Controlling Interference in Ambients

Francesca Levi
University of Pisa, Italy
levifran@di.unipi.it

Davide Sangiorgi
INRIA Sophia-Antipolis, France
davide.sangiorgi@sophia.inria.fr

Abstract

Two forms of interferences are individuated in Cardelli and Gordon's *Mobile Ambients* (MA): *plain interferences*, which are similar to the interferences one finds in CCS and π-calculus; and *grave interferences*, which are more dangerous and may be regarded as programming errors. To control interferences, the MA movement primitives are modified. On the new calculus, the *Mobile Safe Ambients* (SA), a type system is defined that: controls the mobility of ambients; removes all grave interferences. Other advantages of SA are: a useful algebraic theory; programs sometimes more robust (they require milder conditions for correctness) and/or simpler. These points are illustrated on several examples.

1 Motivations

The calculus of *Mobile Ambients* [3] (MA) is proposed both as a core programming language for the Web and as a model for reasoning about properties of mobile processes, including security. In contrast with previous formalisms for mobile processes such as the π-calculus, whose computational model is based on the notion of *communication*, the MA computational model is based on the notion of *movement*. An ambient, which may be thought of as a named location, is the unit of movement. Processes within the same ambient may exchange messages; ambients may be nested, so to form a hierarchical structure. The three primitives for movement allow: an ambient to enter another ambient, $n[\,\mathtt{in}\,m.\,P \mid Q\,] \mid m[\,R\,] \longrightarrow m[\,n[\,P \mid Q\,] \mid R\,]$; an ambient to exit another ambient, $m[\,n[\,\mathtt{out}\,m.\,P \mid Q\,] \mid R\,] \longrightarrow n[\,P \mid Q\,] \mid m[R]$; a process to dissolve an ambient boundary thus obtaining access to its content, $\mathtt{open}\,n.\,P \mid n[\,Q\,] \longrightarrow P \mid Q$. We call the pairs of ambients or processes involved in a reduction a *redex*, and the corresponding reduction the *contraction* of the redex (by contrast with the λ-calculus, a redex may not form a subterm). Elegant type systems for MA have been given; they control the type of values exchanged within an ambient [7] and the mobility of ambients [2].

The work reported in this paper began with a study of *interference* in MA. An interference occurs when the activity of a process is damaged or corrupted because of the activities of other processes. Interference is one of the most tricky and subtle phenomena in concurrency, and may make both the activities of programming and of reasoning on programs hard. In MA, the most important interference is *redex overlapping* [1]. It is caused by two, or more, redexes sharing one of the interacting partners, as in

$$n[\,\mathtt{in}\,m.\,P\,] \mid m[\,Q\,] \mid m[\,R\,] \tag{1}$$

$$\mathtt{open}\,n.\,P \mid \mathtt{open}\,n.\,Q \mid n[\,R\,] \tag{2}$$

In (1), the redex formed by $n[\,\mathtt{in}\,m.\,P\,]$ and $m[\,Q\,]$ interferes with that formed by $n[\,\mathtt{in}\,m.\,P\,]$ and $m[\,R\,]$. The contraction of a redex destroys the other one. The same happens in (2). We find similar interferences in calculi for concurrency such as CCS and π-calculus [11, 12]. For instance, in the π-calculus process

$$\overline{x}z.\,P \mid x(y).\,Q \mid x(y).\,R \tag{3}$$

the output $\overline{x}z$ may be consumed either by input $x(y).\,Q$ or by $x(y).\,R$. We call interferences such as (1-3) *plain interferences*. They occur when a process may engage the "same" interaction with two different partners. Intuitively, a process requests or offers a service and finds at least two processes willing to react. Plain interferences are sometimes desirable, for instance to program useful non-deterministic behaviours. An example is a request of printing a job that is non-deterministically served by one out of several printers.

In CCS or π-calculus[2] all redex interferences are plain interferences. In MA, however, we find a new, and much more dangerous, form of interference, which we call *grave interference*. Consider the processes

$$\mathtt{open}\,n \mid n[\,\mathtt{in}\,m.\,P\,] \mid m[\,Q\,] \tag{4}$$

$$h[\,n[\,\mathtt{in}\,m.\,P \mid \mathtt{out}\,h.\,R\,] \mid m[\,Q\,]\,] \tag{5}$$

$$h[\,P\,] \mid n[\,\mathtt{in}\,h \mid m[\,\mathtt{out}\,n.\,Q\,]\,] \tag{6}$$

[1] Another source of interferences is aliasing, similar to that one finds in imperative programs [17]; but the problems causes by aliasing interference often show up as forms of redex interference.

[2] We are thinking at the sum-free calculi, since MA has no sum or other operators for expressing a choice between two possible behaviours.

To appear in Proc. 27th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages Boston, Massachusetts, January 19-21, 2000.

In (4), n may be opened, but may also jump into m; in (5) n may move out of the parent h, but also into the sibling m; in (6), n can let m go out of it, or jump into the sibling h. As in (1-2), so in (4-6) the same ambient n is shared by two different redexes. But, in contrast with the previous examples, here the shared ambient may perform logically different interactions. The choice of contraction between the two redexes has dramatic consequences on n: in (4) n is either dissolved or will find refuge inside m; in (5) n will end up either outside h or inside a son of h. In (4) and (5) the contraction of a redex destroys the other one; in (6) the second redex remains, but is corrupted (i.e., it will produce an unexpected outcome when later contracted). The capability $\mathtt{out}$ can be exercised both before and after the $\mathtt{in}\, h$ capability; but one of the results can be quite upsetting for m. The systems (4-6) are, from n's point of view, inconsistent. In our opinion, the occurrence of grave interferences should be regarded as a programming error. Unfortunately, finding conditions that guarantee absence of grave interferences in a process and all its derivatives, without imposing draconian restrictions on the calculus, appears hard (type systems for controlling mobility [2], for instance, are too weak).

The coexistence of plain and grave interferences has serious consequences: (1) the algebraic theory is poor; (2) it is difficult to write correct programs, that is, programs that behave in the expected way in *all* contexts; (3) it is difficult to prove behavioural properties of processes. The MA paper serie presents several examples of modelisation of entire calculi or specific systems. But (with the exception of the firewall-crossing protocol [5]) none of the examples has been proved correct. Only simulations of the reductions of the source processes by the corresponding MA processes are given. While such simulations may be satisfactory for sequential languages, they are not in concurrency, for they say nothing about possible other reductions that the target processes may take. In Section 6 we shall indeed suggest that for some of the examples strong conditions on the surrounding contexts are necessary for correctness.

We tried to develop an algebraic theory for MA, but failed (even taking into account causality or type information) precisely due to the interference problems. Techniques for proving behavioural properties and laws on MA are proposed in [5, 6] but, aside structural laws, the only laws given are $\boldsymbol{\nu} n\ (n[\,P\,]) = \mathbf{0}$ and $\boldsymbol{\nu} n\ (n[\,\mathbf{0}\,] \mid \mathtt{open}\, n.\, P) = P$ (in both cases n should not be free in P); and, although the laws are simple, their proofs are fairly involved. Especially difficult in MA is the formulation of laws that relate the reduction relation to behavioural equality. These laws are at the heart of the algebraic theory of any calculus.

To remedy the above problems while maintaining the essence of the computational model, we propose a modification of the MA primitives. We call *Mobile Safe Ambients* (SA) the new calculus. The modifications do not reduce the expressiveness. In some cases the SA programs are actually simpler. An example is the final encoding of channels in Section 6: we can write in SA the original encoding of channels into MA [7, 2], but also a new one that is simpler and, in contrast with the original one, is purely compositional. Even when the programs are similar, the SA program may require fewer conditions for correctness; examples are the protocols for renaming and for routable packets in Section 6. Existing type systems for MA can be easily adapted to SA. The type systems for controlling mobility, however, appear to be more powerful in SA, in that (i) type systems for MA may give more precise information when transplanted onto SA, and (ii) new type systems may be defined. We discuss in Section 3 type systems for SA that allow us to remove all grave interferences. Exploiting these type systems, in Section 5 we formulate a sample of algebraic laws. As an indication of their power, we use them to prove, purely algebraically, correctness properties for several examples.

2 Mobile Safe Ambients: syntax and semantics

In the MA interaction rules, an ambient may enter, exit, or open another ambient. The second ambient undergoes the action; it has no control on *when* the action takes place. In *Mobile Safe Ambients* (SA) this is rectified: any movement (in, out, open) takes place only if both participants agree. This is achieved by means of the *coactions* $\overline{\mathtt{in}}\, n$, $\overline{\mathtt{out}}\, n$, $\overline{\mathtt{open}}\, n$, whose addition is the the main difference from the syntax of MA. The basic movement rules thus become:

$$\text{(R-in)} \qquad n[\,\mathtt{in}\, m.\, P_1 \mid P_2\,] \mid m[\,\overline{\mathtt{in}}\, m.\, Q_1 \mid Q_2\,] \;\longrightarrow\; m[\,n[\,P_1 \mid P_2\,] \mid Q_1 \mid Q_2\,]$$

$$\text{(R-out)} \qquad m[\,n[\,\mathtt{out}\, m.\, P_1 \mid P_2\,] \mid \overline{\mathtt{out}}\, m.\, Q_1 \mid Q_2\,] \;\longrightarrow\; n[\,P_1 \mid P_2\,] \mid m[\,Q_1 \mid Q_2\,]$$

$$\text{(R-open)} \qquad \mathtt{open}\, n.\, P \mid n[\,\overline{\mathtt{open}}\, n.\, Q_1 \mid Q_2\,] \;\longrightarrow\; P \mid Q_1 \mid Q_2$$

The syntax of SA is presented in Table 1. In calculi for mobility, infinite behaviours are described using either the replication construct $!P$, which represents an unbounded number of copies of P in parallel, or the recursion construct $\mathtt{rec}\, X.\, P$. In MA one usually finds the former; for SA we prefer the latter. To have simpler typing rules, we require that in $\mathtt{rec}\, X.\, P$ the variable X does not appear free inside ambients in the body P. We use $!P$ as an abbreviation for $\mathtt{rec}\, X.\, (P \mid X)$. Expressions that are not variables or names are called *capabilities*; actions and coactions are the *simple capabilities*. We often omit the trailing $\mathbf{0}$ in processes $M.\,\mathbf{0}$. Parallel composition has the least syntactic precedence, thus $m[\,M.\,P \mid Q\,]$ reads $m[\,(M.\,P) \mid Q\,]$. A process is *closed* if it has no free variables. We often omit type annotations in restrictions and abstractions whenever not important or clear from the context.

