Math. Struct. in Comp. Science (2008), *vol.* 18, *pp.* 221–290. © 2008 Cambridge University Press
doi:10.1017/S0960129508006658 Printed in the United Kingdom

Types for ambient and process mobility[†]

MARIO COPPO, MARIANGIOLA DEZANI-CIANCAGLINI and ELIO GIOVANNETTI

Università di Torino, Dipartimento di Informatica, Corso Svizzera 185,
10149 Torino, Italy
Email: {coppo,dezani,elio}@di.unito.it

Received 18 June 2006; revised 10 April 2007

In Memory of Sauro Tulipani

We present a new kind of ambient calculus in which the open capability is replaced by direct mobility of generic processes. 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. As usual, types express the communication, access and mobility properties of the modelled system, and inferred types express the minimal constraints required for the system to be well behaved.

1. Introduction

Spatial distribution and movement have taken on a prominent role in current information technologies: computational entities spread across the World exchange data, move from one location to another and interact with each other (either cooperatively or competitively). An adequate theoretical foundation for this new dimension of computing has therefore become a crucial objective, where computation has to be abstractly described as something that develops not only in time and in memory space, either sequentially (λ-calculus) or as a dynamic set of concurrent processes (π-calculus – see Milner *et al.* (1992) and Milner (1993)), but also in a wide geographical and administrative space.

One of the first theoretical models integrating spatial aspects was the Distributed π-calculus (Hennessy and Riely 2002), which we will refer to as Dπ for short. Dπ is an extended π-calculus with immobile locations and processes moving between locations.

In the area of programming languages, starting from a different background (loosely connected with logic programming), KLAIM was one of the first proposals explicitly containing the notion of location (De Nicola *et al.* 1998), which later developed into a fully-fledged project oriented to mobile and global computing (see Bettini *et al.* (2003) and the references therein).

The calculus of Mobile Ambients (Cardelli and Gordon 2000), which we will refer to as MA for short, is also built on the concurrency paradigm represented by the π-calculus

[†] This research was partially supported by the EU FP6-2004-510996 Coordination Action TYPES, the MURST PRIN'04 project 'McTafi', and the MURST PRIN'05 project 'Logical Foundations of Distributed Systems and Mobile Code'.

and introduced the notion of an ambient as ‘a bounded place where (generally multi-threaded) computation happens’: it can contain nested subambients in a dynamic tree structure, and can move in and out of other ambients, that is, 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 the dissolution of ambient boundaries.

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

For this reason, the distinction between processes and (possibly mobile) containers of processes is intentionally blurred in ambient calculi. In MA there are, implicitly, two main kinds of processes, which we will call *lightweight processes* (or *naked processes*) and *ambient processes* (or simply *ambients*).

— Lightweight processes are unnamed lists of actions[†] $\mathsf{act}_1.\mathsf{act}_2\ldots\mathsf{act}_m$ to be executed sequentially, and generally concurrently with other processes: they may perform communication and drive their containers through the spatial hierarchy, but in the original MA calculus, they cannot individually go from one ambient to another.

— Ambient processes 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 perform communication directly.

In MA, therefore, mobile processes must be represented by ambient-processes, with communication between them represented by the exchange of other ambients of usually shorter life, which have their boundaries dissolved by an open action so as to expose their internal lightweight processes performing the input–output proper. The capability of opening an ambient in this way has, however, 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 (for a survey, see Giovannetti (2003)) handling this issue, the calculus of Safe Ambients (Levi and Sangiorgi 2003; Bugliesi and Castagna 2002) introduced the notion of coaction, through which, among other things, an ambient cannot be opened without its agreement.

In the calculus of Boxed Ambients (Bugliesi *et al.* 2004), on the other hand, open is dropped altogether, and its absence is compensated for by the possibility of direct communication between the parent and child ambients.

In the present paper, 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 lightweight processes, thus avoiding the more general opening mechanism.

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

We do this by recovering the idea, present in Dπ (Hennessy and Riely 2002), that a lightweight process may move directly from one location to another, without needing to be enclosed in an ambient. Mobile lightweight processes also seem to represent *strong software mobility* more closely: with this, a procedure (function, 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. Further discussion of the relations between MA, Boxed Ambients and the present calculus can be found in Section 7.

All ambient calculi come with type systems (Cardelli *et al.* 1999) as essential components, since they are intended, like any formal description, as foundations for reasoning about program behaviours in the new global computing reality. In our proposal also, the calculus is an elementary basis for a type discipline that can control communication as well as access and mobility properties. Among the many complex features that could be envisaged, we have singled out a system that is non-trivial but simple enough to be easily readable and understandable. We have named this system M^3: a system of *Mobile processes and Mobile ambients with Mobility types.*

Following Cardelli *et al.* (2002) and Merro and Sassone (2002), the notion of a *group* is the key notion: groups characterise the possible movements and communications of ambients and processes.

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

In addition to the usual reduction semantics, where types play no role (since the rules are assumed to apply to well-typed terms only), a labelled transition system (LTS for short) semantics is given, which allows us to define a notion of bisimilarity and to compare it with a contextual equivalence, namely a barbed congruence. As M^3 is a typed calculus, the definition of the LTS involves types in an essential way: typed processes can only perform actions that are allowed by their types. The transition relation, therefore, is relative to an environment Ξ and to a process type g, and is, accordingly, denoted by the symbol $\xrightarrow{\alpha}_{\Xi,g}$. Spatial mobility is handled using the notions of concretion and higher-order transition (Merro and Hennessy 2002; Bugliesi *et al.* 2005). The resulting notion of bisimilarity is proved to be a congruence and to represent a sound approximation to barbed (observational) equivalence. Natural notions of process and ambient mobility, which are definable in terms of types, may be characterised using our LTS, and useful algebraic laws may be proved.

In spite of its simplicity, the calculus seems to possess sufficient expressive power, as shown by the compositional encodings in it of two standard calculi of concurrency: π-calculus and Dπ. In our encoding of the π-calculus, as in the encoding given in Cardelli and Gordon (1999), there are ambients that play the role of communication buffers: the advantage of moving lightweight processes as well as ambients (which need to be open for communicating) is that we can type such buffer-ambients as immobile, without introducing objective moves as in Cardelli and Gordon (1999). This feature also appears in the encoding in Mobile Safe Ambients (Levi and Sangiorgi 2003), thanks to the refined type system and the presence of co-actions.

The type system enjoys a kind of principal typing property, which is an adaptation to our setting of Wells' standard definition (Wells 2002). For every *raw* term, that is, a term in which type annotations are missing, there exists a 'most general' way of completing it with such annotations, where types provide the maximum amount of information that they can give about the behaviour of the term. A type reconstruction algorithm is then specified, which, given a raw term, yields its most general typing.

The algorithm is proved to be sound and complete, and a formal certification in Coq (of a preliminary version of it) has been given (Honsell and Scagnetto 2004); it has also been implemented in Prolog, through the intermediate step of a more algorithmic formulation of the type system itself (Giovannetti 2004).

This paper is an extended and improved version of Coppo *et al.* (2003), which was presented at CATS'03: in particular, the LTS behavioural semantics has been added.

The paper is organised as follows. In Section 2 we present the syntax of the calculus (without the type syntax) with the reduction operational semantics. The type system is defined in Section 3, where the usual subject reduction property is proved.

We define the relation of barbed congruence between processes in Section 4. We also present the new behavioural semantics and define a notion of full bisimilarity, which we prove to be sound (but not complete) with respect to the barbed congruence. We also state a number of useful algebraic laws and sketch the proofs.

In Section 5, we present some examples showing the expressiveness of the calculus and type system. These show how we can encode some well-known calculi for concurrency, and some common protocols.

In Section 6 we specify a type reconstruction algorithm, along with the proof of its soundness and completeness with respect to the type system. Section 7 compares M^3 with other ambient calculi, and Section 8 gives a brief account of a Prolog implementation. Finally, we present some concluding remarks in Section 9.

2. The calculus

The structural syntax of the pre-terms of the M^3 calculus is shown in Figure 1. It is the same as the MA syntax (Cardelli and Gordon 2000) apart from the absence of open and the presence of the new primitive to for lightweight process mobility. Also, synchronous output is allowed: the asynchronous version is a particular case of this.

M^3 is a typed calculus where types occur in the term syntax: namely, in the input construct and in the restrictions with respect to ambient names and group names. Figure 1 is thus to be read in conjunction with Figure 4, which describes the type syntax. More precisely, the pre-terms defined in Figure 1 are not exactly the terms of the calculus, because type constraints are not considered; the well-formed terms are only those pre-terms that are well typed (with respect to a typing environment) according to the typing rules given in Figure 5.

The notions of reduction and structural equivalence are therefore defined relative to a typing environment and a process type, just like the relations concerning the behavioural

N	$::=$		**ambients**
		$m, n, \ldots$	ambient names
		$x, y, \ldots$	variables
C	$::=$		**capabilities**
		$\mathsf{in}\ N$	moves the containing ambient into ambient N
		$\mathsf{out}\ N$	moves the containing ambient out of ambient N
		$\mathsf{to}\ N$	moves a process from its ambient into sibling ambient N
		$C.C$	path
		$x, y, \ldots$	variables
$M ::=$			**messages**
		N	ambients
		C	capabilities
π	$::=$		**prefixes**
		$C\,.$	capability
		$\langle M \rangle$	synchronous output
		$(x\colon W)$	typed input
$P,\ Q,\ R$	$::=$		**processes**
		$\mathbf{0}$	null
		$\pi\, P$	prefixed
		$P \mid Q$	parallel composition
		$N[P]$	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 of pre-terms.

semantics that will be presented in Section 4. In a well-typed term, however, reduction and structural equivalence are always consistent with the typing of the term (Theorem 3.3). So type information plays no explicit role in these rules, and is therefore omitted in their definition.

Since group types contain group names, the possibility of simultaneously restricting a set of group names becomes crucial: while ambient name restriction is, as usual, monadic, the fact that groups can have mutually dependent group types requires group restriction to be polyadic. We use $v\{g_1 : G_1, \ldots, g_k : G_k\}$ to denote the simultaneous restriction of the group names $g_1, \ldots, g_k$ having group types $G_1, \ldots, G_k$, respectively. 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* standing for both ambient names and capabilities.

Free and bound names and variables are defined in the usual way. A capability or a process is *closed* if it does not contain free variables (though it may contain free names). We identify processes up to α-renaming of bound names and bound variables.

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

The structural congruence rules, which are shown in Figure 2, are standard for the usual ambient constructors (Cardelli *et al.* 1999). The rules for group restriction allow us, under suitable conditions, to permute, split and erase simultaneous restrictions. Despite equivalence:

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

congruence:

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

prefix associativity:

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

parallel composition – associativity, commutativity, zero:

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

replication:

$$!\pi P \equiv \pi P \,|\, !\pi P$$

restriction swapping and group restriction splitting :

$$(\nu n\!:\!\mathsf{amb}(g))(\nu m\!:\!\mathsf{amb}(g'))P \equiv (\nu m\!:\!\mathsf{amb}(g'))(\nu n\!:\!\mathsf{amb}(g))P \quad \text{if } m \neq n$$

$$\begin{array}{l} g_i \neq g'_j \& g_i \notin GN(G'_j) \& g'_j \notin GN(G_i)(1 \leq i \leq k)(1 \leq j \leq h) \\ \implies (\nu\{\overrightarrow{\mathbf{g}\!:\!\mathbf{G}}\}_{(k)})(\nu\{\overrightarrow{\mathbf{g}'\!:\!\mathbf{G}'}\}_{(h)})P \equiv (\nu\{\overrightarrow{\mathbf{g}'\!:\!\mathbf{G}'}\}_{(h)})(\nu\{\overrightarrow{\mathbf{g}\!:\!\mathbf{G}}\}_{(k)})P \end{array}$$

$$(\nu n\!:\!\mathsf{amb}(g))(\nu\{\overrightarrow{\mathbf{g}\!:\!\mathbf{G}}\}_{(k)})P \equiv (\nu\{\overrightarrow{\mathbf{g}\!:\!\mathbf{G}}\}_{(k)})(\nu n\!:\!\mathsf{amb}(g))P \quad \text{if } g \neq g_i \ (1 \leq i \leq k)$$

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

scope extrusion:

$$\begin{array}{lll} (\nu n\!:\!\mathsf{amb}(g))P \,|\, Q \equiv (\nu n\!:\!\mathsf{amb}(g))(P \,|\, Q) & \text{if } n \notin AN(Q) \\ N[(\nu n\!:\!\mathsf{amb}(g))P] \equiv (\nu n\!:\!\mathsf{amb}(g))N[P] & \text{if } n \neq N \\ (\nu\{\overrightarrow{\mathbf{g}\!:\!\mathbf{G}}\}_{(k)})P \,|\, Q \equiv (\nu\{\overrightarrow{\mathbf{g}\!:\!\mathbf{G}}\}_{(k)})(P \,|\, Q) & \text{if } \vec{g} \notin GN(Q) \\ N[(\nu\{\overrightarrow{\mathbf{g}\!:\!\mathbf{G}}\}_{(k)})P] \equiv (\nu\{\overrightarrow{\mathbf{g}\!:\!\mathbf{G}}\}_{(k)})N[P] & \end{array}$$

equivalence to zero:

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

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.

their awkward aspects, they are basically analogous to the rules for name restriction; what complicates the notation is the fact that mutually dependent group types must be handled simultaneously.

The reduction rules, shown in Figure 3, are the same as those for MA, with the obvious differences arising from the inclusion of synchronous output and omission of open, and with the new rule for the to action, which is 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, unlike the case when moving upward or downward, the process does not change its nesting level.

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

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:

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

Fig. 3. Reduction.

In Dπ, by contrast, the go action moves processes independently of the formal presence of the target location, so $m[\mathsf{go}\, m.P]$ reduces to $m[P]$.

As usual, $\rightarrow^*$ will denote the reflexive and transitive closure of the relation $\rightarrow$.

3. The type system

As noted in the previous section, the syntax definition of Figure 1 is not complete without the type syntax and typing rules. As is usual in ambient calculi, a first (trivial) role of types is to ensure that no meaningless terms can be defined or can result from a computation. In addition, types control access and mobility. As already observed in Cardelli *et al.* (2002), 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 along these lines can be found in Lhoussaine and Sassone (2004). Following the seminal work in Cardelli *et al.* (2002), and, among others, Merro and Sassone (2002), we instead adopt the more common approach based on ambient *groups*, which enables us to avoid the direct dependence of types on values.

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

We use a simplification with respect to the systems in Cardelli *et al.* (2002) and Merro and Sassone (2002), in which ambient types have the schematic form $\mathsf{amb}(g,B)$, where B is the expression of some behavioural properties concerning mobility and communication. In our proposal the property B is instead (the content of) the type of the group, and the typing judgment $m\colon\mathsf{amb}(g,B)$ becomes $m\colon\mathsf{amb}(g),\ g\colon B$.

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

Fig. 4. Types.

The first form is more general, allowing different ambient types for the same group. In our approach, however, a group represents a set of ambients that are guaranteed to share 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 of a process type $\mathsf{proc}(g)$ is a group name g, whose type[†] 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 prevent types from directly 'pointing to' (that is, depending on) values.

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

The form of capability types is relatively novel in that they are (very particular) sorts of function types from processes to processes, corresponding to the fact that, from a syntactic point of view, a capability turns a process into another process, which is designed to run in a possibly different ambient. $\mathsf{proc}(g_1) \rightarrow \mathsf{proc}(g_2)$ 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, the type of a capability that, when exercised by a process of type $\mathsf{proc}(g_2)$, which is, 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 the one in Amtoft *et al.* (2001), where a capability type is a type context that, when filled with a process type, yields another process type.

Notational remark. We shall write g for both $\mathsf{amb}(g)$ and $\mathsf{proc}(g)$ as the distinction will always be clear from the context. As a consequence, capability types will be written concisely in the form $g_1 \rightarrow g_2$.

[†] Note that a group type is the type of a type, that is, following a fairly standard terminology, a *kind*. Moreover, since it contains group names, it might be considered a 'kind dependent on types'. This double level, however, is clearly used only in a very limited and *ad hoc* way, with no real stratification. We are, therefore, justified in not using the expression *group kind*, but simply sticking to *group type*.

We will also use the abbreviations *g-ambients* and *g-processes* for *the ambients of group g* and *the processes of group g*, respectively.

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, which is typical of ambient systems, is not the type of empty messages (which can be used for synchronisation), but the type 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 $\mathsf{gr}(\mathscr{S},\mathscr{C},\mathscr{E},T)$, the intuitive meanings of the type's components are as follows:

- $\mathscr{S}$ is the set of ambient groups where the g-ambients can stay.
- $\mathscr{C}$ is the set of ambient groups that g-ambients can cross, that is, those that they may be driven into or out of by in or out actions, respectively. Clearly, $\mathscr{C} \subseteq \mathscr{S}$ (and $\mathscr{C}$ is empty if the g-ambients are immobile).
- $\mathscr{E}$ is the set of ambients that (lightweight) g-processes can 'enter'. More precisely, those ambients 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 contained in the $\mathscr{S}$ and $\mathscr{C}$ components was also considered in Merro and Sassone (2002).

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

The $\mathscr{C}$ component is essential for controlling ambient mobility. For instance, ambients of a group whose group type has $\mathscr{C} = \varnothing$ are immobile. The novel $\mathscr{E}$ component is needed to control the potentially most dangerous to-moves. So $\mathscr{C}$ and $\mathscr{E}$ serve different purposes, and are not interrelated. The former is intended to control ambient mobility, the latter is relevant for security. The $\mathscr{S}$ component is a superset of $\mathscr{C}$. It is not directly connected with security, and the reasons for its presence are not compelling. However, if we omit $\mathscr{S}$, the control of ambient mobility becomes rather lop-sided, because in the standard $\mathsf{out}\, m$ construct, the argument m is the ambient to be exited, rather than the one to be entered (as in the $\mathsf{to}\, m$ construct). Thus the $\mathscr{C}$ component cannot control which ambients are allowed to enter a given ambient from underneath, and an additional more general set of permissions like $\mathscr{S}$ is required. Using the $\mathscr{S}$ component, we require that the ambient moved by an $\mathsf{out}\, m$ capability is allowed to stay in every ambient where m itself is allowed to stay.

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

$$Dom(\varnothing) = \varnothing \qquad Dom(\Gamma,\, g : G) = Dom(\Gamma) \cup \{g\} \qquad Dom(\Delta,\, \xi : W) = Dom(\Delta) \cup \{\xi\}\,.$$

$GN(G)$ 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 just those in $Dom(\Gamma)$ but also those in the components of the types in Ξ. Environments are considered as sets of statements, and therefore modulo permutations and duplications.

A variable environment Δ is *well formed* if for each $\xi \in Dom(\Delta)$ there is exactly one type associated with it in Δ, that is, 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 with 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.

We use the standard notation $\Delta,\ \xi : W$ to denote a variable environment containing a statement $\xi : W$, assuming that $\xi \notin Dom(\Delta)$. We write $\xi : W \in \Delta$ as an abbreviation for $\Delta = \Delta', \xi : W$ for some Δ', and write $\xi : W \in \Xi$ if $\Xi = \Gamma, \Delta$ and $\xi : W \in \Delta$. Also, we use the notation Δ_1, Δ_2 to represent the variable environment consisting of the set union of Δ_1 and Δ_2 (that is, with elimination of duplicates). We adopt similar conventions for group environments and environments.

The formal definition of the type assignment rules is shown in Figure 5. The system's fundamental rule (Amb) is quite standard: it requires that in a term $N[P]$ the ambient N and its content P be of the same group, while the process $N[P]$, as it is a completely passive object that cannot either communicate or move other ambients, may, in turn, stay in any ambient of any group g' (that is, it may be of any group g'), *provided* its 'membrane' N, of type g, has permission from the specification G to stay in a g'-ambient.

Since a process executing an action to N goes from its ambient (in)to an ambient N, the rule (To) states that the action to N, if performed by a g_2-process (in a g_2-ambient), leaves as continuation a g_1-process, if g_1 is the group of N and, moreover, is one of the groups to which g_2-processes are allowed to go (that is, to send their continuations) by a to.

The rules (In) and (Out) state that a process exercising an in/out N 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 N. Moreover, in the case of (Out), the g_2-ambient, which is being driven out of N, becomes a sibling of N, and must therefore

[†] Note that this syntactic definition of ξ as a variable or an ambient name is exactly the same as the definition of an ambient N. The reason we introduce a different notation here is that ξ may be an ambient name, an ambient variable or a capability variable, while the third alternative is clearly excluded for N. The two notions only appear to coincide in the pre-term syntax because there, following a well-established use in ambient calculi, we do not distinguish between the two kinds of variables, but use the same letters $x, y, \ldots$ for both (which amounts to having a single syntactic category for all variables).

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

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

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

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

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

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

$$\frac{\Gamma;\, \Delta, x : W \vdash P : g \quad g : \mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, W) \in \Gamma}{\Gamma;\, \Delta \vdash (x : W)P : g}\ (\textsc{Input})$$

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

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

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

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

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

Fig. 5. Typing rules.

have permission to stay where N stays (that is, the condition $\mathscr{S}(G_1) \subseteq \mathscr{S}(G_2)$). The analogous side condition in the rule (In), which ensures that the moving g_2-ambient has the permission to stay inside the g_1-ambient N ($g_1 \in \mathscr{S}(G_2)$), 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, sorts of function composition and function application, respectively.

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 (that is, moved from the left-hand side to the right-hand side of the turnstile) cannot contain the group g of the restricted term.

Note that our system has an implicit notion of subtyping since, for example, we can derive $\Gamma, g : G; \Delta \vdash P : g$ with $T(G) \neq \mathsf{shh}$ also for a silent process P (that is, for a process offering no communication), and this allows us to send silent processes to ambients where the topic of conversation is different from shh.

The type assignment system is clearly syntax-directed, so a Generation Lemma holds trivially.

Lemma 3.1 (Generation Lemma).

1 If $\Xi \vdash \xi : W$, then $\xi : W \in \Xi$.
2 If $\Xi \vdash \mathsf{in}\ N : g_2 \to g_2$, then $g_2 : G_2 \in \Xi$, and $N : g_1 \in \Xi$, and $g_1 \in \mathscr{C}(G_2)$ for unique g_1, G_2.
3 If $\Xi \vdash \mathsf{out}\ N : g_2 \to g_2$, then $g_1 : G_1 \in \Xi$, and $g_2 : G_2 \in \Xi$, and $N : g_1 \in \Xi$, and $g_1 \in \mathscr{C}(G_2)$, and $\mathscr{S}(G_1) \subseteq \mathscr{S}(G_2)$ for unique g_1, G_1, G_2.
4 If $\Xi \vdash \mathsf{to}\ N : g_1 \to g_2$, then $g_2 : G_2 \in \Xi$, and $N : g_1 \in \Xi$, and $g_1 \in \mathscr{E}(G_2)$ for a unique G_2.
5 If $\Xi \vdash C.C' : g_1 \to g_2$, then $\Xi \vdash C : g_3 \to g_2$, and $\Xi \vdash C' : g_1 \to g_3$ for a unique g_3.
6 If $\Xi \vdash C.P : g_2$, then $\Xi \vdash C : g_1 \to g_2$, and $\Xi \vdash P : g_1$ for a unique g_1.
7 If $\Gamma; \Delta \vdash (x : W)P : g$, then $\Gamma; \Delta, x : W \vdash P : g$ and $g : \mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, W) \in \Gamma$ for unique $\mathscr{S}, \mathscr{C}, \mathscr{E}$.
8 If $\Xi \vdash \langle M \rangle P : g$, then $\Xi \vdash P : g$, and $\Xi \vdash M : W$, and $g : \mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, W) \in \Xi$ for a unique $\mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, W)$.
9 If $\Xi \vdash N[P] : g$, then $\Xi \vdash P : g'$, and $N : g' \in \Xi$, and $g' : G \in \Xi$, and $g \in \mathscr{S}(G)$ for unique g', G.
10 If $\Xi \vdash P \mid Q : g$, then $\Xi \vdash P : g$, and $\Xi \vdash Q : g$.
11 If $\Xi \vdash !\, \pi . P : g$, then $\Xi \vdash \pi . P : g$.
12 If $\Gamma; \Delta \vdash (\nu m : g')P : g$, then $\Gamma; \Delta, m : g' \vdash P : g$.
13 If $\Gamma, \Delta \vdash (\nu\{g_1 : G_1, \ldots, g_k : G_k\})P : g$, then $\Gamma, g_1 : G_1, \ldots, g_k : G_k; \Delta \vdash P : g$, and $g_i \notin GN(\Gamma; \Delta)$ and $g_i \neq g$ $(1 \leqslant i \leqslant k)$.

Note that this lemma implies that every typing judgement has a unique derivation.

It is easy to verify by induction on derivations that the type assignment system enjoys the weakening and substitution properties.

Lemma 3.2 (Weakening and substitution Lemma). Let Π denote a process P or a message M, and τ a process type g or a message type W. Then we have:

1 If $\Gamma; \Delta \vdash \Pi : \tau$, the environments $\Gamma'; \Delta'$ are well formed and $\Gamma \subseteq \Gamma'$, $\Delta \subseteq \Delta'$, then $\Gamma'; \Delta' \vdash \Pi : \tau$.
2 If $\Gamma; \Delta, x : W \vdash \Pi : \tau$ and $\Gamma; \Delta \vdash M : W$, then $\Gamma; \Delta \vdash \Pi\{x := M\} : \tau$.

The usual property of subject reduction holds, which guarantees the soundness of the system by ensuring that typing is preserved by computation and structural equivalence. Note that we do not need to expand environments as in Cardelli *et al.* (2002) since we allow a type in a variable environment to contain a group name even though such a group is not assigned a group type by the associated group environment (provided this is compatible with the assignment rules), that is, we allow $\Gamma; \Delta$ with $\xi : g \in \Delta$ even if $g \notin Dom(\Gamma)$.

Theorem 3.3 (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 by induction on the derivations of $P \equiv Q$ and $P \;\rightarrow\; Q$ using the Weakening, Substitution and Generation Lemmas is standard. We will 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 3.1(10), we have $\Xi \vdash n[\mathsf{to}\, m \,.\, P \mid Q] : g$ and $\Xi \vdash m[R] : g$. By Lemma 3.1(9) we must have

$$\begin{aligned} \Xi \vdash \mathsf{to}\, m \,.\, P \mid Q &: g_n \\ n : g_n &\in \Xi \\ g_n : G_n &\in \Xi \\ g &\in \mathscr{S}(G_n) \\ \Xi \vdash R &: g_m \\ m : g_m &\in \Xi \\ g_m : G_m &\in \Xi \\ g &\in \mathscr{S}(G_m) \end{aligned}$$

for some g_n, G_n, g_m, G_m.

From $\Xi \vdash \mathsf{to}\, m \,.\, P \mid Q : g_n$, by Lemma 3.1(10), we have $\Xi \vdash Q : g_n$ and $\Xi \vdash \mathsf{to}\, m \,.\, P : g_n$, which by Lemma 3.1(6 and 4) implies $\Xi \vdash P : g_m$.
Rule (Amb) applied to $\Xi \vdash Q : g_n$ gives $\Xi \vdash n[Q] : g$ being $n : g_n \in \Xi$, and $g_n : G_n \in \Xi$, and $g \in \mathscr{S}(G_n)$. Rule (Par) applied to $\Xi \vdash P : g_m$, and $\Xi \vdash R : g_m$ gives $\Xi \vdash P \mid R : g_m$. Since

$$\begin{aligned} m : g_m &\in \Xi \\ g_m : G_m &\in \Xi \\ g &\in \mathscr{S}(G_m)\,, \end{aligned}$$

we can deduce $\Xi \vdash m[P \mid R] : g$ using rule (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 (Par). □

A process containing variables represents a set of closed processes, which can be obtained by replacing the variables by ambient names or closed capabilities, depending on their types. In an untyped setting, the set of possible replacements is in general infinite and never empty. On the other hand, group environments can prevent the replacement of capability variables. Processes containing variables that cannot be replaced are ‘dead’ in the sense we will discuss in Section 4.

We conclude this section with some definitions that formalise this notion of non-replaceable variable and the related notion of a *closing* substitution.

Definition 3.4 (Ξ-ghost variables and processes).

1 Let Γ be a group environment. A capability type V is Γ-*ghost* if there is no well-formed environment $\Gamma' ; \Delta$, with $\Gamma' \supseteq \Gamma$, such that the typing $\Gamma' ; \Delta \vdash C : V$ holds for some closed capability C.

2 A variable x is $\Gamma ; \Delta$-ghost if $x : V \in \Delta$ where V is Γ-ghost.

3 A process is Ξ-*ghost* if $\Xi \vdash P : g$ for some g and P contains a Ξ-ghost variable.

For example, if $\Xi_0 = \{ g : \mathsf{gr}(\emptyset, \emptyset, \emptyset, \mathsf{shh})\} ; \{x : g \to g\}$, the variable x is Ξ_0-ghost since the typing rules prevent the possibility of defining a closed capability of type $g \to g$ (an ambient n having group type g can neither move nor send processes) from $\mathsf{gr}(\emptyset, \emptyset, \emptyset, \mathsf{shh})$. An effective characterisation of the Ξ-ghost variables for a fixed environment Ξ can be given, but this is not relevant here.

Intuitively, a ghost type with respect to an environment is a type that cannot have (closed) inhabitants with respect to that environment, not even by extending it. The definition only mentions capability types because, for an arbitrary ambient type g, we can always add a fresh ambient name with type g to the environment.

If P is Ξ-ghost and $P \equiv Q$ holds, then Q is Ξ-ghost also (the property of being Ξ-ghost is preserved by structural equivalence).

Let Γ be a group environment occurring in a statement of a typing derivation and Γ' be the group environment used in the conclusion of the same derivation. If the type V is Γ-ghost, then V is also Γ'-ghost. More precisely, we have the following proposition.

Proposition 3.5. Let $\Gamma ; \Delta \vdash P : g$ be a statement occurring in a derivation ending with $\Gamma' ; \Delta' \vdash Q : g'$. If the type V is Γ-ghost, then V is also Γ'-ghost.

Proof. We prove that there exists no closed capability C such that $\Gamma_e ; \Delta' \vdash C : V$ for some Γ_e extending Γ'. The proof is by induction on the deduction of $\Gamma' ; \Delta' \vdash Q : g'$.

