup \mathscr{S}(G')$
until there are no more g^* satisfying the above condition
Then erase all *.

Note that if Γ is well-formed also $\triangle(\Gamma)$ is well-formed and $\Gamma \leq \triangle(\Gamma)$ since the effect of closure is only that of increasing the $\mathscr{S}$ component of group types.

Lemma 9. **[closureprop]** *Let* $\Gamma; \Delta \vdash_{\mathtt{I}} \Pi : \tau$. *Then*

(i) $\triangle(\Gamma); \Delta \vdash_{\nu g}^{+\mathcal{V}} \Pi : \tau$.
(ii) Moreover if $\Gamma'; \Delta \vdash_{\nu g}^{+\mathcal{V}} \Pi : \tau$ *for some well-formed* Γ' *such that* $\Gamma \leq \Gamma'$ *then* $\triangle(\Gamma) \leq \Gamma'$.

Proof. (i) By Lemma 5 (i) Γ is well-formed and then $\triangle(\Gamma)$ is well-formed too. By Lemmas 8 and 7 (ii) there is a deduction of $\triangle(\Gamma); \Delta \vdash_{\nu g \_ \mathscr{S}}^{+\mathcal{V}} P : g$ which, by the generation Lemma 1, is unique. Now the only reason why a deduction in $\vdash_{\nu g \_ \mathscr{S}}^{+\mathcal{V}}$ can fail to be a deduction in $\vdash_{\nu g}^{+\mathcal{V}}$ is that the condition on rule $(\textsc{Out})$ is not satisfied somewhere, i.e. that for some action $\mathtt{out}\ \xi$ occurring in P we have that to $\mathtt{out}\ \xi$ is assigned a type $g_2 \to g_2$ and that $\xi : g_1 \in \Delta,\ g_1 : G_1,\ g_2 : G_2 \in \Gamma$ and $\mathscr{S}(G_1) \not\subseteq \mathscr{S}(G_2)$. But in this case g_1 is starred in $\mathscr{S}(G_2)$ and at the end of the closure algorithm all starred groups g_1 occurring in $\mathscr{S}(G_2)$ are such that $\mathscr{S}(G_1) \subseteq \mathscr{S}(G_2)$.
(ii) By induction on the number of steps of the algorithm of Def. 13 we show that the environment Γ_n resulting at the n-th step must be such that $\Gamma_n \leq \Gamma'$.

Definition 14. **[finishing]** *The* finishing *substitution* σ_f *for a judgement* $\Gamma; \Delta \vdash_{\mathtt{I}} \Pi : \tau$ *is a substitution whichreplaces with* shh *all the communication-type variables* t *that occur only in the fourth components of group types in* Γ.

Note that finishing substitutions are not inference substitutions in the sense of Definition 7.
By an inspection of the inference rules, it is easy to see that the variables mentioned in point 1. can occur only in one group type and never as components of a capability type. So they characterize the groups in which no communication is done and then can be replaced by shh. In the resulting term, there can still be left some type variables as those mentioned in point 2. which can be replaced by arbitrary communication types. Note that if an input of the shape $(x : w)$ occurs in some process then no output has been offered in the ambient containing this process.

Lastly we remark that all when the raw process is structurally equivalent to $\mathsf{0}$ the group environment contain one useless assumption of the shape $g : \mathsf{gr}(\varnothing, \varnothing, \varnothing, t)$. We convene that we erase this assumption in this particular case.

The soundness result is then a consequence of Lemma 9 (i) observing that the finishing substitution eliminates all variables preserving well-formedness of environments.

Theorem 9 (Soundness). *If $\Gamma;\Delta \vdash_{\mathtt{F}} \Pi : \tau$ and σ_f is a finishing substitution for it then $\sigma_f(\triangle(\Gamma);\Delta) \vdash_{\text{-}\nu g} \sigma_f(\Pi) : \sigma_f(\tau)$.*

Completeness of the type inference procedure can be proved by induction on deductions using Lemmas 4, 5, 9.