The one-step reduction relation $\longrightarrow$ of SA is defined in the usual way. It exploits the auxiliary relation $\equiv$ of structural congruence to bring the participants of a potential interaction into contiguous positions; the definition of $\equiv$ is standard, and includes rules for commuting the positions of parallel components, for stretching the scope of a restriction, for unfolding recursion. The only differences with the reduction semantics of MA are the basic movement rules R-in, R-out, and R-open. Here are the remaining reduction rules, including the axiom that allows the exchange of messages (both in MA and in SA, communication is asynchronous, takes place inside ambients, and is anonymous—it does not use channel or process names):

(R-Msg) $\langle M \rangle \mid (x : W)P \longrightarrow P\{ {}^{M}/_{x} \}$

(R-Eps) $\epsilon. P \longrightarrow P$

(R-Rec) $\texttt{rec}\, X. P \longrightarrow P\{ {}^{\texttt{rec}\, X.\, P}/_{X} \}$

$$\text{(R-Res)}\ \frac{P \longrightarrow P'}{(\boldsymbol{\nu} n : Amb)\ P \longrightarrow (\boldsymbol{\nu} n : Amb)\ P'}$$

$$\text{(R-Par)}\ \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q}$$

$$\text{(R-Amb)}\ \frac{P \longrightarrow P'}{n[\, P\,] \longrightarrow n[\, P'\,]}$$

$$\text{(R-Struct)}\ \frac{P \equiv P' \qquad P' \longrightarrow P'' \qquad P'' \equiv P'''}{P \longrightarrow P'''}$$

We write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$. The notions of plain and grave interference can be formalised, both in MA and in SA, using, for instance, an SOS semantics of the calculi and *proved transitions* [1, 16] defined on top of them. The informal explanation of interferences in the Introduction will be sufficient for our purposes.

3 Types

We have three main types for both ambients, processes and capabilities: *basic*, *immobile*, and *single-threaded*.

Basic types. It is straightforward to adapt the basic type system for MA [7] to SA; productions and rules are in Table 2. $\texttt{BAmb}\,[T]$ describes ambients inside which messages of type T may be exchanged; $\texttt{BProc}\,[T]$ describes processes local to basic ambients; $\texttt{BCap}\,[T]$ describes capabilities that, when used, may cause the unleashing of messages of type T (this will only happen upon opening an ambient; $\texttt{BCap}\,[T]$ does not record any information on the other capabilities). The messages exchanged may be ambient or capability expressions; or there may be no exchange of messages, which is indicated by the type $\texttt{Shh}$. Other useful type constructs can be added to the grammar of message types; for instance products, to allow the exchange of tuples of messages, or primitive constructs such as integers and booleans. In the rules, bi in front of $\texttt{Amb}$, $\texttt{Cap}$, or $\texttt{Proc}$ is intended as a metavariable, that in the basic system may only take the value "$\texttt{B}$" (thus $bi\texttt{Amb}\,[T]$ stands for $\texttt{BAmb}\,[T]$); in later extensions, however, bi may take other values. We use bi as a notational device to compact similar rules into a single one. A basic ambient can only open, and be opened by, other basic ambients; it can however jump into any ambient. Jumping into an ambient without being opened can nevertheless be useful, for instance for loading subambients and carrying them away.

Immobility types. The first refinement of the basic type system tracks ambients that: (i) cannot jump into or out of other ambients; (ii) cannot be opened. We call them *immobile ambients*. Basic ambients can jump into an immobile ambient, but cannot be opened there. An immobile ambient can however be inside another one that jumps. For immobile ambients, we add these productions to the type grammars:

			Types of expressions
Amb	$:=$	$\texttt{IAmb}\,[T]$	*(immobile ambient)*
Cap	$:=$	$\texttt{ICap}\,[T]$	*(immobile capability)*
			Types of processes
$Proc$	$:=$	$\texttt{IProc}\,[T]$	*(immobile process)*

$\texttt{IAmb}\,[T]$ is the type of an ambient that is immobile and in whose interior messages of type T can be exchanged. $\texttt{ICap}\,[T]$ is the type of a capability that can be exercised inside an immobile ambient and that may unleash messages of type T. $\texttt{IProc}\,[T]$ is the type of a process that is local to an immobile ambient and may unleash messages of type T. Immobile ambients have behavioural restrictions that basic ambients do not have; the typing rules of the former are indeed a subset of those for of latter. The rules for immobile ambients are obtained by simply allowing, in Table 2, the metavariable bi to range over both $\texttt{B}$ and $\texttt{I}$. With these rules, immobile ambients cannot open any ambients; but they will be allowed to open (some of) the single-threaded ambients of the next section.

Single-threadness types. The capabilities of the local processes of an ambient control the activities of that ambient. In a basic or immobile ambient such control is distributed over the local processes: any of them may exercise a capability. In a *single-threaded* (ST) ambient, by contrast, at any moment at most *one* process has the control thread, and may therefore use a capability. An ST ambient n is willing to engage in at most one interaction at a time with external or internal ambients. Inside n, however, several activities may take place concurrently: a subambient may reduce, two subambients may interact with each other, messages may be consumed (if messages in n carry no thread, see below).

The control thread may move between processes local to an ST ambient by means of an open action. Consider, for instance, a reduction

$$n[\, \texttt{open}\, m.\, P \mid m[\, \overline{\texttt{open}}\, m.\, Q\,]\,] \longrightarrow n[\, P \mid Q\,] \qquad (7)$$

where n and m are ST ambients. Initially $\texttt{open}\, m.\, P$ has the control thread over n, and $\overline{\texttt{open}}\, m.\, Q$ over m. At the end, m has disappeared; the control thread over n may or may not have moved from P to Q, depending on the type of m. If the movement occurs, Q can immediately exercise a capability, whereas P cannot; to use further capabilities within n, P will have to get the thread back.

To gain flexibility, we also allow messages carrying the thread. The type of an ambient says whether messages exchanged within it carry the thread or not. If they do, then the control thread over that ambient may move, other than by an open operation, as a result of the consumption of a message by an abstraction. This happens for instance in

$$n[\, \texttt{in}\, h.\, \langle M \rangle \mid (x)\, x.\, Q\,] \mid h[\, \overline{\texttt{in}}\, h.\, R\,] \ \longrightarrow\longrightarrow \ h[\, R \mid n[\, M.\, Q\,]\,] \qquad (8)$$

The thread over n is first held by $\texttt{in}\, h.\, \langle M \rangle$, but after the consumption of M the thread passes to (the derivative of) $x.\, Q$. In an ST ambient, when messages carry a thread, at

$h, k, \ldots n, m$			*Names*	x, y, z			*Variables*
				X, Y, Z			*Recursion variables*
			Expressions				
M, N	$:=$	x	*(variable)*				*Processes*
	$\mid$	n	*(name)*	P, Q, R	$:=$	$\mathbf{0}$	*(nil)*
	$\mid$	$\texttt{in}\, M$	*(enter)*		$\mid$	$P \mid Q$	*(parallel)*
	$\mid$	$\overline{\texttt{in}}\, M$	*(allow enter)*		$\mid$	$(\boldsymbol{\nu} n : W)\, P$	*(restriction)*
	$\mid$	$\texttt{out}\, M$	*(exit)*		$\mid$	$M.\, P$	*(prefixing)*
	$\mid$	$\overline{\texttt{out}}\, M$	*(allow exit)*		$\mid$	$M[\, P\,]$	*(ambient)*
	$\mid$	$\texttt{open}\, M$	*(open)*		$\mid$	$\langle M \rangle$	*(message)*
	$\mid$	$\overline{\texttt{open}}\, M$	*(allow open)*		$\mid$	$(x : W) P$	*(abstraction)*
	$\mid$	$M.\, N$	*(path)*		$\mid$	X	*(recursion variable)*
	$\mid$	ϵ	*(empty path)*		$\mid$	$\texttt{rec}\, X.\, P$	*(recursive process)*

Table 1: The syntax of Mobile Safe Ambients

Type grammars

			Types of expressions				*Types of messages*
W	$:=$	Amb	*(ambient type)*	T	$:=$	W	*(expression)*
	$\mid$	Cap	*(capability type)*		$\mid$	$\texttt{Shh}$	*(no exchange)*
Amb	$:=$	$\texttt{BAmb}\,[T]$	*(basic ambient)*				
Cap	$:=$	$\texttt{BCap}\,[T]$	*(basic capability)*				*Types of processes*
				$Proc$	$:=$	$\texttt{BProc}\,[T]$	*(basic process)*

Type environments $\quad \Gamma := n : Amb, \Gamma \;\mid\; x : W, \Gamma \;\mid\; X : Proc, \Gamma \;\mid\; \emptyset$

Typing rules for expressions

$$\text{(B-opencoopen)}\ \frac{M \in \{\texttt{open}\, N, \overline{\texttt{open}}\, N\} \qquad \Gamma \vdash N\ : \texttt{BAmb}\,[T]}{\Gamma \vdash M\ : \texttt{BCap}\,[T]} \qquad \text{(B-inout)}\ \frac{M \in \{\texttt{in}\, N, \texttt{out}\, N\} \qquad \Gamma \vdash N\ : Amb}{\Gamma \vdash M\ : \texttt{BCap}\,[S]}$$

$$\text{(B-co-inout)}\ \frac{M \in \{\overline{\texttt{in}}\, N, \overline{\texttt{out}}\, N\} \qquad \Gamma \vdash N\ : bi\texttt{Amb}\,[T]}{\Gamma \vdash M\ : bi\texttt{Cap}\,[S]} \qquad \text{(B-path)}\ \frac{\Gamma \vdash M_1\ : bi\texttt{Cap}\,[T] \qquad \Gamma \vdash M_2\ : bi\texttt{Cap}\,[T]}{\Gamma \vdash M_1.\, M_2\ : bi\texttt{Cap}\,[T]}$$

$$\text{(B-eps)}\ \frac{}{\Gamma \vdash \epsilon\ : Cap} \qquad \text{(B-name)}\ \frac{\Gamma(n) = W}{\Gamma \vdash n\ : W} \qquad \text{(B-varExp)}\ \frac{\Gamma(x) = W}{\Gamma \vdash x\ : W}$$