The base step is when Q coincides with P. In this case the proof follows immediately from the definition of ghost type.

The only non-trivial case for the induction step is for rule (GrpRes). In this case, let $Q = (\nu \{\overrightarrow{\mathbf{g} : \mathbf{G}}\}_{(k)})\, Q'$. Then we have

$$\frac{\Gamma', \overrightarrow{\mathbf{g} : \mathbf{G}}_{(k)} ; \Delta' \vdash Q' : g \quad g_i \notin GN(\Gamma' ; \Delta') \quad g_i \neq g \quad (1 \leqslant i \leqslant k)}{\Gamma' ; \Delta' \vdash (\nu \{\overrightarrow{\mathbf{g} : \mathbf{G}}\}_{(k)})\, Q' : g}.$$

Assume, in order to show a contradiction, that Γ_e is an extension of Γ' such that we have $\Gamma_e ; \Delta' \vdash C : V$ for some closed capability C. Since the group names $\overrightarrow{\mathbf{g}}_{(k)}$ are bound in Q, we can assume without loss of generality that they do not occur in the domain of Γ_e. This implies that $\Gamma_e, \overrightarrow{\mathbf{g} : \mathbf{G}}_{(k)}$ is well formed. By the Weakening Lemma (Lemma 3.2(1)), we get $\Gamma_e, \overrightarrow{\mathbf{g} : \mathbf{G}}_{(k)} ; \Delta' \vdash C : V$, and this contradicts the induction hypothesis, since $\Gamma_e, \overrightarrow{\mathbf{g} : \mathbf{G}}_{(k)}$ is an extension of $\Gamma', \overrightarrow{\mathbf{g} : \mathbf{G}}_{(k)}$. □

The next definition is useful for singling out the environments that allow all non-ghost variables to be replaced by closed messages.

Definition 3.6 (Complete environments). An environment Ξ is *complete* if:

1 for all $x : g \in \Xi$ there is $n : g \in \Xi$ for some n;
2 for all $x : g \rightarrow g' \in \Xi$ either there exists a closed capability C such that $\Xi \vdash C : g \rightarrow g'$ or x is Ξ-ghost.

Definition 3.7 (Ξ-closing substitutions). Let Ξ be a complete environment. A Ξ-*closing substitution* $\mathbf{s}$ is a partial mapping from variables to ambient names and closed capabilities such that $\Xi \vdash \mathbf{s}(x) : W$ whenever $x : W \in \Xi$ and x is not Ξ-ghost.

It is easy to verify that a Ξ-closing substitution is undefined for all and only the Ξ-ghost variables. Also, if $\mathbf{s}$ is a Ξ-closing substitution and we assume $\mathbf{s}(x) = x$ for the Ξ-ghost variables x, then $\Xi \vdash \mathbf{s}(P) : g$ whenever $\Xi \vdash P : g$. Finally, $\mathbf{s}(P)$ is a closed process if and only if P is not Ξ-ghost.

4. Behavioural semantics

In order to develop a theory of equivalence for M^3, we start, as usual, with a notion of observability predicate (Milner and Sangiorgi 1992), which denotes the fact that a process can interact with the environment. A standard notion of observable for ambient calculi (Cardelli and Gordon 1999) is the exhibition of an ambient at the top level. Other choices of observables could be considered, such as the possibility of performing an in action. The relations between these different choices have been investigated in Merro and Hennessy (2002) for another variant of the ambient calculus: all the different notions of observable are shown to lead to the same observational equality. The same property holds for our calculus with a similar proof.

As M^3 is a typed calculus, we take the following definition of *observability of an ambient name n with respect to an environment Ξ and a group g*.

Definition 4.1 (Observability). Let P be a closed process. We say that P *exhibits the ambient name n with respect to an environment Ξ and a group g* (notation $P \downarrow_n^{\Xi,g}$) if

$$P \equiv (\nu \widetilde{g' : G})(\nu \widetilde{p : g})(n[P'] \mid Q),$$

where $n \notin \widetilde{p}$ and $\Xi \vdash P : g$.

We will use $P \Downarrow_n^{\Xi,g}$ as an abbreviation for $P \rightarrow^* P'$ with $P' \downarrow_n^{\Xi,g}$.

Note that group restrictions cannot in any way affect the computational properties of processes. As remarked in Cardelli *et al.* (2002), group restrictions are mainly a tool for preventing restricted groups from ever being known in the outside environment, thus increasing security against external malicious agents. But note that a group name g can only be restricted if there are neither variables nor ambient names whose types mention g in the variable environment: all the variables and names of this kind must have already been abstracted or restricted so that they are no longer visible.

To formalise the above argument, we define $\overline{P}$ to be the process obtained from P by pushing outwards all group restrictions. It is easy to check that

$$P \downarrow_n^{\Xi,g} \text{ if and only if } \overline{P} \downarrow_n^{\Xi,g} .$$

Furthermore, if $P \to^* P'$ holds, $\overline{P} \to^* \overline{P'}$ holds also, and if $\overline{P} \to^* Q$ holds, then $P \to^* P'$ holds for some P' such that $\overline{P'} = Q$. Therefore, group restrictions cannot change the observability of ambient names, so in this section we are allowed, without loss of generality, to consider only those processes that have no occurrences of group restrictions. This leads to a simplified formulation of the labelled transition system. Note, in fact, that none of the labelled transition in Figures 8, 9 or 10 contains any group names: transitions are transparent to group restrictions.

Since observability is relative to environments and groups, both *barbed bisimulation* and *reduction barbed congruence* (Sangiorgi and Milner 1992; Levi and Sangiorgi 2003; Sangiorgi and Walker 2001) are also defined with respect to environments and groups. More precisely, environments and groups are used both to test observability and to type the processes that are to be compared with the enclosing contexts.

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*. In the following we will often consider a family of relations with disjoint domains indexed by the pair (Ξ, g), rather than a single relation. With some abuse of terminology, we will just call them ‘relations’ rather than ‘families of relations’.

Definition 4.2.

1 A (Ξ, g)-relation is *reduction-closed* if $P\,\mathcal{R}^{\Xi,g}\,Q$ and $P \to P'$ imply the existence of some Q' such that $Q \to^* Q'$ and $P'\,\mathcal{R}^{\Xi,g}\,Q'$.
2 A (Ξ, g)-relation is *barb-preserving* if $P\,\mathcal{R}^{\Xi,g}\,Q$ and $P \downarrow_n^{\Xi,g}$ imply $Q \Downarrow_n^{\Xi,g}$.

In order to introduce reduction barbed congruence, we need the notion of a *context* $\mathcal{C}[\;]$, which is defined in the usual way as a process with a hole in it. As remarked above, behavioural properties are transparent to group restrictions, so, in the following, we will only consider contexts that have no occurrences of group restrictions.

Following Sangiorgi and Walker (2001), we say that a context is *agreeing* with an environment Ξ and a group g, and denote it by $\mathcal{C}[\;]^{\Xi,g}$, if the judgment $\Xi' \vdash \mathcal{C}[\;] : g'$, for some environment Ξ' and some group g', can be derived using the following typing rule for the hole:

$$\frac{\Xi \subseteq \Xi''}{\Xi'' \vdash [\;] : g}.$$

In this case we say that Ξ' and g' are *compatible* with $\mathcal{C}[\;]^{\Xi,g}$. Notice that this implies $\Xi' \subseteq \Xi''$.

To simplify the notation, we will sometimes omit the superscript decoration of contexts agreeing with (Ξ, g) when it is obvious from the surrounding text.

Definition 4.3 (Reduction barbed congruence).

1 We say that a relation $\mathcal{R}^{\Xi,g}$ is *contextual* if $P\,\mathcal{R}^{\Xi,g}\,Q$ implies that for all contexts $\mathcal{C}[\ ]^{\Xi,g}$ the relation $\mathcal{C}[P]\,\mathcal{R}^{\Xi',g'}\,\mathcal{C}[Q]$ holds for any (Ξ',g') compatible with $\mathcal{C}[\ ]^{\Xi,g}$.

2 *Reduction barbed congruence* $\cong^{\Xi,g}$ is the maximal contextual $(\Xi,\ g)$-equivalence relation that, when restricted to closed processes, is reduction-closed and barb-preserving.

It is worth noting that type constraints affect congruence. Similar phenomena occur for the π-calculus, see Sangiorgi and Walker (2001). For example, assume $\Xi \vdash P\!:\!g$ and $n\!:\!g$, $g\!:\!G \in \Xi$ for some n, g, G such that $\mathscr{E}(G) = \emptyset$ and $\mathscr{C}(G) = \emptyset$. Assume also that $n \notin AN(P)$. Then $(\nu n\!:\!g)n[P] \cong^{\Xi,g'} \mathsf{0}$ because:

— n is unknown to the external world;

— no ambient occurring in P can cross n because $n \notin AN(P)$;

— n cannot move since $\mathscr{C}(G) = \emptyset$;

— n cannot send processes because $\mathscr{E}(G) = \emptyset$.

So no interaction is possible between P and the external world (this property will be formalised by Theorem 4.23(6) below). As another example, take

$$\Xi = \{g : \mathsf{gr}(\{g'\}, \emptyset, \emptyset, \mathsf{shh})\}; \{n : g, x : g \to g\}\,.$$

Now x is Ξ-ghost since in any extension of Ξ there can be no closed message of type $g \to g$ (see the remark after Definition 3.4). All contexts that agree with Ξ, g' and close $n[x]$ must be of the form $\mathcal{C}_1[(x : g \to g)\mathcal{C}_2[\ ]^{\Xi,g'}]^{\Xi',g''}$. Since there is no closed message of type $g \to g$, no reduction inside $\mathcal{C}_2[n[x]]^{\Xi,g'}$ will ever be possible. So any subterm of $\mathcal{C}_2[n[x]]^{\Xi,g'}$ can be equivalently replaced by $\mathsf{0}$.

A general result is that all Ξ-ghost processes are 'dead', in the sense that they are (Ξ, g)-reduction barbed congruent to the process $\mathsf{0}$.

Lemma 4.4. If $\Xi \vdash P : g$ and P is a Ξ-ghost process, then $P \cong^{\Xi,g} \mathsf{0}$.

Proof. Note first that the property of being Ξ-ghost is invariant under structural equivalence. Take the relation $\mathcal{R}^{\Xi,g}$ defined by

$$P\,\mathcal{R}^{\Xi,g}\,\mathsf{0} \quad \text{if } P \text{ is } \Xi\text{-ghost}\,,$$

and let $\asymp^{\Xi,g}$ denote the least equivalence relation induced by the closure of $\mathcal{R}^{\Xi,g}$ under context formation and structural equivalence. We will show that $\asymp^{\Xi,g}$ when restricted to closed processes is barb-preserving and reduction-closed.

By definition, a Ξ-ghost process P contains at least one Ξ-ghost variable – let it be x, and let $x : W \in \Xi$ where x is Ξ-ghost. Hence all contexts that agree with Ξ and g and close P must be of the form

$$\mathcal{C}_1[(x : W)\mathcal{C}_2[\ ]^{\Xi,g}]^{\Xi',g'}.$$

If $\mathcal{C}_1[(x : W)\mathcal{C}_2[P]^{\Xi,g}]^{\Xi',g'} \downarrow_n^{\Xi'',g''}$ holds, then $\mathcal{C}_1[(x : W)\mathcal{C}_2[\mathsf{0}]^{\Xi,g}]^{\Xi',g'} \downarrow_n^{\Xi'',g''}$ also holds, and *vice versa*, as can be verified immediately.

To prove reduction closure, assume $\Xi_0 \vdash \mathcal{C}_1[(x : W)\mathcal{C}_2[P]^{\Xi,g}]^{\Xi',g'} : g_0$. We prove that if $\mathcal{C}_1[(x : W)\mathcal{C}_2[P]^{\Xi,g}]^{\Xi',g'} \to R$, then $\mathcal{C}_1[(x : W)\mathcal{C}_2[\mathsf{0}]^{\Xi,g}]^{\Xi',g'} \to R'$ for some R' such that $R \asymp^{\Xi_0,g_0} R'$.

First note that since $\asymp^{\Xi,g}$ is closed under structural equivalence, we can ignore the rule (R-$\equiv$) and can assume that the $\rightarrow$ reduction step has been obtained by one of the basic reduction rules (R-in), (R-out), (R-to) or (R-comm) combined with the structural rules other than (R-$\equiv$). Note also that $(x:W)\mathcal{C}_2[P]^{\Xi,g}$ is either disjoint from the reduced redex or is a proper subterm of it.

Now there are only two possible cases:

1 The reduction has been obtained by (R-in), (R-out), (R-to) or (R-comm) without affecting $\mathcal{C}_2[P]^{\Xi,g}$. In this case we have $R = \mathcal{C}'_1[(x:W)\mathcal{C}_2[P]^{\Xi,g}]^{\Xi',g'}$. Hence

$$\mathcal{C}_1[(x:W)\,\mathcal{C}_2[\mathbf{0}]^{\Xi,g}]^{\Xi',g'} \;\rightarrow\; \mathcal{C}'_1[(x:W)\,\mathcal{C}_2[\mathbf{0}]^{\Xi,g}]^{\Xi',g'} \asymp^{\Xi_0,g_0} R,$$

and we are done.

2 The reduction is obtained by an application of rule (R-comm) involving an input variable that occurs in $\mathcal{C}_2[P]^{\Xi,g}$. Note that it is not possible that the communication is performed using x, since we should have

$$\mathcal{C}_1[(x:W)\,\mathcal{C}_2[P]^{\Xi,g}]^{\Xi',g'} = \mathcal{C}_3[(x:W)\,\mathcal{C}_2[P]^{\Xi,g} \mid <C> S]^{\Xi',g'},$$

and this would imply $\Xi' \vdash C:W$ for some closed capability C. But this is impossible since W is Γ-ghost (where $\Xi = \Gamma;\Delta$) by hypothesis, so it is Γ'-ghost (where $\Xi' = \Gamma';\Delta'$) by Proposition 3.5. So we have $R = \mathcal{C}'_1[(x:W)\,\mathcal{C}'_2[P\{y := M\}]^{\Xi,g}]^{\Xi',g'}$ where $y \neq x$ and $\mathcal{C}'_2[\;]^{\Xi,g} = \mathcal{C}_2[\;]^{\Xi,g}\{y := M\}$. Obviously, $P\{y := M\}$ is Ξ-ghost also. On the other hand, with the null process we have

$$\mathcal{C}_1[(x:W)\mathcal{C}_2[\mathbf{0}]^{\Xi,g}]^{\Xi',g'} \;\rightarrow\; \mathcal{C}'_1[(x:W)\mathcal{C}'_2[\mathbf{0}]^{\Xi,g}]^{\Xi',g'},$$

and we are done. □

As usual, in order to study reduction barbed congruences, we introduce a labelled transition system (LTS). Our labelled transitions have the form

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

where:

— P is a closed process;
— the judgment $\Xi \vdash P:g$ holds;
— the *label* α encodes the transition of the process;
— the *outcome* O can be either a *concretion*, that is, a partial derivative, which needs a contribution from the surrounding context to be completed, or a process.

Figure 6 defines labels and concretions. In $(\nu\widetilde{p:g})\langle\!\langle P\rangle\!\rangle Q$ the process P represents the moving part and the process Q represents the rest of the system not affected by the movement. In $(\nu\widetilde{p:g})\langle\!\langle M\rangle\!\rangle P$, 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 the shared private names.

$$\begin{array}{llll}
\textit{Labels} & \alpha & ::= & \tau \mid \mathsf{in}\ n \mid \mathsf{out}\ n \mid \mathsf{to}\ n \mid [\mathsf{in}\ n] \mid [\mathsf{out}\ n] \mid [\mathsf{to}\ n] \mid \overline{\mathsf{in}\ n} \mid \overline{\mathsf{to}\ n} \mid \langle M\rangle \mid \langle - \rangle \\
\textit{Concretions} & K & ::= & (\nu\widetilde{p:g})\langle\!\langle P\rangle\!\rangle Q \mid (\nu\widetilde{p:g})\langle\!\langle M\rangle\!\rangle P \\
\textit{Outcomes} & O & ::= & P \mid K
\end{array}$$

Fig. 6. Labels, concretions and outcomes.

$$\begin{array}{c}
P \equiv P', Q \equiv Q' \implies \langle\!\langle P\rangle\!\rangle Q \ \equiv\ \langle\!\langle P'\rangle\!\rangle Q' \qquad P \equiv P' \implies \langle\!\langle M\rangle\!\rangle P \ \equiv\ \langle\!\langle M\rangle\!\rangle P' \\
O \equiv O' \implies (\nu n\!:\!g)O \ \equiv\ (\nu n\!:\!g)O' \qquad (\nu n\!:\!g)(\nu m\!:\!g')O \ \equiv\ (\nu m\!:\!g')(\nu n\!:\!g)O \\
(\nu n\!:\!g)\langle\!\langle P\rangle\!\rangle Q \ \equiv\ \langle\!\langle(\nu n\!:\!g)P\rangle\!\rangle Q \ \text{ if } n \notin AN(Q) \\
(\nu n\!:\!g)\langle\!\langle M\rangle\!\rangle P \ \equiv\ \langle\!\langle M\rangle\!\rangle(\nu n\!:\!g)P \ \text{ if } n \notin AN(M) \\
(\nu n\!:\!g)\langle\!\langle P\rangle\!\rangle Q \ \equiv\ \langle\!\langle P\rangle\!\rangle(\nu n\!:\!g)Q \ \text{ if } n \notin AN(P)
\end{array}$$

Fig. 7. Structural congruence for concretions.

$$\textsc{(Cap-In-Out)}\quad \frac{C \in \{\mathsf{in}\ n, \mathsf{out}\ n\}}{C.P \xrightarrow{C}_{\Xi,g} P} \qquad\qquad \textsc{(Cap-To)}\quad \frac{}{\mathsf{to}\ n\,.\,P \xrightarrow{\mathsf{to}\ n}_{\Xi,g} \langle\!\langle P\rangle\!\rangle 0}$$

$$\textsc{(Path)}\quad \frac{C_1.(C_2.P) \xrightarrow{\alpha}_{\Xi,g} O}{(C_1\,.\,C_2)\,.\,P \xrightarrow{\alpha}_{\Xi,g} O} \qquad\qquad \textsc{(In-Out)}\quad \frac{P \xrightarrow{C}_{\Xi,g'} P' \quad C \in \{\mathsf{in}\ n, \mathsf{out}\ n\}}{m[P] \xrightarrow{[C]}_{\Xi,g} \langle\!\langle m[P']\rangle\!\rangle 0}$$

$$\textsc{(To)}\quad \frac{P \xrightarrow{\mathsf{to}\ n}_{\Xi,g'} (\nu\widetilde{p:g})\langle\!\langle P_1\rangle\!\rangle P_2 \quad m \notin \tilde{p}}{m[P] \xrightarrow{[\mathsf{to}\ n]}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle P_1\rangle\!\rangle m[P_2]} \qquad\qquad \textsc{(Co-In-To)}\quad \frac{C \in \{\overline{\mathsf{in}\ n}, \overline{\mathsf{to}\ n}\}}{n[P] \xrightarrow{C}_{\Xi,g} \langle\!\langle P\rangle\!\rangle 0}$$

$$\textsc{(Input)}\quad \frac{\Xi \vdash M : W}{(x\!:\!W)P \xrightarrow{\langle M\rangle}_{\Xi,g} P\{x := M\}} \qquad\qquad \textsc{(Output)}\quad \frac{}{\langle M\rangle P \xrightarrow{\langle - \rangle}_{\Xi,g} \langle\!\langle M\rangle\!\rangle P}$$

Fig. 8. Commitments: visible transitions.

Figures 8, 9 and 10 define the LTS, where $AN(\alpha)$ ($AN(P)$) is the set of free ambient names that occur in α (P). Following a common practice, we omit the symmetric counterparts of the rules dealing with the parallel operator.

To avoid having to write tedious side conditions in transition rules, we decree that a transition $P \xrightarrow{\alpha}_{\Xi,g} O$ is defined only for processes P such that $\Xi \vdash P : g$.

We will also use the following standard conventions:

— If O is the concretion $(\nu\widetilde{p:g})\langle\!\langle P\rangle\!\rangle Q$ and $AN(R) \cap \tilde{p} = \varnothing$, then

$$O \mid R = (\nu\widetilde{p:g})\langle\!\langle P\rangle\!\rangle(Q \mid R).$$

(τ-Enter/τ-To)

$$\frac{P \xrightarrow{[C]}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle P_1\rangle\!\rangle P_2 \quad Q \xrightarrow{\overline{C}}_{\Xi,g} (\nu\widetilde{q:g'})\langle\!\langle Q_1\rangle\!\rangle Q_2 \quad \begin{array}{c} C\in\{\mathsf{in}\,n,\mathsf{to}\,n\} \\ \tilde{p}\cap\tilde{q} = AN(P)\cap\tilde{q} = AN(Q)\cap\tilde{p} = \varnothing \end{array}}{P \mid Q \xrightarrow{\tau}_{\Xi,g} (\nu\widetilde{p:g},\widetilde{q:g'})(n[P_1 \mid Q_1] \mid P_2 \mid Q_2)}$$

(τ-Exit)

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

(τ-Exchange)

$$\frac{P \xrightarrow{\langle M\rangle}_{\Xi,g} P_1 \quad Q \xrightarrow{\langle - \rangle}_{\Xi,g} (\nu\,\widetilde{q:g})\langle\!\langle M\rangle\!\rangle Q_1 \quad AN(P)\cap\tilde{q} = \varnothing}{P \mid Q \xrightarrow{\tau}_{\Xi,g} (\nu\,\widetilde{q:g})(P_1 \mid Q_1)}$$

Fig. 9. Commitments: τ transitions.

(Par)

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

(Res)

$$\frac{P \xrightarrow{\alpha}_{\Xi,g} O \quad n\notin AN(\alpha) \quad \Xi = \Xi' \cup \{n:g'\}}{(\nu n:g')P \xrightarrow{\alpha}_{\Xi',g} (\nu n:g')O}$$

(τ-Amb)

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

(Repl)

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

Fig. 10. Commitments: structural transitions.

— If O is the concretion $(\nu\widetilde{p:g})\langle\!\langle M\rangle\!\rangle P$ and $AN(R)\cap\tilde{p} = \varnothing$, then

$$O \mid R = (\nu\widetilde{p:g})\langle\!\langle M\rangle\!\rangle(P \mid R).$$

All rules are quite standard – see Levi and Sangiorgi (2003), Merro and Hennessy (2002) and Bugliesi *et al.* (2005). Note that rule (τ-Exchange) allows the communication of bound names. Note also that the assumption that processes are well typed in Ξ has implicit effects on the possible transitions. For instance, in the rule (In-Out), the fact that P has type g' with respect to Ξ implies that $n:g'', g':G \in \Xi$ for some g'', G such that $g'' \in \mathscr{C}(G)$, since otherwise P could not be typed, and then the transition would not be possible either.

Figure 7 extends the structural congruence to concretions in the standard way. We identify concretions up to α-renaming of bound variables and bound names.

We start with a lemma that relates the structure of processes to the labels of visible transitions.

Lemma 4.5.

1 If $P \xrightarrow{C}_{\Xi,g} P'$ with $C \in \{\mathsf{in}\ n, \mathsf{out}\ n\}$, then we have $P \equiv (\nu\widetilde{p:g})(C.Q \,|\, R)$ and $P' \equiv (\nu\widetilde{p:g})(Q \,|\, R)$ for some $\tilde{p}$, $\tilde{g}$, Q and R such that $n \notin \tilde{p}$.

2 If $P \xrightarrow{\mathsf{to}\ n}_{\Xi,g} O$, then we have $P \equiv (\nu\widetilde{p:g})(\mathsf{to}\ n.Q \,|\, R)$ and $O \equiv (\nu\widetilde{p:g})\langle\!\langle Q\rangle\!\rangle R$ for some $\tilde{p}$, $\tilde{g}$, Q and R such that $n \notin \tilde{p}$.

3 If $P \xrightarrow{[C]}_{\Xi,g} O$ with $C \in \{\mathsf{in}\ n, \mathsf{out}\ n\}$, then we have $P \equiv (\nu\widetilde{p:g})(m[C.Q \,|\, R] \,|\, S)$ and $O \equiv (\nu\widetilde{p:g})\langle\!\langle m[Q \,|\, R]\rangle\!\rangle S$ for some $\tilde{p}$, $\tilde{g}$, m, Q, R and S such that $n \notin \tilde{p}$.

4 If $P \xrightarrow{[\mathsf{to}\ n]}_{\Xi,g} O$, then $P \equiv (\nu\widetilde{p:g})(m[\mathsf{to}\ n.Q \,|\, R] \,|\, S)$ and $O \equiv (\nu\widetilde{p:g})\langle\!\langle Q\rangle\!\rangle(m[R] \,|\, S)$ for some $\tilde{p}$, $\tilde{g}$, m, Q, R and S such that $n \notin \tilde{p}$.

5 If $P \xrightarrow{C}_{\Xi,g} O$ with $C \in \{\overline{\mathsf{in}\ n}, \overline{\mathsf{to}\ n}\}$, then we have $P \equiv (\nu\widetilde{p:g})(n[Q] \,|\, R)$ and $O \equiv (\nu\widetilde{p:g})\langle\!\langle Q\rangle\!\rangle R$ for some $\tilde{p}$, $\tilde{g}$, Q, and R such that $n \notin \tilde{p}$.

6 If $P \xrightarrow{\langle M\rangle}_{\Xi,g} P'$, then we have

$$\begin{aligned} \Xi &\vdash M : W \\ P &\equiv (\nu\widetilde{p:g})((x:W)Q \,|\, R) \\ P' &\equiv (\nu\widetilde{p:g})(Q\{x := M\} \,|\, R) \end{aligned}$$

for some $\tilde{p}$, $\tilde{g}$, x, W, Q and R such that $\tilde{p} \cap AN(M) = \emptyset$.

7 If $P \xrightarrow{\langle - \rangle}_{\Xi,g} O$, then $P \equiv (\nu\widetilde{p:g})(\langle M\rangle Q \,|\, R)$ and $O \equiv (\nu\widetilde{p:g})\langle\!\langle M\rangle\!\rangle(Q \,|\, R)$ for some $\tilde{p}$, $\tilde{g}$, M, Q and R.

Proof. The proof by induction on the transition rules is quite standard. We only note that if $P \xrightarrow{\alpha}_{\Xi,g} O$ by rule (Repl), we have

$$\begin{gathered} P \equiv\, !\pi\,.\,Q \equiv \pi\,.\,Q \,|\, P \\ O \equiv O' \,|\, P \\ \pi\,.\,Q \xrightarrow{\alpha}_{\Xi,g} O' \end{gathered}$$

for some π, Q, and O'. Note that this implies that α has one of the following forms: $\mathsf{in}\ n$, $\mathsf{out}\ n$, $\mathsf{to}\ n$, $\langle M\rangle$, $\langle - \rangle$. □

The next lemma shows that structurally congruent processes have the same labelled transitions.

Lemma 4.6. If $P \xrightarrow{\alpha}_{\Xi,g} O$ and $P \equiv Q$, there exists an O' such that $Q \xrightarrow{\alpha}_{\Xi,g} O'$ and $O \equiv O'$.

Proof. We prove the two statements simultaneously by induction on the derivation of $P \equiv Q$:

1 If $P \xrightarrow{\alpha}_{\Xi,g} O$ and $P \equiv Q$, there exists an O' such that $Q \xrightarrow{\alpha}_{\Xi,g} O'$ and $O \equiv O'$.

2 If $P \xrightarrow{\alpha}_{\Xi,g} O$ and $Q \equiv P$, there exists an O' such that $Q \xrightarrow{\alpha}_{\Xi,g} O'$ and $O \equiv O'$.

In this way we automatically prove the base case of the symmetry law. The proofs of the many other base cases are all standard, so we just consider one subcase of point (1), the other cases being analogous. Let P be $(\nu n:g)(R \,|\, S)$, and Q be $(\nu n:g)R \,|\, S$, and

$n \notin AN(S)$. Then the labelled transition must have been derived using rule (RES). Let us further consider the subcase in which $\alpha = \tau$ and $\Xi' = \Xi, n:g$ and

$$R \mid S \xrightarrow{\tau}_{\Xi',g} (\nu\widetilde{p:g},\widetilde{q:g'})(m[R_1 \mid S_1] \mid R_2 \mid S_2)$$

follows from an application of rule (τ-To) to

$$R \xrightarrow{[\mathsf{to}\ m]}_{\Xi',g} (\nu\widetilde{p:g})\langle\!\langle R_1\rangle\!\rangle R_2$$

and

$$S \xrightarrow{\overline{\mathsf{to}\ m}}_{\Xi',g} (\nu\widetilde{q:g'})\langle\!\langle S_1\rangle\!\rangle S_2$$

where

$$\tilde{p} \cap \tilde{q} = AN(R) \cap \{\tilde{q}\} = AN(S) \cap \{\tilde{p}\} = \emptyset\,.$$

Note that $S \xrightarrow{\overline{\mathsf{to}\ m}}_{\Xi',g}$ implies $m \in AN(S)$ by Lemma 4.5(5), so $m \neq n$. By applying the rule (RES) to $R \xrightarrow{[\mathsf{to}\ m]}_{\Xi',g} (\nu\widetilde{p:g})\langle\!\langle R_1\rangle\!\rangle R_2$, we get

$$(\nu n:g)R \xrightarrow{[\mathsf{to}\ m]}_{\Xi,g} (\nu n:g)(\nu\widetilde{p:g})\langle\!\langle R_1\rangle\!\rangle R_2\,.$$

Hence, as $n \notin AN(S)$, we can conclude by rule (τ-To) that

$$(\nu n:g)R \mid S \xrightarrow{\tau}_{\Xi,g} (\nu n:g)(\nu\widetilde{p:g},\widetilde{q:g'})(m[R_1 \mid S_1] \mid R_2 \mid S_2)\,,$$

where the term on the right-hand side of the arrow is, with reference to 1 and 2, an O' coinciding with O.

The induction step is trivial. □

Using the previous lemmas, we can check that labelled transitions agree with reductions when restricted to well-typed processes in the current environment.

Theorem 4.7.

1 If $P \xrightarrow{\tau}_{\Xi,g} Q$, then $P \to Q$.

2 If $P \to Q$ and $\Xi \vdash P : g$ for some g, then $P \xrightarrow{\tau}_{\Xi,g} Q'$ for some $Q' \equiv Q$.

Proof.