Theorem 10 (Completeness). **[completeness]** *If $\Xi \vdash_{\text{-}\nu g} \Pi : \tau$ then $\Xi' \vdash_{\mathtt{F}} \Pi' : \tau'$ where $|\Pi| = |\Pi'|$ and there is a substitution σ such that:*

1. $\sigma(\Pi') = \Pi$
2. $\sigma(\tau') = \tau$
3. $\sigma(\Xi') \leq \Xi$.

Proof. By induction on deductions in $\vdash_{\text{-}\nu g}$.
We only show the case of rule (AMB). The other cases are similar. So assume

$$\frac{\Xi \vdash P : g \quad \Xi \vdash \xi : g \quad \Xi \vdash g : G \quad g' \in \mathscr{S}(G)}{\Xi \vdash \xi[P] : g'}$$

where $\Xi = \Gamma;\Delta$, and ξ is either a variable or an ambient name. In both cases we must have $\xi : g \in \Delta$. Moreover $g : G \in \Gamma$.
By induction hypothesis on $\Gamma;\Delta \vdash P : g$ we have that $\Gamma_1;\Delta_1 \vdash_{\mathtt{F}} P_1 : g_1$ and there is a substitution σ_1 such that:
$1'$. $\sigma_1(P_1) = P$
$2'$. $\sigma_1(g_1) = g$
$3'$. $\sigma_1(\Gamma_1;\Delta_1) \leq \Gamma;\Delta$.
Now Γ_1 must contain an assignment for g_1 and so let

$$\Gamma_1 = \Gamma_2,\ g_1 : \mathsf{gr}(\mathscr{S}_1, \mathscr{C}_1, \mathscr{E}_1, T_1).$$

Being $g' \in \mathscr{S}(G)$, $\mathscr{S}(G)$ is not empty and then there exists a substitution σ_2 satisfying:
$1''$. $\sigma_2(P_1) = P$
$2''$. $\sigma_2(g_1) = g$
$3''$. $\sigma_2(\Gamma_1';\Delta_1) \leq \Gamma;\Delta$
4. $\sigma_2(g_2) = g$
where

$$\Gamma_1' = \Gamma_2,\ g_1 : \mathsf{gr}(\mathscr{S}_1 \cup \{g_2\}, \mathscr{C}_1, \mathscr{E}_1, T_1)$$

and g_2 is a fresh group name. Note that σ_2 can always identify g_2 with another group name.
Let $\sigma = \phi(\{\xi : g_1\}, \Delta_1)$, $\Gamma' = \sigma(\Gamma_1')$ and $\sigma' = \Sigma[\Gamma'] \circ \sigma$.
By definition of most general unifier there exists a substitution σ_1' such that $\sigma_1 = \sigma_1' \circ \sigma$. Since $\sigma_1'(\Gamma_1')$ must be a consistent group environment by

Lemma 4 (ii) there is another substitution σ_1'' such that $\sigma_1' = \sigma_1'' \circ \Sigma[\Gamma']$. Now let By rule (I-Amb) we have that

$$\Sigma[\Gamma'](\sigma(\Gamma));\ \sigma'(\Delta) \vdash_{\mathtt{I}} \sigma'(\xi[P_1]) : \sigma'(g_3)$$

Points 1.,2., 3. follow from the fact that $\sigma_1 = \sigma_1'' \circ \sigma'$ using Lemma 9.

7 Conclusion

We have presented a simple calculus that combines ambient mobility with general process mobility, putting together the standard in and out Mobile Ambients actions with the to primitive of the Distributed π-calculus [9]. As observed in section 5, other choices were possible for process mobility, namely moving up to the parent ambient, or down to a child ambient.

With the down primitive alone, even in the presence of the in and out ambient primitives, it is impossible to express the up and to moves.

