The Seal Calculus

G. Castagna
ENS Paris
France

J. Vitek
Purdue University
USA

F. Zappa Nardelli
INRIA Rocquencourt
France

Abstract

The Seal Calculus is a process language for describing mobile computation. Threads and resources are tree structured; the nodes thereof correspond to agents, the units of mobility. The Calculus extends a π-calculus core with synchronous, objective mobility of agents over channels. This paper systematically compares all previous variants of Seal Calculus. We study their operational behaviour with labelled transition systems and bisimulations; by comparing the resulting algebraic theorieswe highlight the differences between these apparently similar approaches. This leads us to identify the dialect of Seal that is most amenable to operational reasoning and can form the basis of a distributed programming language. We propose type systems for characterising the communications in which an agent can engage. The type systems thus enforce a discipline of agent mobility, since the latter is coded in terms of higher-order communication.

1 Introduction

Process mobility adds a new dimension to distributed computing systems. By endowing program logic with location-awareness and control over the logical and physical location of computation, mobile programming systems aim to enable the development of adaptive and reconfigurable systems that respond to changes in their execution environment in both wide area and ad-hoc networks [Ten96, HKM+98, Whi96, Gra98]. The Seal project is among the first attempts to explore the design space of programming languages for mobile systems. The results presented in this paper summarize our work on formal foundations of mobile languages. This paper refines and extends an earlier version of the calculus [VC99]. Many of the changes were also influenced by our experiences with implementations of Seal and the engineering medium-sized mobile applications [BV01, Bin01].

The original impetus for location-aware and mobile computation grew out of experience with distributed programming languages such as Emerald [JLHB87] and Obliq [Car95], and middleware infrastructures such as Linda [DFP97b, Gel85] that strove to facilitate distributed programming by presenting a, mostly, transparent programming model in which remote resources can be accessed without having worry about their location relative to the current computation. While these systems are well suited to applications distributed over centrally administered local area networks, they are not appropriate for building applications on wide area and ad-hoc networks. The shift away from local area networks involves rethinking some of the assumption underlying the design of these systems. While it is usual for local area networks to be under a single common administrative domain and thus share security policies, this is not the case in wide area networks. Hosts are, in general, spread over different administrative domains with a variety of policies controlling access to local resources. The impact of decentralized control is compounded by issues of scale and connectivity. Local area network are usually small, enjoy good connectivity and low latencies, whereas wide area networks are several orders of magnitude larger with intermittent connectivity and highly variable bandwidth. Finally, communication errors and host failures are common. This degree of heterogeneity can not easily be papered over by a location transparent programming model [WWWK97]. This observation holds for process calculi such as the pi-calculus [MPW92] which do not expose locations.

Finding abstractions suited to programming wide area networks is a challenging problem as one has to balance the need for the kind of simple and clean semantics that allows reasoning about program

behavior with the need for efficient implementations. Our approach is to explore the language design space and, in parallel, investigate the foundations of mobile computation following four design principles:

Location-awareness: Locality is crucial in wide area computing. Observables such as latency, communication errors, and security restrictions, impact the semantics of resource access. Locations, be they logical or physical, thus need to be exposed to distinguish remote interactions from local ones.

Self-reliance: As hosts may experience partial connectivity, and systems may have to scale to large configuration, the semantics should not impose availability of global state and the need for distributed synchronization should be minimized.

Reconfigurability: Wide area and ad-hoc networks are inherently dynamic in their topology as well as resources available to computations. Reconfigurability must be accounted for in the semantics of a distributed language.

Hierarchical access control: While security is an essential constituent of any distributed infrastructure, it is unreasonable to assume a global security policy and enforcement mechanism. Hierarchical access control models which allow different parts of network to define their own policies are better suited to distributed systems.

These guidelines led to the definition of the Seal Calculus. In the Seal Calculus the main concepts of distributed computing are distilled down to three abstractions: *processes*, *resources*, and *locations*.

- Process are sequential threads of control modeled on the π-calculus with terms to denote the inert process, sequential and parallel composition, and replication.
- Physical and logical resources are modeled by *channels*, which are named computational structures used to synchronize processes.
- Locations are denoted by terms called *seals* which stand for physical locations, such as hosts or networks, and logical locations, such as operating system processes or finer grained software components. To play these different roles, seals are arranged in a tree-like hierarchy, each level modeling some real world entity.

As an example, a system composed of a computer running a Java virtual machine on top of an operating system can be represented by three nested seals, one for each entity. Furthermore the address space of the virtual machine can be decomposed into several isolated subcomputations, one per applet, modeled as seals. The sample configuration of Figure 1 depicts a network with two hosts running three applications, one of which is executing with a protection domain. The configuration tree in the same figure is an alternate graphical representation of the seal term.

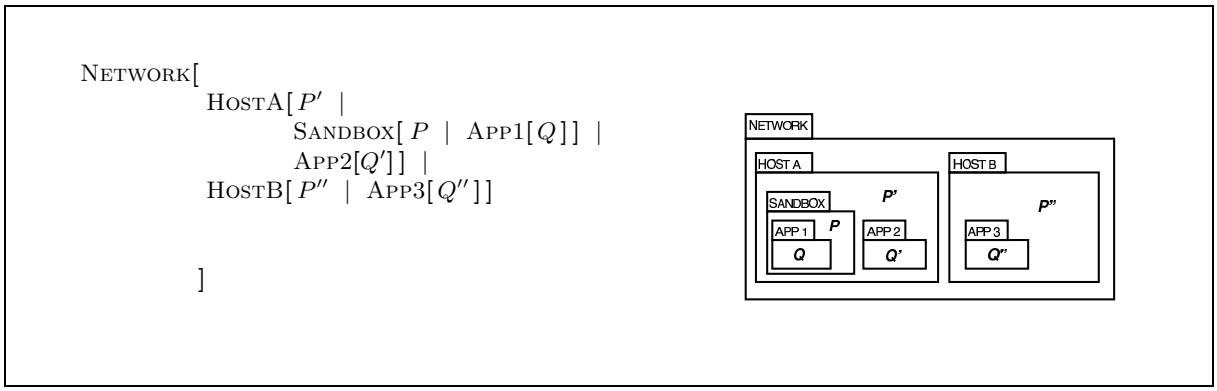

Figure 1: Seal Calculus term and configuration tree.