(1) We use induction on $\xrightarrow{\tau}_{\Xi,g}$. The basic cases are the τ transitions in Figure 9. We consider the rule (τ-To). In this case P is $R \mid S$, and

$$R \xrightarrow{[\mathsf{to}\ n]}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle R_1\rangle\!\rangle R_2$$

and

$$S \xrightarrow{\overline{\mathsf{to}\ n}}_{\Xi,g} (\nu\widetilde{q:g'})\langle\!\langle S_1\rangle\!\rangle S_2$$

where

$$\tilde{p} \cap \tilde{q} = AN(R) \cap \{\tilde{q}\} = AN(S) \cap \{\tilde{p}\} = \emptyset$$

and Q is

$$(\nu\widetilde{p:g},\widetilde{q:g'})(n[R_1 \mid S_1] \mid R_2 \mid S_2)\,.$$

By Lemma 4.5(4 and 5), we have, for some m, T and U, that

$$R \equiv (\nu\widetilde{p:g})(m[\mathsf{to}\,n.R_1 \mid T] \mid U),$$

so $R_2 \equiv m[T] \mid U$ and $S \equiv (\nu\widetilde{q:g'})(n[S_1] \mid S_2)$.
By rule (R-to), we get

$$P \;\to\; (\nu\widetilde{p:g},\widetilde{q:g'})(n[R_1 \mid S_1] \mid m[T] \mid U \mid S_2).$$

The induction case is trivial.

(2) We use induction on $\to$. The only interesting case is rule (R-$\equiv$):

$$P \equiv P',\;\; P \;\to\; Q,\;\; Q \equiv Q' \quad\Rightarrow\quad P' \;\to\; Q'.$$

By induction, $P \;\to\; Q$ implies $P \;\xrightarrow{\tau}_{\Xi,g} Q''$ for some $Q'' \equiv Q$. By Lemma 4.6, there is Q''' such that $P' \;\xrightarrow{\tau}_{\Xi,g} Q'''$ and $Q''' \equiv Q'' \equiv Q \equiv Q'$, and this concludes the proof. □

By comparing the definition of observability with rule (Co-In-To) one can easily conclude that a name n is observable if and only if at least one of the two actions $\overline{\mathsf{in}\,n}$ or $\overline{\mathsf{to}\,n}$ can be performed.

Proposition 4.8. $P \downarrow_n^{\Xi,g}$ if and only if $P \;\xrightarrow{\alpha}_{\Xi,g} (\nu\widetilde{m:g})\langle\!\langle Q\rangle\!\rangle R$ 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 reduction barbed congruence, we follow Merro and Hennessy (2002) and Bugliesi *et al.* (2005) in defining higher-order transitions that allow us to get rid of the transitions whose outcome is a concretion rather than a process (see Figure 11). In these transitions we use labels of the form $\alpha;Q$ or $\alpha;Q;R$, and thus transitions of the form

$$P \;\xrightarrow{\alpha;Q}_{\Xi,g} P' \text{ or } P \;\xrightarrow{\alpha;Q;R}_{\Xi,g} P'$$

where the processes Q, R (which are required to be well typed from Ξ with the proper group type) represent the contribution needed from the context in order to build from P a well-formed term in which action α can fire.

We decree that a transition $P \;\xrightarrow{\alpha;Q}_{\Xi,g} P'$ or $P \;\xrightarrow{\alpha;Q;R}_{\Xi,g} P'$ is defined only for processes P such that $\Xi \vdash P : g$. It is easy to see that this implies $\Xi \vdash P' : g'$, where $g' = g$ for all rules but (HO To) and (HO Out), while g' is in general different from g in the case of these two rules.

We use Λ to denote the set of all the first-order labels occurring in the rules shown in Figure 8 and whose outcome is a process, plus τ, plus the higher-order labels occurring in the transitions in Figure 11. Therefore, when we write

$$P \;\xrightarrow{\lambda}_{\Xi,g} Q,$$

we mean that this is either a first-order transition in Figure 8 whose outcome is a process, or a τ-transition in Figure 9, or a structural transition in Figure 10 applied to a first-order transition, or a higher-order transition in Figure 11.

(HO OUTPUT)

$$\frac{P \xrightarrow{\langle - \rangle}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle M \rangle\!\rangle P' \quad \Xi, x:T(G) \vdash Q:g \quad \begin{array}{c} \Xi = \Gamma, g:G;\, \Delta \\ FV(Q) \subseteq \{x\} \\ AN(Q) \cap \tilde{p} = \varnothing \end{array}}{P \xrightarrow{\langle - \rangle ; Q}_{\Xi,g} (\nu\widetilde{p:g})(P' \mid Q\{x := M\})}$$

(HO TO)

$$\frac{P \xrightarrow{\mathsf{to}\ n}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle P_1 \rangle\!\rangle P_2 \quad \Xi \vdash Q_1 : g' \quad \Xi \vdash Q_2 : g \quad \begin{array}{c} \Xi = \Gamma;\, \Delta, m:g, n:g' \\ AN(Q_1 \mid Q_2) \cap \tilde{p} = \varnothing \end{array}}{P \xrightarrow{\mathsf{to}\ n; Q_1; m[Q_2]}_{\Xi,g} (\nu\widetilde{p:g})(n[P_1 \mid Q_1] \mid m[P_2 \mid Q_2])}$$

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

$$\frac{P \xrightarrow{\alpha}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle P_1 \rangle\!\rangle P_2 \quad \alpha \in \{[\mathsf{in}\ n],\ [\mathsf{to}\ n],\ \overline{\mathsf{in}\ n},\ \overline{\mathsf{to}\ n}\} \quad \Xi \vdash Q : g' \quad \begin{array}{c} \Xi = \Gamma;\, \Delta, n:g' \\ AN(Q) \cap \tilde{p} = \varnothing \end{array}}{P \xrightarrow{\alpha; Q}_{\Xi,g} (\nu\widetilde{p:g})(n[P_1 \mid Q] \mid P_2)}$$

(HO OUT)

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

Fig. 11. Commitments: higher-order transitions.

With $\lambda \in \Lambda$ we decree that:

(i) $\Longrightarrow_{\Xi,g}$ denotes $\xrightarrow{\tau}{}^{*}_{\Xi,g}$;

(ii) $\overset{\lambda}{\Longrightarrow}_{\Xi,g}$ denotes $\Longrightarrow_{\Xi,g} \xrightarrow{\lambda}_{\Xi,g} \Longrightarrow_{\Xi,g'}$ for some g';

(iii) $\overset{\hat{\lambda}}{\Longrightarrow}_{\Xi,g}$ denotes $\Longrightarrow_{\Xi,g}$ if $\lambda = \tau$ and $\overset{\lambda}{\Longrightarrow}_{\Xi,g}$ otherwise.

Note that in point (ii), g differs from g' only when λ is of the form $\mathsf{to}\ n; Q_1; m[Q_2]$ or $[\mathsf{out}\ n]; Q$.

Limiting labelled transitions to be from processes to processes means the standard definition of labelled bisimulation adapts smoothly to our calculus.

Definition 4.9 (Labelled bisimilarity with respect to environments).

1 A symmetric (Ξ, g)-relation $\mathcal{R}^{\Xi,g}$ over closed processes is a (Ξ, g)-*labelled bisimulation* if $P\,\mathcal{R}^{\Xi,g}\,Q$ and $P \xrightarrow{\lambda}_{\Xi',g} P'$, for some $\Xi' \supseteq \Xi$ and some P' such that $\Xi' \vdash P' : g'$, imply that there exists Q' such that $Q \overset{\hat{\lambda}}{\Longrightarrow}_{\Xi',g} Q'$ and $\Xi' \vdash Q' : g'$ and $P'\,\mathcal{R}^{\Xi,g'}\,Q'$.

2 Two closed processes P and Q are (Ξ, g)-*labelled bisimilar*, written $P \approx^{\Xi,g} Q$, if $P\,\mathcal{R}^{\Xi,g}\,Q$ for some (Ξ, g)-labelled bisimulation.

Note that the possibility of extending the environments is crucial in the previous definition since otherwise we would get, for example, $(x:g) \approx^{\varnothing;\varnothing,g} \mathbf{0}$.

It is easy to verify that $\equiv$ is a (Ξ, g)-labelled bisimulation on closed processes typable with g from Ξ.

Lemma 4.10.

1 If $P \equiv Q$ and $P \xrightarrow{\lambda}_{\Xi,g} P'$, then $Q \xrightarrow{\lambda}_{\Xi,g} Q'$ for some $Q' \equiv P'$.

2 $\Xi \vdash P : g$ and $P \equiv Q$ imply $P \approx^{\Xi,g} Q$.

Proof.

(1) When λ is first order, this is Lemma 4.6. If λ is higher order, all the proofs are similar. Take, for example, $\lambda = \mathsf{to}\, n ; R_1 ; m[R_2]$. In this case, $P \xrightarrow{\lambda}_{\Xi,g} P'$ implies

$$P \xrightarrow{\mathsf{to}\, n}_{\Xi,g} (\nu \widetilde{p : g})\langle\langle P_1 \rangle\rangle P_2$$

and

$$P' \equiv (\nu \widetilde{p : g})(n[P_1 \mid R_1] \mid m[P_2 \mid R_2]) .$$

By Lemmas 4.5(2) and 4.6, we get

$$Q \xrightarrow{\mathsf{to}\, n}_{\Xi,g} (\nu \widetilde{q : g'})\langle\langle Q_1 \rangle\rangle Q_2$$

and

$$(\nu \widetilde{p : g})\langle\langle P_1 \rangle\rangle P_2 \equiv (\nu \widetilde{q : g'})\langle\langle Q_1 \rangle\rangle Q_2 .$$

Then by rule (HO To), we get

$$Q \xrightarrow{\lambda}_{\Xi,g} (\nu \widetilde{q : g'})(n[Q_1 \mid R_1] \mid m[Q_2 \mid R_2]) .$$

(2) This part follows from (1). □

In order to extend bisimilarity to open processes, we need to consider Ξ-ghost variables. Lemma 4.4 proves that all Ξ-ghost processes are (Ξ, g)-reduction barbed congruent (for a suitable g) to the $\mathbf{0}$ process. This is also substantiated by the absence of transitions from processes that are abstractions with respect to ghost variables, as proved in the following lemma.

Lemma 4.11. If x is Ξ-ghost, $\Xi = \Xi', x : W$, and $\Xi \vdash P : g$ with $FV(P) \subseteq \{x\}$, then there is no λ such that $(x : W)P \xrightarrow{\lambda}_{\Xi,g}$.

Proof. By inspection of the transition rules of Figures 8, 9, 10 and 11, the only possible transitions would have label $\langle C \rangle$ for some closed capability C such that $\Xi' \vdash C : W$. But, by the definition of a Ξ-ghost variable, no such C can exist. □

Definition 4.12 (Full bisimilarity with respect to environments and groups). Two processes P and Q such that $\Xi \vdash P : g$ and $\Xi \vdash Q : g$ are *fully* (Ξ, g)-bisimilar, written $P \approx_{\mathrm{c}}^{\Xi,g} Q$, if one of the following conditions holds:

1 Both P and Q are Ξ-ghost.

2 Both P and Q are not Ξ-ghost and $\mathbf{s}(P) \approx^{\Xi',g} \mathbf{s}(Q)$ for every complete environment Ξ' such that $\Xi' \supseteq \Xi$ and every Ξ'-closing substitution $\mathbf{s}$.

3 P is Ξ-ghost, Q is not Ξ-ghost and $\mathbf{s}(Q) \approx^{\Xi,g} \mathsf{0}$ for every complete environment Ξ' such that $\Xi' \supseteq \Xi$ and every Ξ'-closing substitution $\mathbf{s}$.

4 Q is Ξ-ghost, P is not Ξ-ghost and $\mathbf{s}(P) \approx^{\Xi,g} \mathsf{0}$ for every complete environment Ξ' such that $\Xi' \supseteq \Xi$ and every Ξ'-closing substitution $\mathbf{s}$.

Since $\Xi' \supseteq \Xi$ implies that more variables can be ghost and that we quantify over fewer substitutions, we have the following proposition.

Proposition 4.13. If $P \approx_{\mathrm{c}}^{\Xi,g} Q$ and $\Xi' \supseteq \Xi$, then $P \approx_{\mathrm{c}}^{\Xi',g} Q$.

The converse is false. For example, take $\Xi = \emptyset; \Delta$ and $\Xi' = \{g : \mathsf{gr}(\{g'\}, \emptyset, \emptyset, \mathsf{shh})\}; \Delta$, where $\Delta = \{x : g \to g, n : g\}$. Then $n[x] \not\approx_{\mathrm{c}}^{\Xi,g'} \mathsf{0}$ but $n[x] \approx_{\mathrm{c}}^{\Xi',g'} \mathsf{0}$, as x is not Ξ-ghost but Ξ'-ghost.

As an immediate consequence of Lemma 4.11, we have that if x is $\Xi, x : W$-ghost, then $(x : W)P$ is fully bisimilar to $\mathsf{0}$ for all P such that $(x : W)P$ is a well-formed process.

Corollary 4.14. If x is $\Xi, x : W$-ghost and $\Xi, x : W \vdash P : g$, then $(x : W)P \approx_{\mathrm{c}}^{\Xi,g} \mathsf{0}$.

It is easy to verify that full bisimilarity includes structural equivalence.

Lemma 4.15. $\Xi \vdash P : g$ and $P \equiv Q$ imply $P \approx_{\mathrm{c}}^{\Xi,g} Q$.

Proof. The statement follows by Lemma 4.10(2), Definition 4.12 and the fact that the property of being Ξ-ghost is preserved by $\equiv$. □

The following lemma will be used to prove that $\approx_{\mathrm{c}}^{\Xi,g}$ is closed under contexts.

Lemma 4.16.

1 If $P \xrightarrow{\lambda}_{\Xi,g} P'$ and $\lambda \notin \{\mathsf{to}\, n; Q_1; m[Q_2],\ [\mathsf{out}\, n]; Q\}$, then $P \mid R \xrightarrow{\lambda}_{\Xi,g} P' \mid R$ for all processes R such that $\Xi \vdash R : g$.

2 If $P \overset{\hat{\lambda}}{\Longrightarrow}_{\Xi,g} P'$ and $\lambda \notin \{\mathsf{to}\, n; Q_1; m[Q_2], [\mathsf{out}\, n]; Q\}$, then $P \mid R \overset{\hat{\lambda}}{\Longrightarrow}_{\Xi,g} P' \mid R$ for all processes R such that $\Xi \vdash R : g$.

3 If $P \xrightarrow{\mathsf{to}\, n; Q_1; m[R \mid Q_2]}_{\Xi,g} P'$, then $P \mid R \xrightarrow{\mathsf{to}\, n; Q_1; m[Q_2]}_{\Xi,g} P'$ for all processes R such that $\Xi \vdash R : g$.

4 If $P \xrightarrow{[\mathsf{out}\ n]; Q \mid R}_{\Xi,g} P'$, then $P \mid R \xrightarrow{[\mathsf{out}\ n]; Q}_{\Xi,g} P'$ for all processes R such that $\Xi \vdash R : g$.

5 If $P \xrightarrow{\langle - \rangle; Q}_{\Xi,g} P'$ and $R \xrightarrow{\langle M \rangle}_{\Xi,g} (\nu \widetilde{r : g})(Q\{x := M\} \mid R')$, then $P \mid R \xrightarrow{\tau}_{\Xi,g} Q'$ for some $Q' \equiv (\nu \widetilde{r : g})(P' \mid R')$.

6 If we have $P \xrightarrow{[C]; Q}_{\Xi,g} P'$ and $R \xrightarrow{\overline{C}}_{\Xi,g} (\nu \widetilde{r : g})\langle\!\langle Q \rangle\!\rangle R'$, and $C \in \{\mathsf{in}\, n, \mathsf{to}\, n\}$, then we have $P \mid R \xrightarrow{\tau}_{\Xi,g} Q'$ for some $Q' \equiv (\nu \widetilde{r : g})(P' \mid R')$.

7 If we have $P \xrightarrow{\overline{C}; Q}_{\Xi,g} P'$ and $R \xrightarrow{[C]}_{\Xi,g} (\nu \widetilde{r : g})\langle\!\langle Q \rangle\!\rangle R'$, and $C \in \{\mathsf{in}\, n, \mathsf{to}\, n\}$, then we have $P \mid R \xrightarrow{\tau}_{\Xi,g} Q'$ for some $Q' \equiv (\nu \widetilde{r : g})(P' \mid R')$.

8 If $P \xrightarrow{\lambda}_{\Xi,g} Q$ and $n \notin AN(\lambda)$, then $(\nu n : g')P \xrightarrow{\lambda}_{\Xi',g} (\nu n : g')Q$, where $\Xi = \Xi' \cup \{n : g'\}$.

Proof. In all cases, when the labels are higher order we derive the transitions of processes and their forms by inspection of the transition rules.

(1) If λ is a first-order label, the statement follows from rule (Par). Otherwise, $\lambda = \alpha;Q$ and either

$$P \xrightarrow{\alpha}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle M\rangle\!\rangle P_2 \quad \text{and} \quad P' \equiv (\nu\widetilde{p:g})(P_2 \mid Q\{x := M\})$$

or

$$P \xrightarrow{\alpha}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle P_1\rangle\!\rangle P_2 \quad \text{and} \quad P' \equiv (\nu\widetilde{p:g})(n[P_1 \mid Q] \mid P_2).$$

In the first case we have $P \mid R \xrightarrow{\alpha}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle M\rangle\!\rangle P_2 \mid R$ by rule (Par), so we conclude $P \mid R \xrightarrow{\lambda}_{\Xi,g} (\nu\widetilde{p:g})(P_2 \mid Q\{x := M\}) \mid R$ by rule (HO Output). And similarly in the second case.

(2) We have that $P \overset{\hat{\lambda}}{\Longrightarrow}_{\Xi,g} P'$ means either

$$P \Longrightarrow_{\Xi,g} P' \quad \text{if} \quad \lambda = \tau$$

or

$$P \Longrightarrow_{\Xi,g} Q \xrightarrow{\lambda}_{\Xi,g} Q' \Longrightarrow_{\Xi,g'} P' \quad \text{for some} \quad g', Q, Q'.$$

In the first case, (2) follows from rule (Par); and in the second case, it follows from rule (Par) and (1).

(3) We have $P \xrightarrow{\text{to } n}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle P_1\rangle\!\rangle P_2$ and $P' \equiv (\nu\widetilde{p:g})(n[P_1 \mid Q_1] \mid m[P_2 \mid R \mid Q_2])$. By rule (Par), we have $P \mid R \xrightarrow{\text{to } n}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle P_1\rangle\!\rangle P_2 \mid R$, so we conclude

$$P \mid R \xrightarrow{\text{to } n;Q_1;m[Q_2]}_{\Xi,g} (\nu\widetilde{p:g})(n[P_1 \mid Q_1] \mid m[P_2 \mid R \mid Q_2])$$

by rule (HO To).

(4) The proof of this is similar to the proof of (3).

(5) The proof of this is similar to the proof of (6).

(6) We have $P \xrightarrow{[C]}_{\Xi,g} (\nu\widetilde{p:g'})\langle\!\langle P_1\rangle\!\rangle P_2$ and $P' \equiv (\nu\widetilde{p:g'})(n[P_1 \mid Q] \mid P_2)$, so

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

by rule (τ-Enter) or (τ-To).

(7) The proof of this is similar to the proof of (6).

(8) The case of λ first-order is immediate by rule (Res). For higher-order labels we consider the case $\lambda = \text{to } q;Q_1;m[Q_2]$, since the other cases are similar. In this case

$$P \xrightarrow{\text{to } q}_{\Xi,g} (\nu\widetilde{p:g'})\langle\!\langle P_1\rangle\!\rangle P_2$$

and

$$Q \equiv (\nu\widetilde{p:g'})(q[P_1 \mid Q_1] \mid m[P_2 \mid Q_2]).$$

By rule (Res), we get

$$(\nu n:g)P \xrightarrow{\text{to } q}_{\Xi',g} (\nu n:g)(\nu\widetilde{p:g'})\langle\!\langle P_1\rangle\!\rangle P_2\,,$$

so we conclude

$$(\nu n:g)P \xrightarrow{\text{to } q;Q_1;m[Q_2]}_{\Xi',g} (\nu n:g)(\nu\widetilde{p:g'})(q[P_1 \mid Q_1] \mid m[P_2 \mid Q_2])$$

by rule (HO To). □

Following Merro and Hennessy (2002) and Bugliesi *et al.* (2005), we can show the closure under contexts of $\approx_{\mathrm{c}}^{\Xi,g}$.

Theorem 4.17. Full bisimilarity is contextual.

Proof. The proof is organised in the following three steps:

Step A Full bisimilarity is preserved by input prefixes.
Step B Full bisimilarity is preserved by capability and output prefixes and by parallel composition, ambient construction and restriction.
Step C Full bisimilarity is preserved by replication.

A For input prefixes, we show that if $P \approx_{\mathrm{c}}^{\Xi,g} Q$ and either $\Xi = \Xi', x:W$ or $x \notin \Xi = \Xi'$, then $(x:W)P \approx_{\mathrm{c}}^{\Xi',g} (x:W)Q$. We distinguish two cases depending on whether P and Q are Ξ-ghost or not.

If P and Q are both non-Ξ-ghost and we assume $P \approx_{\mathrm{c}}^{\Xi,g} Q$, then, by definition, $\mathbf{s}(P) \approx^{\Xi'',g} \mathbf{s}(Q)$ for any $\Xi'' \supseteq \Xi$ and any complete Ξ''-closing substitution $\mathbf{s}$. This implies $\mathbf{s}(P\{x := M\}) \approx^{\Xi'',g} \mathbf{s}(Q\{x := M\})$ for any complete $\Xi'' \supseteq \Xi$ and any Ξ''-closing substitution $\mathbf{s}$ and any x, M such that either $x:W \in \Xi$ or $x \notin \Xi$, and in both cases $\Xi'' \vdash M:W$ for some W. We conclude that $(x:W)P \approx_{\mathrm{c}}^{\Xi',g} (x:W)Q$.

Note that if a process R is Ξ-ghost and $\Xi \vdash R:g$, then either $(x:W)R$ is Ξ-ghost or x is a Ξ-ghost variable, so, by Corollary 4.14, we have $(x:W)R \approx^{\Xi',g} \mathbf{0}$. We conclude that if one between P and Q is Ξ-ghost and $P \approx_{\mathrm{c}}^{\Xi,g} Q$, then $(x:W)P \approx_{\mathrm{c}}^{\Xi',g} (x:W)Q$.

B Since we know that $\approx_{\mathrm{c}}^{\Xi,g}$ is preserved by input prefixes, we can consider $\approx^{\Xi,g}$. We define $\bowtie^{\Xi,g}$ as the *contextual* closure of $\approx^{\Xi,g}$ with respect to capability and output prefixes and by parallel composition, ambient construction and restriction, that is, as the least symmetric relation such that:

1 $\approx^{\Xi,g} \subseteq \bowtie^{\Xi,g}$.

2 $P \bowtie^{\Xi,g} Q$ and $\Xi \vdash C:g \rightarrow g'$ imply $C\,.\,P \bowtie^{\Xi,g'} C\,.\,Q$.

3 $P \bowtie^{\Xi,g} Q$ and $\Xi \vdash M:W$ and $\Xi \vdash g:\mathsf{gr}(\mathscr{S},\mathscr{C},\mathscr{E},W)$ imply $\langle M\rangle P \bowtie^{\Xi,g} \langle M\rangle Q$.

4 $P \bowtie^{\Xi,g} Q$ and $\Xi \vdash R:g$ imply $P \mid R \bowtie^{\Xi,g} Q \mid R$ and $R \mid P \bowtie^{\Xi,g} R \mid Q$.

5 $P \bowtie^{\Xi,g} Q$ and $n:g \in \Xi$ and $g:G \in \Xi$ and $g' \in \mathscr{S}(G)$ imply $n[P] \bowtie^{\Xi,g'} n[Q]$.

6 $P \bowtie^{\Xi,g} Q$ and $\Xi = \Gamma;\Delta, n:g'$ imply $(\nu n:g')P \bowtie^{\Gamma;\Delta,g} (\nu n:g')Q$.

Adapting the definition in Sangiorgi and Milner (1992), we say that $\bowtie^{\Xi,g}$ is a (Ξ,g)-*labelled bisimulation up to* $\equiv$ if it is symmetric and $P \bowtie^{\Xi,g} Q$ and $P \xrightarrow{\lambda}_{\Xi',g} P'$, for some $\Xi' \supseteq \Xi$ and some P', g' such that $\Xi' \vdash P':g'$, imply that there exist P'', Q', Q'' such that $Q \overset{\hat{\lambda}}{\Longrightarrow}_{\Xi',g} Q'$ and $\Xi' \vdash Q':g'$ and $P' \equiv P'' \bowtie^{\Xi',g'} Q'' \equiv Q'$. It is clearly enough to show that $\bowtie^{\Xi,g}$ is a (Ξ,g)-labelled bisimulation up to $\equiv$, since this implies $\bowtie^{\Xi,g} \subseteq \approx^{\Xi,g}$, and we conclude $\bowtie^{\Xi,g} = \approx^{\Xi,g}$, which proves that $\approx^{\Xi,g}$ (and thus $\approx_{\mathrm{c}}^{\Xi,g}$) is preserved by the listed process constructors. The proof is by induction on the definition of $\bowtie^{\Xi,g}$ using Lemma 4.16.

(1) This case follows by definition.

(2) Note that if $C\,.\,P\xrightarrow{\lambda}_{\Xi,g'} P'$, then either

$$\lambda \in \{\mathsf{in}\ n, \mathsf{out}\ n\} \quad \text{and} \quad P' \equiv P$$

or

$$\lambda = \mathsf{to}\ n; R_1\,; m[R_2] \quad \text{and} \quad P' \equiv n[P \mid R_1] \mid m[R_2]\,.$$

Since we also have $C\,.\,Q\xrightarrow{C}_{\Xi,g'} Q$ for $C \in \{\mathsf{in}\ n, \mathsf{out}\ n\}$ and

$$\mathsf{to}\ n\,.\,Q \xrightarrow{\mathsf{to}\ n;R_1\,;m[R_2]}_{\Xi,g} n[Q \mid R_1] \mid m[R_2]\,,$$

we are done using the contextuality of $\bowtie$.

(3) This proof is similar to and simpler than the proof of (2).

(4) For this proof we need to consider many different subcases:

– If $P \mid R\xrightarrow{\lambda}_{\Xi,g} P \mid R'$ because $R\xrightarrow{\lambda}_{\Xi,g} R'$ and $\lambda \notin \{\mathsf{to}\ n; S_1\,; m[S_2],\ [\mathsf{out}\ n]\,; S\}$ the proof follows trivially by Lemma 4.16(1) and the contextuality of $\bowtie$.

– Let $P \mid R\xrightarrow{\lambda}_{\Xi,g} P' \mid R$ because $P\xrightarrow{\lambda}_{\Xi,g} P'$ and

$$\lambda \notin \{\mathsf{to}\ n; S_1\,; m[S_2],\ [\mathsf{out}\ n]\,; S\}\,.$$

Then, by induction, we have $Q\overset{\hat{\lambda}}{\Longrightarrow}_{\Xi,g} Q'$ for some Q' such that $P' \bowtie^{\Xi,g} Q'$. By Lemma 4.16(2), we get $Q \mid R\overset{\hat{\lambda}}{\Longrightarrow}_{\Xi,g} Q' \mid R$, and thus $P' \mid R \bowtie^{\Xi,g} Q' \mid R$.

– Let $P \mid R \xrightarrow{\mathsf{to}\ n;S_1\,;m[S_2]}_{\Xi,g} R'$ because $R \xrightarrow{\mathsf{to}\ n}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle R_1\rangle\!\rangle R_2$. Then

$$\Xi \vdash P : g$$
$$\Xi = \Gamma;\ \Delta, m : g, n : g'$$
$$\Xi \vdash S_1 : g'$$
$$\Xi \vdash S_2 : g$$
$$R' \equiv (\nu\widetilde{p:g})(n[R_1 \mid S_1] \mid m[P \mid R_2 \mid S_2])\,.$$

By rule (Par), we have $Q \mid R\overset{\mathsf{to}\ n}{\Longrightarrow}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle R_1\rangle\!\rangle Q \mid R_2$, so by rule (HO To), we get

$$Q \mid R \overset{\mathsf{to}\ n;S_1\,;m[S_2]}{\Longrightarrow}_{\Xi,g} (\nu\widetilde{p:g})(n[R_1 \mid S_1] \mid m[Q \mid R_2 \mid S_2])\,,$$

and, by the contextuality of $\bowtie$, we are done.

The proof for $P \mid R \xrightarrow{[\mathsf{out}\ n]\,;S}_{\Xi,g} R'$ because $R \xrightarrow{[\mathsf{out}\ n]}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle R_1\rangle\!\rangle R_2$ is similar.

– Let $P \mid R \xrightarrow{\mathsf{to}\ n;S_1\,;m[S_2]}_{\Xi,g} P'$ because $P \xrightarrow{\mathsf{to}\ n}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle P_1\rangle\!\rangle P_2$. Then

$$\Xi \vdash R : g$$
$$\Xi = \Gamma;\ \Delta, m : g, n : g'$$
$$\Xi \vdash S_1 : g'$$
$$\Xi \vdash S_2 : g$$
$$P' \equiv (\nu\widetilde{p:g})(n[P_1 \mid S_1] \mid m[P_2 \mid R \mid S_2])\,.$$

By rule (HO To), we get $P \xrightarrow{\text{to } n;S_1;m[R\,|\,S_2]}_{\Xi,g} P'$. Then, by induction,

$$Q \overset{\text{to } n;S_1;m[R\,|\,S_2]}{\Longrightarrow}_{\Xi,g} Q'$$

for some Q' such that $P' \bowtie^{\Xi,g''} Q'$ for some g''. By Lemma 4.16(3), we have $Q\,|\,R \overset{\text{to } n;S_1;m[S_2]}{\Longrightarrow}_{\Xi,g} Q'$, so we are done.
The proof for $P\,|\,R \xrightarrow{[\text{out } n];S}_{\Xi,g} P'$ because $P \xrightarrow{[\text{out } n]}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle P_1\rangle\!\rangle P_2$ is similar.