The up $m\,.\,P$ construct (m being the destination ambient) is the most powerful of the three, since it allows the other two kinds of process movements to be expressed by means of it, with the help of in and out; in particular, the down $m\,.\,P$ construct may be obtained in a context-free way as $(\nu p)p[\mathsf{in}\ m\,.\,\mathsf{up}\ m\,.\,P]$, while a to $m.P$ process may be contextually encoded, within an ambient n, by the term $(\nu p)p[\mathsf{out}\ n\,.\,\mathsf{in}\ m\,.\,\mathsf{up}\ m\,.\,P]$.

Since the up primitive is deterministic (the parent ambient is only one), the encodings of the other two constructs do not contain any additional nondeterminism and are therefore correct in a strong sense, in contrast with the opposite encoding of up by means of to, where the latter's inherent nondeterminism w.r.t. homonymous ambients does not allow an expression of the former that is correct in all contexts, as remarked in section 5.

Nevertheless, we have chosen to adopt the to primitive for its greater meaningfulness in the context of mobile computing, and its grounding in a well-known foundational system such as D-π (whose untyped calculus can thus be very easily encoded in our system).

We also have chosen, for the moment, to privilege simplicity. The type system is indeed defined in such a way that subject reduction almost holds by definition. The seeming complexity of the four components of the group types is not real. From the point of view of the type inference, the $\mathscr{C}$ component merely records in and out actions, while the $\mathscr{E}$ component records to moves. From the type-checking perspective, $\mathscr{C}$ simply lists the ambient groups whereinto or wherefrom a given ambient is permitted to move (driven by in or out, respectively), while $\mathscr{E}$ lists the groups whereto a given ambient may send processes.

The $\mathscr{C}$ component, already present in other type systems (for example, in [15]), is essential for controlling ambient mobility, as shown in all the

examples of section 5: for instance, the $\mathscr{C}$ components of the various types state that the mail server and the mailboxes are immobile, and so is the firewall, and Troy and the king's palace; at the same time they permit the user to enter its own mailbox only, they permit an agent to cross the firewall, etc.

The novel $\mathscr{E}$ component is needed to control the potentially most dangerous to moves, as is also apparent from the examples: the agent ambient is not authorized to send processes into the firewall-protected site, while the firewall is authorized to send the in capability to the agent. Observe that, as usual, types are only static preconditions that do not prevent more restrictive properties from being checked at runtime; for example, though the agent is granted permission to cross the firewall by its group type, it cannot actually do it if in addition it is not provided at runtime with the appropriate capability.

$\mathscr{C}$ and $\mathscr{E}$ serve therefore different purposes, and are not interrelated: as is apparent, the former is mainly intended to control mobility, the latter is relevant for security.

The $\mathscr{S}$ component, on the other hand, is a superset of $\mathscr{C}$; it is not directly connected with security, and the reasons for its presence are not compelling (as a matter of fact, in the first version of the system it was absent, and was added later). However, without $\mathscr{S}$ the control of ambient mobility is rather lop-sided, owing to the fact that in the standard out m construct the argument m is the ambient one comes out of, instead of that whereinto one enters (like in the to m construct). Thus the $\mathscr{C}$ component cannot control which ambients are allowed to enter a given ambient from downwards, and another more general set of permissions is necessary.

For instance, without $\mathscr{S}$ the Trojans have no way of knowing that Ulysses may come into Troy, since he comes in hidden within the horse. The case is well-known, and in previous ambient systems this knowledge may be directly relevant for security. In our system, on the contrary, the danger represented by Ulysses is already clear from his permission to send general processes into the palace. Actually, with the to mechanism, the Greeks do not need the horse to set Troy on fire: they might merely send incendiary processes to Troy from the outside (the $\mathscr{E}$ component serves to protect Trojans exactly from these missiles, either coming from the outside or from the inside).

In the absence of open, the simple Ulysses' presence in Troy is not dangerous: if he is not allowed to send out processes, he can only take a harmless stroll in the city. However, the knowledge is desirable of which ambients may move where, and completely forbidding Ulysses the access to Troy is, after all, a good policy.