Seals differ from *ambients* [CG98] in three important ways. First, the association between names and seals is weak. Names are merely tags used by parent seals to tell their children apart. They can be changed at the parent's whim. Contrast this to the Ambient Calculus in which names are capabilities used for access control. Second, unlike ambients, the boundary around seals cannot be dissolved. Thus a process contained in a seal can be deactivated by blocking every possible interaction it can have with the surrounding context, it can also be destroyed or passed along, but it can never be separated from its boundary.[1] Finally, while the Ambient Calculus uses *subjective* mobility (an agent moves itself) in the Seal Calculus mobility is *objective* (an agent is moved by its context) as this makes the enforcement of security requirements easier.

The Seal Calculus has been designed so as to allow seals to confine subseals and control their access to resources. Several features of Seal have been specifically chosen with this goal in mind. Seal boundaries give us a way to delineate groups of processes. This makes it possible, for instance, to assign trust levels to each group. The choice to disallow boundary dissolution greatly simplifies reasoning as we do not have to worry about arbitrary processes set loose in the local environment. The main resource access control mechanism of the Seal Calculus is complete mediation. Since interaction is limited to direct parent-child, any remote interaction between seals that are not in direct contact is mediated by all the intervening seals. When coupled with the hierarchical organization of seals this gives rise to programming styles that emphasize interposition techniques [PSB95]. For instance a seal can set up a monitor process which through careful use of restrictions can intercept all communications of a child seal and filter or route outgoing messages. An important precondition to mediation is the ability to choose subseal names. When a parent performs the receive action that allows a new seal to move in, it chooses the name under which the seal will operate.

We opted for a symmetric interaction model. Remote interaction requires an explicit agreement between two partners. This should be contrasted with Mobile Ambients where actions are asymmetric as one of the two partners simply undergoes move actions. Symmetry implies that either parent or child may prevent the other from accessing its local resources. Mobility actions are objective in the sense that seals are moved by their parent seal. Again, we contrast this with the subjective mobility of Ambients where moves are initiated by the ambient that is the subject of the move. Migration is thus totally under the control of a seal's environment which decides whether and when it occurs. For example, Trojan Horses can be detected in Seal, while in a model with subjective mobility complex coarse-grained analysis techniques are needed (e.g. see [BC01] and [NNHJ99]).

Overview This article is intended to be a reference publication for the Seal Calculus. Several process languages are today know as Seal Calculus: our investigation aims to tidy up the picture offered by these dialects, as well as to develop solid foundations for Seal. In Section 2 we give a uniform presentation of the syntax and of the reduction semantics of all of the dialects. We also summarize the evolution of the Seal Calculus and we examine the influence that the research on Seal had on other process calculi. Section 3 opens with a preliminary investigation of behavioural theories of the dialects that highlights the deep differences between these process calculi. We concentrate on the dialect that appears to be, at the same time, a suitable kernel for a programming language and a process calculus amenable to theoretical investigation. We propose a labelled transition system, its definition points out some of the particularities of the Seal model of computation. On top of the LTS we build a labelled bisimilarity and we prove that it is a sound proof technique for reduction barbed congruence. The labelled bisimilarity is subsequently used to prove some algebraic laws. In Section 4 a type system aimed at understanding what typing mobility means is presented. Related works are discussed throughout the article.

Besides their technical importance, the results on equivalences reflect a common theme in mobile calculi. Our results demonstrate that properties that required extensions in the original Ambient calculus (such as the results of [LS00] and [MH02]) hold for Seal as it is. *A posteriori* this is not surprising as expressive equivalence theories require tight control on interaction, exactly as security—one of the main Seal's design guidelines—does. Therefore this confirms the pertinence of our design choices. Finally it is noteworthy that the work presented here is the only one we are aware of that accounts for process

[1] This feature is the one that is at the origin of the name of "Seal", as we thought it as a calculus of "sealed objects".

duplication. Although Sangiorgi's research on higher-order π-calculus [San92, San96] accounts for (inactive) process duplication, mobility frameworks usually restrict their analyses to the far simpler case where duplication is not allowed.

The version of Seal Calculus presented here is based on [VC99] and on the second author's PhD Thesis [Vit99] from which it inherits syntax and most design choices. This calculus is essentially the one defined in [CZ02], to which the introduction of the equivalence theory of Section 3 is also due. The development of the equivalence theory is taken from Chapter 3 of third author's PhD Thesis [Zap03]. The type system of Section 4, although based on the one in [CGZ01], is essentially original to this work as it uses new forms of judgments and a much more precise typing technique that types agent generators that are parametric on channel names.

2 The Seal Calculus

The Seal Calculus was introduced in [VC99] as formal counterpart to a kernel for programming mobile agents [BV01] on top of the Java Virtual Machine. Since then, the name Seal Calculus has been used to denote a family of process calculi that originate from the same design choices and provide the programmer with the same set of abstractions. At the same time, the little differences that characterise their reduction semantics have a big impact on the programming pragmatics. We introduce these different dialects, and we define their reduction semantics in a uniform framework. All along the paper, the different developments will highlight the differences between these apparently similar dialects, and will point out a dialect which is both amenable of theoretical investigation and reasonable as core of a programming language.

Located and Shared Seal A distinctive feature of the Seal model of computation is that process interactions are restricted to channels that are 'close enough' in the seal hierarchy. We distinguish two interpretations of this notion of proximity, called respectively *located channels* and *shared channels*. In the first case, channels are associated to seals and channel denotations specify the seal in which a channel is located. In the second interpretation, channels are shared uniquely between communicating entities, so that channel denotations specify the partner the channel is shared with. Figure 2.a illustrates a configuration in the located interpretation. Channels x and y are localised in two different seals. Synchronisation between these seals can happen over both channels: to synchronise over x, process P must specify that it refers to a local channel, while Q will specify access to a channel located in its direct parent. Thus, channel x will be referred as x^* by process P, and as channel $x^\uparrow$ by Q, while channel y is referred as y^b by P, and as y^* by Q. Figure 2.b depicts a similar configuration in the shared interpretation. In this interpretation only the channel x can be used to synchronise processes P and Q, and it will be referred to as x^b by P and as $x^\uparrow$ by Q.

a. Localized channels b. Shared channels

Figure 2: Channels in the Seal Calculus.

Mobility and Extrusion of Names Scoping is lexical in the Seal Calculus and, as in π-calculus, the scope of a name can be enlarged after an interaction. Although scope extrusion is fundamental to expressivity, it desirable to limit it in some cases. In particular there are reasons to prevent extrusion when it entails a name escaping the enclosing seal's boundary as a consequence of a mobility interaction.

Intuitively, this is a security risk as the name may represent a password or capability. We refer to this constraint as the *e*-condition.