– Let $P\,|\,R \xrightarrow{\tau}_{\Xi,g} P'$ because

$$P \xrightarrow{[C]}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle P_1\rangle\!\rangle P_2$$

and

$$R \xrightarrow{\overline{C}}_{\Xi,g} (\nu\widetilde{r:g'})\langle\!\langle R_1\rangle\!\rangle R_2$$

and $C \in \{\text{in } n, \text{to } n\}$. Then $P' \equiv (\nu\widetilde{p:g},\widetilde{r:g'})(n[P_1\,|\,R_1]\,|\,P_2\,|\,R_2)$.
By rule (HO [In]) or (HO [To]), we get $P \xrightarrow{[C];R_1}_{\Xi,g} (\nu\widetilde{p:g})(n[P_1\,|\,R_1]\,|\,P_2)$.
Then, by induction, $Q \overset{[C];R_1}{\Longrightarrow}_{\Xi,g} Q'$ for some Q' such that

$$(\nu\widetilde{p:g})(n[P_1\,|\,R_1]\,|\,P_2) \bowtie^{\Xi,g} Q'\,.$$

By Lemma 4.16(6), we have $Q\,|\,R \Longrightarrow_{\Xi,g} Q''$ for some $Q'' \equiv (\nu\widetilde{r:g'})(Q'\,|\,R_2)$, and this, by the contextuality of $\bowtie$, concludes the proof.
The proof for $P\,|\,R \xrightarrow{\tau}_{\Xi,g} P'$ because

$$P \xrightarrow{\langle M\rangle}_{\Xi,g} P_1$$

and

$$R \xrightarrow{\langle - \rangle}_{\Xi,g} (\nu\,\widetilde{q:g'})\langle\!\langle M\rangle\!\rangle R_1$$

is similar.

– Let $P\,|\,R \xrightarrow{\tau}_{\Xi,g} P'$ because

$$P \xrightarrow{\overline{C}}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle P_1\rangle\!\rangle P_2$$

and

$$R \xrightarrow{[C]}_{\Xi,g} (\nu\widetilde{r:g'})\langle\!\langle R_1\rangle\!\rangle R_2$$

and $C \in \{\text{in } n, \text{to } n\}$. Then $P' \equiv (\nu\widetilde{p:g},\widetilde{r:g'})(n[P_1\,|\,R_1]\,|\,P_2\,|\,R_2)$.
By rule (HO Co-In) or (HO Co-To), we get

$$P \xrightarrow{\overline{C};R_1}_{\Xi,g} (\nu\widetilde{p:g})(n[P_1\,|\,R_1]\,|\,P_2)\,.$$

Then, by induction, $Q \overset{\overline{C};R_1}{\Longrightarrow}_{\Xi,g} Q'$ for some Q' such that

$$(\nu\widetilde{p:g})(n[P_1\,|\,R_1]\,|\,P_2) \bowtie^{\Xi,g} Q'\,.$$

By Lemma 4.16(7), we have $Q\,|\,R \Longrightarrow_{\Xi,g} (\nu\widetilde{r:g'})(Q'\,|\,R_2)$ and this, by the contextuality of $\bowtie$, concludes the proof.

The proof for $P \mid R \xrightarrow{\tau}_{\Xi,g} P'$ because

$$P \xrightarrow{\langle - \rangle}_{\Xi,g} ; P_1 \quad \text{and} \quad R \xrightarrow{\langle M \rangle}_{\Xi,g} (\nu\,\widetilde{q:g'})\langle\!\langle M \rangle\!\rangle R_1$$

is similar.

(5) This proof also requires us to examine different cases:

- The case $m[P] \xrightarrow{\tau}_{\Xi,g} m[P']$ because $P \xrightarrow{\tau}_{\Xi,g'} P'$ is trivial.
- Let $m[P] \xrightarrow{[\mathsf{in}\ n];R}_{\Xi,g} P'$ because $P \xrightarrow{\mathsf{in}\ n}_{\Xi,g'} P_1$, $\Xi = \Gamma; \Delta, n:g'$ and $\Xi \vdash R:g'$. Then $P' \equiv n[m[P_1] \mid R]$. By induction,

$$Q \Longrightarrow_{\Xi,g'} S \xrightarrow{\mathsf{in}\ n}_{\Xi,g'} V \Longrightarrow_{\Xi,g'} Q_1$$

and $P_1 \bowtie^{\Xi,g'} Q_1$ for some S, V, Q_1. Now, $m[Q] \Longrightarrow_{\Xi,g} m[S]$ by rule (τ-Amb). By rules (In) and (HO [In]), we have $m[S] \xrightarrow{[\mathsf{in}\ n];R}_{\Xi,g} n[m[V] \mid R]$. Then, by rules (τ-Amb) and (Par), we have $n[m[V] \mid R] \Longrightarrow_{\Xi,g} n[m[Q_1] \mid R]$. This concludes the proof, owing to the contextuality of $\bowtie$.

- Let $m[P] \xrightarrow{[\mathsf{to}\ n];R}_{\Xi,g} P'$ because

$$P \xrightarrow{\mathsf{to}\ n}_{\Xi,g'} (\nu\widetilde{p:g})\langle\!\langle P_1 \rangle\!\rangle P_2$$

and $\Xi = \Gamma; \Delta, n:g'$ and $\Xi \vdash R:g'$. Then $P' \equiv (\nu\widetilde{p:g})(n[P_1 \mid R] \mid m[P_2])$. By rule (HO To),

$$P \xrightarrow{\mathsf{to}\ n;R;m[0]}_{\Xi,g'} (\nu\widetilde{p:g})(n[P_1 \mid R] \mid m[P_2 \mid 0]) \equiv P'.$$

By induction,

$$Q \Longrightarrow_{\Xi,g'} S \xrightarrow{\mathsf{to}\ n;R;m[0]}_{\Xi,g'} V \Longrightarrow_{\Xi,g''} Q'$$

and $P' \bowtie^{\Xi,g''} Q'$ for some g'', S, V, Q'. Now, we have $m[Q] \Longrightarrow_{\Xi,g} m[S]$ by rule (τ-Amb). If $S \xrightarrow{\mathsf{to}\ n;R;m[0]}_{\Xi,g'} V$, then

$$S \xrightarrow{\mathsf{to}\ n}_{\Xi,g'} (\nu\widetilde{s:g''})\langle\!\langle S_1 \rangle\!\rangle S_2$$

and

$$V \equiv (\nu\widetilde{s:g''})(n[S_1 \mid R] \mid m[S_2]).$$

By rule (To), we have $m[S] \xrightarrow{[\mathsf{to}\ n]}_{\Xi,g} (\nu\widetilde{s:g''})\langle\!\langle S_1 \rangle\!\rangle m[S_2]$ and by rule (HO [To]), we have $m[S] \xrightarrow{[\mathsf{to}\ n];R}_{\Xi,g} V$, and this concludes the proof.

- Let $m[P] \xrightarrow{\alpha;R}_{\Xi,g} P'$ because

$$P \xrightarrow{\alpha}_{\Xi,g'} O_1$$

$$\alpha \in \{\overline{\mathsf{in}\ m}, \overline{\mathsf{to}\ m}\}$$

$$\Xi = \Gamma, g'':G; \Delta, m:g', n:g''$$

$$g \in \mathscr{S}(G)$$

and

$$\begin{array}{lll} g' \in \mathscr{C}(G) & \text{if} & \alpha = \overline{\mathsf{in}\ m} \\ g' \in \mathscr{E}(G) & \text{if} & \alpha = \overline{\mathsf{to}\ m} \end{array}$$

and $\Xi \vdash R : g'$.
Then $O_1 \equiv \langle\!\langle m[P] \rangle\!\rangle \mathbf{0}$ and $P' \equiv m[P \mid R]$. By rule (Co-In-To), we get

$$m[Q] \xrightarrow{\alpha}_{\Xi,g} \langle\!\langle m[Q] \rangle\!\rangle \mathbf{0}\,.$$

Then, using either (HO Co-In) or (HO Co-To), we get

$$m[Q] \xrightarrow{\alpha;R}_{\Xi,g} m[Q \mid R]\,.$$

This concludes the proof by the contextuality of $\bowtie$.

– Let $m[P] \xrightarrow{\tau}_{\Xi,g} P'$ because $P \xrightarrow{[\mathsf{out}\ m]}_{\Xi,g'} (\nu \widetilde{p:g}) \langle\!\langle P_1 \rangle\!\rangle P_2$.
Then $P' \equiv (\nu \widetilde{p:g})(P_1 \mid m[P_2])$ and $m \notin \tilde{p}$. By rule (HO Out), we have

$$P \xrightarrow{[\mathsf{out}\ m];\mathbf{0}}_{\Xi,g'} (\nu \widetilde{p:g})(P_1 \mid m[P_2 \mid \mathbf{0}]) \equiv P'\,.$$

By induction,

$$Q \Longrightarrow_{\Xi,g'} S \xrightarrow{[\mathsf{out}\ m];\mathbf{0}}_{\Xi,g'} V \Longrightarrow_{\Xi,g''} Q'$$

and $P' \bowtie^{\Xi,g''} Q'$ for some g'', S, V, Q'. Now, by rule (τ-Amb), we have that $m[Q] \Longrightarrow_{\Xi,g} m[S]$. If $S \xrightarrow{[\mathsf{out}\ m];\mathbf{0}}_{\Xi,g'} V$, then

$$S \xrightarrow{[\mathsf{out}\ m]}_{\Xi,g'} (\nu \widetilde{s:g''}) \langle\!\langle S_1 \rangle\!\rangle S_2$$

and

$$V \equiv (\nu \widetilde{s:g''})(S_1 \mid m[S_2]).$$

By rule (τ-Exit), we have $m[S] \xrightarrow{\tau}_{\Xi,g} V$, and this concludes the proof.

(6) Again, we need to examine several cases.
Let

$$(\nu n : g')P \xrightarrow{\mathsf{to}\ m;R_1;q[R_2]}_{\Xi,g} P'$$

because

$$P \xrightarrow{\mathsf{to}\ m}_{\Xi',g} (\nu \widetilde{p:g''}) \langle\!\langle P_1 \rangle\!\rangle P_2$$

where $\Xi' = \Xi, n : g'$.
Then $P' \equiv (\nu n : g')P''$ and $P \xrightarrow{\mathsf{to}\ m;R_1;q[R_2]}_{\Xi',g} P''$ where

$$P'' \equiv (\nu \widetilde{p:g''})(m[P_1 \mid R_1] | q[P_2 \mid R_2])\,.$$

By induction,

$$Q \Longrightarrow_{\Xi',g} S \xrightarrow{\mathsf{to}\ m;R_1;q[R_2]}_{\Xi',g} V \Longrightarrow_{\Xi',g''} Q''$$

and $P'' \bowtie^{\Xi',g''} Q''$ for some g'', S, V, Q''.

As n is bound, we can assume $n \notin AN(\mathsf{to}\, m; R_1; q[R_2])$. By Lemma 4.16(8), we have

$$(\nu n : g)Q \Longrightarrow_{\Xi,g} (\nu n : g)S \xrightarrow{\mathsf{to}\, m; R_1; q[R_2]}_{\Xi,g} (\nu n : g)V \Longrightarrow_{\Xi,g''} (\nu n : g)Q'',$$

and

$$(\nu n : g)Q'' \bowtie^{\Xi,g''} (\nu n : g)P'' \equiv P'$$

by the contextuality of $\bowtie$.

The proof in the other cases is similar and simpler.

C Note first that the operator ! can only be applied to guarded processes. So we can assume that both P and Q are guarded, and hence that they cannot do τ-transitions. We consider the relation $\widehat{\bowtie}^{\Xi,g}$ defined as $\bowtie^{\Xi,g}$ plus a rule for replication.

1 $\approx^{\Xi,g} \subseteq \widehat{\bowtie}^{\Xi,g}$.

2 $P \widehat{\bowtie}^{\Xi,g} Q$ and $\Xi \vdash C : g \to g'$ imply $C . P \widehat{\bowtie}^{\Xi,g'} C . Q$.

3 $P \widehat{\bowtie}^{\Xi,g} Q$, and $\Xi \vdash M : W$ and $\Xi \vdash g : \mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, W)$ imply $\langle M \rangle P \widehat{\bowtie}^{\Xi,g} \langle M \rangle Q$.

4 $P \widehat{\bowtie}^{\Xi,g} Q$ and $\Xi \vdash R : g$ imply $P \mid R \widehat{\bowtie}^{\Xi,g} Q \mid R$ and $R \mid P \widehat{\bowtie}^{\Xi,g} R \mid P$.

5 $P \widehat{\bowtie}^{\Xi,g} Q$, and $n : g \in \Xi$ and $g : G \in \Xi$, and $g' \in \mathscr{S}(G)$ imply $n[P] \widehat{\bowtie}^{\Xi,g'} n[Q]$.

6 $P \widehat{\bowtie}^{\Xi,g} Q$ and $\Xi = \Gamma; \Delta, n : g'$ imply $(\nu n : g')P \widehat{\bowtie}^{\Gamma;\Delta,g} (\nu n : g')Q$.

7 $P \approx^{\Xi,g} Q$ implies $!P \widehat{\bowtie}^{\Xi,g} !Q$, where P, Q are prefixed processes.

We prove that

- If $P \xrightarrow{\lambda}_{\Xi,g} P'$ where $\lambda \neq \tau$, then $Q \overset{\lambda}{\Longrightarrow}_{\Xi,g} Q'$ and $P' \approx^{\Xi,g} \widehat{\bowtie}^{\Xi,g} \approx^{\Xi,g} Q'$.
- If $P \xrightarrow{\tau}_{\Xi,g} P'$, then $Q \Longrightarrow_{\Xi,g} Q'$ and $P' \equiv \widehat{\bowtie}^{\Xi,g} \equiv Q'$.
- If $Q \xrightarrow{\lambda}_{\Xi,g} Q'$ where $\lambda \neq \tau$, then $P \overset{\lambda}{\Longrightarrow}_{\Xi,g} P'$ and $Q' \approx^{\Xi,g} \widehat{\bowtie}^{\Xi,g} \approx^{\Xi,g} P'$.
- If $Q \xrightarrow{\tau}_{\Xi,g} Q'$, then $P \Longrightarrow_{\Xi,g} P'$ and $Q' \equiv \widehat{\bowtie}^{\Xi,g} \equiv P'$.

From the above and using Sangiorgi and Walker (2001, Exercise 2.4.64), we can conclude that $\widehat{\bowtie}^{\Xi,g}$ is a bisimilarity, and hence, $\widehat{\bowtie}^{\Xi,g} = \approx^{\Xi,g}$.

(1–6) The proofs for the first six parts of the definition, for both $\xrightarrow{\lambda}_{\Xi,g}$ and $\xrightarrow{\tau}_{\Xi,g}$, are exactly as in Step **B**, taking into account the fact that $\equiv$ is included in $\approx^{\Xi,g}$ for suitable Ξ, g by Lemma 4.10.

(7) Let $!P \xrightarrow{\alpha}_{\Xi,g} P'$ because $P \xrightarrow{\alpha}_{\Xi,g} P_1$ where $\alpha \neq \tau$ is a first-order label. Then $P' \equiv !P \mid P_1$. By definition, $Q \xrightarrow{\alpha}_{\Xi,g} S \Longrightarrow_{\Xi,g} Q_1$ and $P_1 \approx^{\Xi,g} Q_1$ for some S, Q_1. By rules (Repl) and (Par), we have $!Q \xrightarrow{\alpha}_{\Xi,g} !Q \mid S \Longrightarrow_{\Xi,g} !Q \mid Q_1$. Finally, note that $!P \mid P_1 \approx^{\Xi,g} !P \mid Q_1 \widehat{\bowtie}^{\Xi,g} !Q \mid Q_1$ since in Step **B** we proved that $\approx^{\Xi,g}$ is preserved by parallel composition. Indeed, this was the reason for separating Steps **B** and **C**.

Otherwise, let $!P \xrightarrow{\lambda}_{\Xi,g} P'$ where λ is a higher-order label. We will only give the proof for $\lambda = \mathsf{to}\, n; R_1; m[R_2]$, since the other cases are similar and simpler.

Let $!P \xrightarrow{\mathsf{to}\, n; R_1; m[R_2]}_{\Xi,g} P'$ because $P \xrightarrow{\mathsf{to}\, n}_{\Xi,g} (\nu \widetilde{p : g})\langle\langle P_1 \rangle\rangle P_2$.

Then $P' = (\nu \widetilde{p : g})(n[P_1 \mid R_1] | m[P_2 \mid !P \mid R_2])$. By rule (HO To), we have

$$P \xrightarrow{\mathsf{to}\, n; R_1; m[!Q \mid R_2]}_{\Xi,g} P''$$

where

$$P'' = (\nu\widetilde{p:g})(n[P_1 \mid R_1]|m[P_2 \mid !Q \mid R_2])\,.$$

Note that $P'\widehat{\bowtie}^{\Xi,g}P''$ by the definition of $\widehat{\bowtie}^{\Xi,g}$. Since Q cannot do silent actions, by induction,

$$Q \xrightarrow{\mathsf{to}\ n;R_1\,;m[!Q \mid R_2]}{}_{\Xi,g}\ Q' \Longrightarrow_{\Xi,g''} Q''$$

for some g'', Q', Q'' such that $P'' \approx^{\Xi,g''} Q''$. If $Q \overset{\mathsf{to}\ n;R_1\,;m[!Q \mid R_2]}{\longrightarrow}{}_{\Xi,g}\ Q'$, then by Lemma 4.16(3), we have

$$Q \mid !Q \xrightarrow{\mathsf{to}\ n;R_1\,;m[R_2]}{}_{\Xi,g}\ Q'$$

and by Lemma 4.10(1), we have

$$!Q \xrightarrow{\mathsf{to}\ n;R_1\,;m[R_2]}{}_{\Xi,g}\ S'$$

for some $S' \equiv Q'$, and $S' \Longrightarrow_{\Xi,g''} T$ for some $T \equiv Q''$, and we are done. □

Full bisimilarity is sound but not complete with respect to the reduction barbed congruence.

Theorem 4.18 (Soundness of full bisimilarity). If $P \approx_{\mathrm{c}}^{\Xi,g} Q$, then $P \cong^{\Xi,g} Q$.

Proof. By Theorem 4.17, it is enough to show that $\approx_{\mathrm{c}}^{\Xi,g}$ is reduction-closed and barb-preserving when restricted to closed processes. Take P, Q closed and assume $P \approx^{\Xi,g} Q$.

If $P \to P'$, by Theorem 4.7(2), we have $P \xrightarrow{\tau}_{\Xi,g} P''$ for some $P'' \equiv P'$. Since $P \approx^{\Xi,g} Q$, there exists Q' such that $Q \Longrightarrow_{\Xi,g} Q'$ and $P'' \approx^{\Xi,g} Q'$. Thus $P' \approx^{\Xi,g} Q'$, since clearly $\approx^{\Xi,g}$ up to $\equiv$ coincides with $\approx^{\Xi,g}$.

If $P \downarrow_n$, then, using Proposition 4.8, and the rules (HO Co-In) and (HO Co-To), we have $P \xrightarrow{C;R}_{\Xi,g} S$, where $C \in \{\overline{\mathsf{in}\ n}, \overline{\mathsf{to}\ n}\}$, for some R, S. Then, since $P \approx^{\Xi,g} Q$, we know that $Q \overset{C;R}{\Longrightarrow} S'$ for some $S' \approx^{\Xi,g} S$, from which $Q \Downarrow_n$, as desired. □

The failure of completeness is due to the fact that contexts are insensitive to movements of restricted ambients and to sending dead processes. For example, no context can distinguish between the processes

$$(\nu n:g)(n[\mathsf{in}\ m]) \quad (\nu n:g)(n[\mathsf{out}\ m]) \quad (\nu n:g)(n[\mathsf{to}\ m]) \quad \mathbf{0}\,,$$

which are not fully bisimilar. We conjecture that we could obtain a complete labelled transition system by refining the notion of bisimilarity so that it distinguishes between visible and invisible transitions along the lines of Merro and Zappa Nardelli (2005).

We conclude this section by discussing some properties of processes and ambients in a fixed environment; they will allow us to prove a few interesting algebraic laws.

In fact, we can use higher-order transitions to define some useful notions for ambients and processes that have natural sufficient conditions in terms of type information. We write $P \Longrightarrow_{\Xi,g} \xrightarrow{\mathsf{in}\ n}_{\Xi,g} O$ as an abbreviation for $P \Longrightarrow_{\Xi,g} Q \xrightarrow{\mathsf{in}\ n}_{\Xi,g} O$ for some Q.

Definition 4.19. Let $\Xi \vdash P : g$. We say that:

1 A process P is a (Ξ, g)-mover if $P \Longrightarrow_{\Xi,g} \xrightarrow{\mathsf{in}\ n}_{\Xi,g} P'$ or $P \Longrightarrow_{\Xi,g} \xrightarrow{\mathsf{out}\ n}_{\Xi,g} P'$ for some n, P'.

2 A process P is a (Ξ, g)-sender if $P \Longrightarrow_{\Xi,g} \xrightarrow{\mathsf{to}\ n}_{\Xi,g} O$ for some n, O.

3 A process P is a (Ξ, g)-communicator if $P \Longrightarrow_{\Xi,g} \xrightarrow{\langle M \rangle}_{\Xi,g} P'$ or $P \Longrightarrow_{\Xi,g} \xrightarrow{\langle - \rangle}_{\Xi,g} O$ for some M, P', O.

4 An ambient n is (Ξ, g)-mobile if $n[P] \xrightarrow{[\mathsf{in}\ m]}_{\Xi',g} O$ for some P, m, O and well-formed $\Xi' \supseteq \Xi$.

In point (4) we have chosen the action $\xrightarrow{[\mathsf{in}\ m]}_{\Xi,g}$ to characterise the mobility of ambients, since one can easily see that if $n[P] \xrightarrow{[\mathsf{out}\ m]}_{\Xi,g} O$ for some process P and concretion O, then $n[P'] \xrightarrow{[\mathsf{in}\ m]}_{\Xi,g} O'$ for some process P' and concretion O'. The converse is not true, since the conditions of the typing rule (Out) are more restrictive than those of the typing rule (In). In point 4 we take $\Xi' \supseteq \Xi$ since we want ambient properties to be preserved when ambients are included in contexts. Without that condition, an ambient could be moved, for instance, by a parallel process. In fact, assume that Ξ contains $n : g'$, $g' : G$ with $\mathscr{C}(G) = \{g''\}$, but no ambient of type g''. Then there is no m such that $n[P] \xrightarrow{[\mathsf{in}\ m]}_{\Xi,g} O$; but if we take $n[\,] \mid (\nu m : g'')m[\mathsf{to}\ n\,.\,\mathsf{in}\ m]$, we see that $n[\,]$ can be moved inside m.

Types give necessary conditions for the process properties, as stated in the following Lemma.

Lemma 4.20. Let $\Xi \vdash P : g$ and $\Xi = \Gamma, g : G; \Delta$.

1 If P is a (Ξ, g)-*mover*, then $\mathscr{C}(G) \neq \emptyset$.

2 If P is a (Ξ, g)-*sender*, then $\mathscr{E}(G) \neq \emptyset$.

3 If P is a (Ξ, g)-*communicator*, then $T(G) \neq \mathsf{shh}$.

Ambient mobility can be fully characterised by types.

Lemma 4.21. Let $\Xi = \Gamma, g : G; \Delta, n : g$ and $g' \in \mathscr{S}(G)$. An ambient n is (Ξ, g')-*mobile* if and only if $\mathscr{C}(G) \neq \emptyset$.

Proof. Assume $n[P] \xrightarrow{[\mathsf{in}\ m]}_{\Xi',g'} O$ for some $\Xi' \supseteq \Xi$. This implies $P \equiv (\nu \widetilde{p : g''})(\mathsf{in}\ m.Q \mid R)$ for some $\tilde{p}, \tilde{g}'', Q, R$ by Lemma 4.5(3). By definition, $\Xi' \vdash n[P] : g'$, so by Theorem 3.3(1) and Lemma 3.1(9), we get $\Xi' \vdash (\nu \widetilde{p : g''})(\mathsf{in}\ m.Q \mid R) : g$ and $g' \in \mathscr{S}(G)$.

We get $\Xi', \widetilde{p : g''} \vdash \mathsf{in}\ m.Q : g$ by Lemma 3.1(12 and 10), and this implies

$$\Xi', \widetilde{p : g''} \vdash \mathsf{in}\ m : g \to g\,,$$

and $m : g_m \in \Xi'$ and $g_m \in \mathscr{C}(G)$ by Lemma 3.1(6 and 2).

For the converse, take g_m such that $g_m \in \mathscr{C}(G)$ and define $\Delta' = \Delta$ if $m : g_m \in \Delta$ for some ambient m and $\Delta' = \Delta, m : g_m$ otherwise, where m is fresh for Δ. We then get $n[\mathsf{in}\ m.0] \xrightarrow{[\mathsf{in}\ m]}_{\Xi',g'} \langle\!\langle n[0] \rangle\!\rangle 0$, where $\Xi' = \Gamma, g : G; \Delta', n : g$. □

Using the previous definitions, we can formulate a number of algebraic laws about processes. Some of them will be used in the following section. A preliminary lemma will be used in their proofs.

Lemma 4.22.

1 If $P \xrightarrow{\alpha}_{\Xi,g} P'$, then $P \equiv (\nu\widetilde{p:g})(\pi S \mid R)$, and $P' \equiv (\nu\widetilde{p:g})(S \mid R)$ for some $\tilde{p}, \tilde{g}, \pi, S, R$, and, for all R' such that $\Xi \vdash (\nu\widetilde{p:g})(\pi S \mid R') : g$, we get

$$(\nu\widetilde{p:g})(\pi S \mid R') \xrightarrow{\alpha}_{\Xi,g} (\nu\widetilde{p:g})(\pi S \mid R')\,.$$

2 If $P \xrightarrow{\langle - \rangle;Q}_{\Xi,g} P'$, then $P \equiv (\nu\widetilde{p:g})(\langle M \rangle S \mid R)$, and $P' \equiv (\nu\widetilde{p:g})(S \mid Q\{x := M\} \mid R)$ for some $\tilde{p}, \tilde{g}, M, S, R$, and, for all R' such that $\Xi \vdash (\nu\widetilde{p:g})(\langle M \rangle S \mid R') : g$, we get

$$(\nu\widetilde{p:g})(\langle M \rangle S \mid R') \xrightarrow{\langle - \rangle;Q}_{\Xi,g} (\nu\widetilde{p:g})(S \mid Q\{x := M\} \mid R')\,.$$

3 If $P \overset{\mathsf{to}\ n;Q_1;m[Q_2]}{\longrightarrow}_{\Xi,g} P'$, then $P \equiv (\nu\widetilde{p:g})(\mathsf{to}\ n.S \mid R)$, and

$$P' \equiv (\nu\widetilde{p:g})(n[S \mid Q_1] \mid m[R \mid Q_2])$$

for some $\tilde{p}, \tilde{g}, S, R$, and, for all R' such that $\Xi \vdash (\nu\widetilde{p:g})(\mathsf{to}\ n.S \mid R') : g$, we get

$$(\nu\widetilde{p:g})(\mathsf{to}\ n.S \mid R') \overset{\mathsf{to}\ n;Q_1;m[Q_2]}{\longrightarrow}_{\Xi,g} (\nu\widetilde{p:g})(n[S \mid Q_1] \mid m[R' \mid Q_2])\,.$$

4 If $P \overset{[\mathsf{in}\ n];Q}{\longrightarrow}_{\Xi,g} P'$, then $P \equiv (\nu\widetilde{p:g})(m[\mathsf{in}\ n.S_1 \mid S_2] \mid R)$, and

$$P' \equiv (\nu\widetilde{p:g})(n[m[S_1 \mid S_2] \mid Q] \mid R)$$

for some $\tilde{p}, \tilde{g}, m, S_1, S_2, R$, and, for all S_2', R' such that

$$\Xi \vdash (\nu\widetilde{p:g})(m[\mathsf{in}\ n.S_1 \mid S_2'] \mid R') : g\,,$$

we get

$$(\nu\widetilde{p:g})(m[\mathsf{in}\ n.S_1 \mid S_2'] \mid R') \overset{[\mathsf{in}\ n];Q}{\longrightarrow}_{\Xi,g} (\nu\widetilde{p:g})(n[m[S_1 \mid S_2'] \mid Q] \mid R')\,.$$

5 If $P \overset{[\mathsf{to}\ n];Q}{\longrightarrow}_{\Xi,g} P'$, then $P \equiv (\nu\widetilde{p:g})(m[\mathsf{to}\ n.S_1 \mid S_2] \mid R)$, and

$$P' \equiv (\nu\widetilde{p:g})(n[S_1 \mid Q] \mid m[S_2] \mid R)$$

for some $\tilde{p}, \tilde{g}, m, S_1, S_2, R$, and, for all S_2', R' such that

$$\Xi \vdash (\nu\widetilde{p:g})(m[\mathsf{to}\ n.S_1 \mid S_2'] \mid R') : g\,,$$

we get

$$(\nu\widetilde{p:g})(m[\mathsf{to}\ n.S_1 \mid S_2'] \mid R') \overset{[\mathsf{to}\ n];Q}{\longrightarrow}_{\Xi,g} (\nu\widetilde{p:g})(n[S_1 \mid Q] \mid m[S_2'] \mid R')\,.$$

6 If $P \xrightarrow{\alpha;Q}_{\Xi,g} P'$ with $\alpha \in \{\overline{\mathsf{in}\ n}, \overline{\mathsf{to}\ n}\}$, then $P \equiv (\nu\widetilde{p:g})(n[S] \mid R)$, and

$$P' \equiv (\nu\widetilde{p:g})(n[S \mid Q] \mid R)$$

for some $\tilde{p}, \tilde{g}, S, R$, and for all S', R' such that $\Xi \vdash (\nu\widetilde{p:g})(n[S'] \mid R') : g$, we get

$$(\nu\widetilde{p:g})(n[S'] \mid R') \xrightarrow{\alpha;Q}_{\Xi,g} (\nu\widetilde{p:g})(n[S' \mid Q] \mid R')\,.$$

7 If $P \xrightarrow{[\mathsf{out}\ n];Q}_{\Xi,g} P'$, then $P \equiv (\nu\widetilde{p:g})(m[\mathsf{out}\ n.S_1 \mid S_2] \mid R)$, and

$$P' \equiv (\nu\widetilde{p:g})(m[S_1 \mid S_2] \mid n[R \mid Q])$$

for some $\tilde{p}, \tilde{g}, m, S_1, S_2, R$, and, for all S_2', R' such that

$$\Xi \vdash (\nu\widetilde{p:g})(m[\mathsf{out}\ n.S_1 \mid S_2'] \mid R') : g\,,$$

we get

$$(\nu\widetilde{p:g})(m[\mathsf{out}\ n.S_1 \mid S_2'] \mid R') \xrightarrow{[\mathsf{out}\ n];Q}_{\Xi,g} (\nu\widetilde{p:g})(m[S_1 \mid S_2'] \mid n[R' \mid Q])\,.$$

Proof. We show (7), the other proofs being similar.