As a final remark, we observe that the very simplicity of our type system, which grants it an easy readability and usability, does not allow the control of finer properties, expressible through much more sophisticated types such as the ones by Pierce-Sangiorgi [**?**] for the π-calculus. Even the basic type system for D-π [9] is only incompletely rendered, since our system cannot encode the assignment of different types to homonymous channels belonging to different locations; this is made possible in D-π by the presence of local typing judgments, which cannot be simulated by our only global judgments.

References

1. Torben Amtoft, Assaf J. Kfoury, and Santiago M. Pericas-Geertsen. What are polymorphically-typed ambients? In David Sands, editor, *ESOP '01*, volume 2028 of *LNCS*, pages 206–220, Berlin, 2001. Springer-Verlag.
2. H[endrik] P[ieter] Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, revised edition, 1984.
3. M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication Interference in Mobile Boxed Ambients. In *FSTTCS'02, Int. Conf. on Foundations of Software Technology and Theoretical Computer Science*, number 2556 in Lecture Notes in Computer Science, pages 71–84. Springer–Verlag, 2002.
4. M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication and Mobility Control in Boxed Ambients. Submitted for publication. Extended and revised version of [3], April 2003.
5. Michele Bugliesi and Giuseppe Castagna. Secure safe ambients. In *POPL'01*, pages 222–235, New York, 2001. ACM Press.
6. Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Boxed ambients. In Benjamin Pierce, editor, *TACS'01*, volume 2215 of *LNCS*, pages 38–63. Springer-Verlag, 2001.
7. Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Mobility types for mobile ambients. In J. Wiederman, P. van Emde Boas, and M. Nielsen, editors, *ICALP'99*, volume 1644 of *LNCS*, pages 230–239, Berlin, 1999. Springer-Verlag.
8. Luca Cardelli and Andrew D. Gordon. Mobile ambients. In Maurice Nivat, editor, *FoSSaCS'98*, volume 1378 of *LNCS*, pages 140–155, Berlin, 1998. Springer-Verlag.
9. Matthew Hennessy and James Riely. Resource access control in systems of mobile agents (extended abstract). In U. Nestmann and B.C. Pierce, editors, *HLCL'98*, volume 16(3) of *ENTCS*, Amsterdam, 1998. Elsevier Science B. V.
10. J.R. Hindley. The principal type scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.
11. Francesca Levi and Davide Sangiorgi. Controlling interference in ambients. In *POPL'00*, pages 352–364, New York, 2000. ACM Press.
12. Francesca Levi and Davide Sangiorgi. Controlling interference in Ambients. In *POPL'00*, pages 352–364. ACM Press, New York, 2000.
13. C. Lhoussaine and V. Sassone. A dependently typed ambient calculus. In *Proceedings of European Symposium on Programming, ESOP'03*, LNCS. Springer, 2003. To appear.
14. M. Merro and M. Hennessy. Bisimulation Congruences in Safe Ambients. In *POPL'02*, pages 71–80, New York, 2002. ACM Press.

15. M. Merro and V. Sassone. Typing and subtyping mobility in boxed ambients. In L. Brim, P. Jančar, M. Křetínský, and A. Kučera, editors, *CONCUR'02*, volume 2421 of *LNCS*, pages 304–320, Berlin, 2002. Springer-Verlag.
16. R. Milner. The polyadic π-calculus: A tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *NATO ASI Series F: Computer and Systems Sciences*, pages 203–246, Berlin, 1993. Springer-Verlag.
17. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts 1-2. *Information and Computation*, 100(1):1–77, 1992.
18. D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". In *Proc. CONCUR '92*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. springer, 1992.
19. Joe Wells. The essence of principal typings. In P. Widmayer, F. Triguero, R. Morales, M. Hennessy, S. Eidenbez, and R. Conejo, editors, *ICALP'02*, volume 2380 of *LNCS*, pages 913–925, Berlin, 2002. Springer-Verlag.