Typing rules for processes

$$\text{(B-pref)}\ \frac{\Gamma \vdash M\ : bi\texttt{Cap}\,[T] \qquad \Gamma \vdash P\ : bi\texttt{Proc}\,[T]}{\Gamma \vdash M.\, P\ : bi\texttt{Proc}\,[T]} \qquad \text{(B-amb)}\ \frac{\Gamma \vdash M\ : bi\texttt{Amb}\,[T] \qquad \Gamma \vdash P\ : bi\texttt{Proc}\,[T]}{\Gamma \vdash M[\, P\,]\ : Proc}$$

$$\text{(B-par)}\ \frac{\Gamma \vdash P_1\ : bi\texttt{Proc}\,[T] \qquad \Gamma \vdash P_2\ : bi\texttt{Proc}\,[T]}{\Gamma \vdash P_1 \mid P_2\ : bi\texttt{Proc}\,[T]} \qquad \text{(B-res)}\ \frac{\Gamma, n : Amb \vdash P\ : Proc}{\Gamma \vdash (\boldsymbol{\nu} n : Amb)\ P\ : Proc}$$

$$\text{(B-msg)}\ \frac{\Gamma \vdash M\ : W}{\Gamma \vdash \langle M \rangle\ : bi\texttt{Proc}\,[W]} \qquad \text{(B-abs)}\ \frac{\Gamma, x : W \vdash P\ : bi\texttt{Proc}\,[W]}{\Gamma \vdash (x : W)P\ : bi\texttt{Proc}\,[W]}$$

$$\text{(B-nil)}\ \frac{}{\Gamma \vdash \mathbf{0}\ : Proc} \qquad \text{(B-rec)}\ \frac{\Gamma, X : Proc \vdash P\ : Proc}{\Gamma \vdash \texttt{rec}\, X.\, P\ : Proc} \qquad \text{(B-varProc)}\ \frac{\Gamma(X) = Proc}{\Gamma \vdash X\ : Proc}$$

Table 2: The rules for basic types

Type grammars

			Thread rights				*Types of messages*
I, J	$:=$	$\uparrow$	*(thread)*	Tt	$:=$	$I\,W$	*(ST message)*
	$\mid$	$\dagger$	*(no thread)*		$\mid$	$\mathtt{Shh}$	*(no message)*
			Types of expressions				*Types of processes*
Amb	$:=$	$\mathtt{STAmb}\,I\,[Tt]$	*(ST ambient)*	$Proc$	$:=$	$\mathtt{STProc}\,I\,[Tt]$	*(ST process)*
Cap	$:=$	$\mathtt{STCap}\,I\,[Tt]$	*(ST capability)*				

Typing rules for expressions

$$\text{(ST-open1)}\ \frac{\Gamma \vdash M\ :\mathtt{STAmb}\,I\,[Tt]}{\Gamma \vdash \mathtt{open}\,M\ :\mathtt{STCap}\,I^{-1}\,[Tt]}$$

$$\text{(ST-open2)}\ \frac{\Gamma \vdash M\ :\mathtt{STAmb}\ \dagger\ [\dagger\,W]}{\Gamma \vdash \mathtt{open}\,M\ :bi\mathtt{Cap}\,[W]} \qquad \text{(ST-open3)}\ \frac{\Gamma \vdash M\ :\mathtt{STAmb}\ \dagger\ [\mathtt{Shh}]}{\Gamma \vdash \mathtt{open}\,M\ :bi\mathtt{Cap}\,[\mathtt{Shh}]}$$

$$\text{(ST-coopen)}\ \frac{\Gamma \vdash M\ :\mathtt{STAmb}\,I\,[Tt]}{\Gamma \vdash \overline{\mathtt{open}}\,M\ :\mathtt{STCap}\,I\,[Tt]} \qquad \text{(ST-path)}\ \frac{\Gamma \vdash M_1\ :\mathtt{STCap}\ \uparrow\ [Tt] \qquad \Gamma \vdash M_2\ :\mathtt{STCap}\,I\,[Tt]}{\Gamma \vdash M_1.\,M_2\ :\mathtt{STCap}\,I\,[Tt]}$$

$$\text{(ST-inout)}\ \frac{M \in \{\mathtt{in}\,N, \mathtt{out}\,N\} \qquad \Gamma \vdash N\ :Amb}{\Gamma \vdash M\ :\mathtt{STCap}\ \uparrow\ [Tt]} \qquad \text{(ST-co-inout)}\ \frac{M \in \{\overline{\mathtt{in}}\,N, \overline{\mathtt{out}}\,N\} \qquad \Gamma \vdash N\ :\mathtt{STAmb}\,I\,[Tt]}{\Gamma \vdash M\ :\mathtt{STCap}\ \uparrow\ [Tt']}$$

Typing rules for processes

$$\text{(ST-pref)}\ \frac{\Gamma \vdash M\ :\mathtt{STCap}\,I\,[Tt] \qquad \Gamma \vdash P\ :\mathtt{STProc}\,I\,[Tt]}{\Gamma \vdash M.\,P\ :\mathtt{STProc}\ \uparrow\ [Tt]}$$

$$\text{(ST-amb)}\ \frac{\Gamma \vdash M\ :\mathtt{STAmb}\,I\,[Tt] \qquad \Gamma \vdash P\ :\mathtt{STProc}\ \uparrow\ [Tt]}{\Gamma \vdash M[\,P\,]\ :Proc}$$

$$\text{(ST-par1)}\ \frac{\Gamma \vdash P_1\ :\mathtt{STProc}\ \uparrow\ [Tt] \qquad \Gamma \vdash P_2\ :\mathtt{STProc}\ \dagger\ [Tt]}{\Gamma \vdash P_1 \mid P_2\ :\mathtt{STProc}\ \uparrow\ [Tt]}$$

$$\text{(ST-par2)}\ \frac{\Gamma \vdash P_1\ :\mathtt{STProc}\ \dagger\ [Tt] \qquad \Gamma \vdash P_2\ :\mathtt{STProc}\ \uparrow\ [Tt]}{\Gamma \vdash P_1 \mid P_2\ :\mathtt{STProc}\ \uparrow\ [Tt]}$$

$$\text{(ST-par3)}\ \frac{\Gamma \vdash P_1\ :\mathtt{STProc}\ \dagger\ [Tt] \qquad \Gamma \vdash P_2\ :\mathtt{STProc}\ \dagger\ [Tt]}{\Gamma \vdash P_1 \mid P_2\ :\mathtt{STProc}\ \dagger\ [Tt]}$$

$$\text{(ST-msg1)}\ \frac{\Gamma \vdash M\ :W}{\Gamma \vdash \langle M \rangle\ :\mathtt{STProc}\ \uparrow\ [\uparrow\ W]} \qquad \text{(ST-msg2)}\ \frac{\Gamma \vdash M\ :W}{\Gamma \vdash \langle M \rangle\ :\mathtt{STProc}\,J\,[\dagger\,W]}$$

$$\text{(ST-abs)}\ \frac{\Gamma, x : W \vdash P\ :\mathtt{STProc}\,I\,[\,I\,W]}{\Gamma \vdash (x : W)P\ :\mathtt{STProc}\,J\,[\,I\,W]} \qquad \text{(ST-rec)}\ \frac{\Gamma \vdash \mathtt{rec}\,X.\,P\ :\mathtt{STProc}\ \dagger\ [Tt]}{\Gamma \vdash \mathtt{rec}\,X.\,P\ :\mathtt{STProc}\ \uparrow\ [Tt]}$$

Table 3: Additional productions and rules for ST types

any moment at most one capability or message are ready to fire. Therefore there can be non-determinism on reception of messages (two abstractions competing for the same message) but not on emission (two messages competing for the same abstraction).

Additional productions and rules for ST types are in Table 3. We use I and J as thread tags, indicating absence ($\dagger$) or presence ($\upharpoonleft$) of thread. Type $\mathtt{STAmb}\, I\,[Tt]$ describes ambients that when opened: have thread right I; may unleash messages of type Tt. Type $\mathtt{STCap}\, I\,[Tt]$ describes capabilities that when used: leave thread right I; may cause the unleash of messages of type Tt. Type $\mathtt{STProc}\, I\,[Tt]$ describes processes that have thread right I and can unleash messages of type Tt. Type $I\,W$ describes messages that carry: a thread right I (that is, if $I = \upharpoonleft$ then consuming the message means acquiring the thread, whereas if $I = \dagger$ then the thread is not acquired); an expression of type W as a content.

For instance, if in (7) the type of m is of the form $\mathtt{STAmb} \upharpoonleft [Tt]$ then, upon opening m, the thread over n moves from P to Q; the thread rights for P and Q appear in the types of $\mathtt{open}\, m$ and $\overline{\mathtt{open}}\, m$, which are, respectively, $\mathtt{STCap} \dagger [Tt]$ and $\mathtt{STCap} \upharpoonleft [Tt]$. By contrast, if the type of m is of the form $\mathtt{STAmb} \dagger [Tt]$, then P keeps the thread and Q remains with no thread. In this case the types of $\mathtt{open}\, m$ and $\overline{\mathtt{open}}\, m$ are $\mathtt{STCap} \upharpoonleft [Tt]$ and $\mathtt{STCap} \dagger [Tt]$. In (8) the type of n must be of the form $\mathtt{STAmb}\, I\,[\upharpoonleft W]$, to indicate that messages in n carry a thread. This allows to type the body $x.Q$ of the abstraction as a process with the thread.

The rules for ST ambients are rather different from the rules of existing type systems for MA, and is therefore worth explaining some of them. Rule ST-open1 is used when an ST ambient is opened by another ST ambient, as in (7) above; in the rule, $\upharpoonleft^{-1}$ is $\dagger$, and conversely. Rules ST-open2/3 show that ST ambients can be opened by a basic or immobile ambient; the process so liberated has no thread, nor can it acquire the thread by means of message consumption. For immobile ambients, such strict rules are necessary: subject reduction would otherwise fail. For basic ambients, the motivation is the following: in an ST ambient, a capability or a message carrying the thread are "expected" to be exercised without interferences from the other local processes; this would not be assured if the ambient is opened by a basic ambient. Rule ST-inout shows that an ST ambient can enter or exit any type of ambient (basic, immobile, ST). Rule ST-amb assigns any process type to a well-typed ST ambient, which may therefore be used as a local process inside any type of ambient. Rule ST-pref shows that a prefixed process has always the thread; its continuation keeps the thread iff the capabilities in the prefix allow so. The rules for parallel composition guarantee that at most one of the components has the thread. The rules for messages and abstractions have to take into account whether messages carry a thread; for instance, in ST-abs the body of the abstraction is typed as a process whose thread right is dictated by the thread right of the messages. Certain parts of the system (the occurrences of tag J in ST-msg2 and ST-abs, as well as generic rules such as B-nil) show that the thread is an *affine* (at most one process can hold it) rather than a *linear* (exactly one process can hold it) resource; see Lemma 3.5 below. Rule ST-rec has been added precisely to have this lemma.

The following example shows the usefulness of ST types for the typing of immobile ambients.