If $P \xrightarrow{[\mathsf{out}\ n];Q}_{\Xi,g} P'$, then by rule (HO Out) and Lemma 4.5(3), we get

$$P \equiv (\nu\widetilde{p:g})(m[\mathsf{out}\ n.S_1 \mid S_2] \mid R)\,,$$

and

$$P' \equiv (\nu\widetilde{p:g})(m[S_1 \mid S_2] \mid n[R \mid Q])$$

for some $\tilde{p}, \tilde{g}, m, S_1, S_2, R$. Moreover, if

$$\Xi \vdash (\nu\widetilde{p:g})(m[\mathsf{out}\ n.S_1 \mid S_2'] \mid R') : g\,,$$

we get

$$(\nu\widetilde{p:g})(m[\mathsf{out}\ n.S_1 \mid S_2'] \mid R') \xrightarrow{[\mathsf{out}\ n]}_{\Xi,g} (\nu\widetilde{p:g})\langle\!\langle (m[S_1 \mid S_2'] \rangle\!\rangle R'$$

by rules (Cap-Out), (Out), (Par) and (Res), and hence by rule (HO Out), we conclude

$$(\nu\widetilde{p:g})(m[\mathsf{out}\ n.S_1 \mid S_2'] \mid R') \xrightarrow{[\mathsf{out}\ n];Q}_{\Xi,g} (\nu\widetilde{p:g})(m[S_1 \mid S_2'] \mid n[R' \mid Q])\,. \qquad \square$$

In points (1), (3) and (4) of the following theorem we need to put some restrictions on the possible occurrences of the ambient named m. This is accomplished by different conditions on m, which will be required when we use point (3) in the firewall example (Subsection 5.1) and point (4) in the encoding of the π-calculus (Subsection 5.2).

Theorem 4.23. Let both sides of the following equalities be well-typed closed processes with type g from Ξ. Then we have:

(1) Let $\Xi' = \Xi, m : g'$. If Q has only τ transitions, m is not (Ξ', g)-mobile and R contains occurrences of m only in capabilities, then

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

(2) If Q has only τ transitions and m is not (Ξ, g)-mobile, then

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

(3) Let $\Xi' = \Xi, m:g'$. If m, n are not (Ξ', g)-mobile and Q, R contain occurrences of m only in capabilities, then

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

(4) Let $\Xi' = \Xi, m:g', n:g''$. If m is not (Ξ', g)-mobile and the ambient names in Q, R are neither m nor variables, then

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

(5) Let $\Xi' = \Xi, n:g'$. If $n \notin AN(R)$ and R is not a (Ξ', g')-communicator, then

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

(6) Let $\Xi' = \Xi, n:g'$. If $n \notin AN(P)$ and P is neither a (Ξ', g')-sender nor a (Ξ', g')-mover, then

$$(\nu n:g')(n[P]) \cong^{\Xi,g} \mathsf{0}\,.$$

Proof. Let $\mathcal{I}^{\Xi,g}$ denote the identity relation on processes typable by g in the environment Ξ.

(1) Take

$$\begin{aligned}\mathcal{B}^{\Xi,g} \ = \ \mathcal{I}^{\Xi,g} \cup \{&(\nu m:g')\mathcal{C}[(\nu n:g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid m[R]]^{\Xi_0,g_0},\\ &(\nu m:g')\mathcal{C}[(m[(\nu n:g'')\, n[P \mid Q]) \mid R]]^{\Xi_0,g_0}\}\,,\end{aligned}$$

where n, m, Q, R satisfy the hypotheses of (1) with Ξ', g replaced by Ξ_0, g_0, and $\mathcal{C}[\ ]^{\Xi_0,g_0}$ is a closed context built only by name restrictions, ambient formation and parallel composition, which contains occurrences of m only in capabilities, and such that

$$\Xi \vdash \mathcal{C}[(\nu n:g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid m[R]]^{\Xi_0,g_0} : g\,.$$

We will show that $\mathcal{B}^{\Xi,g}$ is a bisimulation, and hence that it is included in $\approx_c$.
For $\mathcal{B}^{\Xi,g}$ to be a bisimulation, it must satisfy:

$$T \xrightarrow{\lambda}_{\Xi,g} T' \text{ and } T\mathcal{B}^{\Xi,g}U \text{ imply } U \overset{\hat{\lambda}}{\Longrightarrow}_{\Xi,g} U' \text{ for some } T'\mathcal{B}^{\Xi,g}U'\,. \tag{4.1}$$

$$U \xrightarrow{\lambda}_{\Xi,g} U' \text{ and } T\mathcal{B}^{\Xi,g}U \text{ imply } T \overset{\hat{\lambda}}{\Longrightarrow}_{\Xi,g} T' \text{ for some } T'\mathcal{B}^{\Xi,g}U'\,. \tag{4.2}$$

If $(T, U) \in \mathcal{I}^{\Xi,g}$, the property is immediate. Otherwise, the proof of (4.2) is trivial as $T \xrightarrow{\tau}_{\Xi,g} U$. For (4.1), let $T \xrightarrow{\lambda}_{\Xi,g} T'$. The proof is by cases on λ. We will only consider the following cases:

(i) $\lambda = \mathsf{to}\ q\,; V_1\,; r[V_2]$. This case is paradigmatic for higher-order transitions involving the context only.

(ii) $\lambda = [\mathsf{to}\ q]\,; V$. This is the only higher-order transition involving $m[R]$.

(iii) $\lambda = \tau$.

However, before we consider these cases, first note that by looking at the transition rules, one can easily see that:

- Since Q has only τ-transitions, the only transitions of $(\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q])$ are

$$\begin{array}{lll}(\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) & \xrightarrow{[\mathsf{in}\ m];V}_{\Xi_0,g_0} & m[(\nu n : g'')(n[P \mid Q]) \mid V] \\ (\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) & \xrightarrow{\tau}_{\Xi_0,g_0} & (\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q']) .\end{array}$$

- Since m is not (Ξ_0, g_0)-mobile, the only transitions of $m[R]$ are

$$\begin{array}{lll} m[R] & \xrightarrow{[\mathsf{out}\ m];0}_{\Xi_0,g_0} & m[R_1] \mid p[R_2] \\ m[R] & \xrightarrow{[\mathsf{to}\ q];V}_{\Xi_0,g_0} & q[R_1 \mid V] \mid m[R_2] \\ m[R] & \xrightarrow{C;V}_{\Xi_0,g_0} & m[R \mid V] \\ m[R] & \xrightarrow{\tau}_{\Xi_0,g_0} & m[R'] ,\end{array}$$

  where $C \in \{\overline{\mathsf{in}\ m}, \overline{\mathsf{to}\ m}\}$, for some $p \neq m$ (since R only contains m within capabilities), q, R_1, R_2, R'.

We now consider the three cases listed above:

(i) $T \xrightarrow{\mathsf{to}\ q;V_1;r[V_2]}_{\Xi,g} T'$.

By Lemma 4.22(3), we have

$$T \equiv (\nu m : g')(\widetilde{\nu p : g})(\mathsf{to}\ q.S \mid Z)$$

and

$$T' \equiv (\nu m : g')(\widetilde{\nu p : g})(q[S \mid V_1] \mid r[Z \mid V_2])$$

for some r, S, Z. This implies

$$\mathcal{C}[(\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid m[R]]^{\Xi_0,g_0} \equiv (\widetilde{\nu p : g})(\mathsf{to}\ q.S \mid Z)$$

and

$$Z \equiv \mathcal{C}_1[(\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid m[R]]^{\Xi_0,g_0}$$

for some context $\mathcal{C}_1[\ ]$. By Lemma 4.22(3), we have

$$\begin{array}{l}(\nu m : g')(\widetilde{\nu p : g})(\mathsf{to}\ q.S \mid \mathcal{C}_1[m[(\nu n : g'')(n[P \mid Q]) \mid R]]^{\Xi_0,g_0}) \xrightarrow{\mathsf{to}\ q;V_1;r[V_2]}_{\Xi,g} \\ \qquad (\nu m : g')(\widetilde{\nu p : g})(q[S \mid V_1] \mid r[\mathcal{C}_1[m[(\nu n : g'')(n[P \mid Q]) \mid R]]^{\Xi_0,g_0} \mid V_2]) \equiv U'\end{array}$$

and $T' \mathcal{B}^{\Xi,g} U'$ through the context $(\widetilde{\nu p : g})(q[S \mid V_1] \mid r[\mathcal{C}_1[\ ]^{\Xi_0,g_0} \mid V_2])$.

(ii) $T \xrightarrow{[\mathsf{to}\ q];V}_{\Xi,g} T'$.

By Lemma 4.22(5), we have

$$T \equiv (\nu m : g')(\widetilde{\nu p : g})(r[\mathsf{to}\ q.S_1 \mid S_2] \mid Z)$$

and

$$T' \equiv (\nu m : g')(\widetilde{\nu p : g})(r[S_1] \mid q[S_2 \mid V] \mid Z)$$

for some r, S_1, S_2, Z. Hence, either

(a) $\mathcal{C}[(\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid m[R]]^{\Xi_0,g_0} \equiv (\widetilde{\nu p : g})(r[\mathsf{to}\ q.S_1 \mid S_2] \mid Z)$ and

$$Z \equiv \mathcal{C}_1[(\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid m[R]]^{\Xi_0,g_0}$$

for some context $\mathcal{C}_1[\ ]$; or

(b) $r = m$ and $R \equiv \mathsf{to}\ q.S_1 \mid S_2$ and $Z \equiv (\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid Z'$ for some Z'.

In the first case the proof is similar to that of case (i).
In the second case,

$$U \equiv (\nu m : g')(\widetilde{\nu p : g})(m[\mathsf{to}\ q.S_1 \mid S_2 \mid (\nu n : g'')(n[P \mid Q])] \mid Z'),$$

so, by Lemma 4.22(5), we have

$$U \xrightarrow{[\mathsf{to}\ q]\,;V}{}_{\Xi,g}\ (\nu m : g')(\widetilde{\nu p : g})(m[S_1 \mid (\nu n : g'')(n[P \mid Q])] \mid q[S_2 \mid V] \mid Z') \equiv U'$$

and $T' \mathcal{B}^{\Xi,g} U'$ through the context $(\widetilde{\nu p : g})([\ ]^{\Xi_0,g_0} \mid q[S_2 \mid V] \mid Z')$.

(iii) $T \xrightarrow{\tau}{}_{\Xi,g}\ T'$.

The only interesting sub-cases are those in which the transition involves both the context and the process $m[R]$, that is, when

$$T \equiv \mathcal{C}'[(\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid m[R] \mid Z]^{\Xi_0,g_0}$$

and:

(a) $R \equiv \mathsf{to}\ q\,.\,R_1 \mid R_2$,
$Z \equiv q[Z']$,
$T' \equiv \mathcal{C}'[(\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid m[R_2] \mid q[R_1 \mid Z']]^{\Xi_0,g_0}$;

or

(b) $Z \equiv q[\mathsf{in}\ m\,.\,Z_1 \mid Z_2]$,
$T' \equiv \mathcal{C}'[(\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid m[R \mid q[Z_1 \mid Z_2]]]^{\Xi_0,g_0}$;

or

(c) $Z \equiv q[\mathsf{to}\ m\,.\,Z_1 \mid Z_2]$,
$T' \equiv \mathcal{C}'[(\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid m[R \mid Z_1] \mid q[Z_2]]^{\Xi_0,g_0}$;

or

(d) $R \equiv q[\mathsf{out}\ m\,.\,R_1 \mid R_2] \mid R_3$,
$Z \equiv 0$,
$T' \equiv \mathcal{C}'[(\nu n : g'')(n[\mathsf{in}\ m\,.\,P \mid Q]) \mid m[R_3] \mid q[R_1 \mid R_2]]^{\Xi_0,g_0}$.

Note that in all cases $q \neq m$ since both R and $C[\ ]$ only contain m within capabilities. In case (a),

$$U \equiv \mathcal{C}'[m[\mathsf{to}\ q\,.\,R_1 \mid R_2 \mid (\nu n : g'')(n[P \mid Q])] \mid q[Z']]^{\Xi_0,g_0},$$

so, by rules (Cap-To), (To), (Par), (Res), (τ-To) and (Amb), we get

$$U \xrightarrow{\tau}{}_{\Xi,g}\ \mathcal{C}'[m[R_2 \mid (\nu n : g'')(n[P \mid Q])] \mid q[R_1 \mid Z']]^{\Xi_0,g_0}.$$

Hence, $T' \mathcal{B}^{\Xi,g} U'$ through the context $\mathcal{C}'[[\ ]^{\Xi_0,g_0} \mid q[R_1 \mid Z']]^{\Xi_0,g_0}$. The context $\mathcal{C}'[\ ]$ again satisfies the condition that m occurs only within capabilities.

The proofs of the other cases are similar.

(2) The proof of (2) is similar to but simpler than the proof of (1).

(3) Assume $g : G,\ g' : G' \in \Xi$ and take

$$\begin{aligned}\mathcal{B}^{\Xi,g} &= \mathcal{I}^{\Xi,g} \cup \{(\nu m : g')\mathcal{C}[n[\mathsf{to}\ m\,.\,P \mid Q] \mid m[R]]^{\Xi_0,g_0},\\ &\qquad (\nu m : g')\mathcal{C}[n[Q] \mid m[P \mid R]]^{\Xi_0,g_0}\},\end{aligned}$$

where n, m, Q, R satisfy the hypothesis of (3) with Ξ', g replaced by Ξ_0, g_0, and $\mathcal{C}[\ ]^{\Xi_0,g_0}$ is a closed context built only by name restrictions, ambient formation and parallel composition, and such that any occurrences of m are contained within capabilities, and such that $\Xi \vdash \mathcal{C}[n[\mathsf{to}\ m\,.\,P \mid Q] \mid m[R]]^{\Xi_0,g_0} : g$.

By looking at the transition rules, one can easily see that:

— The only transitions of $n[\mathsf{to}\ m\,.\,P \mid Q]$ are:

$$\begin{array}{lll}
n[\mathsf{to}\ m\,.\,P \mid Q] & \xrightarrow{[\mathsf{to}\ m]\,;V}_{\Xi_0,g_0} & m[P \mid V] \mid n[Q]\\
n[\mathsf{to}\ m\,.\,P \mid Q] & \xrightarrow{[\mathsf{out}\ n]\,;0}_{\Xi_0,g_0} & n[\mathsf{to}\ m\,.\,P \mid Q_1] \mid p[Q_2]\\
n[\mathsf{to}\ m\,.\,P \mid Q] & \xrightarrow{[\mathsf{to}\ q]\,;V}_{\Xi_0,g_0} & q[Q_1 \mid V] \mid n[\mathsf{to}\ m\,.\,P \mid Q_2]\\
n[\mathsf{to}\ m\,.\,P \mid Q] & \xrightarrow{C\,;V}_{\Xi_0,g_0} & n[\mathsf{to}\ m\,.\,P \mid Q \mid V]\\
n[\mathsf{to}\ m\,.\,P \mid Q] & \xrightarrow{\tau}_{\Xi_0,g_0} & n[\mathsf{to}\ m\,.\,P \mid Q'],
\end{array}$$

where $C \in \{\overline{\mathsf{in}\ n}, \overline{\mathsf{to}\ n}\}$, for some $p \neq m$ (since Q only contains m within capabilities), q, Q_1, Q_2, Q', since n is not (Ξ_0, g_0)-mobile.

— The only transitions of $m[R]$ are:

$$\begin{array}{lll}
m[R] & \xrightarrow{[\mathsf{out}\ m]\,;0}_{\Xi_0,g_0} & m[R_1] \mid p[R_2]\\
m[R] & \xrightarrow{[\mathsf{to}\ q]\,;V}_{\Xi_0,g_0} & q[R_1 \mid V] \mid m[R_2]\\
m[R] & \xrightarrow{C\,;V}_{\Xi_0,g_0} & m[R \mid V]\\
m[R] & \xrightarrow{\tau}_{\Xi_0,g_0} & m[R'],
\end{array}$$

where $C \in \{\overline{\mathsf{in}\ m}, \overline{\mathsf{to}\ m}\}$, for some $p \neq m$ (since R only contains m within capabilities), q, R_1, R_2, R', since m is not (Ξ_0, g_0)-mobile.

We only consider the case $T \xrightarrow{\overline{\mathsf{to}\ n}\,;V}_{\Xi,g} T'$ when

$$T \equiv (\nu m : g')(\nu \widetilde{p : g})(n[\mathsf{to}\ m\,.\,P \mid Q] \mid m[R] \mid Z)$$

and

$$T' \equiv (\nu m : g')(\nu \widetilde{p : g})(n[\mathsf{to}\ m\,.\,P \mid Q \mid V] \mid m[R] \mid Z).$$

In this case

$$U \equiv (\nu m : g')(\nu \widetilde{p : g})(n[Q] \mid m[P \mid R] \mid Z)$$

and

$$U' \equiv (\nu m : g')(\nu \widetilde{p : g})(n[Q \mid V] \mid m[P \mid R] \mid Z),$$

so $T'\mathcal{B}^{\Xi,g}U'$ through the context $(\nu \widetilde{p : g})([\ ]^{\Xi_0,g_0} \mid Z)$.

(4) This proof is similar to that of (3).

(5) The key observation is that no process can enter n as n is private and n does not occur in R. Also, R cannot offer communications since it is not a (Ξ', g')-communicator.

(6) Take

$$\mathcal{B}^{\Xi,g} = \mathcal{I}^{\Xi,g} \cup \{(\nu n : g')(n[P]), 0)\},$$

where n and P satisfy the hypothesis of (6).

We need to prove that, if $(\nu n : g')(n[P]) \xrightarrow{\lambda}_{\Xi,g} P'$, then $\lambda = \tau$. The transitions mentioning n are blocked by name restriction. No ambient can go out of $(\nu n : g')(n[P])$, since $n \notin AN(P)$. No process can go out of $(\nu n : g')(n[P])$ since P is not a (Ξ', g')-sender. Finally, $n[P]$ cannot move, since P is not a (Ξ', g')-mover. □

5. Examples

In this section we test the expressiveness of M^3 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 two well-known calculi for modelling concurrent (π-calculus) and distributed (Dπ-calculus) 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 actual name) that allows the process P to enter it.

We give two encodings: in the first the processes inside the firewall can also perform computations before the process P has entered the firewall; while in the second they are blocked until the process P has entered the firewall. In both cases we assume that the name *agent* occurs only once.

The first encoding is

$$\text{AGENT} = agent[(x : g_{fw} \to g_{agent})x.P]$$

$$\text{FW} = (\nu\, fw : g_{fw}) fw[\mathsf{to}\ agent.\langle \mathsf{to}\ fw\rangle \mid Q],$$

where we assume P and Q do not contain occurrences of *agent*, Q only contains occurrences of *fw* within capabilities and x does not occur in P.

In this encoding the system agent-plus-firewall is represented by the top-level process $(\nu\, agent : g_{agent})(\text{AGENT} \mid \text{FW})$. It can be typed with the group g_0 by assuming $agent : g_{agent}$ and $fw : g_{fw}$, where the groups are typed as follows:

$$g_{agent} : \mathsf{gr}(\{g_0\}, \emptyset, \{g_{fw}\}, g_{fw} \to g_{agent}))$$

$$g_{fw} : \mathsf{gr}(\{g_0\}, \emptyset, \{g_{agent}\}, \mathsf{shh}).$$

shh can be replaced by a suitable type if we allow communication within the ambient fw. Let Ξ contain the assumptions above, so the typing $\Xi \vdash (\nu\, agent : g_{agent})(\mathrm{AGENT} \mid \mathrm{FW}) : g_0$ holds. Note that the ambients fw and $agent$ are not (Ξ, g_0)-mobile.

So we have

$$\begin{aligned}
&(\nu\, agent : g_{agent})(\mathrm{AGENT} \mid \mathrm{FW}) \\
&\quad\equiv\ \ (\nu\, agent : g_{agent})(\nu\, fw : g_{gw}) \\
&\qquad\qquad (agent[(x : g_{fw} \to g_{agent})x.P] \mid fw[\mathsf{to}\ agent.\langle \mathsf{to}\ fw\rangle \mid Q]) \\
&\quad\cong^{\Xi,g_0}\ (\nu\, agent : g_{agent})(\nu\, fw : g_{gw}) \\
&\qquad\qquad (agent[(x : g_{fw} \to g_{agent})x.P \mid \langle \mathsf{to}\ fw\rangle] \mid fw[Q]) && \text{by Theorem 4.23(3)} \\
&\quad\cong^{\Xi,g_0}\ (\nu\, agent : g_{agent})(\nu\, fw : g_{gw})(agent[\mathsf{to}\ fw.P] \mid fw[Q]) && \text{by Theorem 4.23(5)} \\
&\quad\cong^{\Xi,g_0}\ (\nu\, agent : g_{agent})(\nu\, fw : g_{gw})(agent[\,] \mid fw[P \mid Q]) && \text{by Theorem 4.23(3)} \\
&\quad\equiv\ \ (\nu\, agent : g_{agent})(agent[\,]) \mid (\nu\, fw : g_{gw})(fw[P \mid Q]) && \text{since } agent \text{ does not} \\
& && \text{occur in } P \text{ or } Q \\
&\quad\cong^{\Xi,g_0}\ (\nu\, fw : g_{gw})fw[P \mid Q] && \text{by Theorem 4.23(6).}
\end{aligned}$$

In the second encoding we block, by means of a communication, the process Q until process P has entered the firewall:

$$\begin{aligned}
\mathrm{AGENT} &= agent[(x : g_{fw} \to g_{agent})x.\langle M\rangle P] \\
\mathrm{FW} &= (\nu\, fw : g_{fw})fw[\mathsf{to}\ agent.\langle \mathsf{to}\ fw\rangle \mid (y : W)Q]\,,
\end{aligned}$$

where W is the type of messages being exchanged within fw, M has type W, x does not occur in P and y does not occur in Q; if no communication takes place within the firewall, W is an arbitrary message type.

In this encoding the system agent-plus-firewall is again represented by the top-level process $(\nu\, agent : g_{agent})(\mathrm{AGENT} \mid \mathrm{FW})$. The types are as before, except for the type of g_{fw}, which must allow the communication, that is,

$$g_{fw} : \mathsf{gr}(\{g_0\}, \varnothing, \{g_{agent}\}, W)\,.$$

As in the previous case, one can show that

$$(\nu\, agent : g_{agent})(\mathrm{AGENT} \mid \mathrm{FW}) \cong^{\Xi,g_0} (\nu\, fw : g_{fw})(fw[\langle M\rangle P \mid (y : W)Q])\,,$$

and hence conclude by Theorem 4.23(5) that

$$(\nu\, agent : g_{agent})(\mathrm{AGENT} \mid \mathrm{FW}) \cong^{\Xi,g_0} (\nu\, fw : g_{fw})(fw[P \mid Q])\,.$$

Types play a crucial role in showing the correctness of this encoding, since, for example, we could not apply Theorem 4.23(3) if the $\mathscr{C}$ components of the g_{agent} and g_{fw} group types were not empty.

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

The Trojan horse attack

In this example we will show how our type system can detect a Trojan horse attack. Ulysses is naturally encoded as a mobile ambient that enters a *horse* ambient 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 process MYTH defined as:

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

If ambients $\mathit{ulysses}, \mathit{horse}, \ldots$ belong to groups $g_{\mathit{ulysses}}, g_{\mathit{horse}}, \ldots$, respectively, the whole MYTHic process can be typed by a group g_{myth} with respect to an environment Ξ that contains the following assumptions:

$$\begin{array}{rl}
g_{\mathit{ulysses}} & : \mathsf{gr}(\{g_{\mathit{myth}}, g_{\mathit{troy}}, g_{\mathit{horse}}\}, \{g_{\mathit{horse}}\}, \{g_{\mathit{palace}}\}, \mathsf{shh}) \\
g_{\mathit{horse}} & : \mathsf{gr}(\{g_{\mathit{myth}}, g_{\mathit{troy}}\}, \{g_{\mathit{troy}}\}, \varnothing, \mathsf{shh}) \\
g_{\mathit{troy}} & : \mathsf{gr}(\{g_{\mathit{myth}}\}, \varnothing, \varnothing, \mathsf{shh}) \\
g_{\mathit{palace}} & : \mathsf{gr}(\{g_{\mathit{troy}}\}, \varnothing, \varnothing, \mathsf{shh})\,.
\end{array}$$

It is clear from this 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.

Observe that without the $\mathscr{S}$ component the Trojans would have no way of knowing that Ulysses might enter Troy, since he enters hidden within the horse.

5.2. *Encoding process calculi*

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 asynchronous π-calculus given by the following grammar (we use letters *a*–*d* for channel names and *x*–*z* for variables):

$$P \ ::= \ \mathsf{0} \mid \xi(x\,{:}\,T)P \mid \xi\langle\xi'\rangle \mid (\nu c\,{:}\,T)P \mid P|Q \mid !\xi(x\,{:}\,T)P \mid !\xi\langle\xi'\rangle\,,$$

where ξ, ξ' are either a variable or a channel name and T ranges over a type hierarchy:

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

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

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

and structural equivalence and other reduction rules similar to those of M^3.

The type system, defined by the typing rules in Figure 12, derives judgements of the form $\Theta \vdash P$, where Θ is a set of assumptions of the form $c\,{:}\,Ch(T)$. The informal meaning of $\Theta \vdash P$ is that P is a well-typed π-process with respect to the environment Θ, that is, communication is well typed in P with respect to assumptions Θ. Note that the typing rules only guarantee the safeness of communication. We will use the labelled transition

$$\frac{}{\Theta \vdash 0} \qquad \frac{\xi : Ch(T) \in \Theta \quad \Theta, x : T \vdash P}{\Theta \vdash \xi(x : T)P} \qquad \frac{\xi : Ch(T) \in \Theta \quad \xi' : T \in \Theta}{\Theta \vdash \xi\langle \xi' \rangle}$$

$$\frac{\Theta, c : T \vdash P}{\Theta \vdash (\nu c : T)P} \qquad \frac{\Theta \vdash P \quad \Theta \vdash Q}{\Theta \vdash P \,|\, Q} \qquad \frac{\Theta \vdash \xi(x : T)P}{\Theta \vdash !\xi(x : T)P} \qquad \frac{\Theta \vdash \xi\langle \xi' \rangle}{\Theta \vdash !\xi\langle \xi' \rangle}$$

Fig. 12. Type system for π-calculus.

system of Section 4, based on our richer type system, to discuss the correctness of our translation.

As usual, the basic idea of the encoding is to represent each channel as an ambient: in particular, we will associate with each public π-calculus channel c, a private ambient c and a public ambient $\check{c}$; and when the channel c is private, both ambients c and $\check{c}$ will be private. The whole encoding of processes is placed within a private ambient l. A process prefixed with an input on a channel c is encoded as a mobile process that immediately goes (in)to the ambient c where communication will take place. Asynchronous outputs on channel c also go into ambient c. After the input is performed, the encoding creates a process that goes to the ambient $\check{c}$, moves $\check{c}$ out of ambient l and then moves it into l again, where it finally deposits the encoding of the continuation of the input action. The only reason for this movement of $\check{c}$ is to make the communication on public channels visible. The public ambients $\check{c}$ have all the same group g_{com}, while the private channel-ambients are typed by the encodings of the types of the corresponding channels. The movements of processes are encoded by means of auxiliary private ambients of group g_{aux}. The channel-ambients and the encoding proper $[\![P]\!]$ of the top-level π-calculus term P run within the ambient l of group g_l.

We choose a configuration like this, that is, with the ambient l containing both the encoding of the process and the ambient-channels, as this paves the way for the encoding of $D\pi$ in the next subsection, even though ambient-channels in parallel with l would make the encoding of the π-calculus slightly simpler.

The encoding is defined in a *polyadic* version of M^3, where tuples of values, instead of single values, are exchanged in communication. So a process offering an input has the form $(x_1 : W_1, ..., x_n : W_n)P$, and communication types are of the form $W_1 \times ... \times W_n$ or shh. All the notions introduced for the monadic calculus can be standardly generalised to the polyadic version. Of course, our encoding technique also works perfectly for the polyadic π-calculus, but, for simplicity, we will only describe the encoding of the monadic version.

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 $G_0, G_1, \ldots, G_{n-1}, \ldots$ defined by

$$\begin{array}{rclrcl} g_0 & : & G_0 & \text{with} \quad G_0 & = & \mathsf{gr}(\{g_l\}, \emptyset, \{g_{com}\}, \mathsf{shh}) \\ g_{j+1} & : & G_{j+1} & \text{with} \quad G_{j+1} & = & \mathsf{gr}(\{g_l\}, \emptyset, \{g_{com}\}, g_j \times g_{com}), \end{array}$$

where g_{com} is the group of all the public ambients.

$$\begin{array}{rcl}
\llbracket \xi(x{:}T)P \rrbracket & = & (\nu n{:}g_{aux})n[\mathsf{to}\ \xi.(x{:}\llbracket T\rrbracket, \check{x}{:}g_{com})\ \mathsf{to}\ \check{\xi}.\mathsf{out}\ l.\mathsf{in}\ l.\ \mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P\rrbracket] \\
\llbracket !\xi(x{:}T)P \rrbracket & = & !\llbracket \xi(x{:}T)P \rrbracket \\
\llbracket \xi\langle \xi' \rangle \rrbracket & = & (\nu n{:}g_{aux})n[\mathsf{to}\ \xi.\langle \xi', \check{\xi}' \rangle] \\
\llbracket !\xi\langle \xi' \rangle \rrbracket & = & !\llbracket \xi\langle \xi' \rangle \rrbracket \\
\llbracket (\nu c{:}T)P \rrbracket & = & (\nu c{:}\llbracket T\rrbracket)(\nu \check{c}{:}\llbracket T\rrbracket)(\llbracket P\rrbracket \mid c[\,] \mid \check{c}[\,]) \\
\llbracket P \mid Q \rrbracket & = & \llbracket P\rrbracket \mid \llbracket Q\rrbracket \\
\llbracket \mathsf{0} \rrbracket & = & \mathsf{0}
\end{array}$$

Fig. 13. Encoding of π-calculus.