Seal Dialects As localisation of channels and the *e*-condition are orthogonal features, they define four dialects of the Seal Calculus. All of them have been investigated in previous work, and should not be confused as they have rather different properties:

- *L-Seal*: located channels; (almost) the original Seal Calculus, as presented in [VC99];
- *eL-Seal*: located channels and *e*-condition; introduced in [CZ02];
- *S-Seal*: shared channels; appears in [CGZ01];
- *eS-Seal*: shared channels and *e*-condition; introduced in [CZ02].

One of the aims of this paper is to precisely characterize the differences between these calculi, and to develop solid foundations for the Seal Calculus.

2.1 Syntax and reduction semantics

The syntax of the Seal Calculus is defined in Table 1, where $\mathbf{N}$ denotes an infinite set of names. Following the π-calculus, the inert process is denoted by $\mathbf{0}$. A process $\alpha.P$ is composed of an action α and a continuation P; it denotes a process waiting to perform α and then behave as P. The term $P \mid Q$ denotes a process composed of two subprocesses, P and Q running in parallel. A seal is represented by the term $a[P]$, where the process P is running at a. A seal is itself a process, it can thus be nested within another seal; so if P contains one or more seals, say $m_1 \dots m_n$, then n is called the *direct parent* of each m_i. The term $!\alpha.P$ creates an unbound number of copies of $\alpha.P$ running in parallel. Guarded replication was chosen because it yields a simpler LTS (Section 3.3) without loss of generality. In the π-calculus replicated input has the same expressive power as full replication [HY95a] and recursion [Mil91, SW02].

Names:	$a, \dots, u, v, x, y, z, \cdots \in \mathbf{N}$		
Locations:			
η	::=	$*$	local
	$\mid$	$\uparrow$	up
	$\mid$	z	down
Actions:			
α	::=	$\overline{x}^{\eta}(y_1, \cdots, y_n)$	output
	$\mid$	$x^{\eta}(y_1, \cdots, y_n)$	input
	$\mid$	$\overline{x}^{\eta}\{y\}$	send
	$\mid$	$x^{\eta}\{y_1, \cdots, y_n\}$	receive
for $n \geq 0$.			
Processes:			
P, Q, R	::=	$\mathbf{0}$	inactivity
	$\mid$	$P \mid Q$	parallel composition
	$\mid$	$(\boldsymbol{\nu} x)P$	restriction
	$\mid$	$\alpha.P$	prefixing
	$\mid$	$n[P]$	seal
	$\mid$	$!\alpha.P$	replication

Table 1: The Syntax of the Seal Calculus.

We work modulo α-conversion and thus assume all bound variables in the same process to be distinct. Some useful notations: we write $\vec{x}_n$ (or simply $\vec{x}$) to denote the tuple $x_1, \cdots, x_n$. We also write $(\boldsymbol{\nu}\vec{x}_n)P$

(or $(\boldsymbol{\nu}\vec{x})P$), for $(\boldsymbol{\nu}x_1)\ldots(\boldsymbol{\nu}x_n)P$. We omit $*$ locations and trailing $\mathbf{0}$ processes. In the input action $x^\eta(y_1,\cdots,y_n)$ the y_i's are required to be pairwise distinct (however this is not required in the receiving action, see below). We use $P[y/x]$ to denote the process obtained from P by substituting y for all free occurrences of x, and use $P[\vec{y}_n/\vec{x}_n]$ to denote the process obtained from P by simultaneous substitution of y_i for x_i. The latter is not defined for vectors of different arity.

The free names of actions and processes are defined as:

$$\begin{array}{rclcrcl}
\mathrm{fn}(\mathbf{0}) & = & \varnothing & & \mathrm{fn}(\uparrow) & = & \varnothing \\
\mathrm{fn}(*) & = & \varnothing & & \mathrm{fn}(x) & = & \{x\} \\
\mathrm{fn}(P \mid Q) & = & \mathrm{fn}(P) \cup \mathrm{fn}(Q) & & \mathrm{fn}(!\alpha.P) & = & \mathrm{fn}(\alpha.P) \\
\mathrm{fn}(x[P]) & = & \mathrm{fn}(P) \cup \{x\} & & \mathrm{fn}((\boldsymbol{\nu}x)P) & = & \mathrm{fn}(P) \setminus \{x\} \\
\mathrm{fn}(x^\eta(\vec{y}).P) & = & (\mathrm{fn}(P) \setminus \vec{y}) \cup \{x\} \cup \mathrm{fn}(\eta) & & \mathrm{fn}(\overline{x}^\eta(\vec{y}).P) & = & \mathrm{fn}(P) \cup \vec{y} \cup \{x\} \cup \mathrm{fn}(\eta) \\
\mathrm{fn}(\overline{x}^\eta\{y\}.P) & = & \mathrm{fn}(P) \cup \{y\} \cup \{x\} \cup \mathrm{fn}(\eta) & & \mathrm{fn}(x^\eta\{\vec{y}\}.P) & = & \mathrm{fn}(P) \cup \vec{y} \cup \{x\} \cup \mathrm{fn}(\eta)
\end{array}$$

Contrary to what happens with the input action, the receive action is not a binding operation. As a consequence, the names y_i are not required to be pairwise distinct. As we will see shortly, a name specified in the receive action denotes the name of a seal that will run in parallel with the receiving process. It seems an undue restriction to limit the scope of the names of incoming seals to the sole continuation of the receive action. If a similar behaviour is needed, it can be easily programmed.

Actions Every interaction takes place over named, localised channels. The Seal Calculus differentiates between local and remote interaction. Interaction is local if the processes that synchronise are located in the same seal. Interaction is remote if the processes are located in seals in parent-child relationship: depending on the interpretation, they synchronise over a shared channel or a channel located in one of the host seals. Processes that are not in seals in parent-child relationship can not communicate, thus any interaction that spans more than a single seal boundary has to be encoded explicitly. Channel synchronisation is used both for communication (exchange of names) and for mobility (exchange of seals).

***Communication*:** $\overline{x}^\eta(\vec{y}).P$ denotes a process ready to output $\vec{y}$ on channel x^η and then continue as P, and $x^\eta(\vec{z}).Q$ denotes a process that will read from channel x^η and then continue as Q where the free occurrences of $\vec{z}_i$ are replaced by the corresponding messages.

As an example, in the configuration of Figure 2.a, the reduction below illustrates the exchange of the name w between P and Q over the channel y located inside the seal b.

$$a[\ \underbrace{\overline{y}^b(w).R}_{P} \mid b[\ \underbrace{y^*(z).S}_{Q}\ ]\ ] \quad \rightarrowtriangle \quad a[R \mid b[S[w/z]]]$$

Referring to Figure 2.b, a similar interaction carried on in the shared interpretation looks like:

$$a[\ \underbrace{x^b(z).R}_{P} \mid b[\ \underbrace{\overline{x}^\uparrow(w).S}_{Q}\ ]\ ] \quad \rightarrowtriangle \quad a[R[w/z] \mid b[S]]$$

Remark that the (sum and matching free) polyadic π-calculus is a sub-calculus of the Seal Calculus as it can be obtained by forbidding the use of seal processes, of send and receive actions, and of up and down locations.