Example 3.1 *Consider the process*

$$Q \stackrel{def}{=} n[\,\overline{\mathtt{in}}\, n \mid \mathtt{open}\, h \mid (x : W)P\,] \mid h[\,\mathtt{in}\, n.\, \overline{\mathtt{open}}\, h.\, \langle M \rangle\,]$$

Ambient h moves into n, where h is opened and the message $\langle M \rangle$ is emitted. Ambient h is mobile—it moves. Even though n opens a mobile ambient, n can be typed as immobile. Precisely, suppose $\Gamma \vdash M : W$, and $\Gamma \vdash h : \mathtt{STAmb} \dagger [\dagger W]$, and $\Gamma \vdash n : \mathtt{IAmb}\,[W]$, and $\Gamma \vdash (x : W)P : \mathtt{IProc}\,[W]$; then we have

$$\Gamma \vdash Q : Proc\ .$$

*The ST type $\mathtt{STAmb} \dagger [\dagger W]$ for h shows that this ambient will not keep the thread after being opened. This means that all movement capabilities (*in *and* out*) inside h are consumed* before *h is opened. It is therefore safe for an immobile ambient, such as n, to open h.*

In MA, without coactions and hence without ST types, types for immobility are less precise. For instance, without the coaction $\overline{\mathtt{open}}\, h$, the syntax of ambient h would not show whether h is opened before or after its jump. Indeed, the type system for controlling mobility in MA [2] forbids immobile ambients from opening any *mobile ambients, precisely because the processes so liberated could, in principle, use* in *or* out *capabilities. Immobile ambients may only open other immobile ambients. Hence, in the MA version of process Q above, process n would not be typable as immobile.*

We shall find similar situations in the examples of the firewall protocol and of the channel encoding in Section 6: the SA versions of the firewall agent w and the channel ambients p^{ch} will be typable as immobile, whereas their MA versions are not.

The three theorems below assert the soundness of the typing rules. The first two show that immobile and ST processes do respect the behavioural restrictions described above. We say that an occurrence of an expression M in a process P is *unguarded* if that occurrence is not underneath a prefix, an ambient, or an abstraction; likewise for unguarded messages. The third theorem shows preservation of types under reduction.

Theorem 3.2 (immobility types) *If $\Gamma \vdash P : \mathtt{IProc}\,[T]$ then P has no unguarded capabilities of the form $\mathtt{in}\, h$, $\mathtt{out}\, h$ or $\overline{\mathtt{open}}\, h$.*

Theorem 3.3 (single-threadness types)
1. *If $\Gamma \vdash P : \mathtt{STProc} \dagger [Tt]$ then P has no unguarded capabilities; moreover if $Tt = \upharpoonleft W$ (for some W) then P has also no unguarded message.*
2. *If $\Gamma \vdash P : \mathtt{STProc} \upharpoonleft [Tt]$ then P has at most one unguarded capability; moreover if $Tt = \upharpoonleft W$ (for some W) then P has at most one unguarded capability or message.*

Theorem 3.4 (Subject Reduction) *If $\Gamma \vdash P : Proc$ and $P \longrightarrow P'$ then also $\Gamma \vdash P' : Proc$.*

As a consequence of Theorems 3.2 and 3.3, a system in which each ambient is either immobile or ST presents no grave interferences.

Lemmas 3.5 and 3.6 show that the type systems allow a (limited) form of subtyping.

Lemma 3.5 *Suppose P has no free recursion variables. If $\Gamma \vdash P : \mathtt{STProc} \dagger [Tt]$ then also $\Gamma \vdash P : \mathtt{STProc} \upharpoonleft [Tt]$.*

Lemma 3.6 *Suppose P has no free recursion variables.*

1. *If* $\Gamma \vdash P : \texttt{STProc} \dagger [\dagger W]$ *(resp.* $\Gamma \vdash P : \texttt{STProc} \dagger [\texttt{Shh}]$ *) then also* $\Gamma \vdash P : \texttt{BProc}[W]$ *(resp.* $\Gamma \vdash P : \texttt{BProc}[\texttt{Shh}]$ *).*
2. *If* $\Gamma \vdash P : \texttt{STProc} \dagger [\dagger W]$ *(resp.* $\Gamma \vdash P : \texttt{STProc} \dagger [\texttt{Shh}]$ *) then also* $\Gamma \vdash P : \texttt{IProc}[W]$ *(resp.* $\Gamma \vdash P : \texttt{IProc}[\texttt{Shh}]$ *).*

4 Behavioural equivalence

The behavioural equality we use is *barbed congruence* [13], a contextually-defined, bisimulation-based relation. Other contextual equivalences, such as forms of testing equivalence, would do. Intuitively two processes P and Q are behaviourally equal, written $P \simeq Q$, if no closing contexts can tell them apart. In an untyped calculus, no other constraints are made on processes or contexts. In a typed setting, processes compared should have the same type, and should only be tested in contexts that respect such a type. We write $\Gamma \triangleright P_1 \simeq P_2 : Proc$ if $\Gamma \vdash P_i : Proc \ \ (i = 1, 2)$, and P_1 and P_2 are indistinguishable by closing contexts that respect the type information Γ and *Proc*. We omit the formal definitions of the behavioural equality, for they are not essential to understand the contents of the paper.

Capability constraints. In the algebraic laws of Section 5, we shall control plain interferences among processes by constraining the use of capabilities in processes. We use two kinds of constraints. The first kind is for laws of the form

$$\Gamma\{L\} \triangleright P_1 \simeq P_2 : Proc$$

where L is a set of simple capabilities. This means that $\Gamma \vdash P_i : Proc$ and that processes P_1 and P_2 cannot be distinguished by contexts that, besides respecting the type information Γ and *Proc*, never use the capabilities in L. This requirement on L can be formalised along the lines of existing type systems for capability control in the π-calculus such as [15, 10, 19]. A simpler—though more restrictive—syntactic condition is the following: the contexts in which P_1 and P_2 may be placed (i) do not use the capabilities in L in prefixes or messages, (ii) do not transmit names appearing in L in messages (otherwise, a recipient could reconstruct and use a capability of L).

Our second form of capability constraint uses the notion of *uniform receptiveness* [18]. Roughly, a simple capability μ is uniformly receptive if the unique process that may use it is of the form $!\mu. P'$ (where P' itself cannot use μ). Assuming that process $!\mu. P'$ does not move (which is the case if the process is local to an immobile ambient), this condition ensures that processes attempting to consume μ do not interfere among them: μ stays in the same place, is always available, and always activates the same continuation P'. Again, the uniform receptiveness condition is best formalised by means of a type system [18]. But, as the laws in which we shall use it are very simple, in this paper we simply regard it as a syntactic condition, restricting the use of capabilities both in the processes of the laws and in the contexts in which the processes may be tested. We write $\Gamma\{rec : L\} \triangleright P_1 \simeq P_2 : Proc$ to mean that P_1 and P_2 are indistinguishable under the additional hypothesis of uniform receptiveness for the capabilities in L.

We write $\Gamma \triangleright P_1 \simeq P_2$ to mean that $\Gamma \triangleright P_1 \simeq P_2 : Proc$ holds for any type *Proc* such that $\Gamma \vdash P_i : Proc \ \ (i = 1, 2)$. Likewise for $\Gamma\{L\} \triangleright P_1 \simeq P_2$ and $\Gamma\{rec : L\} \triangleright P_1 \simeq P_2$.

5 Algebraic laws

Key algebraic laws for both sequential and concurrent calculi are those that make a reduction step into a behavioural equality. If the reduction relation is confluent, then all reduction steps are also behavioural equalities. But if reduction is not confluent, as usual in concurrency, then some conditions have to be imposed to obtain equalities. This section presents a sample of laws of this kind, for both untyped and typed SA. Therefore in all the laws in this section the left-hand side process may reduce in one step to the right-hand side. (Other algebraic laws are of course those for structural equivalence.) The laws of Lemma 5.1 are already valid in untyped SA, hence also in its typed versions.

Lemma 5.1 *We have:*

1. $\boldsymbol{\nu} n\, (m[\, \texttt{in}\, n. P \,] \mid n[\, \overline{\texttt{in}}\, n. Q \,]) \simeq \boldsymbol{\nu} n\, (n[\, Q \mid m[\, P \,]\,])$;
2. $\boldsymbol{\nu} m\, (\texttt{open}\, m. P \mid m[\, \overline{\texttt{open}}\, m. Q \,]) \simeq \boldsymbol{\nu} m\, (P \mid Q)$;
3. $k[\, \texttt{open}\, m. P \mid m[\, \overline{\texttt{open}}\, m. Q \,]\,] \simeq k[\, P \mid Q \,]$;
4. $n[\, \overline{\texttt{out}}\, n. P \mid m[\, \texttt{out}\, n. Q \,]\,] \simeq m[\, Q \,] \mid n[\, P \,]$;
5. $n[\, \langle M \rangle \mid (x)\, P \,] \simeq n[\, P\{ {}^{M}\!/x \} \,]$.

The laws of Lemma 5.1 are sometimes useful (for instance, they are almost enough for the algebraic proof of the firewall crossing protocol and the second encoding of channels in Section 6). In general, however, they are too restrictive, in that they are only applicable to ambients with a single local process. We obtain more powerful laws exploiting ST and immobility types. The reason why these types are useful is that they remove all forms of grave interferences. We therefore only need conditions for controlling plain interferences. The first set of laws is about interactions between ST ambients. We write $P \overset{\texttt{out}\, n}{\not\Longrightarrow}$ if no ambient in P demands to exit n; formally, there are no $m, \tilde{n}, Q_1, Q_2, Q_3$ such that $P \Longrightarrow\equiv \boldsymbol{\nu} \tilde{n}\, (m[\, \texttt{out}\, n. Q_1 \mid Q_2 \,] \mid Q_3)$ with $n \notin \tilde{n}$ ($P \overset{\texttt{out}\, n}{\not\Longrightarrow}$ is undecidable, but one may think of various syntactic conditions that will imply it, for instance P being an abstraction or not containing n free).