The encoding of π-calculus types is then defined by

$$\llbracket Ch() \rrbracket = g_0 \qquad\qquad \llbracket Ch(T) \rrbracket = g_{j+1} \text{ if } \llbracket T \rrbracket = g_j\,.$$

In the following, Ψ_k denotes the group environment

$$\{g_0 : G_0, \ldots, g_k : G_k, g_l : G_l, g_{aux} : G_{aux}^{(k)}, g_{com} : G_{com}\},$$

where g_{aux} is the group of the auxiliary private ambients, and

$$\begin{array}{rcl}
G_l & = & \mathsf{gr}(\{g_*\}, \emptyset, \emptyset, \mathsf{shh}) \\
G_{aux}^{(k)} & = & \mathsf{gr}(\{g_*, g_l\}, \{g_l\}, \{g_1, \ldots, g_k, g_l\}, \mathsf{shh}) \\
G_{com} & = & \mathsf{gr}(\{g_*, g_l\}, \{g_l\}, \{g_{aux}\}, \mathsf{shh})\,,
\end{array}$$

and g_* is the group of the top-level M^3 processes (in our case, ambient-processes), that is, the type of what can be called the *top anonymous ambient*, where the ambient $l[\,]$ stays. Note that, instead of the unique group g_{aux}, we could have two different groups for the private ambients used in the encodings of inputs and outputs, respectively, and the group type for the output-encoding ambients could be the less permissive $\mathsf{gr}(\{g_l\}, \{\}, \{g_1, \ldots, g_k\}, \mathsf{shh})$. This more refined typing, however, would be useless here.

If Θ is a π-calculus environment, its encoding $\llbracket\Theta\rrbracket$ is the variable environment that contains the assumptions for all the public ambient names and all the private and public ambient variables corresponding to the channel names and to the variables of Θ, respectively, that is,

$$\llbracket\Theta\rrbracket = \{\check{c} : g_{com} \mid c : T \in \Theta\} \cup \{x : \llbracket T \rrbracket, \check{x} : g_{com} \mid x : T \in \Theta\}\,.$$

Let $\{c_1 : T_1, ..., c_h : T_h\}$ (denoted $\widetilde{c:T}^{(h)}$) contain the (finite) set of free channel names occurring in P. The global encoding of P, denoted $\mathcal{C}(P, \widetilde{c:T}^{(h)})$, is then the process of group g_* given by

$$\mathcal{C}(P, \widetilde{c:T}^{(h)}) \;=\; (\nu l : g_l)(\nu \widetilde{c : \llbracket T \rrbracket}^{(h)}) l[\llbracket P \rrbracket \mid c_1[\,] \mid \ldots \mid c_h[\,] \mid \check{c}_1[\,] \mid \ldots \mid \check{c}_h[\,]]\,,$$

where $\llbracket P \rrbracket$ is defined in Figure 13. Note that each variable x of the π-calculus is encoded using two variables $x, \check{x}$: the former corresponds to the private channel in which communications take place; while the latter corresponds to the associated public ambient.

Theorem 5.1 states that the translation respects types. Also, the translation is correct in the sense expressed by Theorem 5.2. Obviously, one reduction step in π-calculus is simulated by many reduction steps in M^3. Point 2 of Theorem 5.2 guarantees that any

M^3 reduction starting from the translation of a π-calculus term P must eventually end in an M^3 term Q' that is reduction barbed congruent to a π-calculus reduct of P.

Theorem 5.1. Let the judgment $\Theta \vdash P$ hold, let $\widetilde{c:T}^{(h)}$ contain the set of free channel names occurring in P, and let k be greater than or equal to the maximum nesting of $Ch()$ in types occurring in P and Θ. Then, we have

$$\Psi_k\,;\llbracket\Theta\rrbracket \vdash \mathcal{C}(P,\widetilde{c:T}^{(h)}) : g_* \,.$$

Proof. Let

$$\mathcal{C}(P,\widetilde{c:T}^{(h)}) = (\nu l:g_l)(\nu \widetilde{c:\llbracket T\rrbracket}^{(h)})(l[\llbracket P\rrbracket \mid c_1[\,] \mid \ldots \mid c_h[\,] \mid \check{c}_1[\,] \mid \ldots \mid \check{c}_h[\,]]) \,.$$

We prove by induction on the derivation of $\Theta \vdash P$ that

$$\Psi_k\,;\llbracket\Theta\rrbracket, \Delta \vdash \llbracket P\rrbracket : g_l \,,$$

where $\Delta = l:g_l,\ c_1:\llbracket T_1\rrbracket, \ldots,\ c_h:\llbracket T_h\rrbracket$. As a typical example, let us consider the rule for input:

$$\frac{\xi : Ch(T) \in \Theta \quad \Theta, x:T \vdash P'}{\Theta \vdash \xi(x:T)P'} \,.$$

We get by induction that $\Psi_k\,;\llbracket\Theta, x:T\rrbracket, \Delta \vdash \llbracket P'\rrbracket : g_l$. By definition, $\llbracket\Theta, x:T\rrbracket = \llbracket\Theta\rrbracket, x:\llbracket T\rrbracket, \check{x}:g_{com}$. Let $g_\xi = \llbracket Ch(T)\rrbracket$. Hence, $\xi:g_\xi \in \llbracket\Theta\rrbracket$, and $g_\xi : G_\xi \in \Psi_k$ for some G_ξ, such that $T(G_\xi) = \llbracket T\rrbracket \times g_{com}$. We have the following typing derivation for $\llbracket P\rrbracket$:

$$\dfrac{\dfrac{\Xi' \vdash \mathsf{to}\ \xi : g_\xi \to g_{aux} \qquad \dfrac{\dfrac{\dfrac{\mathcal{D}}{\Xi'' \vdash \mathsf{to}\ \check{\xi}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l : g_l \to g_\xi} \quad \Xi'' \vdash \llbracket P\rrbracket : g_l}{\Xi'' \vdash \mathsf{to}\ \check{\xi}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P\rrbracket : g_\xi}}{\Xi' \vdash (x:\llbracket T\rrbracket, \check{x}:g_{com})\mathsf{to}\ \check{\xi}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P'\rrbracket : g_\xi}}{\dfrac{\Xi' \vdash \mathsf{to}\ \xi.(x:\llbracket T\rrbracket, \check{x}:g_{com})\,\mathsf{to}\ \check{\xi}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P'\rrbracket : g_{aux}}{\dfrac{\Xi' \vdash n[\mathsf{to}\ \xi.(x:\llbracket T\rrbracket, \check{x}:g_{com})\,\mathsf{to}\ \check{\xi}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P'\rrbracket] : g_l}{\Xi \vdash (\nu n:g_{aux})n[\mathsf{to}\ \xi.(x:\llbracket T\rrbracket, \check{x}:g_{com})\,\mathsf{to}\ \check{\xi}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P'\rrbracket] : g_l}}}$$

where

$$\begin{aligned} \Xi &= \Psi_k\,;\llbracket\Theta\rrbracket, \Delta \\ \Xi' &= \Xi, n:g_{aux} \\ \Xi'' &= \Xi', x:\llbracket T\rrbracket, \check{x}:g_{com} \,, \end{aligned}$$

and $\mathcal{D}$ is a (standard) derivation of

$$\Xi'' \vdash \mathsf{to}\ \check{\xi}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l : g_l \to g_\xi \,.$$

The theorem then follows easily by first deriving the typing judgements $\Xi \vdash c_i[\,] : g_l$, $\Xi \vdash \check{c}_i[\,] : g_l$ for $1 \leqslant i \leqslant h$, and then by using the rules (Par), (Amb), and (AmbRes). □

Theorem 5.2. Let P be a term of the π-calculus such that the judgment $\Theta \vdash P$ holds, let $\widetilde{c:T}^{(h)}$ contain the set of free channel names occurring in P, and let k be greater than or equal to the maximum nesting of $Ch()$ in types occurring in P and Θ. Then, we have

1 If $P \rightarrow_{\pi} Q$, then $\mathcal{C}(P, \widetilde{c:T}^{(h)}) \rightarrow^{*} \mathcal{C}(Q, \widetilde{c:T}^{(h)})$.

2 If $\mathcal{C}(P, \widetilde{c:T}^{(h)}) \rightarrow^{*} Q$, then $Q \rightarrow^{*} Q'$ where $Q' \cong^{\Psi_k; \llbracket \Theta \rrbracket, g_*} \mathcal{C}(R, \widetilde{c:T}^{(h)})$, for some π-calculus process R such that $P \rightarrow_{\pi}^{*} R$.

Proof.

1 Consider, as an interesting case, the case where P is of the form $c(x:T)P' \mid c\langle a\rangle \mid S$, which reduces to $Q = P'\{x := a\} \mid S$ by performing a communication on the public channel c. By the definition of $\llbracket \cdot \rrbracket$, we have

$$\begin{aligned} \llbracket P \rrbracket \; = \; & (\nu n : g_{aux}) n[\mathsf{to}\ c.(x : \llbracket T \rrbracket, \check{x} : g_{com})\ \mathsf{to}\ \check{c}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P' \rrbracket] \mid \\ & (\nu n' : g_{aux}) n'[\mathsf{to}\ c.\langle a, \check{a}\rangle] \mid \llbracket S \rrbracket \,. \end{aligned}$$

Of course, c is a free variable of P, and in the term $\mathcal{C}(P, \widetilde{c:T}^{(h)})$ there are, by hypothesis, the ambients c (restricted) and $\check{c}$ (public) that run in parallel with the process $\llbracket P \rrbracket$. We therefore have the following reduction (we only show the relevant subprocesses):

$$\begin{aligned} & l[(\nu n : g_{aux}) n[\mathsf{to}\ c.(x : \llbracket T \rrbracket, \check{x} : g_{com})\ \mathsf{to}\ \check{c}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P' \rrbracket] \mid \\ & \qquad\qquad (\nu n' : g_{aux}) n'[\mathsf{to}\ c.\langle a, \check{a}\rangle] \mid c[\,] \mid \check{c}[\,] \mid \ldots] \\ \rightarrow^{*}\ & l[(\nu n : g_{aux}) n[\,] \mid (\nu n' : g_{aux}) n'[\,] \mid \check{c}[\,] \mid \\ & \qquad c[(x : \llbracket T \rrbracket, \check{x} : g_{com})\ \mathsf{to}\ \check{c}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P' \rrbracket \mid \langle a, \check{a}\rangle] \mid \ldots] \\ \rightarrow\ & l[(\nu n : g_{aux}) n[\,] \mid \check{c}[\,] \mid \\ & \qquad c[\mathsf{to}\ \check{c}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P' \rrbracket\{x, \check{x} := a, \check{a}\}] \mid \ldots] \\ \rightarrow\ & l[(\nu n : g_{aux}) n[\,] \mid \check{c}[\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P' \rrbracket\{x, \check{x} := a, \check{a}\}] \mid c[\,] \mid \ldots] \\ \rightarrow^{*}\ & l[(\nu n : g_{aux}) n[\,] \mid \check{c}[\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P' \rrbracket\{x, \check{x} := a, \check{a}\}] \mid c[\,] \mid \ldots] \\ \rightarrow^{*}\ & l[(\nu n : g_{aux}) n[\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P' \rrbracket\{x, \check{x} := a, \check{a}\}] \mid \check{c}[\,] \mid c[\,] \mid \ldots] \\ \rightarrow^{*}\ & l[\llbracket P' \rrbracket\{x, \check{x} := a, \check{a}\} \mid c[\,] \mid \check{c}[\,] \mid \ldots]\,, \end{aligned}$$

whereas $\llbracket Q \rrbracket$ is $\llbracket P'\{x := a\} \rrbracket \mid \llbracket S \rrbracket$.
An easy induction on the definition of the translation concludes the proof by showing that $\llbracket P'\{x := a\} \rrbracket = \llbracket P' \rrbracket\{x, \check{x} := a, \check{a}\}$.

2 Let $\rightarrow_{-}$ denote the reduction relation defined in the same way as $\rightarrow$ but without the communication rule (R-comm). Define $\mathcal{E}(P, \widetilde{c:T}^{(h)})$ as the set of all the processes Q such that $\mathcal{C}(P, \widetilde{c:T}^{(h)}) \rightarrow_{-}^{*} Q$. Note that this is the set of all the processes that can be obtained from $\mathcal{C}(P, \widetilde{c:T}^{(h)})$ by performing 'to' actions of the form

$$\begin{aligned} & (\nu n : g_{aux}) n[\mathsf{to}\ c.(x : \llbracket T \rrbracket, \check{x} : \llbracket T \rrbracket)\ \mathsf{to}\ \check{\xi}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P' \rrbracket] \mid c[Q'] \rightarrow \\ & \qquad (\nu n : g_{aux}) n[\,] \mid c[(x : \llbracket T \rrbracket, \check{x} : \llbracket T \rrbracket)\ \mathsf{to}\ \check{\xi}.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n.\mathsf{out}\ l.\mathsf{to}\ l.\llbracket P' \rrbracket \mid Q']\,. \end{aligned}$$

Processes that encode output actions can be treated similarly. By Theorem 4.23(4), all the processes in $\mathcal{E}(P, \widetilde{c:T}^{(h)})$ are reduction barbed congruent to $\mathcal{C}(P, \widetilde{c:T}^{(h)})$.
We will prove the following property by induction on n:

($\flat$) Let $\mathcal{C}(P,\widetilde{c:T}^{(h)}) \rightarrow^n Q$. Then all the $\rightarrow_{-}$-reduction sequences starting from Q must eventually reach an element of $\mathcal{E}(R,\widetilde{c:T}^{(h)})$ for some π-calculus process R such that $P \rightarrow_\pi^* R$.

The statement of the theorem will then follow immediately.

If $n = 1$ the proof is trivial, since in one step $\mathcal{C}(P,\widetilde{c:T}^{(h)})$ can only move into an element of $\mathcal{E}(P,\widetilde{c:T}^{(h)})$.

For the induction step, assume that $\mathcal{C}(P,\widetilde{c:T}^{(h)}) \rightarrow^n Q_0 \rightarrow Q$, and let $(\widetilde{\nu})$ stand for $(\nu l:g_l)(\nu\widetilde{c:[\![T]\!]}^{(h)})$. If the $n+1$-th reduction step is not a communication step, the proof is trivial by the induction hypothesis. Otherwise, note that communications can only take place in (restricted) ambients representing channels. So let

$$Q_0 = (\widetilde{\nu})\ (l[S_1 \mid c[U \mid V \mid S_2]] \mid S_3)\,,$$

where $U = (x:[\![T]\!], \check{x}:g_{com})\,\mathsf{to}\,\check{c}.\mathsf{out}\,l.\mathsf{in}\,l.\mathsf{to}\,n.\mathsf{out}\,l.\mathsf{to}\,l.[\![P']\!]$ and $V = \langle a, \check{a}\rangle$. Note that, by construction, $c \neq x$. By the induction hypothesis, any $\rightarrow_{-}$-reduction sequence starting from Q_0 must eventually reach a term of the form

$$Q_1 = (\widetilde{\nu})\ (l[S_1' \mid c[U \mid V \mid S_2']] \mid S_3')$$

belonging to $\mathcal{E}(R,\widetilde{c:T}^{(h)})$ for some R such that $P \rightarrow_\pi^* R$. Note that U and V cannot interfere with this reduction since they are blocked by their input and output actions, respectively.

This implies that $R = c(x:T)P' \mid c\langle a\rangle \mid S$ for some π-calculus process S where

$$(\widetilde{\nu})\ (l[S_1' \mid c[S_2']] \mid S_3') \in \mathcal{E}(S,\widetilde{c:T}^{(h)})\,. \tag{5.1}$$

Now the $n+1$-th step is a communication of the form

$$U \mid V \rightarrow U'\,,$$

where $U' = \mathsf{to}\,\check{c}.\mathsf{out}\,l.\mathsf{in}\,l.\mathsf{to}\,n.\mathsf{out}\,l.\mathsf{to}\,l.[\![P']\!]\{x,\check{x} := a,\check{a}\}$. Hence,

$$Q = (\widetilde{\nu})\ (l[S_1 \mid c[U' \mid S_2]] \mid S_3)\,.$$

Now note that U' can only perform a fixed sequence of moves and that, as n is a private ambient, it can only interact with the reduction $Q_0 \rightarrow_{-}^* Q_1$ through the moves involving the ambient $\check{c}$. In particular, in some step of the reduction, U' may not be able to perform the to action of entering $\check{c}$ because the ambient $\check{c}$ has been driven out of l by some other process. Note, however, that each process entering $\check{c}[\,]$ is of the shape $\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n''.Z$, where n'' is some private auxiliary ambient and Z is a process. So when $\check{c}$ is out of the ambient l, it can only perform an $\mathsf{in}\ l$ action and thus go back into l. Hence, before reaching a configuration in which no $\rightarrow_{-}$-reduction is possible, U' must eventually be able to enter $\check{c}$ to drive it out of l, and then back into l again, and to leave it without affecting the moves of the other processes involved in the reduction. Thus in the end we have

$$Q \rightarrow_{-}^* Q' = (\widetilde{\nu})\ (l[[\![P']\!]\{x,\check{x} := a,\check{a}\} \mid S_1' \mid c[S_2']] \mid S_3')\,,$$

$$
\begin{array}{lcl@{\qquad\qquad}lcl}
\textit{Thread} & & & \textit{System} & & \\
p, q, r & = & \mathsf{stop} & P, Q, R & = & \mathsf{0} \\
 & & \mathsf{go}\ w.p & & & l[P] \\
 & & u(x:T)p & & & (\nu_l\, c:T)p \\
 & & u\langle\xi\rangle & & & P \mid Q \\
 & & (\nu c:T)p & & & \\
 & & p \mid q & & & \\
 & & !u(x:T)p & & & \\
 & & !u\langle\xi\rangle & & &
\end{array}
$$

Fig. 14. Syntax of Dπ-calculus.

and from (5.1), we immediately have $Q' \in \mathcal{E}(P'\{x := a\} \mid S, \widetilde{c:T}^{(h)})$. Finally, note that $c(x:T)P' \mid c\langle a\rangle \mid S \rightarrow_\pi P'\{x := a\} \mid S$. □

Encoding the Dπ-calculus

We refer, essentially, to the version of Dπ presented in Hennessy and Riely (2002), with some simplifications (for instance the output is asynchronous) and a much easier type system. We also use a slightly different notation to provide uniformity with our treatment of the π-calculus. The basic syntactic categories are shown in Figure 14, where l, k range over locations, c over channels, u, v over channel names and variables, w over location names and variables, x over location and channel variables, and ξ over channel and location names or variables.

In this presentation we assume a very basic type system, which is similar to that of π and is aimed purely at preventing communication errors. The type syntax is

$$T ::= \mathsf{loc} \mid \mathsf{ch}(T).$$

The reduction semantics of the Dπ-calculus, denoted $\rightarrow_{\mathrm{D}}$, has, basically, the following reduction rules:

$$
\begin{array}{rcl}
l[c(x:T)p] \mid l[c\langle\xi\rangle] & \rightarrow_{\mathrm{D}} & l[p\{x := \xi\}] \\
l[\mathrm{go}\ l'.p] & \rightarrow_{\mathrm{D}} & l'[p]\,.
\end{array}
$$

As usual, the operational semantics is given together with rules that define the structural equivalence. In addition to the standard rules, it is worth mentioning the following rules, which are peculiar to Dπ and state that two locations with the same name are the same location (unlike M^3), and that we can extend the scope of a channel restriction outside a location, provided we keep the information about the location itself:

$$l[p \mid q] \equiv l[p] \mid l[q] \tag{5.2}$$

$$l[(\nu c:\mathsf{ch}(T))\ p] \equiv (\nu_l\, c:\mathsf{ch}(T))\ l[p]\,. \tag{5.3}$$

The only function of the type system is to ensure the consistency of the values passed in channels. The typing rules we consider are given in Figure 15. In our version of Dπ, both channel and location names or variables can be communicated, but channels

THREAD

$$\frac{}{\Omega \vdash_w \mathsf{stop}} \qquad \frac{\Omega \vdash_w p}{\Omega \vdash_{w'} \mathsf{go}\; w.p} \qquad \frac{w : \mathsf{loc} \in \Omega}{\Omega, u@w : Ch(T) \vdash_w u : Ch(T)} \qquad \frac{\Omega \vdash_w p \quad \Omega \vdash_w q}{\Omega \vdash_w p \mid q}$$

$$\frac{\Omega, x : \mathsf{loc} \vdash_w u : Ch(\mathsf{loc}) \quad \Omega, x : \mathsf{loc} \vdash_w p \quad x \notin \Omega}{\Omega \vdash_w u(x : \mathsf{loc})\; p} \qquad \frac{\Omega \vdash_w u : Ch(T) \quad \Omega, x@w' : T \vdash_w p}{\Omega \vdash_w u(x : T)\; p}$$

$$\frac{\Omega \vdash_w u : Ch(T) \quad \Omega \vdash_w \xi : T}{\Omega \vdash_w u\langle \xi \rangle} \qquad \frac{\Omega, c@w : Ch(T) \vdash_w p}{\Omega \vdash_w (\nu c : Ch(T))p} \qquad \frac{\Omega \vdash_w u(x : T)\; p}{\Omega \vdash_w !u(x : T)\; p} \qquad \frac{\Omega \vdash_w u\langle \xi \rangle}{\Omega \vdash_w !u\langle \xi \rangle}$$

SYSTEM

$$\frac{}{\Omega \vdash \mathsf{0}} \qquad \frac{\Omega \vdash_l p}{\Omega \vdash l[p]} \qquad \frac{\Omega, c@l : \mathsf{ch}(T) \vdash P}{\Omega \vdash (\nu_l c : \mathsf{ch}(T))\; P} \qquad \frac{\Omega \vdash P \quad \Omega \vdash Q}{\Omega \vdash P \mid Q}$$

Fig. 15. Type system for Dπ-calculus.

must always be *located*, that is, they must be explicitly assigned to locations. In type environments, denoted by Ω, types are assigned to channels by statements like $u@w : T$, where u represents the name of the channel and w the name of the location in which u is located. Note that in a type environment we can have $u@w : T$ and $u@w' : T'$ with $w \neq w'$. The type of locations is simply loc, and is assigned to names in statements of the form $w : \mathsf{loc}$.

The typing judgments are of two kinds:

— $\vdash_w$ to represent assignments for threads relative to the location w (note that w can also be a variable); and

— $\vdash$ to represent assignments for systems.

This allows us to type those systems where the same channel name has different types in different locations.

Note that if a subexpression of a thread is typed with a statement of the form $\Omega, u@x : Ch(T) \vdash_x p$, then either u is a variable or a channel name that will eventually be restricted in $\vdash_x$. In fact, since x is a variable, p must occur in the final system in the scope of an input $(x : \mathsf{loc})$ (there are no location variables at the system level). But this can be done only if the statement $u@x : Ch(T)$ is eventually removed from the type environment (by the condition $x \notin \Omega$ in the rule for abstraction of locations), and this can happen only if u is a variable (input), or u is restricted. This observation will be useful for understanding our translation.

As in the encoding of the π-calculus, each channel is represented by two ambients: one private and the other public if the channel is public; both private otherwise. For uniformity, to allow the same public channel names at different locations, we need to choose the names of the representing ambients carefully. If a channel is declared as $c@l : Ch(T)$ in the environment or in a restriction, we represent it with the ambients c_l

and $\check{c}_l$. If a channel is declared as $c@x:Ch(T)$, we represent it with the ambients c and $\check{c}$. Note that in this last case, c must be restricted, as observed above. In order that we can always communicate two values, we also associate two ambient names l and $\check{l}$ with each location name l.

We define the following group and group types:

$$\begin{aligned} g_0 &= g_{loc}:G_{loc} = \mathsf{gr}(\{g_D\},\emptyset,\{g_{loc}\},\mathsf{shh}) \\ g_1:G_1 &= \mathsf{gr}(\{g_{loc}\},\emptyset,\{g_{aux},g_{com}\},g_{loc}\times g_{loc}) \\ g_{i+2}:G_{i+2} &= \mathsf{gr}(\{g_{loc}\},\emptyset,\{g_{aux},g_{com}\},g_{i+1}\times g_{com}) \\ g_{aux}:G^{(k)}_{aux} &= \mathsf{gr}(\{g_{loc},g_D\},\{g_1,\ldots,g_k,g_{loc}\},\{g_{loc}\},\mathsf{shh}) \\ g_{com}:G_{com} &= \mathsf{gr}(\{g_{loc},g_D\},\{g_{loc}\},\{g_{aux}\},\mathsf{shh}), \end{aligned}$$

where g_{loc} is the group of locations, g_D is the group of the whole system, g_1 is the group of ambients representing channels that send locations, g_{i+2} is the group of ambients representing channels that send channels of group g_{i+1}, and g_{aux}, g_{com} are as in the translation of the π-calculus.

The Dπ types are encoded by

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

We can assume, without loss of generality, that, in typing judgements for systems, the type environments do not contain variables (variables can only occur in threads). The translation $[\![\Omega]\!]$ of such an environment Ω is defined as the minimal set such that:

— for each statement $c@l:Ch(T)\in\Omega$, there is one statement $\check{c}_l:g_{com}\in[\![\Omega]\!]$;
— for each statement $l:\mathsf{loc}\in\Omega$, there are two statements $l:g_{loc}\in[\![\Omega]\!]$ and $\check{l}:g_{loc}\in[\![\Omega]\!]$.

Similarly, we define the translation $[\![\Omega]\!]^w$relative to a location w as the minimal set such that:

— for each statement $u@w':Ch(T)\in\Omega$, there is one statement $\phi:g_{com}\in[\![\Omega]\!]^w$, where $\phi=\check{u}_{w'}$ if u is a channel name and w' is a location name, and $\phi=\check{u}$ otherwise;
— for each statement $w':\mathsf{loc}\in\Omega$, there are the two statements $w':g_{loc}\in[\![\Omega]\!]^w$ and $\check{w}':g_{loc}\in[\![\Omega]\!]^w$.

The basic ideas behind the translation are as follows. As anticipated above, each public channel is encoded by two ambients, one public and the other private, exactly as in the π-calculus translation. Communication takes place in private channels only; public channels are just used to make the communication events observable. The migration of processes into private channels is observationally invisible, as stated by Theorem 4.23(3). Each location l is represented by an ambient l that contains as subambients the encodings of channels used in communications local to l.

We only consider systems P of the form

$$l_1[p_1]\,|\,\ldots\,|\,l_n[p_n]\,, \tag{5.4}$$

where $l_i\neq l_j$ for all $1\leqslant i,j\leqslant n$. This is not restrictive, since every system can be transformed into this form by applying the structural rules (5.2) and (5.3).

$$
\begin{array}{rcl}
\llbracket \mathrm{stop} \rrbracket^{w} & = & \mathsf{0} \\
\llbracket \mathrm{go}\ w' . p \rrbracket^{w} & = & \mathsf{to}\ w' . \llbracket p \rrbracket^{w'} \\
\llbracket u(x{:}\,T)p \rrbracket^{w} & = & (\nu n : g_{aux}) n[\mathsf{to}\ \phi . (x{:}\,\llbracket T \rrbracket, \check{x} : \overset{\diamond}{g}) \mathsf{to}\ \check{\phi} . \mathsf{out}\ w . \mathsf{in}\ w . \mathsf{to}\ n . \mathsf{out}\ w . \mathsf{to}\ w . \llbracket p \rrbracket^{w}] \\
\llbracket u\langle v \rangle \rrbracket^{w} & = & (\nu n : g_{aux}) n[\mathsf{to}\ \phi . \langle v, \check{v} \rangle] \\
\llbracket (\nu c : \mathsf{ch}(T))\ p \rrbracket^{w} & = & (\nu \psi : g_{i+1})(\nu \check{\psi} : g_{com})\ (\llbracket p \rrbracket^{w} \mid \psi[\,] \mid \check{\psi}[\,]) \\
\llbracket p \mid q \rrbracket^{w} & = & \llbracket p \rrbracket^{w} \mid \llbracket q \rrbracket^{w} \\
\llbracket !\, u(x{:}\,T)\ p \rrbracket^{w} & = & !\, \llbracket u(x{:}\,T)\ p \rrbracket^{w} \\
\llbracket ! u\langle \xi \rangle \rrbracket^{w} & = & !\, \llbracket u\langle \xi \rangle \rrbracket^{w}
\end{array}
$$

$$
\begin{array}{rcl}
\text{where} \quad \overset{\diamond}{g} & = & \begin{cases} g_{loc} & \text{if } \llbracket T \rrbracket = g_{loc}, \\ g_{com} & \text{otherwise.} \end{cases} \\
\phi & = & \begin{cases} u_{w} & \text{if } u \text{ is a channel name and } w \text{ a location name;} \\ u & \text{if } u \text{ or } w \text{ are variables}^{\dagger} \end{cases} \\
\llbracket T \rrbracket & = & g_{i} \\
\psi & = & \begin{cases} c_{w} & \text{if } w \text{ is a location name;} \\ c & \text{if } w \text{ is a variable.} \end{cases}
\end{array}
$$

Fig. 16. Encoding of Dπ-calculus.