***Mobility*:** $\overline{x}^\eta\{y\}.P$ denotes a process ready to send a child seal named y along channel x^η; $x^\eta\{\vec{z}_n\}.P$ denotes a process that will receive a seal along channel x^η and reactivate n copies of it inside newly created seals called respectively z_i.

The nesting of seals is represented as a tree, and mobility corresponds to a tree rewriting operation, as shown in Figure 3. A move disconnects a subtree rooted at some seal y and grafts it either onto the parent of y (configuration (b)), onto one of the children of y (configuration (c)), or back onto y itself (configuration (d)). The rewriting operation relabels the edge associated to the moved seal, and can also create a finite number of copies of the subtree rooted at the moved seal. These tree transformations correspond to mobility reduction rules of the calculus.

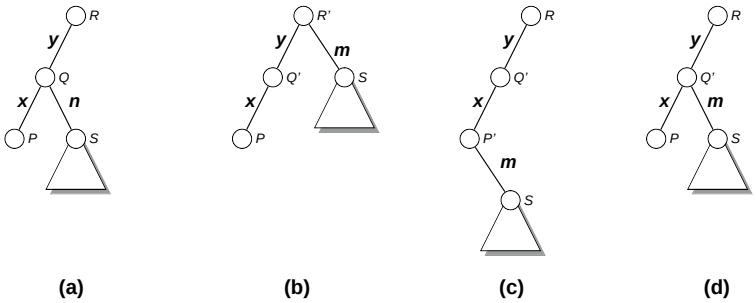

Figure 3: Mobility as restructuring the tree of locations.

So for example the transformation from (a) to (b) using a channel c located in y corresponds to the following reduction

$$\underbrace{c^{y}\{m\}.R'}_{R} \mid y[\ \underbrace{\overline{c}^{*}\{n\}}_{Q} \mid x[P] \mid n[S]] \qquad \rightarrow \qquad R' \mid m[S] \mid y[x[P]]\ .$$

Seal Duplication A straightforward application of the semantics of mobility is the implementation of a copy operation:

$$(\mathsf{copy}\, x\, \mathsf{as}\, z).P \quad \stackrel{def}{=} \quad (\boldsymbol{\nu}\, y)\,(\,\overline{y}^{*}\{x\} \mid y^{*}\{x\,, z\}.P\,)$$

The term $\mathsf{copy}\, n\, \mathsf{as}\, m \mid n[P]$ reduces to $n[P] \mid m[P]$. The copy process creates a brand new channel name y to prevent other processes from interfering with the protocol. Then, the left hand side subprocess attempts to move n over the local channel, while the right hand side process receives it and instantiates two copies of it under names n and m. The same technique can be used to destroy a seal:

$$(\mathsf{destroy}\, x\,).P \quad \stackrel{def}{=} \quad (\boldsymbol{\nu}\, y)\,(\,\overline{y}^{*}\{x\} \mid y^{*}\{\}.P\,)$$

Here the receive action instantiates zero copies of the sent seal, thus destroying it. Duplication differs from replication, $!\alpha.P$, because replication creates copies of inert processes while duplication copies running processes. This is similar to higher-order π-calculi [San92, San96] in which duplication is also restricted to inactive processes.

Synchronisation We define the reduction semantics of the calculus in chemical style, using a structural congruence relation and a set of reduction rules. The semantics is parametric on the interpretation of channels. For that, we introduce two predicates

$$\mathsf{synch}^{S}, \mathsf{synch}^{L} : \mathbf{Var} \times \mathbf{Loc} \times \mathbf{Loc} \rightarrow \mathbf{Bool}$$

ranged over by synch. Intuitively, $\mathsf{synch}_{y}(\eta_1, \eta_2)$ holds if and only if for any channel x an action on x^{η_1} performed in some parent seal may synchronise with a coaction on x^{η_2} performed in a child seal named y (in this case we say that η_1 and η_2 are 'y-corresponding' locations).

Definition 2.1 (y-correspondence) *Let η_1, η_2 be locations and y a name. We define:*

Shared channels: $\mathsf{synch}^{S}_{y}(\eta_1, \eta_2) \stackrel{def}{=} (\eta_1 = y \wedge \eta_2 = \uparrow)$*;*

Located channels: $\mathsf{synch}^{L}_{y}(\eta_1, \eta_2) \stackrel{def}{=} (\eta_1 = y \wedge \eta_2 = *) \vee (\eta_1 = * \wedge \eta_2 = \uparrow)$.

Definition 2.2 (Contexts) *A* context *is a term containing a hole, denoted $-$, generated by the following grammar in which P is an arbitrary process, α an arbitrary action, and n an arbitrary name:*

$$C[-] \quad ::= \quad - \quad \mid \quad C[-] \mid P \quad \mid \quad P \mid C[-] \quad \mid \quad n[C[-]] \quad \mid \quad \alpha.C[-] \quad \mid \quad (\boldsymbol{\nu} n)C[-]$$

A static context *is a context where the hole does not appear under prefix.*

Definition 2.3 (Structural Congruence) *Structural congruence is the smallest relation over processes that is preserved by static contexts and that satisfies the following axioms:*

$$\begin{array}{rcl@{\qquad}rcll} P \mid \mathbf{0} & \equiv & P & (\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P & \equiv & (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P & x \neq y \\ P \mid Q & \equiv & Q \mid P & (\boldsymbol{\nu} x)(P \mid Q) & \equiv & P \mid (\boldsymbol{\nu} x)Q & x \notin \mathrm{fn}(P) \\ P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & (\boldsymbol{\nu} x)\mathbf{0} & \equiv & \mathbf{0} & \\ !\alpha.P & \equiv & \alpha.P \mid !\alpha.P & & & & \end{array}$$

Seal duplication has an interesting consequence on the semantics of the Seal Calculus: the structural rule

$$(\boldsymbol{\nu} x)y[P] \quad \equiv \quad y[(\boldsymbol{\nu} x)P] \qquad x \neq y \tag{1}$$

found in the semantics of Mobile Ambients is not sound as processes $(\boldsymbol{\nu} x)y[P]$ and $y[(\boldsymbol{\nu} x)P]$ are not equivalent. If we compose the terms with the process $Q = \mathsf{copy}\ y\ \mathsf{as}\ z$ we obtain

$$y[(\boldsymbol{\nu} x)P] \mid Q \ \twoheadrightarrow\ y[(\boldsymbol{\nu} x)P] \mid z[(\boldsymbol{\nu} x)P] \quad \text{and} \quad (\boldsymbol{\nu} x)y[P] \mid Q \ \twoheadrightarrow\ (\boldsymbol{\nu} x)(y[P] \mid z[P]) \quad .$$

The first process yields a configuration where seals y and z have each a private channel x, while the second process reduces to a configuration where y and z share a common channel x. This unsoundness result is proved in Section 3.2.