Lemma 5.2 *Suppose ambients n and m are ST in Γ. We have*

1. $\Gamma\{\texttt{in}\, n, \overline{\texttt{in}}\, n\} \triangleright m[\, \texttt{in}\, n. P_1 \mid P_2 \,] \mid n[\, \overline{\texttt{in}}\, n. Q_1 \mid Q_2 \,] \simeq n[\, Q_1 \mid Q_2 \mid m[\, P_1 \mid P_2 \,]\,]$;
2. $\Gamma\{\texttt{open}\, m, \overline{\texttt{open}}\, m\} \triangleright \texttt{open}\, m. P \mid m[\, \overline{\texttt{open}}\, m. Q_1 \mid Q_2 \,] \simeq P \mid Q_1 \mid Q_2$;
3. $\Gamma \triangleright n[\, \overline{\texttt{out}}\, n. P_1 \mid m[\, \texttt{out}\, n. Q_1 \mid Q_2 \,] \mid P_2 \,] \simeq m[\, Q_1 \mid Q_2 \,] \mid n[\, P_1 \mid P_2 \,]$, *if* $P_2 \overset{\texttt{out}\, n}{\not\Longrightarrow}$;
4. $\Gamma \triangleright n[\, \langle M \rangle \mid (x)\, P \mid Q \,] \simeq m[\, P\{ {}^{M}\!/x \} \mid Q \,]$, *if messages in n carry the thread (i.e., $\Gamma(n)$ is of the form* $\texttt{STAmb}\, I\, [\uparrow\ W]$*) and no abstractions appear unguarded in Q.*

Law (1) does not hold without the capability constraints on $\texttt{in}\, n$, $\overline{\texttt{in}}\, n$, for the surrounding contexts may contain an ambient that wishes to enter n (thus interfering with m) or another ambient n that allows to be entered (thus interfering with the existing n). Similarly, the constraints on $\texttt{open}\, m, \overline{\texttt{open}}\, m$ are necessary in law (2). The capability constraints in laws (1) and (2) may be removed by means of restrictions or ambients:

Corollary 5.3 *Suppose ambients n and m are ST in Γ. We have*

1. $\Gamma \triangleright \boldsymbol{\nu} n\left(m[\mathtt{in}\, n.\, P_1 \mid P_2] \mid n[\overline{\mathtt{in}}\, n.\, Q_1 \mid Q_2]\right) \simeq \boldsymbol{\nu} n\left(n[Q_1 \mid Q_2 \mid m[P_1 \mid P_2]]\right)$;

2a. $\Gamma \triangleright \boldsymbol{\nu} m\, (\mathtt{open}\, m.\, P \mid m[\overline{\mathtt{open}}\, m.\, Q_1 \mid Q_2]) \simeq \boldsymbol{\nu} m\, (P \mid Q_1 \mid Q_2)$;

2b. $\Gamma \triangleright k[\mathtt{open}\, m.\, P \mid m[\overline{\mathtt{open}}\, m.\, Q_1 \mid Q_2]] \simeq k[P \mid Q_1 \mid Q_2]$.

Remark 5.4 *In the laws (1) and (2) of Lemma 5.2, types (in particular the ST type condition on ambients n and m) can be removed; the price is that the equalities of the laws have to be syntactically strengthened by eliminating processes P_2 and Q_2. The resulting laws, like those of Lemma 5.1, are only applicable to ambients with a single local process.*

Next we formulate laws concerning interactions between ST and immobile ambients. Laws similar to those of Lemma 5.2 and Corollary 5.3 hold. For lack of space we present only some laws that are specific to immobile ambients, and which involve uniformly receptive capabilities. An immobile ambient n often uses capabilities such as $\overline{\mathtt{in}}\, n$, $\overline{\mathtt{out}}\, n$, or $\mathtt{open}\, m$ in a uniformly receptive way. This happens if n continuously enables movements into and out of it, or is continuously ready to open ambients named m (examples are the channel ambients in the first encoding of channels, Section 6). Here are some laws for uniform receptiveness in SA.

Lemma 5.5 *Suppose that ambient n is immobile in Γ, and ambient m is ST. We have:*

1. $\Gamma\{rec : \overline{\mathtt{in}}\, n\} \triangleright m[\mathtt{in}\, n.\, P_1 \mid P_2] \mid n[!\overline{\mathtt{in}}\, n.\, Q_1 \mid Q_2] \simeq n[!\overline{\mathtt{in}}\, n.\, Q_1 \mid Q_1 \mid Q_2 \mid m[P_1 \mid P_2]]$;
2. $\Gamma\{rec : \overline{\mathtt{out}}\, n\} \triangleright n[!\overline{\mathtt{out}}\, n.\, P_1 \mid m[\mathtt{out}\, n.\, Q_1 \mid Q_2] \mid P_2] \simeq m[Q_1 \mid Q_2] \mid n[!\overline{\mathtt{out}}\, n.\, P_1 \mid P_1 \mid P_2]$;
3. $\Gamma\{rec : \mathtt{open}\, m\} \triangleright n[!\mathtt{open}\, m.\, P_1 \mid m[\overline{\mathtt{open}}\, m.\, Q_1 \mid Q_2] \mid P_2] \simeq n[!\mathtt{open}\, m.\, P_1 \mid P_1 \mid Q_1 \mid Q_2 \mid P_2]$.

In law (1), the uniform receptiveness hypothesis on $\overline{\mathtt{in}}\, n$ makes sure that this capability is used neither by P_i, Q_i ($i = 1, 2$) nor by other ambients in the surrounding environment. Similarly for laws (2) and (3).

For the proofs of the laws in this section we have exploited an SOS presentation of the semantics of SA, along the lines of that for MA in [4].

6 Examples

There is an obvious encoding of MA into SA, where each ambient n is given unlimited availability of the coactions $\overline{\mathtt{in}}\, n$, $\overline{\mathtt{out}}\, n$, $\overline{\mathtt{open}}\, n$ (that is, process $!\overline{\mathtt{in}}\, n \mid !\overline{\mathtt{out}}\, n \mid !\overline{\mathtt{open}}\, n$ is placed inside n). There is, however, a natural and more conscientious way of rewriting MA programs into SA, whereby coactions are inserted only when actually needed, rather than "by saturation". The firewall protocol and the first encoding of channels below are examples of this. We also look at the protocols for renaming and for routing packets, from [3]. In MA they suffer interference problems, which undermine their correctness. In SA we are able to remove most of the interferences, using coactions and types. Finally we present a different encoding of channels that has various desirable properties over the previous one. This encoding only makes sense in SA: removing coactions we obtain useless MA programs. In all examples: the ambients presented are typable; the behavioural equivalences are proved algebraically, from the laws of structural equivalence and of Section 5.

Renaming. Here is the MA construct for changing the name of an ambient, from [3]:

$$n\ \mathtt{be}\ m.\, P \stackrel{\text{def}}{=} m[\mathtt{out}\, n.\, \mathtt{open}\, n.\, P] \mid \mathtt{in}\, m$$

A process $n\ \mathtt{be}\ m.\, P$ placed inside an ambient n may cause the renaming of the ambient n to m thus: $n[n\ \mathtt{be}\ m.\, P \mid Q] \Longrightarrow m[P \mid Q]$. For the correctness of the protocol, we look at the conditions under which $n[n\ \mathtt{be}\ m.\, P \mid Q]$ and $m[P \mid Q]$ are behaviourally the same. This is stronger than requiring successful completion of the protocol implementing the renaming, but the correctness conditions are essentially the same and the statement is simpler. The equality may fail in MA because, for instance: (i) the external environment contains ambients named n or m; (ii) the external environment contains ambients that try to enter n, or to open n or m; (iii) Q causes n to jump somewhere else; (iv) Q contains an ambient that tries to exit n. There are various reasons why an ambient entering n may disrupt the protocol; for instance the ambient entering could be $h[\mathtt{in}\, n.\, \mathtt{in}\, k]$ and Q could be $\mathtt{open}\, h$, in which case n may jump into k. Similarly an ambient exiting n could interfere: if the ambient that exits is $h[\mathtt{out}\, n.\, \mathtt{open}\, n]$, then this ambient, once out of n, can be opened and then can open n (note that this interference arises even if n is unknown to the external environment).

Conditions (i-iv) are strong: in particular, (i) and (ii) essentially mean that *both n and m* are unknown to the external environment, which probably makes the renaming construct little useful. Renaming becomes, in SA:

$$n\ \mathtt{be}\ m.\, P \stackrel{\text{def}}{=} m[\mathtt{out}\, n.\, \overline{\mathtt{in}}\, m.\, \mathtt{open}\, n.\, P] \mid \overline{\mathtt{out}}\, n.\, \mathtt{in}\, m.\, \overline{\mathtt{open}}\, n$$

Process $n[n\ \mathtt{be}\ m.\, P \mid Q]$ is well typed in type environments in which n is ST. For instance, it holds that $\Gamma \vdash n[n\ \mathtt{be}\ m.\, P \mid Q] : \mathit{Proc}$ if Γ is such that

- $\Gamma \vdash m : \mathtt{STAmb}\, I\, [Tt]$ (for any I),
- $\Gamma \vdash n : \mathtt{STAmb}\ \dagger\ [Tt]$,
- $\Gamma \vdash P : \mathtt{STProc}\ \uparrow\ [Tt]$,
- $\Gamma \vdash Q : \mathtt{STProc}\ \dagger\ [Tt]$,

or if Γ is such that

- $\Gamma \vdash m : \mathtt{BAmb}\, [W]$,
- $\Gamma \vdash n : \mathtt{STAmb}\ \dagger\ [\dagger W]$,
- $\Gamma \vdash P : \mathtt{BProc}\, [W]$,
- $\Gamma \vdash Q : \mathtt{STProc}\ \dagger\ [\dagger W]$.

Behaviourally, we have: if $Q \overset{\mathtt{out}\, n}{\not\Longrightarrow}$, then

$$\Gamma\{\overline{\mathtt{in}}\, m, \mathtt{in}\, m\} \triangleright n[n\ \mathtt{be}\ m.\, P \mid Q] \simeq m[P \mid Q].$$

The single threadedness of n guarantees that Q does not cause unexpected movements of n; the capability constraints on $\mathtt{in}\, m, \overline{\mathtt{in}}\, m$ guarantee that no processes in the environment interfere with the movement of ambient n into m; the semantic condition on Q guarantees that no ambients in Q try to exit n, stealing the coaction $\overline{\mathtt{out}}\, n$ to the ambient m. An ambient in the external environment trying to enter n, or to open n or m, does not interfere, since the complementary coaction is not available.

If the *target* name m is only known to P, then the capability constraints on $\overline{\mathtt{in}}\, m$ and $\mathtt{in}\, m$ are not necessary:

$$\Gamma \triangleright n[\boldsymbol{\nu} m\, (n \;\mathtt{be}\; m.\, P) \mid Q] \simeq \boldsymbol{\nu} m\, (m[P \mid Q])$$

Conversely, suppose the *source* name n is only known to P, and define

$$n \;\mathtt{be}^{+}\; m.\, P \stackrel{\mathrm{def}}{=} \boldsymbol{\nu} h(\;\; h[\mathtt{out}\, n.\, \overline{\mathtt{in}}\, h.\, \overline{\mathtt{out}}\, h \mid m[\overline{\mathtt{in}}\, m.\, \mathtt{open}\, n]] \mid \overline{\mathtt{out}}\, n.\, \mathtt{in}\, h.\, \mathtt{in}\, m.\, \overline{\mathtt{open}}\, n.\, \mathtt{out}\, h.\, P)$$

where h is any fresh name. In this case correctness holds without capability constraints and the semantic condition on Q:

$$\Gamma \triangleright \boldsymbol{\nu} n\, (n[n \;\mathtt{be}^{+}\; m.\, P \mid Q]) \simeq \boldsymbol{\nu} n\, (m[P \mid Q]).$$

Combining the two last results, we obtain a renaming construct that works under the only conditions that n is ST in Γ and $Q \stackrel{\mathtt{out}\, n}{\not\Longrightarrow}$:

$$\Gamma \triangleright n[\boldsymbol{\nu} k\, (n \;\mathtt{be}\; k.\, (k \;\mathtt{be}^{+}\; m.\, P)) \mid Q] \simeq m[P \mid Q]$$

where k is any fresh name.