Let P be a system of the form (5.4) such that $\Omega \vdash P$. Assume that location l_1 has free channels $c^1@l_1, \ \ldots, c^{j_1}@l_1, \ j_1 \geqslant 0, \ \ldots,$ and location l_n has free channels $c^1@l_n, \ldots, \ c^{j_n}@l_n, \ j_n \geqslant 0$. Also, let $m_1, \ldots, m_r$ be all the location names occurring in P that are distinct from $l_1, \ldots, l_n$ (that is, they occur only as outputs and/or as arguments of the go action) and such that $m_i \neq m_j$ for all $1 \leqslant i, j \leqslant r$. In Ω there must be a typing statement for each of these locations and channels, so we can assume that Ω characterises the locations and free channels of the system. We then define

$$
\begin{array}{rcl}
\mathcal{D}(P, \Omega) & = & (\nu c^{1}_{l_1} : \llbracket T^{1}_{1} \rrbracket) \ldots (\nu c^{j_1}_{l_1} : \llbracket T^{J_1}_{1} \rrbracket) \ldots (\nu c^{1}_{l_n} : \llbracket T^{1}_{n} \rrbracket) \ldots (\nu c^{j_n}_{l_n} : \llbracket T^{j_n}_{n} \rrbracket) \\
 & & (l_1[\llbracket p_1 \rrbracket^{l_1} \mid c^{1}_{l_1}[\,] \mid \ldots \mid c^{j_1}_{l_1}[\,] \mid \check{c}^{1}_{l_1}[\,] \mid \ldots \mid \check{c}^{j_1}_{l_1}[\,]] \mid \ldots \\
 & & \mid l_n[\llbracket p_n \rrbracket^{l_n} \mid c^{1}_{l_n}[\,] \mid \ldots \mid c^{j_n}_{l_n}[\,] \mid \check{c}^{1}_{l_n}[\,] \mid \ldots \mid \check{c}^{j_n}_{l_n}[\,]]) \mid \\
 & & m_1[\,] \mid \ldots m_r[\,],
\end{array}
$$

where $\llbracket\ \rrbracket^{w}$ is defined in Figure 16.

For example, let

$$
\begin{array}{rcl}
p & = & c(x : \mathsf{loc}) p_0 \\
p_0 & = & \mathrm{go}\ x . (\nu c_1 @ x : Ch(\mathsf{loc}))(p_1 \mid p_2) \\
p_1 & = & c_1 \langle l \rangle \\
p_2 & = & c_1(y : \mathsf{loc}) \\
q & = & c\langle m \rangle \\
\Omega_0 & = & l : \mathsf{loc}, \ m : \mathsf{loc}, \ c@l : Ch(\mathsf{loc}).
\end{array}
$$

[†] Recall that if w is a variable, u is either a variable or a name that will eventually be restricted.

The encoding of $P = l[p \,|\, q]$ in the environment Ω_0 is defined through the following steps:

$$\begin{array}{rcll}
\llbracket c_1\langle l\rangle \rrbracket^x & = & (\nu n_1 : g_{aux})(n_1[\mathsf{to}\ c_1.\langle l, \check{l}\rangle]) & = \ P_1 \\
\llbracket c_1(y : g_{loc}) \rrbracket^x & = & (\nu n_2 : g_{aux})(n_2[\mathsf{to}\ c_1.(y : g_{loc}, \check{y} : g_{loc}) & \\
 & & \quad \{\mathsf{to}\ \check{c}_1.\mathsf{out}\ x.\mathsf{in}\ x.\mathsf{to}\ n_2.\mathsf{out}\ x.\mathsf{to}\ x]) & = \ P_2 \\
\llbracket (\nu c_1 @ x : Ch(\mathsf{loc}))(p_1 \,|\, p_2) \rrbracket^x & = & (\nu c_1 : g_1)(\nu \check{c}_1 : g_{com})(P_1 \,|\, P_2 \,|\, c_1[\,] \,|\, \check{c}_1[\,]) & \\
\llbracket \mathsf{go}\ x.(\nu c_1 @ x : Ch(\mathsf{loc}))(p_1 \,|\, p_2) \rrbracket^l & = & \mathsf{to}\ x.(\nu c_1 : g_1)(\nu \check{c}_1 : g_{com})(P_1 \,|\, P_2 \,|\, c_1[\,] \,|\, \check{c}_1[\,]) & = \ P_0 \\
\llbracket c(x : g_{loc}) \,.\, p_0 \rrbracket^l & = & (\nu n_3 : g_{aux})(n_3[\mathsf{to}\ c_l.(x : g_{loc}, \check{x} : g_{loc}) & \\
 & & \quad \mathsf{to}\ \check{c}_l.\mathsf{out}\ l.\mathsf{in}\ l.\mathsf{to}\ n_3.\mathsf{out}\ l.\mathsf{to}\ l \,.\, P_0]) & = \ P_3 \\
\llbracket c\langle m\rangle \rrbracket^l & = & (\nu n_4 : g_{aux})(n_4[\mathsf{to}\ c_l.\langle m, \check{m}\rangle]) & = \ P_4
\end{array}$$

which leads to

$$\mathcal{D}(P, \Omega_0) \ = \ (\nu c_l : Ch(\mathsf{loc}))\,(l[P_3 \,|\, P_4]) \,|\, m[\,]\,.$$

The correctness of the translation, in the same sense as for the encoding of the π-calculus, is ensured by analogous theorems.

Let Ψ^{D}_k be the group environment

$$\{g_{loc} : G_{loc}, g_1 : G_1, \ldots, g_k : G_k, g_{aux} : G^{(k)}_{aux}, g_{com} : G_{com}\}\,.$$

Lemma 5.3. Let $\Omega \vdash_w p$. Then we have $\Psi^{\mathrm{D}}_k; \llbracket \Omega \rrbracket^w \vdash \llbracket p \rrbracket^w : g_{loc}$, where k is greater than or equal to the maximum nesting of $Ch()$ in types occurring in p, Ω.

Proof. The proof is a standard induction on the proof of $\Omega \vdash_w p$ similar to the proof of Theorem 5.1. □

From Lemma 5.3 and the definition of $\llbracket \Omega \rrbracket$, we immediately get the type correctness of our translation.

Theorem 5.4. Let P be a Dπ system such that $\Omega \vdash P$ and let k be greater than or equal to the maximum nesting of $Ch()$ in types occurring in P, Ω. Then we have

$$\Psi^{\mathrm{D}}_k; \llbracket \Omega \rrbracket \vdash \mathcal{D}(P, \Omega) : g_D.$$

Theorem 5.5. Let P be a Dπ system such that $\Omega \vdash P$ and let k be equal to the maximum nesting of $Ch()$ in types occurring in P, Ω. Then we have:

1 If $P \rightarrow_D Q$, then $\mathcal{D}(P, \Omega) \ \rightarrow^* \ \mathcal{D}(Q, \Omega)$.

2 If $\mathcal{D}(P, \Omega) \ \rightarrow^* \ Q$, then $Q \ \rightarrow^* \ Q'$ and $P \rightarrow^*_D R$ with $Q' \cong^{\Psi^{\mathrm{D}}_k; \llbracket \Omega \rrbracket, g_D} \mathcal{D}(R, \Omega)$, for some M^3 process Q' and $D\pi$ system R.

Proof.

1 The proof is just an exercise in reduction similar to that in the proof of Theorem 5.2(1).

2 Again, the proof is similar to the proof of Theorem 5.2(2). Let $\rightarrow_=$ denote the reduction relation defined in the same way as $\rightarrow_-$, but without reduction of actions of the form '$\mathsf{to}\ l$' with l a location name. Let $\mathcal{E}^D(P, \Omega)$ be the set of all M^3 processes Q such that $\mathcal{D}(P, \Omega) \rightarrow^*_= Q$. In this case we also prove by induction on n that if $\mathcal{D}(P, \Omega) \rightarrow^n Q$ holds, then all the $\rightarrow_=$-reduction sequences starting from Q must eventually reach an element of $\mathcal{E}^D(R, \Omega)$ for some $D\pi$ system R such that $P \rightarrow^*_D R$. The proof is similar to that of property ($\flat$) in the proof of Theorem 5.2(2). □

6. Type inference

Although M^3 is a typed language, for its use in the design of a computational system it would be helpful, following the consolidated approach of functional languages like ML, to be able to define a first draft of the system by focusing purely on the computational behaviours of processes, without bothering about types. It is then crucial to be able to check whether this draft can be completed with type information so as to become a well-formed system, and whether the inserted type constraints are consistent with the intended behaviours.

In this section we present a type inference algorithm for M^3: the algorithm builds a typing whenever possible, and fails otherwise. Type inference allows us to complete the definition of a process when type annotations are incomplete or even missing. From another point of view, type inference can be seen as an analysis tool that provides information about the mobility properties of raw processes (represented by the $\mathscr{S}$, $\mathscr{C}, \mathscr{E}$ components of group types), and checks the consistency of values exchanged in communications.

Given a raw term R, that is, a pre-term in which all type annotations have been erased, our algorithm either fails or computes an environment scheme (defined below) Ξ and a well-typed version P of R (obtained by assigning type schemes – as defined in Figure 17 – to the bound names occurring in R) such that $\Xi \vdash P : g$ for some group g. Note that the algorithm succeeds whenever R can be obtained from a well-formed process by erasing the type information. This is formalised in Theorem 6.17. Furthermore, Ξ, P, g are 'most general', in the sense that all other typings Ξ', P', g' (that can be given to processes P' obtained from R by adding type annotations) can be derived from Ξ, P, g using substitution, weakening and a kind of subtyping.

Note that our notion of most general typing, although in some sense classical (see, for instance, Hindley (1969)), is formally different from that of Wells (2002), where 'principal' typing is defined using a partial order between typings, independently of terms. The reason is that Wells (2002) deals with inference systems where types are assigned to initially untyped terms. This means that no type commitment is written within terms, and all typing information is contained in the environment scheme and final type scheme, so it is sensible to consider the intended typings as pairs of an environment scheme and a type scheme, without reference to the terms to which they are assigned. In our system, however, types are also written within terms, and this information cannot, in general, be retrieved from the environment scheme and the final type scheme: the complete type information about a raw term can only be obtained by taking the term itself into consideration as well. Apart from these technicalities, however, the two definitions represent essentially the same concept.

Type variables, substitutions and environment operations

We first introduce some technical tools, which will be useful in defining the inference procedure and its properties. The inference algorithm deals with type variables, substitutions, unifiers and the merging of type environment schemes; moreover, a partial order relation

$\mathtt{W}$	$::=$		message type schemes
		t	type variable
		W	message type
$\mathtt{T}$	$::=$		communication type schemes
		t	type variable
		T	communication type
$\mathtt{G}$	$::=$	$\mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, \mathtt{T})$	group type schemes

Fig. 17. Type schemes.

on group type schemes and type environment schemes is needed for the derivation of a principal typing property.

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

In order to describe and prove properties of the inference algorithm, the original M^3 syntax of communication types has to be extended by a set $\mathcal{V}_{\mathcal{T}}$ of type variables (denoted by t). The resulting set is, in the standard nomenclature (Hindley 1969), a set of *type schemes*.

The new syntax of type schemes, where most of the syntactic sugar has been eliminated, is shown in Figure 17, starting from the definitions in Figure 4.

The inference algorithm only builds environment schemes, which are environments with type schemes instead of types. More precisely, an *environment scheme* consists of a *group environment scheme*, containing group type schemes instead of types, and a *variable environment scheme*, which, in addition to message types, may also include type variables.

As usual, in computing type schemes and type environment schemes, the algorithm is driven by the syntax of the raw term. It has, therefore, to put together distinct environment schemes whenever the term has more than one subterm. A fresh group name is assigned to each name, but when different environment schemes 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 type variables to communication type schemes. Let $\mathtt{T}$ be the set of communication type schemes as defined in Figure 17.

Definition 6.1. An *inference substitution* (substitution for short) is a finite mapping in $(\mathbb{G} \to \mathbb{G}) \cup (\mathcal{V}_{\mathcal{T}} \to \mathtt{T})$. Let φ_i range over $\mathbb{G} \cup \mathcal{V}_{\mathcal{T}}$ and A_i over $\mathbb{G} \cup \mathtt{T}$. 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

$$\sigma(\varphi) = \begin{cases} A_j & \text{if } \varphi = \varphi_j \text{ for some } 1 \leqslant j \leqslant n \\ \varphi & \text{otherwise}. \end{cases}$$

The application of a substitution σ to a variable environment scheme Δ (denoted $\sigma(\Delta)$) is defined in the standard way. Well-formed variable environment schemes are closed under substitution. In applying substitutions to group environment schemes, we must take into account the fact that in a statement $g:\mathtt{G}$, group names also occur in the subjects (left-hand sides) of the statements. Taking a well-formed group environment scheme Γ and an arbitrary substitution σ, the group environment scheme $\sigma(\Gamma)$ may not be well formed. For instance, if $g_1:\mathtt{G}_1, g_2:\mathtt{G}_2 \in \Gamma$, we could have $\sigma(g_1) = \sigma(g_2)$ but $\sigma(\mathtt{G}_1) \neq \sigma(\mathtt{G}_2)$. Consequently, type inference is not, in general, closed under substitution. However, one can easily see that if $\Gamma; \Delta \vdash P:g$ and $\sigma(\Gamma)$, $\sigma(\Delta)$ are well formed, then $\sigma(\Gamma); \sigma(\Delta) \vdash \sigma(P):\sigma(g)$ holds.

When two different environment schemes are put together, it may be necessary to *unify* different communication type schemes and group names. This is impossible if they are an ambient type and a capability type: in which case we get a *failure*. Therefore, in the definitions below, operations on type schemes and environment schemes may yield an *undefined* result denoting failure. A failure occurs when none of the cases considered in definitions can be applied. Failure propagates: an operation produces an *undefined* result whenever some step in its evaluation gives rise to *undefined*. To simplify the notation, we assume that failure propagation is implicit, and will not be shown explicitly in the definitions.

Definition 6.2. We use $\phi(\{(A_i, A'_i)|1 \leqslant i \leqslant n\})$ to denote the *two-sorted most general unifier* of the set of equations $\{A_i = A'_i \mid 1 \leqslant i \leqslant n\}$, if it exists, which is a substitution according to Definition 6.1. We will simply call $\phi(\{(A_i, A'_i)|1 \leqslant i \leqslant n\})$ the *unifier* of $\{(A_i, A'_i)|1 \leqslant i \leqslant n\}$.

Let Δ, Δ' be two variable environment schemes. We use (Δ, Δ') to denote the set

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

We will write $\phi(\Delta, \Delta')$ as a shortened form 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 scheme Γ gives an environment scheme 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 following, we will show how to recover a well-formed group environment scheme after the application of substitutions. This task is performed in two steps:

1 by unifying the communication type schemes in different type assumptions for the same group name (*completion-unification*, Definition 6.3);

2 by merging (through componentwise set union) group type schemes having the same communication type scheme (*compression*, Definition 6.5).

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

We say that a group environment scheme Γ is *consistent* if for all $g:\mathtt{G}_1, g:\mathtt{G}_2 \in \Gamma$, we have $T(\mathtt{G}_1) = T(\mathtt{G}_2)$. This condition does not imply $\mathtt{G}_1 = \mathtt{G}_2$, so consistent environment schemes are in general not well formed.

Definition 6.3. Let Γ be an arbitrary group environment scheme. The *completion-unifier* of Γ, denoted $\Sigma[\Gamma]$, is the substitution defined inductively as follows:

1 If Γ is consistent, then $\Sigma[\Gamma]$ is the empty substitution.
2 Otherwise, $\Sigma[\Gamma]$ is the substitution $\Sigma[\sigma(\Gamma)] \circ \sigma$, where

$$\sigma = \phi\{(T(\mathtt{G}), T(\mathtt{G}')) \mid \exists g \,.\, g : \mathtt{G}, g : \mathtt{G}' \in \Gamma \text{ such that } T(\mathtt{G}) \neq T(\mathtt{G}')\}.$$

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

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

Lemma 6.4. Let Γ be an arbitrary group environment scheme. If $\Sigma[\Gamma]$ is defined, then:

1 $\Sigma[\Gamma](\Gamma)$ is consistent.
2 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.

1 This part of the lemma follows by definition.
2 Assume that $\Sigma[\Gamma]$ is obtained by applying step 2 of Definition 6.3 n times. The proof is by induction on n.
For $n = 0$, the lemma is trivially true.
For $n > 0$, let

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

Note that since $\sigma(\Gamma)$ is consistent, we must have $\sigma(\mathtt{T}) = \sigma(\mathtt{T}')$ for all pairs $(\mathtt{T}, \mathtt{T}') \in S$. If σ_0 is the most general unifier of S, we must have $\sigma = \sigma' \circ \sigma_0$ for some substitution σ'. So we have that $\sigma(\Gamma) = \sigma'(\sigma_0(\Gamma))$. Now observe that $\Sigma[\sigma_0(\Gamma)]$ is defined in $n-1$ steps. By the induction hypothesis, there is a σ'' such that $\sigma' = \sigma'' \circ \Sigma[\sigma_0(\Gamma)]$. Since $\Sigma[\Gamma] = \Sigma[\sigma_0(\Gamma)] \circ \sigma_0$ by definition, the result is proved. □

The final (non-recursive) step required to transform a consistent environment scheme into a well-formed one is simply to put together, by set union, the group type schemes of all the different assumptions for the same group name.

Let $\mathtt{G}_1 = \mathsf{gr}(\mathscr{S}_1, \mathscr{C}_1, \mathscr{E}_1, \mathtt{T})$ and $\mathtt{G}_2 = \mathsf{gr}(\mathscr{S}_2, \mathscr{C}_2, \mathscr{E}_2, \mathtt{T})$. Let the union between group type schemes be defined by

$$\mathtt{G}_1 \cup \mathtt{G}_2 = \mathsf{gr}(\mathscr{S}_1 \cup \mathscr{S}_2, \mathscr{C}_1 \cup \mathscr{C}_2, \mathscr{E}_1 \cup \mathscr{E}_2, \mathtt{T}).$$

It is easy to see that this is associative and commutative.

Definition 6.5. The *compression* $\uplus(\Gamma)$ of a consistent group environment scheme Γ is the group environment scheme defined by

$$\uplus(\Gamma) = \{g : \bigcup_{i \in I_g} \mathtt{G}_i \;\mid\; g \in Dom(\Gamma),\ I_g = \{i \;\mid\; g : \mathtt{G}_i \in \Gamma\}\}.$$

One can easily check the following lemma.

Lemma 6.6. If Γ is consistent, then $\uplus(\Gamma)$ is well formed.

In order to discuss the properties of completion-unification and compression (and of the algorithm), we introduce a partial order on type schemes and environment schemes.

Definition 6.7. The relation $\leqslant$ on group type schemes is defined by

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

if $\mathscr{S} \subseteq \mathscr{S}'$, $\mathscr{C} \subseteq \mathscr{C}'$, $\mathscr{E} \subseteq \mathscr{E}'$ and $\mathtt{T} = \mathtt{T}'$.

This partial order is extended monotonically to arbitrary environment schemes by adding set inclusion:

$$\begin{array}{rl} \Gamma \leqslant \Gamma' & \text{if} \quad \forall\,(g:\mathtt{G}) \in \Gamma\,.\,\exists\,(g:\mathtt{G}') \in \Gamma'\,.\,\mathtt{G} \leqslant \mathtt{G}' \\ \Gamma;\Delta \leqslant \Gamma';\Delta' & \text{if} \quad \Gamma \leqslant \Gamma' \text{ and } \Delta \subseteq \Delta'. \end{array}$$

$\Delta \subseteq \Delta'$ reflects the absence of subtyping between communication type schemes.

The meaning of the relation $\Gamma \leqslant \Gamma'$ is that Γ' is more permissive than Γ on $Dom(\Gamma)$, but it can contain statements that are not present in Γ. 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.

The following properties of $\leqslant$ are useful.

Lemma 6.8.

1 If Ξ is well formed and $\Xi' \leqslant \Xi$, then Ξ' is consistent.

2 Let Γ be a consistent group environment scheme such that $\sigma(\Gamma) \leqslant \Gamma'$, where Γ' is well formed. Then $\sigma(\uplus(\Gamma)) \leqslant \Gamma'$ also.

Proof.

1 This part of the lemma is trivial since $\leqslant$ on group type schemes implies identity of communication type schemes.

2 If $I_g = \{i \mid g:\mathtt{G}_i \in \Gamma\}$, then $\bigcup_{i\in I_g} \mathtt{G}_i$ is the group type scheme assigned to g in $\uplus(\Gamma)$. Now let $\sigma(g):\mathtt{G}' \in \Gamma'$. By hypothesis, we have that $\sigma(\mathtt{G}_i) \leqslant \mathtt{G}'$ for all $i \in I_g$, so $\sigma(\bigcup_{i\in I_g} \mathtt{G}_i) \leqslant \mathtt{G}'$ also. □

Type inference algorithm

We now give an algorithm for reconstructing the type information from a raw process in the most general way. The inference algorithm can infer type schemes and annotate terms with them: the result is thus a *process scheme*, which is defined as follows.

Definition 6.9. A *process scheme* is defined by the same syntax as given for processes in Figure 1, except that inputs are decorated by message type schemes instead of message types.

The algorithm is defined through a set of ‘natural semantics’ rules (Kahn 1987). A reasonable judgement produced by these rules would be of the form

$$R \Rightarrow \Xi, P, g$$

where R is a raw term, P is a process scheme that is a typed version of R (that is, $|P| = R$), and Ξ is an environment scheme. Furthermore, Ξ and g are such that $\Xi \vdash P : g$ is the 'most general' typing judgement for R, in the sense that any other typing $\Xi' \vdash Q : g'$ for a process Q such that $|Q| = R$ can be obtained from it by substitution and $\leqslant$. To make the algorithm more readable, however, we will use a judgement of the form

$$\Xi \vdash_{\mathrm{I}} P : g\,,$$

which is intended to represent the most general typing for the raw term $|P|$. Note that this could be expressed in the previous form by writing $|P| \Rightarrow \Xi, P, g$. Similarly, we will use a judgement of the form

$$\Xi \vdash_{\mathrm{I}} C : g \to g'$$

to represent the (most general) typing for a raw capability C.

As a preliminary remark, note that raw terms do not contain any indications of possible occurrences of group restrictions. Moreover, typings of groups given by group restrictions may always be shifted, with no change, from the process to its environment. This can be formalised as follows.

Lemma 6.10. 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

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

Proof. Let $\mathcal{C}[\,]$ be a context not containing group restrictions. Using Lemma 3.1 it is easy to show by induction on $\mathcal{C}[\,]$ that

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

The base step is Lemma 3.1(13).

For the induction step, consider the case $\mathcal{C}[\,] = N[\mathcal{C}'[\,]]$. By Lemma 3.1(9), we have that $\Xi \vdash N[\mathcal{C}'[(\nu\{\overrightarrow{\mathbf{g} : \mathbf{G}}\}_{(k)})P]] : g$ implies $\Xi \vdash \mathcal{C}'[(\nu\{\overrightarrow{\mathbf{g} : \mathbf{G}}\}_{(k)})P] : g'$ and $\Xi \vdash N : g'$, $\Xi \vdash g' : G$, and $g \in \mathscr{S}(G)$ for unique g', G. By induction, $\Xi, \{\overrightarrow{\mathbf{g} : \mathbf{G}}\}_{(k)} \vdash \mathcal{C}'[P] : g'$, and by rule (Amb), we have $\Xi, \{\overrightarrow{\mathbf{g} : \mathbf{G}}\}_{(k)} \vdash \mathcal{C}[P] : g'$. □

This Lemma means that the type inference algorithm can ignore group restrictions as they can always be introduced without affecting typings (while satisfying the relative side conditions).

The inference procedure is defined in a natural semantics style. Figures 18 and 19 give the inference rules for messages and processes, respectively. 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. Rule (I-out) in Figure 18 distinguishes some occurrences of group names in the generated environment by marking them with a star (*) for later use. The marked occurrences are preserved by substitution: if an occurrence $g^\star$ is in the domain

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

$$\{g_2 : \mathsf{gr}(\varnothing, \varnothing, \{g_1\}, t)\}; \{N : g_1\} \vdash_{\mathtt{I}} \mathsf{to}\ N : g_1 \to g_2 \quad \text{(I-to)}$$

$$\{g_2 : \mathsf{gr}(\{g_1\}, \{g_1\}, \varnothing, t)\}; \{N : g_1\} \vdash_{\mathtt{I}} \mathsf{in}\ N : g_2 \to g_2 \quad \text{(I-in)}$$

$$\{g_2 : \mathsf{gr}(\{g_1^\star\}, \{g_1\}, \varnothing, t)\}; \{N : g_1\} \vdash_{\mathtt{I}} \mathsf{out}\ N : g_2 \to g_2 \quad \text{(I-out)}$$

$$\frac{\Gamma; \Delta \vdash_{\mathtt{I}} C : W \qquad \Gamma'; \Delta' \vdash_{\mathtt{I}} C' : W'}{\uplus(\sigma'(\Gamma, \Gamma'));\ \sigma'(\Delta, \Delta') \vdash_{\mathtt{I}} C.C' : \sigma'(g_3 \to g_2)} \quad \text{(I-Path)}$$
$$\text{where } \sigma' = \Sigma[\sigma(\Gamma, \Gamma')] \circ \sigma \text{ and } \sigma = \phi((\Delta, \Delta') \cup \{(W, g_1 \to g_2), (W', g_3 \to g_1)\})$$

Fig. 18. Type reconstruction for capabilities/messages.

$$\varnothing; \varnothing \vdash_{\mathtt{I}} \mathsf{0} : g \quad \text{(I-Null)}$$

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

$$\frac{\Gamma;\ \Delta \vdash_{\mathtt{I}} P : g}{\uplus(\sigma'(\Gamma, g : \mathsf{G}));\ \sigma'(\Delta, x : t) \vdash_{\mathtt{I}} \sigma'((x : t)P) : \sigma'(g)} \quad \text{(I-Input)}$$
$$\text{where } \sigma' = \Sigma[\sigma(\Gamma, g : \mathsf{G})] \circ \sigma,\ \ \sigma = \phi(\Delta, \{x : t\}) \text{ and } \mathsf{G} = g : \mathsf{gr}(\varnothing, \varnothing, \varnothing, t)$$

$$\frac{\Gamma;\ \Delta \vdash_{\mathtt{I}} P : g \qquad \Gamma'; \Delta' \vdash_{\mathtt{I}} M : \mathtt{W}}{\uplus(\sigma'(\Gamma, g : \mathsf{G}, \Gamma'));\ \sigma'(\Delta, \Delta') \vdash_{\mathtt{I}} \langle M \rangle\, \sigma'(P) : \sigma'(g)} \quad \text{(I-Output)}$$
$$\text{where } \sigma' = \Sigma[\sigma(\Gamma, g : \mathsf{G}, \Gamma')] \circ \sigma,\ \ \sigma = \phi((\Delta, \Delta') \cup \{(\mathtt{W}, t)\}) \text{ and } \mathsf{G} = g : \mathsf{gr}(\varnothing, \varnothing, \varnothing, t)$$

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

$$\frac{\Gamma;\ \Delta \vdash_{\mathtt{I}} P : g}{\uplus(\sigma'(\Gamma, g : \mathsf{G}));\ \sigma'(\Delta, N : g) \vdash_{\mathtt{I}} N[\sigma'(P)] : g'} \quad \text{(I-Amb)}$$
$$\text{where } \sigma' = \Sigma[\sigma(\Gamma, g : \mathsf{G})] \circ \sigma,\ \ \sigma = \phi(\Delta, \{N : g\}) \text{ and } \mathsf{G} = g : \mathsf{gr}(\{g'\}, \varnothing, \varnothing, t)$$

$$\frac{\Gamma;\ \Delta, n : g' \vdash_{\mathtt{I}} P : g}{\Gamma;\ \Delta \vdash_{\mathtt{I}} (\nu n : g')P : g} \quad \text{(I-Res)} \qquad \frac{\Gamma;\ \Delta \vdash_{\mathtt{I}} P : g \quad n \notin Dom(\Delta)}{\Gamma;\ \Delta \vdash_{\mathtt{I}} (\nu n : g')P : g} \quad \text{(I-Res1)}$$

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

Fig. 19. Type reconstruction for raw terms.

of a substitution replacing g by g', then the occurrence is changed into $g'^{\star}$[†]. Marking is completely transparent to all operations introduced in this section.

In all the rules with two premises (that is, (I-Path), (I-Prefix-Cap), (I-Output) and (I-Par)), the algorithm merges the two environment schemes of the premises by using completion-unification and compression. In rules (I-Path), (I-Prefix-Cap) and (I-Par) two groups are identified. More precisely:

— in (I-Path), the output group of C' is identified with the input group of C;

— in (I-Prefix-Cap), the input group of C is identified with the group of the process scheme P;

— in (I-Par), the groups of the process schemes P and Q are identified.

In the rule (I-Output), the algorithm identifies the communication type schemes of P and M by means of the unifier defined in Definition 6.2. The same kind of unification is performed by the rule (I-Input). In the rule (I-Amb), the ambient name N must be typed with the group g of process scheme P. The type of the resulting process scheme is a fresh group name g'. The group g' is added to the set of the ambient groups where g-ambients can stay.

Soundness and completeness

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

Let $\vdash_{\text{-}\nu g}$ denote the type assignment system for the variant of our language in which there are no group restrictions (that is, rule (GrpRes)).

Also, let $\vdash_{\text{-}\nu g}^{+\mathcal{V}}$ denote the type assignment for the variant of $\vdash_{\text{-}\nu g}$ in which type variables are also allowed to occur in communication type schemes. The typing rules in Figure 5 are not affected by the presence of variables. Obviously, $\vdash_{\text{-}\nu g}^{+\mathcal{V}}$ is an extension of $\vdash_{\text{-}\nu g}$ in the sense that every statement valid in $\vdash_{\text{-}\nu g}$ is also valid in $\vdash_{\text{-}\nu g}^{+\mathcal{V}}$.

Finally, let $\vdash_{\text{-}\nu g-\mathscr{S}}^{+\mathcal{V}}$ denote the type inference system obtained from $\vdash_{\text{-}\nu g}^{+\mathcal{V}}$ by ignoring the condition on inclusion of $\mathscr{S}$ components in rule (Out).

It is easy to see that the generation Lemma 3.1 also holds for deductions in $\vdash_{\text{-}\nu g-\mathscr{S}}^{+\mathcal{V}}$. This implies that there is a unique deduction for each valid statement. One can show by induction on derivations that $\vdash_{\text{-}\nu g-\mathscr{S}}^{+\mathcal{V}}$ is closed under substitutions respecting consistency and under applications of $\uplus$, and is preserved when the current environment scheme is replaced by a well-formed and greater (with respect to $\leqslant$ as defined in Definition 6.7) environment scheme.

Notational convention. In the following, Π generically denotes a well-formed process scheme P or a message M, and τ denotes either a process type g or a capability type $g_1 \rightarrow g_2$. In both cases the meaning will be clear from the context.

Lemma 6.11.