Reduction Since we work modulo α-conversion we can suppose that in the term $(\boldsymbol{\nu}\vec{x})P$ the names $x_1, \ldots, x_n$ are all distinct. Structural congruence states that their order is not important as they can be freely permuted. This justifies the use of set-theoretic operations such as $(\boldsymbol{\nu}\vec{x} \cap \vec{y})P$ or $(\boldsymbol{\nu}\vec{x} \setminus \vec{y})P$, with the convention that $(\boldsymbol{\nu}\varnothing)P = P$.

Definition 2.4 (Reduction Relation) *The reduction relation, $\twoheadrightarrow$ is the smallest relation between processes that satisfies the rules of Table 2 and is preserved by static contexts. Its reflexive and transitive closure is denoted $\twoheadrightarrow^*$.*

As structural congruence cannot move restrictions across seal boundaries, the reduction rules (write out) and (move out) explicitly extrude restrictions through seal boundaries. The non-local rules are parametric in synch: different remote interaction patterns are obtained according to whether synch is replaced by synch^S (shared channels), or synch^L (located channels).

(write local)	$x^*(\vec{u}).P \mid \overline{x}^*(\vec{v}).Q$	$\twoheadrightarrow$	$P[\vec{v}/\vec{u}] \mid Q$
(write in)	$\overline{x}^{\eta_1}(\vec{w}).P \mid y[(\boldsymbol{\nu}\vec{z})(x^{\eta_2}(\vec{u}).Q_1 \mid Q_2)]$	$\twoheadrightarrow$	$P \mid y[(\boldsymbol{\nu}\vec{z})(Q_1[\vec{w}/\vec{u}] \mid Q_2)]$
(write out)	$x^{\eta_1}(\vec{u}).P \mid y[(\boldsymbol{\nu}\vec{z})(\overline{x}^{\eta_2}(\vec{v}).Q_1 \mid Q_2)]$	$\twoheadrightarrow$	$(\boldsymbol{\nu}\vec{v} \cap \vec{z})(P[\vec{v}/\vec{u}] \mid y[(\boldsymbol{\nu}\vec{z} \setminus \vec{v})(Q_1 \mid Q_2)])$
(move local)	$x^*\{\vec{u}\}.P_1 \mid \overline{x}^*\{v\}.P_2 \mid v[Q]$	$\twoheadrightarrow$	$P_1 \mid u_1[Q] \mid \cdots \mid u_n[Q] \mid P_2$
(move in)	$\overline{x}^{\eta_1}\{v\}.P \mid v[S] \mid y[(\boldsymbol{\nu}\vec{z})(x^{\eta_2}\{\vec{u}\}.Q_1 \mid Q_2)]$	$\twoheadrightarrow$	$P \mid y[(\boldsymbol{\nu}\vec{z})(Q_1 \mid Q_2 \mid u_1[S] \mid \cdots \mid u_n[S])]$
(move out)	$x^{\eta_1}\{\vec{u}\}.P \mid y[(\boldsymbol{\nu}\vec{z})(\overline{x}^{\eta_2}\{v\}.Q_1 \mid v[R] \mid Q_2)]$ $\twoheadrightarrow$		$P \mid (\boldsymbol{\nu}\,\mathrm{fn}(R) \cap \vec{z})(u_1[R] \mid \cdots \mid u_n[R] \mid y[(\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(R))(Q_1 \mid Q_2)])$
(red struct)	$P \equiv P' \ \wedge\ P' \twoheadrightarrow Q' \ \wedge\ Q' \equiv Q$ implies $P \twoheadrightarrow Q$		

Table 2: Reduction rules for the Seal calculus. With $x \notin \vec{z}$, $\vec{w} \cap \vec{z} = \emptyset$, $\mathrm{fn}(S) \cap \vec{z} = \emptyset$ and $\mathsf{synch}_y(\eta_1, \eta_2)$.

The first three rules define the semantics of communications. Rule (write local) gives local communication the same semantics as found in polyadic π-calculus.

Rule (write in) describes the communication of a tuple $\vec{w}$ from a parent to its child y. Reduction takes place provided that channel x is not locally restricted (i.e., $x \notin \vec{z}$), and no name capture arises. (i.e.,

$\vec{w} \cap \vec{z} = \emptyset$). Rule (write out) captures the case when a child y communicates a vector of names $\vec{v}$ to its parent. Names local to y in $\vec{v}$ may be extruded across the boundaries of y. The three remaining rules define the semantics of mobility. In local mobility (rule (move local)) the body of the seal specified by the send action is copied n times, named as specified by the receive action. A seal can be moved inside a child y (rule (move in)) provided that no variable free in the moved process is captured (i.e., $\mathrm{fn}(R) \cap \vec{z} = \varnothing$). A seal can be moved out of its enclosing seal (rule (move out)). Names local to y but free in R may be extruded across the boundaries of y. All non-local rules require that η_1 and η_2 are y-corresponding locations and that channel x is not locally restricted (i.e., $x \notin \vec{z}$).

The *e*-condition There is a tension whether to allow extrusion of names as a consequence of a mobility action. Names private to a seal can be considered its private resources, and it is legitimate to prevent these names to be extruded to the enclosing seal as a consequence of a mobility action. In a distributed implementation locally restricted channels would correspond to local variables (e.g. pointers). Moving these channels outside the enclosing location then requires to implicitly transform them into globally unique identifiers. If the restrictive option is chosen, then all variables free in a seal must already be known by the parent either because they are non-local or because they have been previously communicated. In the example below, the output action extrudes the name y, so that there is no harm in moving the seal m out of n

$$n[(\boldsymbol{\nu} y)(\overline{u}^{\uparrow}(y).\overline{x}^{\uparrow}\{m\} \mid m[\overline{y}^{\uparrow}()])] \ .$$

Extrusion of names across seal boundaries as a consequence of mobility can be forbidden by adding the condition

$$\mathrm{fn}(R) \cap \vec{z} = \emptyset \tag{2}$$

to rule (move out). We refer to the equation (2) as the *e-condition*. In this case, the (move out) rule can be simplified, and rewritten as:

$$x^{\eta_1}\{\vec{u}\}.P \mid y[(\boldsymbol{\nu}\vec{z})(\overline{x}^{\eta_2}\{v\}.Q_1 \mid v[R] \mid Q_2)] \ \rightarrowtriangle \ P \mid u_1[R] \mid \cdots \mid u_n[R] \mid y[(\boldsymbol{\nu}\vec{z})(Q_1 \mid Q_2)]$$

where $\mathrm{fn}(R) \cap \vec{z} = \varnothing$, $x \notin \vec{z}$, and $\mathsf{synch}_y(\eta_1, \eta_2)$ holds.