Routable Packets. In this protocol, from [3], a packet can be repeatedly routed to various destinations. The content P and the destination M are contained in an ambient *route*. The act of sending P to destination M is realized by the following steps. Ambient *route* enters inside the packet and is opened. This liberates a message $\langle M\rangle$, which is then consumed so that the path M can be executed. At the end, the packet, which contains P, has reached destination. Here is the MA program:

$$\begin{array}{lcll} \mathit{PKT} & \stackrel{\mathrm{def}}{=} & \mathit{pkt}[\,!(x)\, x \mid !\mathtt{open}\, \mathit{route}\,] & \text{(the packet)} \\ \langle P, M\rangle & \stackrel{\mathrm{def}}{=} & \mathit{route}[\mathtt{in}\, \mathit{pkt}.\, \langle M\rangle \mid P] & \\ & & & (P \text{ is routed to destination } M\,) \end{array}$$

The surrounding environment may damage the protocol: two routers could enter PKT and interfere on the path to follow; *pkt* and *route* could be opened. (P itself could interfere.) To see an interference between two routers, here a system with a reduction in which none of the contents P_1, P_2 reaches destination:

$$\begin{array}{lll} \mathit{SYS} & \stackrel{\mathrm{def}}{=} & \mathit{PKT} \\ & & \mid \langle P_1, \mathtt{in}\, n.\, \mathtt{in}\, k\rangle \\ & & \mid \langle P_2, \mathtt{in}\, m\rangle \\ & & \mid n[Q_1 \mid m[Q_2] \mid k[Q_3]] \\ & & \mid m[Q_2] \\ & \Longrightarrow & n[\;\; Q_1 \\ & & \quad \mid m[Q_2 \mid \mathit{pkt}[\;\; !(x)\, x \\ & & \qquad\qquad \mid !\mathtt{open}\, \mathit{route} \\ & & \qquad\qquad \mid P_1 \\ & & \qquad\qquad \mid \langle \mathtt{in}\, k\rangle \\ & & \qquad\qquad \mid P_2] \;\;] \\ & & \quad \mid k[Q_3] \;\;] \\ & & \mid m[Q_2] \end{array}$$

The packet has ended up in the inner ambient m, rather than, as expected, in the ambient k or in the outer ambient m.

Further complications would arise if we wished a router content P to exit *pkt* upon arrival at destination (P should not exit before arrival, but must exit before *pkt* moves again.) To avoid all these problems the protocol has to be substantially complicated, along the lines of the taxi protocol in [7]. Below is the SA protocol, with the additional demand on exit for P (which explains why P is placed inside an ambient *dest*).

$$\begin{array}{lcl} \mathit{PKT} & \stackrel{\mathrm{def}}{=} & \mathit{pkt}[R] \\ \langle P, M\rangle & \stackrel{\mathrm{def}}{=} & \\ & & \mathit{route}[\mathtt{in}\, \mathit{pkt}.\, \overline{\mathtt{open}}\, \mathit{route}.\, \langle M\rangle \mid \mathit{dest}[\mathtt{out}\, \mathit{pkt}.\, P]] \end{array}$$

where

$$R \stackrel{\mathrm{def}}{=} \mathtt{rec}\; X.\, \overline{\mathtt{in}}\, \mathit{pkt}.\, \mathtt{open}\, \mathit{route}.\, (x)\, x.\, \overline{\mathtt{out}}\, \mathit{pkt}.\, X$$

Ambients *pkt* and *route* can be typed as ST ambients. Hence the above mentioned (grave) interferences disappear. For instance, ambients *pkt* and *route* cannot be opened by the environment since opening is controlled by $\overline{\mathtt{open}}\, \mathit{pkt}$ and $\overline{\mathtt{open}}\, \mathit{route}$; and two routers cannot enter *pkt* at the same time since entering is controlled by $\overline{\mathtt{in}}\, \mathit{pkt}$.
Consider the SA program corresponding to SYS, where Q_a, for $a \in \{n, m, k\}$, is of the form $!\overline{\mathtt{in}}\, a.\, Q'_a \mid Q''_a$:

$$\begin{array}{c} \mathit{PKT} \mid \langle P_1, \mathtt{in}\, n.\, \mathtt{in}\, k\rangle \mid \langle P_2, \mathtt{in}\, m\rangle \\ \mid n[Q_n \mid m[Q_m] \mid k[Q_k]] \mid m[Q_m] \end{array}$$

It can make two reductions, depending on whether the first or the second router enters. Suppose it is the first; the derivative is then

$$\begin{array}{lcl} \mathit{SYS}_1 & \stackrel{\mathrm{def}}{=} & \mathit{pkt}[\;\; \mathtt{open}\, \mathit{route}.\, (x)\, x.\, \overline{\mathtt{out}}\, \mathit{pkt}.\, R \mid \\ & & \qquad \mathit{route}[\;\; \overline{\mathtt{open}}\, \mathit{route}.\, \langle \mathtt{in}\, n.\, \mathtt{in}\, k\rangle \\ & & \qquad\qquad \mid \mathit{dest}[\mathtt{out}\, \mathit{pkt}.\, P_1] \;\;]\,] \\ & & \mid \mathit{SYS}'_1 \end{array}$$

where

$$\begin{array}{lcl} \mathit{SYS}'_1 & \stackrel{\mathrm{def}}{=} & \\ & & \langle P_2, \mathtt{in}\, m\rangle \mid n[Q_n \mid m[Q_m] \mid k[Q_k]] \mid m[Q_m] \end{array}$$

Let Γ be a type environment in which ambients *pkt* and *route* are ST and n, m, k are immobile. Assuming also that $\overline{\mathtt{in}}\, n, \overline{\mathtt{in}}\, k$ are uniformly receptive, we have

$$\begin{array}{lcl} \Gamma\{\mathit{rec} : \overline{\mathtt{in}}\, n, \overline{\mathtt{in}}\, k\} & \triangleright & \\ \mathit{SYS}_1 & \simeq & \langle P_2, \mathtt{in}\, m\rangle \\ & & \mid n[\;\; Q'_n \mid Q_n \mid m[Q_m] \\ & & \quad \mid k[\;\; Q'_k \mid Q_k \\ & & \qquad \mid \mathit{PKT} \mid \mathit{dest}[P_1] \;\;] \;\;] \\ & & \mid m[Q_m] \end{array}$$

(The uniform receptiveness condition on $\overline{\mathtt{in}}\, n$ is satisfied if $\overline{\mathtt{in}}\, n$ is used neither by Q_m, Q_k, Q'_n and Q''_n nor by the surrounding environment. Similarly for $\overline{\mathtt{in}}\, k$.)

The equivalence above shows that P_1 will always correctly reach destination.

The case when the second router enters is more difficult, and for this we need laws concerning interactions between ST and immobile ambients more powerful than those in Lemma 5.5; we leave their formalisation for future work.

Crossing a firewall. A protocol is discussed in [3] for controlling accesses through a firewall. An ambient w represents the firewall; an ambient m a trusted agent inside which is a process Q that is supposed to cross the firewall. The firewall ambient sends into the agent a pilot ambient k with the capability $\mathtt{in}\, w$ for entering the firewall. The agent acquires the capability by opening k. The process Q carried by the agent is finally liberated inside the firewall by the opening of ambients m and k'. Names m, k, k' act like passwords which guarantee the access only to authorized agents. Here is the protocol, in MA:

$$\begin{array}{rcl} AG & \stackrel{\text{def}}{=} & m[\,\mathtt{open}\, k.\, k'[\,Q\,]\,] \\ FW & \stackrel{\text{def}}{=} & \boldsymbol{\nu} w\,(w[\quad \mathtt{open}\, m.\, \mathtt{open}\, k'.\, P \\ & & \qquad\quad \mid k[\,\mathtt{out}\, w.\, \mathtt{in}\, m.\, \mathtt{in}\, w\,] \quad]\ ) \end{array}$$

The correctness of the protocol is shown in [6], proving that

$$(\boldsymbol{\nu} m, k, k'\,)\Big(FW \mid AG\Big) \simeq \boldsymbol{\nu} w\,\Big(w[\,Q \mid P\,]\Big)$$

under the conditions that $w \notin \mathrm{fn}(Q)$ and $(\mathrm{fn}(Q) \cup \mathrm{fn}(P)) \cap \{m, k, k'\} = \emptyset$.

The restrictions on both m and k are necessary to avoid grave interferences. For instance, the protocol would be damaged if external processes opened ambients m and k, or if external ambients entered inside m or k. The ambient k', and the condition $\{k', k, m\} \cap \mathrm{fn}(Q) = \emptyset$ are needed to block process Q, so to avoid interferences due to actions from Q. Without ambient k', process Q would be local to m, and could cause m to move somewhere else rather than into the firewall. Without the condition $\{m\} \cap \mathrm{fn}(Q) = \emptyset$, process Q could cause ambient $k'[\,Q\,]$ to jump out of m before m has entered the firewall. Similarly for the other conditions.
Below is essentially the same protocol, written in SA.

$$\begin{array}{rcl} AG & \stackrel{\text{def}}{=} & m[\,\overline{\mathtt{in}}\, m.\, \mathtt{open}\, k.\, (x)\, x.\, \overline{\mathtt{open}}\, m.\, Q\,] \\ FW & \stackrel{\text{def}}{=} & \boldsymbol{\nu} w\,(w[\quad \overline{\mathtt{out}}\, w.\, \overline{\mathtt{in}}\, w.\, \mathtt{open}\, m.\, P \\ & & \qquad\quad \mid k[\,\mathtt{out}\, w.\, \mathtt{in}\, m.\, \overline{\mathtt{open}}\, k.\, \langle \mathtt{in}\, w\rangle\,] \quad]\ ) \end{array}$$

Coactions aside, there are two differences with the MA protocol: (i) ambient k' is not needed to control the movements of Q; (ii) the capability $\mathtt{in}\, w$ for entering the firewall is communicated to the agent as a message. The simplification (i) is possible because Q is blocked underneath the coaction $\overline{\mathtt{open}}\, m$. The modification (ii) is not essential, but is important for proving correctness with the algebraic laws of section 5.