1 If the judgment $\Gamma; \Delta \vdash_{\text{-}\nu g-\mathscr{S}}^{+\mathcal{V}} \Pi : \tau$ holds, its derivation is unique.

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

2 Let $\Gamma;\Delta \vdash_{\text{-}v\text{g}-\mathscr{S}}^{+\mathcal{V}} \Pi : \tau$, and let σ be a substitution such that $\sigma(\Gamma)$ is consistent. Then $\uplus(\sigma(\Gamma));\sigma(\Delta) \vdash_{\text{-}v\text{g}-\mathscr{S}}^{+\mathcal{V}} \sigma(\Pi) : \sigma(\tau)$.

3 Let $\Xi \vdash_{\text{-}v\text{g}-\mathscr{S}}^{+\mathcal{V}} \Pi : \tau$, and let Ξ' be a well-formed environment scheme such that $\Xi \leqslant \Xi'$. Then $\Xi' \vdash_{\text{-}v\text{g}-\mathscr{S}}^{+\mathcal{V}} \Pi : \tau$.

The following lemma is the main lemma for the soundness proof.

Lemma 6.12. If $\Xi \vdash_{\mathtt{I}} \Pi : \tau$, then $\Xi \vdash_{\text{-}v\text{g}-\mathscr{S}}^{+\mathcal{V}} \Pi : \tau$.

Proof. The proof is by induction on the type inference derivation (system $\vdash_{\mathtt{I}}$). Note that the environment schemes in the conclusions of all rules are well formed by Lemmas 6.4(1) and 6.6. Lemma 6.11(2 and 3) ensure that we can also apply the substitutions and the compression in the conclusions to the corresponding premises, and that we can weaken the environment schemes in the premises.

As an example, we will show the case of rule (I-Prefix-Cap). We have

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

where $\sigma' = \Sigma[\sigma(\Gamma_1,\Gamma_2)] \circ \sigma$ and $\sigma = \phi((\Delta_1,\Delta_2) \cup (\{\mathtt{W}, g_1 \to g_2\}\})$.

By induction, we have that $\Gamma_1;\Delta_1 \vdash_{\text{-}v\text{g}-\mathscr{S}}^{+\mathcal{V}} C : \mathtt{W}$ and $\Gamma_2;\Delta_2 \vdash_{\text{-}v\text{g}-\mathscr{S}}^{+\mathcal{V}} P : g_1$.

By construction, we have that $\sigma'(\Delta_1,\Delta_2)$ is consistent and $\sigma'(\Delta_i) \leqslant \sigma'(\Delta_1,\Delta_2)$ for $i = 1,2$.

By Lemma 6.4(1), we have that $\sigma'(\Gamma_1,\Gamma_2)$ is consistent, so $\uplus(\sigma'(\Gamma_1,\Gamma_2))$ is well formed by Lemma 6.6. Hence, $\uplus(\sigma'(\Gamma_i))$, for $i = 1,2$, are also well formed and such that $\uplus(\sigma'(\Gamma_i)) \leqslant \uplus(\sigma'(\Gamma_1,\Gamma_2))$.

We must also have $\sigma'(\mathtt{W}) = g \to g'$ and $\sigma'(g_1) = g$, $\sigma'(g_2) = g'$ for some groups g, g'.

By Lemma 6.11(2 and 3), we have

$$\uplus(\sigma'(\Gamma_1,\Gamma_2));\sigma'(\Delta_1,\Delta_2) \vdash_{\text{-}v\text{g}-\mathscr{S}}^{+\mathcal{V}} C : g \to g'$$

and

$$\uplus(\sigma'(\Gamma_1,\Gamma_2));\sigma'(\Delta_1,\Delta_2) \vdash_{\text{-}v\text{g}-\mathscr{S}}^{+\mathcal{V}} \sigma'(P) : g\,,$$

and the proof then follows by rule (Prefix-Cap). □

Note that, by Lemmas 6.12 and 6.11(1), if $\Xi \vdash_{\mathtt{I}} \Pi : \tau$ holds, there is a unique deduction of $\Xi \vdash_{\text{-}v\text{g}-\mathscr{S}}^{+\mathcal{V}} \Pi : \tau$. This can indeed be obtained by applying backwards in the deduction tree of $\Xi \vdash_{\mathtt{I}} \Pi : \tau$ the substitutions generated in $\vdash_{\mathtt{I}}$, and by matching the environment schemes with respect to $\leqslant$. In doing this, we can keep track of the starred group names: in particular (using the notation of Definition 6.5), a group name is starred in the union of $\mathscr{S}$ fields during the compression if it is starred in at least one of the $\mathscr{S}_i$.

We can eliminate the stars using the following closure operation.

Definition 6.13. The *closure* of a group environment scheme Γ (written $\Upsilon(\Gamma)$), where Γ may contain occurrences of the star *, is the environment scheme computed as sketched in the following algorithm:

```
**repeat**
  **if** *for some* $g^\star \in \mathscr{S}(\mathtt{G})$ we have $g : \mathtt{G}' \in \Gamma$ and $\mathscr{S}(\mathtt{G}') \not\subseteq \mathscr{S}(\mathtt{G})$, **then** replace $\mathscr{S}(\mathtt{G})$ by
$\mathscr{S}(\mathtt{G}) \cup \mathscr{S}(\mathtt{G}')$
```

until *there are no more* $g^\star$ *satisfying the above condition*
Then erase all *.

Note that if Γ is well formed, then $\Upsilon(\Gamma)$ is well formed also, and $\Gamma \leqslant \Upsilon(\Gamma)$, since the only effect of closure is to increase the $\mathscr{S}$ components of group type schemes.

Lemma 6.14. Let $\Gamma;\Delta \vdash_{\mathtt{I}} \Pi:\tau$. Then we have:

1 $\Upsilon(\Gamma);\Delta \vdash_{\text{-}vg}{}^{+\mathcal{V}} \Pi:\tau$.
2 Moreover if $\Gamma';\Delta \vdash_{\text{-}vg}{}^{+\mathcal{V}} \Pi:\tau$ for some well-formed Γ' such that $\Gamma \leqslant \Gamma'$, then $\Upsilon(\Gamma) \leqslant \Gamma'$.

Proof.

1 By Lemmas 6.12 and 6.11(3), there is a deduction of $\Upsilon(\Gamma);\Delta \vdash_{\text{-}vg-\mathscr{S}}^{+\mathcal{V}} P : g$ that, by Lemma 6.11(1), is unique. Now the only reason a deduction in $\vdash_{\text{-}vg-\mathscr{S}}^{+\mathcal{V}}$ can fail to be a deduction in $\vdash_{\text{-}vg}{}^{+\mathcal{V}}$ is that the condition on rule (OUT) is not satisfied somewhere, that is, that for some action out N occurring in P we have that out N is assigned a type $g_2 \rightarrow g_2$, and $\xi : g_1 \in \Delta$, and $g_1 : \mathtt{G}_1$, $g_2 : \mathtt{G}_2 \in \Gamma$ and $\mathscr{S}(\mathtt{G}_1) \nsubseteq \mathscr{S}(\mathtt{G}_2)$. This is impossible, since in this case we have $g_1^\star \in \mathscr{S}(\mathtt{G}_2)$ inside Γ, and at the end of the closure, that is, in $\Upsilon(\Gamma)$, all starred groups $g_1 : \mathtt{G}_1$ occurring in $\mathscr{S}(\mathtt{G}_2)$ are such that $\mathscr{S}(\mathtt{G}_1) \subseteq \mathscr{S}(\mathtt{G}_2)$.
2 We show by induction on the number of steps of the algorithm of Definition 6.13 that the environment scheme Γ_n resulting at the n-th step must be such that $\Gamma_n \leqslant \Gamma'$.
For $n = 0$, this is trivial.
Let the $(n+1)$-th step replace $\mathscr{S}(\mathtt{G})$ by $\mathscr{S}(\mathtt{G}) \cup \mathscr{S}(\mathtt{G}')$, since $g^\star \in \mathscr{S}(\mathtt{G})$, $g : \mathtt{G}' \in \Gamma$, and $\mathscr{S}(\mathtt{G}') \nsubseteq \mathscr{S}(\mathtt{G})$. An inspection of the type scheme reconstruction rules shows that the only rule introducing starred groups is (I-out), so Π must contain the capability out N, and $N : g \in \Delta$ and $g' : \mathtt{G} \in \Gamma$ for some g'. Therefore, in the deduction of $\Gamma';\Delta \vdash_{\text{-}vg}{}^{+\mathcal{V}} \Pi:\tau$, there will be an application of rule (OUT) for typing out N with premises $\Xi \vdash g : \mathtt{G}_1$, $\Xi \vdash g' : \mathtt{G}_2$ and $\mathscr{S}(\mathtt{G}_1) \subseteq \mathscr{S}(\mathtt{G}_2)$ for some $\mathtt{G}_1, \mathtt{G}_2$. By induction, $\mathscr{S}(\mathtt{G}') \subseteq \mathscr{S}(\mathtt{G}_1)$ and $\mathscr{S}(\mathtt{G}) \subseteq \mathscr{S}(\mathtt{G}_2)$, so we conclude $\mathscr{S}(\mathtt{G}) \cup \mathscr{S}(\mathtt{G}') \subseteq \mathscr{S}(\mathtt{G}_2)$. □

Definition 6.15. The *finishing* substitution σ_f for a judgement $\Gamma;\Delta \vdash_{\mathtt{I}} \Pi:\tau$ is a substitution that:

1 replaces with shh all the type variables t that occur only once in the fourth components of group type schemes in Γ;
2 replaces with arbitrary group names the type variables t that occur more than once, that is, once in the fourth components of group type schemes in Γ and at least once in Π as input type schemes.

Note that finishing substitutions are not inference substitutions in the sense of Definition 6.1.

By inspection of the inference rules, it is easy to see that the variables mentioned in Point (1) of Definition 6.15 can only occur in one group type scheme and never as components of a capability type scheme. So they characterise the groups in which no communication is done, and can therefore be replaced by shh. In the resulting term, there can still be some type variables left, such as those mentioned in Point (2) of Definition 6.15, which can be replaced by arbitrary group names. Note that, by construction, a variable cannot occur in both Γ and Δ.

The soundness result is then a consequence of Lemma 6.14(1), since the finishing substitution eliminates all variables while preserving the well-formedness of environment schemes.

Theorem 6.16 (Soundness). If $\Gamma;\Delta \vdash_{\mathtt{I}} \Pi : \tau$ and σ_f is a finishing substitution for it, then $\sigma_f(\Upsilon(\Gamma);\Delta) \vdash_{\text{-vg}} \sigma_f(\Pi) : \sigma_f(\tau)$.

Completeness of the type inference procedure can be proved by induction on derivations by means of Lemmas 6.4 and 6.8.

Theorem 6.17 (Completeness). If $\Xi \vdash_{\text{-vg}} \Pi : \tau$, then $\Xi' \vdash_{\mathtt{I}} \Pi' : \tau'$ and there is a substitution σ such that:

(1) $\sigma(\Pi') = \Pi$.

(2) $\sigma(\tau') = \tau$.

(3) $\sigma(\Xi') \leqslant \Xi$.

Proof. The proof is by induction on deductions in $\vdash_{\text{-vg}}$. We will only show the case of rule (Amb); the other cases are similar. Hence, we assume

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

where $\Xi = \Gamma;\Delta$, and N is either a variable or an ambient name. In both cases, we must have $N : g \in \Delta$. Moreover, $g : G \in \Gamma$.

By the induction hypothesis on $\Gamma;\Delta \vdash_{\text{-vg}} P : g$, we have that $\Gamma_1;\Delta_1 \vdash_{\mathtt{I}} P_1 : g_1$ holds, and there is a substitution σ_1 such that:

(1′) $\sigma_1(P_1) = P$.

(2′) $\sigma_1(g_1) = g$.

(3′) $\sigma_1(\Gamma_1;\Delta_1) \leqslant \Gamma;\Delta$.

Applying rule (I-Amb) to $\Gamma_1;\Delta_1 \vdash_{\mathtt{I}} P_1 : g_1$, we have

$$\uplus(\sigma_0'(\Gamma_1, g_1 : \mathtt{G}_1));\ \sigma_0'(\Delta_1, N : g_1) \vdash_{\mathtt{I}} N[\sigma_0'(P_1)] : g_1',$$

where $\sigma_0' = \Sigma[\sigma_0(\Gamma_1, g_1 : \mathtt{G}_1)] \circ \sigma_0$, $\sigma_0 = \phi(\Delta_1, \{N : g_1\})$ and $\mathtt{G}_1 = g_1 : \mathsf{gr}(\{g_1'\}, \varnothing, \varnothing, t)$.

As g_1', t are fresh, we can define $\sigma_2 = \sigma_1 \circ \{g_1' := g'\} \circ \{t := T(\mathtt{G})\}$. Clearly, σ_2 satisfies:

(1″) $\sigma_2(P_1) = P$ (by (1′) since g_1', t do not occur in P).

(2″) $\sigma_2(g_1) = g$ (by (2′)).

(2‴) $\sigma_2(g_1') = g'$.

(3″) $\sigma_2(\Gamma_1, g_1 : \mathtt{G}_1;\ \Delta_1, N : g_1) \leqslant \Gamma;\Delta$ (by (3′), (2″) and (2‴) since $N : g \in \Delta$, $g : G \in \Gamma$, $g' \in \mathscr{S}(G)$ and $\sigma_2(t) = T(\mathtt{G})$).

Lemma 6.8(1) and Point (3″) imply that $\sigma_2(\Delta_1, N : g_1)$ is well formed. Hence, by the definition of most general unifier, we must have $\sigma_2 = \sigma_3 \circ \sigma_0$ for some substitution σ_3.

Lemma 6.8(1) and Point (3″) also imply that $\sigma_3(\sigma_0(\mathtt{G}_1, g_1 : \mathtt{G}_1)) \leqslant \Gamma$ is consistent. Hence, by Lemma 6.4(2), there must be a substitution σ such that $\sigma_3 = \sigma \circ \Sigma[\sigma_0(\Gamma_1, g_1 : \mathtt{G}_1)]$. So $\sigma_2 = \sigma \circ \Sigma[\sigma_0(\Gamma_1, g_1 : \mathtt{G}_1)] \circ \sigma_0 = \sigma \circ \sigma_0'$. Now σ is the desired substitution, since:

(1) $\sigma(\sigma_0'(N[P_1])) = N[P]$ (by (1″))
(2) $\sigma(g_1') = g'$ (by (2″))
(3) $\sigma(\uplus(\sigma_0'(\Gamma_1, g_1 : \mathtt{G}_1)); \sigma_0'(\Delta_1, N : g_1)) \leqslant \Gamma; \Delta$

where $\sigma(\uplus(\sigma_0'(\Gamma_1, g_1 : \mathtt{G}_1)) \leqslant \Gamma$ follows from Point (3″) using Lemma 6.8(2). □

7. Related papers

In the original Mobile Ambients (Cardelli and Gordon 2000), as well as the subjective moving capabilities in and out, there is also the open capability. This capability, which allows the dissolving of boundaries, has been criticised for both bringing about serious security concerns (Bugliesi *et al.* 2004) and for being difficult to implement (Sangiorgi and Valente 2001).

The proposal of Boxed Ambients (Bugliesi *et al.* 2004) drops the open capability but allows communications across ambient boundaries between parents and children.

In M^3, however, the open is replaced by the to capability.

The communication of Boxed Ambients can be mimicked in M^3 by simply moving one of the two processes willing to communicate. Consider the example of a parent willing to send a message M to a child ambient n waiting for it, that is, using the syntax of Boxed Ambients,

$$\langle M \rangle^n \mid n[(x:W)^{\uparrow} P] \,.$$

In M^3, a process with a similar behaviour is

$$(\nu m : g)(m[\mathsf{to}\ n.\langle M \rangle] \mid n[(x:W)P] \,.$$

We conjecture that the to capability is more expressive than the communication mechanisms of Boxed Ambients, since we do not know how this capability or the Dπ calculus might be encoded in Boxed Ambients. Of course, this deserves further investigation.

On the other hand, the to capability seems to be less powerful (and less dangerous!) than the open capability, which we do not know how to fully encode in M^3. We are only able to encode an open contained in an ambient, that is, a process of Mobile Ambients of the form

$$n[m[P] \mid \mathsf{open}\ m \mid Q] \,.$$

The corresponding M^3 process is

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

Conversely, the effect of to,

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

can be mimicked in Mobile Ambients as

$$(\nu p)(m[p[\mathsf{out}\ m.\mathsf{in}\ n.P]) \mid Q] \mid n[\mathsf{open}\ p \mid R]) \,.$$

In M^3, as in Mobile and Boxed Ambients, the ambients do not control when other ambients traverse them. Levi and Sangiorgi have proposed adding co-capabilities to Mobile Ambients in such a way that each action needs the agreement of both the active and passive ambient involved (Levi and Sangiorgi 2003). One could clearly devise a version of M^3 with co-capabilities, but we have not done so as M^3 types ensure that active ambients have the 'right' to traverse or send processes to passive ambients.

The type system of M^3 is inspired by Cardelli *et al.* (2002): similarities and differences were pointed out in Section 3. The type inference algorithm of Section 6 could be easily modified to deal with group types for Mobile Ambients. In the literature there are only a few type inference algorithms for ambient calculi: see Zimmer (2000), Barbanera *et al.* (2002), Amtoft *et al.* (2004), Cozzi (2004) and Coppo *et al.* (2005) – only the last two deal with group types. So the design of the present algorithm involved a number of non-trivial choices.

There are many labelled transition systems for ambient calculi in the literature. The labelled transition system of M^3 is similar to those of Merro and Hennessy (2002) and Bugliesi *et al.* (2005): the novelty is the pervasive use of types and environments. We claim that type information is crucial in the design of labelled transition systems for typed calculi, especially when types enforce security policies, as in M^3. The influence of types on the behaviour of π-calculus processes was first observed by Pierce and Sangiorgi (Pierce and Sangiorgi 2000). The labelled transition system we have presented here is sound but not complete with respect to reduction barbed congruence: we conjecture that a complete system could be obtained by distinguishing between visible and invisible transitions, as Merro and Zappa Nardelli do for Mobile Ambients (Merro and Zappa Nardelli 2005). The price would be an increase in the complexity of transition labels, thus making the use of type information less clear, which is why we did not pursue this approach.

Coppo *et al.* (2004) and Coppo *et al.* (2005) present a typed ambient calculus based on M^3 where global type assumptions on ambient names are eliminated: this is realistic in a scenario where interaction may take place between parties whose respective properties are unknown or only partially known to each other. Cozzi (2004) gives a type inference algorithm equipped with a Prolog implementation for the calculus of Coppo *et al.* (2004).

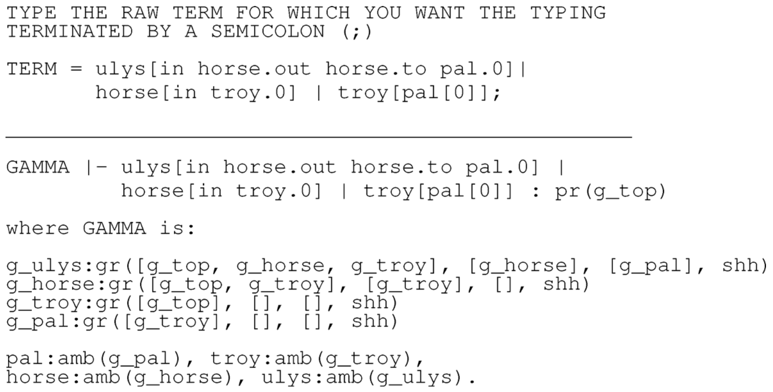
8. Implementation

As shown in Giovannetti (2004), the type inference rules may be almost directly translated into logic programming clauses so that the explicit unification is automatically performed by Prolog; with the addition of functional or imperative Prolog procedures for computing completion and compression, one obtains a program that implements, with slight modifications, the type reconstruction algorithm specified in Section 6.

For example, the rule for parallel composition may be written as

$$\frac{\Gamma_1;\Delta_1 \vdash P \;:\mathsf{pr}(g) \quad \Gamma_2;\Delta_2 \vdash Q \;:\mathsf{pr}(g)}{\Gamma_1 \Cup \Gamma_2;\; \Delta_1 \cup \Delta_2 \vdash P \mid Q \;:\mathsf{pr}(g)} \;(\textsc{Par})$$

where the condition that the two environments in the two premises are compatible is



```
TYPE THE RAW TERM FOR WHICH YOU WANT THE TYPING
TERMINATED BY A SEMICOLON (;)

TERM = ulys[in horse.out horse.to pal.0]|
       horse[in troy.0] | troy[pal[0]];

_________________________________________________

GAMMA |- ulys[in horse.out horse.to pal.0] |
         horse[in troy.0] | troy[pal[0]] : pr(g_top)

where GAMMA is:

g_ulys:gr([g_top, g_horse, g_troy], [g_horse], [g_pal], shh)
g_horse:gr([g_top, g_troy], [g_troy], [], shh)
g_troy:gr([g_top], [], [], shh)
g_pal:gr([g_troy], [], [], shh)

pal:amb(g_pal), troy:amb(g_troy),
horse:amb(g_horse), ulys:amb(g_ulys).
```

Fig. 20. Example session.

omitted. The corresponding Prolog clause is

```
typing(GEnv, Env, P1 par P2, pr(G)):-
  typing(GEnv1, Env1, P1, pr(G)),
  typing(GEnv2, Env, P2, pr(G)),
  sumunion(Env1,Env,GEnv1,GEnv2,GEnv).
```

The procedure `sumunion` is the ‘impure’ logic programming procedure that performs completion and compression.

The type reconstruction algorithm is embodied in a web-based prototype tool, which can be found at `http://lambda.di.unito.it/m3`; the SWI-Prolog source code is available at `http://www.di.unito.it/ elio/dart/m3.pl`.

Figure 20 is the listing of the session for the Trojan horse example, which can only be well typed if the assumptions allow Ulysses’ group to send lightweight (possibly harmful) processes into Priam’s palace.

9. 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 (Hennessy and Riely 2002).

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, which are expressible through much more sophisticated types such as those of Pierce and Sangiorgi for the π-calculus (Pierce and Sangiorgi 1996). Even the basic type system for Dπ (Hennessy and Riely 2002) 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 Dπ by the presence of local typing judgments, which cannot be simulated by our purely global judgments. We plan, therefore, to enrich the type system of M^3 with security controls and local type information.

References

Amtoft, T., Kfoury, A. J. and Pericas-Geertsen, S. M. (2001) What are Polymorphically-Typed Ambients? In: Sands, D. (ed.) ESOP'01. *Springer-Verlag Lecture Notes in Computer Science* **2028** 206–220.

Amtoft, T., Makholm, H. and Wells, J. B. (2004) PolyA: True Type Polymorphism for Mobile Ambients. In: Lévy, J.-J., Mayr, E. W. and Mitchell, J. C. (eds.) *TCS'04*, Kluwer Academic Publishers 5971–611.

Barbanera, F., Dezani-Ciancaglini, M., Salvo, I. and Sassone, V. (2002) A Type Inference Algorithm for Secure Ambients. In: Lenisa, M. and Miculan, M. (eds.) TOSCA'01. *Electronic Notes in Theoretical Computer Science* **62** 83–101.

Bettini, L., Bono, V., De Nicola, R., Ferrari, G., Gorla, D., Loreti, M., Moggi, E., Pugliese, R., Tuosto, E. and Venneri, B. (2003) The Klaim Project: Theory and Practice. In: Priami, C. (ed.) Global Computing. *Springer-Verlag Lecture Notes in Computer Science* **2874** 88–150.

Bugliesi, M. and Castagna, G. (2002) Behavioral Typing for Safe Ambients. *Computer Languages* **28** (1) 61–99.

Bugliesi, M., Castagna, G. and Crafa, S. (2004) Access Control for Mobile Agents: The Calculus of Boxed Ambients. *ACM Transactions on Programming Languages and Systems* **26** (1) 57–124.

Bugliesi, M., Crafa, S., Merro, M. and Sassone, V. (2005) Communication and Mobility Control in Boxed Ambients. *Information and Computation* **202** (1) 39–86.

Cardelli, L., Ghelli, G. and Gordon, A. D. (1999) Mobility Types for Mobile Ambients. In: Wiederman, J., van Emde Boas, P. and Nielsen, M. (eds.) ICALP'99. *Springer-Verlag Lecture Notes in Computer Science* **1644** 230–239.

Cardelli, L., Ghelli, G. and Gordon, A. D. (2002) Types for the Ambient Calculus. *Information and Computation* **177** (2) 160–194.

Cardelli, L. and Gordon, A. D. (1999) Types for Mobile Ambients. In: Aiken, A. (ed.) *POPL'99*, ACM Press 79–92.

Cardelli, L. and Gordon, A. D. (2000) Mobile Ambients. *Theoretical Computer Science* **240** (1) 177–213. (Special Issue on Coordination, Le Métayer, D. (ed.).)

Coppo, M., Cozzi, F., Dezani-Ciancaglini, M., Giovannetti, E. and Pugliese, R. (2005) A Mobility Calculus with Local and Dependent Types. In: Middeldorp, A., van Oostrom, V., van Raamsdonk, F. and de Vrijer, R. (eds.) Processes, Terms and Cycles: Steps on the Road to Infinity. *Springer-Verlag Lecture Notes in Computer Science* **3838** 404–444.

Coppo, M., Dezani-Ciancaglini, M., Giovannetti, E. and Pugliese, R. (2004) Dynamic and Local Typing for Mobile Ambients. In: Lévy, J.-J., Mayr, E. W. and Mitchell, J. C. (eds.) *TCS'04*, Kluwer Academic Publishers 583–596.

Coppo, M., Dezani-Ciancaglini, M., Giovannetti, E. and Salvo, I. (2003) M3: Mobility Types for Mobile Processes in Mobile Ambients. In: Harland, J. (ed.) CATS 2003. *Electronic Notes in Theoretical Computer Science* **78** 1–34.

Cozzi, F. (2004) Type Inference for Local Typing of Mobile Ambients. (Available at `http://homelinux.capitano.unisi.it/ cozzif/`.)

De Nicola, R., Ferrari, G. and Pugliese, R. (1998) Klaim: a Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering* **24** (5) 315–330.

Giovannetti, E. (2003) Ambient Calculi with Types: a Tutorial. In: Priami, C. (ed.) Global Computing. *Springer-Verlag Lecture Notes in Computer Science* **2874** 151–191.

Giovannetti, E. (2004) Type Inference for Mobile Ambients in Prolog. In: Atkinson, M. (ed.) CATS'04. *Electronic Notes in Theoretical Computer Science* **91** 96–115.

Hennessy, M. and Riely, J. (2002) Resource Access Control in Systems of Mobile Agents. *Information and Computation* **173** 82–120.

Hindley, J. (1969) The Principal Type Scheme of an Object in Combinatory Logic. *Transactions of the American Mathematical Society* **146** 29–60.

Honsell, F. and Scagnetto, I. (2004) Mobility Types in Coq. In: Berardi, S., Coppo, M. and Damiani, F. (eds.) TYPES'03. *Springer-Verlag Lecture Notes in Computer Science* **3085** 324–337.

Kahn, G. (1987) Natural Semantics. In: Brandenburg, F.-J., Vidal-Naquet, G. and Wirsing, M. (eds.) STACS'87. *Springer-Verlag Lecture Notes in Computer Science* **247** 22–39.

Levi, F. and Sangiorgi, D. (2003) Controlling Interference in Ambients. *Transactions on Programming Languages and Systems* **25** (1) 1–69.

Lhoussaine, C. and Sassone, V. (2004) A Dependently Typed Ambient Calculus. In: Schmidt, D. A. (ed.) ESOP'04. *Springer-Verlag Lecture Notes in Computer Science* **2986** 171–187.

Merro, M. and Hennessy, M. (2002) Bisimulation Congruences in Safe Ambients. In: Mitchell, J. (ed.) *POPL'02*, ACM Press 71–80.

Merro, M. and Sassone, V. (2002) Typing and Subtyping Mobility in Boxed Ambients. In: Brim, L., Jančar, P., Křetínský, M. and Kučera, A. (eds.) CONCUR'02. *Springer-Verlag Lecture Notes in Computer Science* **2421** 304–320.

Merro, M. and Zappa Nardelli, F. (2005) Behavioural Theory for Mobile Ambients. *Journal of the ACM* **50** (6) 961–1023.

Milner, R. (1993) The Polyadic π-Calculus: A Tutorial. In: Bauer, F. L., Brauer, W. and Schwichtenberg, H. (eds.) Logic and Algebra of Specification. *NATO ASI Series F: Computer and Systems Sciences* **94** 203–246.

Milner, R., Parrow, J. and Walker, D. (1992) A Calculus of Mobile Processes, Parts 1-2. *Information and Computation* **100** (1) 1–77.

Milner, R. and Sangiorgi, D. (1992) Barbed Bisimulation. In: Kuich, W. (ed.) ICALP'92. *Springer-Verlag Lecture Notes in Computer Science* **623** 685–695.

Pierce, B. and Sangiorgi, D. (1996) Typing and Subtyping for Mobile Processes. *Mathematical Structures in Computer Science* **6** (5) 409–454.

Pierce, B. and Sangiorgi, D. (2000) Behavioral Equivalence in the Polymorphic pi-calculus. *Journal of the ACM* **47** (3) 531–584.

Sangiorgi, D. and Milner, R. (1992) The Problem of "Weak Bisimulation up to". In: Cleaveland, W. R. (ed.) CONCUR'92. *Springer-Verlag Lecture Notes in Computer Science* **630** 32–46.

Sangiorgi, D. and Valente, A. (2001) A Distributed Abstract Machine for Safe Ambients. In: Orejas, F., Spirakis, P. and Leeuwen, J. (eds.) ICALP'01. *Springer-Verlag Lecture Notes in Computer Science* **2076** 408–420.

Sangiorgi, D. and Walker, D. (2001) *The π-calculus: a Theory of Mobile Processes*, Cambridge University Press.

Wells, J. (2002) The Essence of Principal Typings. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbez, S. and Conejo, R. (eds.) ICALP'02. *Springer-Verlag Lecture Notes in Computer Science* **2380** 913–925.

Zimmer, P. (2000) Subtyping and Typing Algorithms for Mobile Ambients. In: Tiuryn, J. (ed.) FoSSaCS'00. *Springer-Verlag Lecture Notes in Computer Science* **1784** 375–390.