Adding the *e*-condition to the semantics of the Seal Calculus has surprising effects on its behavioural theory, as we will see shortly.

3 Behavioural theories

The study of the behavioural theories of the dialects highlights the deep differences between these process calculi, only apparently similar. We then concentrate on the dialect that appears at the same time a suitable kernel for a programming language and a process calculus amenable of theoretical investigation.

3.1 Reduction barbed congruence

We introduce an equivalence constructed from natural properties, that will be our reference notion of equivalence. We focus on a slight variant of Milner and Sangiorgi's barbed congruence[MS92], called *reduction barbed congruence*. This relation was first studied by Honda and Yoshida under the name of *maximum sound theory* [HY95b].

Definition 3.1 (Reduction closed) *A relation $\mathcal{R}$ over processes is* reduction closed *if $P \ \mathcal{R} \ Q$ and $P \rightarrowtriangle P'$ implies the existence of some Q' such that $Q \rightarrowtriangle^* Q'$ and $P' \ \mathcal{R} \ Q'$.*

Definition 3.2 (Contextual) *A relation $\mathcal{R}$ over processes is* contextual *if $P \ \mathcal{R} \ Q$ implies $C[P] \ \mathcal{R} \ C[Q]$ for all contexts $C[-]$.*

We build our analysis on the assumption that the basic observable entity in Seal is the presence of a seal whose name is public at top-level. This observation, originally due to Cardelli and Gordon [CG98, CG99a], can be interpreted as the ability of the top-level process to interact with that seal. In Section 3.5.1 we will see that the reduction barbed congruence is insensitive to the exact observation chosen.

Definition 3.3 (Barbs) *We write $P \downarrow n$ if and only if there exist Q, R, $\vec{x}$ such that $P \equiv (\boldsymbol{\nu}\vec{x})(n[Q] \mid R)$ where $n \notin \vec{x}$. We write $P \Downarrow n$ if there exists P' such that $P \rightarrow^* P'$ and $P' \downarrow n$.*

Definition 3.4 (Barb preserving) *We say that a relation $\mathcal{R}$ over processes is* barb preserving *if $P \,\mathcal{R}\, Q$ and $P \downarrow n$ implies $Q \Downarrow n$.*

Definition 3.5 (Reduction barbed congruence) *Reduction barbed congruence, written $\cong$, is the largest symmetric relation over terms which is reduction closed, contextual, and barb preserving.*

In the sequel we will use the following properties of reduction barbed congruence.

Lemma 3.6 *If $P \cong Q$ then*

1. *$P \Downarrow n$ if and only if $Q \Downarrow n$;*
2. *$P \rightarrow^* P'$ implies that there exists a process Q' such that $Q \rightarrow^* Q'$ and $P' \cong Q'$.*

3.2 Seal dialects and reduction barbed congruence

To gain familiarity with the characteristics of reduction barbed congruence, we show that in all Seal dialects the process $P = (\boldsymbol{\nu}x)n[R]$ is not equivalent to $Q = n[(\boldsymbol{\nu}x)R]$. In this way we prove the unsoundness of rule (1) informally discussed in the previous section. Take

$$R \;=\; \overline{y}^{\eta}(x) \mid x^{\eta}()$$

and consider the context

$$C[-] \;=\; \mathsf{copy}\, n \,\mathsf{as}\, m.y^{n}(u).\overline{u}^{m}().b[\mathbf{0}] \mid [-]$$

in which b is fresh, and η is such that $\mathsf{synch}_u(n, \eta)$ holds in the desired channel interpretation (this also implies $\mathsf{synch}_u(m, \eta)$). Then $C[P] \rightarrow^* P'$ and $P' \downarrow b$ (in particular $P' = (\boldsymbol{\nu}x)n[x^{\eta}()] \mid m[\overline{y}^{\eta}(x)] \mid b[\mathbf{0}]$) while there is no Q' such that $C[Q] \rightarrow^* Q'$ and $Q' \Downarrow b$.

Although it may appear that the four dialects of Seal define essentially the same language, deep differences are exposed by the study of their behavioural theory. To avoid confusion between the different semantics, we label $\cong$ with the name of the calculus we are considering. So, $\cong_{eS}$ stands for reduction barbed congruence over Seal with shared channels and the *e*-condition; similarly for the other dialects.

Observing free names Perhaps the biggest surprise of our study: Seal contexts can observe the free names of terms.

It is not difficult to prove this claim in a dialect that includes the *e*-condition. Let P and Q be two processes, and let x be a name such that $x \in \mathrm{fn}(P)$ but $x \notin \mathrm{fn}(Q)$. Independently of their behaviour, the context

$$C[-] \;=\; z^{y}\{n\} \mid y[(\boldsymbol{\nu}x)(\overline{z}^{\uparrow}\{n\} \mid n[\,-\,])]$$

exploits the name x to tell P and Q apart. In fact, while the move of $n[Q]$ out of y can freely occur, the presence of x free in P forbids the move of $n[P]$. The same example applies to *eL*-Seal.

If the dialect does not include the *e*-condition, then we need a fairly complex context to observe the free names of a process.[2] Consider the context

$$C[-] \;=\; m[(\boldsymbol{\nu}x)(a[\,-\,] \mid \overline{w}^{\uparrow}\{a\}.R)] \mid w^{m}\{\}.y^{*}\{m_1, m_2\} \mid \overline{y}^{*}\{m\}$$

and, as before, let P and Q be two processes such that $x \in \mathrm{fn}(P)$ and $x \notin \mathrm{fn}(Q)$. Now,

$$C[P] \rightarrow^* (\boldsymbol{\nu}x)(m_1[R] \mid m_2[R] \mid a[P])$$

[2] We are grateful to Thomas Hildebrandt and Mikkel Bundgaard for pointing us to this example.

while

$$C[Q] \rightarrowtriangle^* m_1[(\boldsymbol{\nu} x)R] \mid m_2[(\boldsymbol{\nu} x)R] \mid a[Q] \ .$$

In general, the two subterms $(\boldsymbol{\nu} x)(m_1[R] \mid m_2[R])$ and $m_1[(\boldsymbol{\nu} x)R] \mid m_2[(\boldsymbol{\nu} x)R]$ are not equivalent, as shown above.