If $w \notin \mathrm{fn}(Q)$ and $m \notin \mathrm{fn}(P)$, we have:

$$\boldsymbol{\nu} m\,\Big(AG \mid FW\Big) \simeq (\boldsymbol{\nu} m, w\,)\Big(w[\,P \mid Q\,]\Big)$$

The restriction on m can be replaced by a capability constraint on $\mathtt{in}\, m$ and $\overline{\mathtt{in}}\, m$:

$$\{\mathtt{in}\, m, \overline{\mathtt{in}}\, m\} \triangleright AG \mid FW \simeq \boldsymbol{\nu} w\,\Big(w[\,P \mid Q\,]\Big)$$

Note that in both SA equivalences no restriction on ambient k is necessary. The two equivalences can be proved purely algebraically (rather than by reasoning over contexts as in [6]), using the untyped laws of Lemma 5.1, and the laws with capability constraints on $\mathtt{in}\, m$ and $\overline{\mathtt{in}}\, m$ indicated in Remark 5.4.

Although types are not needed for the correctness proof, it is worth pointing out that the processes of the protocol may be typed as ST, under appropriate assumptions on the unknown processes P and Q. Let Γ be a type environment such that

- $\Gamma \vdash P\ : \mathtt{STProc} \uparrow [Tt]$,
- $\Gamma \vdash Q\ : \mathtt{STProc} \dagger [Tt]$,
- $\Gamma \vdash m\ : \mathtt{STAmb} \dagger [Tt]$,
- $\Gamma \vdash k\ : \mathtt{STAmb} \uparrow [Tt]$,
- $\Gamma \vdash w\ : \mathtt{STAmb}\, I\, [Tt]$

where $I \in \{\uparrow, \dagger\}$ and Tt is $\uparrow \mathtt{STCap} \uparrow [Tt']$, for some Tt'. We then have

$$\Gamma \vdash FW \mid AG\ : Proc\ .$$

We obtain the same result if the types of P and Q are exchanged. Other typings are possible, for instance one in which w is typed as immobile.

The firewall protocol in MA is also analised by Nielson et al. [14]. The authors employ control-flow analysis techniques to compute the set of ambients that may be at the top-level of a given ambient. Using such an analysis, they prove the property that agents not knowing the appropriate passwords cannot enter the firewall. We think that a similar analysis can be carried out in the SA formulation; it would be interesting however to see if coactions allow one to get sharper results.

Encoding of channels. We present the standard encoding of the polyadic asynchronous π-calculus into MA [7, 2], rewritten in SA (also the target SA calculus is polyadic, that is with the tuple type $W_1 \times \ldots \times W_n$ among the type for messages). The encoding is an important test of expressiveness. It also shows how to program communications along channels (in MA and in SA communications are anonymous).

First, a briefly recall of the polyadic asynchronous π-calculus. The type grammar is

$$T := Ch(T_1 \mathinner{..} T_n) \qquad n \geq 0$$

The type judgements are of the form $\Gamma \vdash P$, where Γ is a finite assignment of types to channels, and asserts that process P respects the type information in Γ. A π-calculus channels p of type $Ch(T_1 \mathinner{..} T_n)$ carries n-uples of channels, whose i-the component has type T_i. The typing rules are straightforward:

$$\frac{\Gamma \vdash p\ : Ch(T_1 \mathinner{..} T_n) \qquad \Gamma, q_1 : T_1 \mathinner{..} q_n : T_n \vdash P}{\Gamma \vdash p(q_1 \mathinner{..} q_n).\, P}$$

$$\frac{\Gamma \vdash p\ : Ch(T_1 \mathinner{..} T_n) \qquad \Gamma \vdash q_i\ : T_i \quad (1 \leq i \leq n)}{\Gamma \vdash \overline{p}\langle q_1 \mathinner{..} q_n\rangle}$$

$$\frac{\Gamma \vdash P_1 \qquad \Gamma \vdash P_1}{\Gamma \vdash P_1 \mid P_2} \qquad\qquad \frac{\Gamma \vdash P}{\Gamma \vdash !P}$$

$$\frac{\Gamma, p : T \vdash P}{\Gamma \vdash (\boldsymbol{\nu} p : T)\, P} \qquad\qquad \frac{}{\Gamma \vdash \mathbf{0}}$$

$$
\begin{array}{rcl}
\mathtt{ch}\,p & \stackrel{\mathrm{def}}{=} & p^{\mathtt{ch}}[\,!\overline{\mathtt{in}}\,p^{\mathtt{ch}} \mid !\overline{\mathtt{out}}\,p^{\mathtt{ch}} \mid !\mathtt{open}\,p^{\mathtt{io}}\,] \\
[\![p(q_1 .. q_n).\,P]\!] & \stackrel{\mathrm{def}}{=} & (\boldsymbol{\nu} k : \mathtt{BAmb}\,[\mathtt{Shh}])\,\Big(p^{\mathtt{io}}[\,\mathtt{in}\,p^{\mathtt{ch}}.\,\overline{\mathtt{open}}\,p^{\mathtt{io}}.\,(q_1^{\mathtt{ch}}, q_1^{\mathtt{io}} .. q_n^{\mathtt{ch}}, q_n^{\mathtt{io}})\;\;(\,k[\,\mathtt{out}\,p^{\mathtt{ch}}.\,\overline{\mathtt{open}}\,k.\,[\![P]\!]\,]\,)\,] \mid \mathtt{open}\,k\Big) \\
[\![\overline{p}\langle q_1 .. q_n\rangle]\!] & \stackrel{\mathrm{def}}{=} & p^{\mathtt{io}}[\,\mathtt{in}\,p^{\mathtt{ch}}.\,\overline{\mathtt{open}}\,p^{\mathtt{io}}.\,\langle q_1^{\mathtt{ch}}, q_1^{\mathtt{io}} .. q_n^{\mathtt{ch}}, q_n^{\mathtt{io}}\rangle\,] \\
[\![(\boldsymbol{\nu} p : T)\,P]\!] & \stackrel{\mathrm{def}}{=} & (\boldsymbol{\nu} p^{\mathtt{ch}} : [\![T]\!]^{\mathtt{ch}})\,(\boldsymbol{\nu} p^{\mathtt{io}} : [\![T]\!]^{\mathtt{io}})\,(\mathtt{ch}\,p \mid [\![P]\!]) \\
[\![P \mid Q]\!] & \stackrel{\mathrm{def}}{=} & [\![P]\!] \mid [\![Q]\!] \\
[\![!P]\!] & \stackrel{\mathrm{def}}{=} & ![\![P]\!] \\
[\![\mathbf{0}]\!] & \stackrel{\mathrm{def}}{=} & \mathbf{0} \\
[\![Ch(T_1 .. T_n)]\!]^{\mathtt{ch}} & \stackrel{\mathrm{def}}{=} & \mathtt{IAmb}\,[[\![T]\!]_1^{\mathtt{ch}} \times [\![T]\!]_1^{\mathtt{io}} \times .. \times [\![T]\!]_n^{\mathtt{ch}} \times [\![T]\!]_n^{\mathtt{io}}] \\
[\![Ch(T_1 .. T_n)]\!]^{\mathtt{io}} & \stackrel{\mathrm{def}}{=} & \mathtt{STAmb}\,\dagger\,[\dagger\;[\![T]\!]_1^{\mathtt{ch}} \times [\![T]\!]_1^{\mathtt{io}} \times .. \times [\![T]\!]_n^{\mathtt{ch}} \times [\![T]\!]_n^{\mathtt{io}}]
\end{array}
$$

Table 4: Encoding of π-calculus operators and types

In the encoding, a channel p is represented as an ambient $p^{\mathtt{ch}}$. Communications along that channel become communications local to $p^{\mathtt{ch}}$. I/O requests on channel p are rendered by special ambients $p^{\mathtt{io}}$. These ambients move into the channel ambient $p^{\mathtt{ch}}$, where they are opened so that the requests may interact. This yields the encoding of the π-calculus operators and types in Table 4. The complete encoding needs however a further encoding step, to add channel ambients for all the free names of a process:

$$\{\![P]\!\} \stackrel{\mathrm{def}}{=} [\![P]\!] \mid \mathtt{ch}\,p_1 \mid \ldots \mid \mathtt{ch}\,p_n \quad \text{where } p_1 \ldots p_n \text{ are free names of } P$$

The typing of the encoding into MA is studied in depth in [2]. The problem raised is the following. A channel ambient $p^{\mathtt{ch}}$ is immobile—it never jumps into or out of another ambient—but the type system for immobility in [2] is unable to type $p^{\mathtt{ch}}$ as immobile. The reason is that $p^{\mathtt{ch}}$ opens the I/O ambients $p^{\mathtt{io}}$, which are mobile (they can jump, precisely into $p^{\mathtt{ch}}$); as pointed out in Example 3.1, the type system in [2] forbids immobile ambients from opening any mobile ambients. The problem is solved by extending the syntax and the type systems of MA with primitives for *objective moves* [2].

In SA, by contrast, coactions and ST types show that the I/O ambients $p^{\mathtt{io}}$ are opened only *after* their jump and that, when opened, they liberate processes that are immobile. The channel ambient $p^{\mathtt{ch}}$ can therefore be typed as immobile, in the same way as the ambient n in Example 3.1 can. Primitives for objective moves are not needed.

Lemma 6.1 *If* $\Gamma \vdash P$, *then* $[\![\Gamma]\!] \vdash \{\![P]\!\}\ : \mathtt{BAmb}\,[\mathtt{Shh}]$.

Encoding of channels, revisited. The previous π-calculus encoding has two drawbacks: (1) it is not purely compositional, for it is is defined in two steps; (2) each channel needs a centralized channel ambient, which handles all I/O requests along that channel. As pointed out in [7], whereas such a centralization point may be acceptable for the π-calculus encoding, it is not as an encoding of communications along channels, for it forces all I/O requests on that channel to be inside the same ambient as the channel ambient. This is in general quite restrictive, because ambients form a hierarchical structure, in which I/O requests may be arbitrarily deep. Table 5 shows an encoding that does not suffer these problems; the solution adopted in [7]—called the coalescing encoding—is behaviourally quite different. Both input and output requests along channel p are represented by ambients called p. The output ambient enters the input one, where it is opened, so that the communication takes place. After that, the continuation of the input exits and is activated.

This encoding, as the previous one, respects typing: if $\Gamma \vdash P$, then

$$[\![\Gamma]\!] \vdash [\![P]\!]\ : \mathtt{BAmb}\,[\mathtt{Shh}]$$

Since the new encoding is compositional and more direct, its operational correctness is (much) simpler to prove.