In the Seal Calculus, contextuality is source of a great discriminating power, difficult to justify purely in terms of *behaviour*. In fact two processes are not equivalent if they contain different free names even in deadlocked subprocesses.

As an aside remark, this is similar to what happens to languages that offer reflection: contextual equivalence becomes dependent not only on the runtime behaviour of a program, but also on the raw code of the program as it is possible to examine the programs definitions at runtime. Whether this is desirable is open to debate, but it is clearly a feature that deserves more attention.

***L*-Seal** Extrusion is subtle in the Located Seal dialect, as demonstrated by the following reduction:

$$x^*(u).P \mid (\boldsymbol{\nu} z)z[\overline{x}^{\uparrow}(v).Q] \ \rightarrowtriangle \ (\boldsymbol{\nu} z)(P[v/u] \mid z[Q]) \ .$$

In L-Seal, a process can synchronise with a child seal without knowing that seal's name. This characteristic of the extrusion rules significantly complicates the task of controlling interference. For instance, $(\boldsymbol{\nu} cxy)\overline{c}\{x\} \mid c\{y\} \mid x[P]$ is *not* equivalent to $(\boldsymbol{\nu} cxy)y[P]$. If P contains an action, e.g. $\overline{z}^{\uparrow}(c)$, that exposes c to its environment, interference will be possible. The consequence is a poor algebraic theory and a programming model that is error prone. For instance, $(\boldsymbol{\nu} x)x[P]$ is in general not equivalent to $\mathbf{0}$ while such an equivalence holds in S-Seal as shown in Section 3.7.

***S*-Seal** The Seal Calculus with shared channels and without the e-condition appears to be the most amenable of theoretical investigation. Our intuitions about its semantics have not been contradicted by basic observations on its behavioural theory. Also it seems to constitute the core of a sensible programming language. For these reasons, in the rest of this section we will focus exclusively on S-Seal, and we will abbreviate its name to Seal. In particular, we will look for a bisimulation based proof method, to learn more about its behaviour, and further investigate its algebraic theory.

3.3 A labelled transition semantics

In Table 4 we report a labelled transition system for Seal. Our labelled transition system uses actions of the form

$$A \vdash P \xrightarrow{\ell} P'$$

where A is a finite set of names such that $\mathrm{fn}(P) \subseteq A$; the judgement should be read as 'in a state where names in A may be known by process P and by its environment, the process P can perform ℓ and become P''. This presentation, borrowed from [SV99] and [Sew00], allows us to drop many side conditions when dealing with the extrusion of names.

In the name environment A we use commas to denote disjoint union, that is, A, y means $A \dot\cup \{y\}$ and $A, \vec{y}$ means $A \dot\cup \vec{y}$. We write $P \xrightarrow{\ell}$ as a shorthand for 'there exist a set of names A and a process Q such that $A \vdash P \xrightarrow{\ell} Q$'. We write $\Rightarrow$ to denote the reflexive and transitive closure of $\xrightarrow{\tau}$. We denote the composition of the relations $\mathcal{R}$ and $\mathcal{S}$ as $\mathcal{R}\mathcal{S}$; this notation is also extended to transitions.

Labels The ℓ labels, defined in Table 3, give a precise description of how the process P evolves, either autonomously, or by interacting with a possible context. We define an *early* semantics, that is, a semantics where receiving actions are instantiated with their actual arguments when the capability is unleashed, rather than when interaction occurs. This avoids dealing explicitly with process substitutions, and also simplifies the definition of the labelled bisimilarity. This is not surprising, as our bisimulation will be a *context bisimulation* in the sense of [San96] and, in general, a transition in an early LTS clearly identifies the context the process is interacting with.

The free names of a label, $\mathrm{fn}(\ell)$, are defined by the rules below:

$$\mathrm{fn}(\tau) = \emptyset \qquad \mathrm{fn}(P_z) = \mathrm{fn}(P^z) = \{z\} \cup \mathrm{fn}(P) \qquad \mathrm{fn}(\gamma[a]) = \mathrm{fn}(\gamma) \cup \mathrm{fn}(a)$$
$$\mathrm{fn}(x^\eta(\vec{y})) = \mathrm{fn}(\overline{x}^\eta(\vec{y})) = \{x, \vec{y}\} \cup \mathrm{fn}(\eta) \qquad \mathrm{fn}(\overline{x}^\eta\{\!|y|\!\}) = \{x, y\} \cup \mathrm{fn}(\eta)$$
$$\mathrm{fn}(\overline{x}^\eta\{\!|P|\!\}) = \mathrm{fn}(x^\eta\{\!|P|\!\}) = \{x\} \cup \mathrm{fn}(\eta) \cup \mathrm{fn}(P) \qquad \mathrm{fn}(x^\eta_z) = \{x, z\} \cup \mathrm{fn}(\eta)$$

While synchronization of communications in Seal is pretty standard, the Seal model of mobility requires a three-party synchronisation between the sender, the receiver, and the seal being moved. The labelled transition system we propose uses intermediate transitions to express partial synchronisations. Actions can be roughly grouped into three groups: the actions that are the direct consequence of the exercise of a capability, those that denote a partial synchronisation, and the internal reduction. Let us describe each group in detail.

Exercise of capabilities A process can exert a capability, emitting the corresponding activity. The corresponding rules are (IN), (OUT), (SND), and (RCV). The activities input $x^\eta(\vec{y})$ and output $\overline{x}^\eta(\vec{y})$ account for communication. They are analogous to the corresponding rules of an early LTS for π-calculus.

The activities send $\overline{x}^\eta\{\!|y|\!\}$ and receive $x^\eta\{\!|Q|\!\}$ account for mobility. The former simply expresses that the process is willing to send a seal named y over the channel x^η. The latter expresses that the process is willing to receive a seal over the channel x^η. The early LTS supposes that the arbitrary seal body Q is provided by the context, and the outcome is exactly the process that would be obtained if the process exerting the capability received the seal body Q.

To this group belong also the P_z label. A consequence of the Seal's mobility model is that a seal can at any time be moved by a context. This implies that a seal $z[P]$ must be able to offer itself to be moved. This is accomplished by the (FREEZE) rule: a seal freezes itself, emitting the P_z label, that is by communicating to the environment its name and its content, and turning into an inactive process.

Partial synchronisation Three labels denote that two of the three components of a (mobility) synchronisation already interacted and are waiting for an action of the third component to yield an internal reduction.