Theorem 6.2 *1. If* $P \longrightarrow P'$ *then* $[\![P]\!] \Longrightarrow\simeq [\![P']\!]$;

2. If $[\![P]\!] \longrightarrow Q$ *then there is* P' *such that* $P \longrightarrow P'$ *and* $Q \simeq [\![P']\!]$.

The laws of untyped SA are sufficient for proving the behavioural equivalences in the theorem. The theorem can be extended to weak reductions. From this and compositionality we derive adequacy and soundness; the latter says if two target SA processes are behaviourally equivalent, then also the two source π-calculus processes are so. (These results hold both on typed and untyped calculi.)

The clause for input prefix may be simplified as below, where the input ambient p is immediately destroyed after the communication:

$$[\![p(q_1 .. q_n).\,P]\!] \stackrel{\mathrm{def}}{=} p[\,\overline{\mathtt{in}}\,p.\,\mathtt{open}\,p.\,(q_1 .. q_n).\,\overline{\mathtt{open}}\,p.\,[\![P]\!]\,] \mid \mathtt{open}\,p$$

This complicates however the correctness proof (we would need for instance type systems for linear types).

In the encoding of Table 5, coactions guarantee that exactly one output ambient enters an input one and, in the simpler input clause above, they control when the input ambient is opened.

7 Related work and conclusions

The introduction of coactions has been inspired by the SOS semantics of MA [4], in which objects similar to coactions

$$
\begin{aligned}
[\![p(q_1\,..\,q_n).\,P]\!] &\stackrel{\text{def}}{=} (\boldsymbol{\nu} k : \mathtt{BAmb}\,[\mathtt{Shh}])\Big(p[\,\overline{\mathtt{in}}\,p.\,\mathtt{open}\,p.\,(q_1\,..\,q_n)\,(k[\,\mathtt{out}\,p.\,\overline{\mathtt{open}}\,k.\,[\![P]\!]\,]\mid \overline{\mathtt{out}}\,p)\,]\mid \mathtt{open}\,k\Big)\\
[\![\overline{p}\langle q_1\,..\,q_n\rangle]\!] &\stackrel{\text{def}}{=} p[\,\mathtt{in}\,p.\,\overline{\mathtt{open}}\,p.\,\langle q_1\,..\,q_n\rangle\,]\\
[\![(\boldsymbol{\nu} p : T)\,P]\!] &\stackrel{\text{def}}{=} (\boldsymbol{\nu} p : [\![T]\!])\,[\![P]\!]\\
[\![P \mid Q]\!] &\stackrel{\text{def}}{=} [\![P]\!] \mid [\![Q]\!]\\
[\![!P]\!] &\stackrel{\text{def}}{=} ![\![P]\!]\\
[\![\mathbf{0}]\!] &\stackrel{\text{def}}{=} \mathbf{0}\\
[\![Ch(T_1\,..\,T_n)]\!] &\stackrel{\text{def}}{=} \mathtt{STAmb}\uparrow[\uparrow\,[\![T_1]\!]\times..\times[\![T_n]\!]]
\end{aligned}
$$

Table 5: Encoding of π-calculus operators and types, revisited

appear in the labels. Interaction by means of coactions also reminds us of interaction in CCS [11] and π-calculus [12], and distributed extensions of them, especially Seal [20], but also Join, Dπ, etc. [8, 9]. Our type system for immobility in SA is similar to the one for MA [2]. In the latter, however, immobility does not implies persistence; a separate annotation records whether an ambient may be opened or not. If needed, such a distinction can also be added to our system. By contrast, the ST (single-threadness) type system is specific to SA: coactions are necessary to make sense of it. The ST system could be formulated on MA, but all properties it gives us, such as removal of grave interferences, algebraic laws, more precise information on mobility, would be lost. (It might be reasonable, by contrast, to adapt the ST type system to Seal.) Single-threadness is a form of linearity, hence the typing rules borrow ideas from ordinary type systems with linear types (on λ-calculi or elsewhere).

The algebraic laws of Section 5 are inspired by the algebraic theory of process calculi such as the π-calculus. Both in typed SA and in the π-calculus, the laws only require conditions to control plain interferences. As mentioned in the Introduction, and in the discussions of the examples of Section 6, the algebraic theory of MA appears to be rather poor, due to the interference problems. Even the MA analogues of the very simple, untyped, laws of Lemma 5.1 would be hard to formulate. For instance, consider law (4) of the lemma. Removing coactions it becomes $n[\,P \mid m[\,\mathtt{out}\,n.\,Q\,]\,] \simeq m[\,Q\,] \mid n[\,P\,]$. This equality may be broken in MA if: (i) P uses a capability; (ii) an ambient in P tries to enter m; (iii) an ambient in the external environment tries to enter n and then m; (iv) a process in the external environment tries to open n. Furthermore, maintaining these invariances through reduction needs further conditions; for instance if $P = k[\,\mathtt{open}\,h.\,\mathtt{out}\,n.\,\mathtt{open}\,n\,]$, then an ambient h from the external environment could enter n and k, be opened; k could then exit n, be opened, and open n. Types to control mobility in MA [2] may be of limited help: they may not be precise enough, as in the example of encoding of channels; they do not rule out various grave interferences (examples: in the law above, declaring n immobile and not-openable does not prevent (ii) and (iii); immobility is of no help for the interference problems discussed in the renaming and routing protocols of Section 6).

Because of coactions, the movement interactions of SA are synchronous. Hence synchronisation mechanisms between potential interacting partners will be required in distributed implementations. Types, including those in this paper, would certainly help. Other tuning might be appropriate, perhaps along the lines of the Join primitives [8]. Synchronisations are also (implicitly) necessary in MA: for instance in all cases of grave interference, and on interfering opening actions.

We discuss some possible refinements of the ST and immobility type systems. The rules for ST could be relaxed so to allow more than one capability to be exercised, in parallel, at the top level. For instance, an $\mathtt{open}\,n$ capability does not really needs the thread if ambient n does not require the thread when opened. Thus we could type as ST an ambient such as $m[\,\mathtt{in}\,n.\,P_1 \mid \mathtt{open}\,n \mid n[\,P\,]\,]$. For this, we would need a new type to describe capabilities, such as $\mathtt{open}\,n$, that can be used without the thread.

At present we are very drastic about ST ambients opened inside non-ST ambients: the liberated process should bear no thread. We could be more flexible, allowing the liberated process to be of the same "class" as the enclosing ambient. For instance, an ST ambient opened inside an immobile ambient would liberate an immobile process.

The type systems allow rather limited forms of subtyping (Lemmas 3.5 and 3.6): an ST process without thread may be also typed: as ST with thread, as basic, as immobile. Moreover, to obtain this we added the constrain on recursive processes that the recursion variable does not appear free underneath an ambient. Subtyping could be developed further: removing the constrain on recursion; but also allowing other forms of subtyping (for instance, allowing any ST or immobile ambient to be typed also as a basic ambient).

In this paper we have ignored deadlocks. For instance, we can type processes like $n[\,\mathbf{0}\,]$, or $n[\,m[\,P\,]\,]$, or $n[\,\overline{\mathtt{out}}\,m.\,P\,]$, or $n[\,\overline{\mathtt{open}}\,m.\,P\,]$ (assuming P itself types), although they are all deadlocked. We consider deadlock an orthogonal issue. Our objective was not preventing deadlocks but, for instance, proving that a deadlocked process is equivalent to $\mathbf{0}$.

We have mentioned that MA can be encoded into SA. We doubt that the converse is possible, especially for typed SA, but we leave proving or disproving it for future work. Another challenging expressiveness issue is the comparison with Seal [20].

Acknowledgments

We have benefited from discussions with Luca Cardelli, Giorgio Ghelli, and Andy Gordon.

References

[1] G. Boudol and I. Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fundamenta Informaticae*, 11(4):433–452, 1988.

[2] C. Cardelli, G. Ghelli, and A.D. Gordon. Mobility types for mobile ambients. In *Proc. ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 230–239. Springer Verlag, 1999.

[3] C. Cardelli and A.D. Gordon. Mobile ambients. In Nivat. M., editor, *Proc. FoSSaCS '98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer Verlag, 1998.

[4] C. Cardelli and A.D. Gordon. Technical annex to [3]. Unpublished notes, 1998.

[5] C. Cardelli and A.D. Gordon. Equational properties of mobile ambients. In W. Thomas, editor, *Proc. FoSSaCS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 212–226. Springer Verlag, 1999. Short version of [6].

[6] C. Cardelli and A.D. Gordon. Equational properties of mobile ambients. Technical Report MSR-TR-99-11, Microsoft Research, 1999.

[7] C. Cardelli and A.D. Gordon. Types for mobile ambients. In *Proc. 26th POPL*, pages 79–92. ACM Press, 1999.

[8] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile processes. In U. Montanari and V. Sassone, editors, *Proc. CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421. Springer Verlag, 1996.

[9] M. Hennessy and J. Riely. A typed language for distributed mobile processes. In *Proc. 25th POPL*. ACM Press, 1998.

[10] N. Kobayashi, B.C. Pierce, and D.N. Turner. Linearity and the pi-calculus. In *Proc. 23th POPL*. ACM Press, 1996.

[11] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[12] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.

[13] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *19th ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer Verlag, 1992.

[14] H.R. Nielson, F. Nielson, R.R. Hansen, and J.G. Jensen. Validating firewalls in mobile ambients. In J. Baeten and S. Mauw, editors, *Proc. CONCUR '99*, volume 1664 of *Lecture Notes in Computer Science*. Springer Verlag, 1999.

[15] B. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Journal of Mathematical Structures in Computer Science*, 6(5):409–454, 1996. An extended abstract in *Proc. LICS 93*, IEEE Computer Society Press.

[16] C. Priami. *Enhanced Operational Semantics for Concurrency*. PhD thesis, Department of Computer Science, Università di Pisa, 1995.

[17] J.C. Reynolds. Syntactic control of interference. In *Proc. 5th POPL*, pages 39–46. ACM Press, January 1978.

[18] D. Sangiorgi. The name discipline of uniform receptiveness. In *24th ICALP*, volume 1256 of *Lecture Notes in Computer Science*. Springer Verlag, 1997. To appear in TCS.

[19] P. Sewell. Global/local subtyping and capability inference for a distributed π-calculus. In *Proc. 25th ICALP*, volume 1443 of *Lecture Notes in Computer Science*. Springer Verlag, 1998.

[20] J. Vitek and G. Castagna. Seal: A framework for secure mobile computations. In *Proc. Internet Programming Languages*. Springer Verlag, 1999. To appear.