As the diagram in Figure 4 illustrates, the LTS can generate τ transitions independently from the order in which labels match. This requires three intermediate labels, called 'capsule' $\overline{x}^\eta\{\!|P|\!\}$, 'lock' x^η_z, and 'seal chained' P^z. The label $\overline{x}^\eta\{\!|P|\!\}$ represents the action of serialising a seal: its emission indicates that a process willing to send a seal over a channel found it, serialised it, and is now waiting to synchronise with a receiver. The label x^η_z denotes that a process willing to receive a seal at x^η synchronised with the corresponding frozen seal named z, and is now looking for a sender. Remark that the names to be extruded are concretized by the (LOCK) rule: the semantics defined by the LTS performs the smallest number of extrusions needed. The label P^z denotes that a process willing to move a seal named z and a process willing to receive a seal with body P synchronised, and are now looking for the frozen seal (named z and whose body is P) to be moved.

Internal reduction The τ label denotes internal synchronisation. Given two matching actions, the (SYNC) rule constructs the process resulting from interaction: it puts the outcomes in parallel and concretizes the extruded names. Matching actions are identified by the $\curlyvee$ relation.

Labels				**Activities**				**Locations**			
ℓ	::=	τ	internal action	a	::=	$x^\eta(\vec{y})$	input	γ	::=	$*$	here
	$\mid$	P_z	seal freeze		$\mid$	$\overline{x}^\eta(\vec{y})$	output		$\mid$	z	inside z
	$\mid$	P^z	seal chained		$\mid$	$\overline{x}^\eta\{\!|y|\!\}$	send				
	$\mid$	$\gamma[a]$	activity a at γ		$\mid$	$\overline{x}^\eta\{\!|P|\!\}$	capsule				
					$\mid$	$x^\eta\{\!|P|\!\}$	receive				
					$\mid$	x^η_z	lock				

Table 3: S-Seal Actions.

Definition 3.7 *Let Υ be the smallest binary* symmetric *relation on labels containing the following relation:*

$$\{\ (\gamma_1[\overline{x}^{\eta_1}(\vec{y})]\ ,\ \gamma_2[x^{\eta_2}(\vec{y})])\ \mid\ \mathcal{M}\ \}\ \cup\ \{\ (\gamma_1[\overline{x}^{\eta_1}\{\!|S|\!\}]\ ,\ \ \gamma_2[x^{\eta_2}\{\!|S|\!\}])\ \mid\ \mathcal{M}\ \}$$
$$\cup\ \{\ (\gamma_1[x_z^{\eta_1}]\ ,\ \gamma_2[\overline{x}^{\eta_2}\{\!|z|\!\}])\ \mid\ \mathcal{M}\ \}\ \cup\ \{\ (S_z\ ,\ S^z)\ \}$$

where $\mathcal{M} \stackrel{def}{=} (\gamma_1 = \eta_1 = \gamma_2 = \eta_2 = *) \vee (\gamma_1 = *\ \wedge\ \mathsf{synch}_{\gamma_2}(\eta_1, \eta_2)) \vee (\gamma_2 = *\ \wedge\ \mathsf{synch}_{\gamma_1}(\eta_2, \eta_1))$.

Congruence rules An action observable at top-level can take place either at top-level, or inside a top-level seal. The labels keep track of where an action a takes place by tagging it with $*[a]$ if the action is a top level action, and with $x[a]$ if the action is unleashed inside a seal x. The (SEAL LABEL) rule transforms a $*[a]$ label into a $x[a]$ label provided that the action a can be observed from the outside of the seal x. For instance, in the process $x[z[P]]$ the seal $z[P]$ cannot be observed and this is reflected in LTS by the absence of a transition $x[z[P]] \xrightarrow{x[P_z]}$. The rules (OPEN FREEZE) and (OPEN CAPSULE) are responsible for the extrusion of names as a consequence of mobility actions.

The remaining congruence rules are fairly standard.

3.4 Basic properties of the LTS

We prove some properties of the labelled transition semantics. In particular, we investigate how transitions are preserved and reflected by injective renaming, and we prove the correspondence between the LTS and the semantics in chemical style.

First of all, we remark that for a given transition $A \vdash P \xrightarrow{\ell} O$ the structure of the process P and of the outcome O can be characterised up to structural congruence.

Lemma 3.8 (Transition analysis) *For* $\mathrm{fn}(P) \subseteq A$*, consider the derivation* $A \vdash P \xrightarrow{\ell} O$.

Let

$$P\ =\ \overline{x}^{\eta}\{\!|z|\!\}.P' \qquad Q\ =\ x^{\eta}\{\!|\vec{y}|\!\}.Q' \qquad R\ =\ z[S]$$

According to the legend

$$(R \mid P) \mid Q \dashrightarrow \qquad (R \mid Q) \mid P \longrightarrow \qquad (P \mid Q) \mid R \cdots\!\!\rightarrow$$

the diagram below depicts the possible interaction paths.

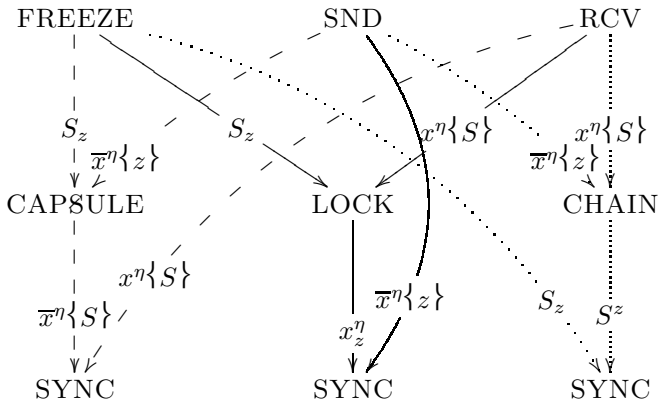

Figure 4: Synchronisation paths for mobility.

Congruence

(PAR) $\dfrac{A \vdash P \xrightarrow{\ell} P'}{A \vdash P \mid Q \xrightarrow{\ell} P' \mid Q}$

(RES) $u \notin \mathrm{fn}(\ell)$ $\dfrac{A, u \vdash P \xrightarrow{\ell} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{\ell} (\boldsymbol{\nu} u)P'}$

(BANG) $\dfrac{A \vdash \alpha.P \xrightarrow{\ell} P'}{A \vdash !\alpha.P \xrightarrow{\ell} P \mid !\alpha.P}$

(OPEN COM) $y, \eta, \gamma \neq u; u \in \vec{x}$ $\dfrac{A, u \vdash P \xrightarrow{\gamma[\overline{y}^{\eta}(\vec{x})]} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{\gamma[\overline{y}^{\eta}(\vec{x})]} P'}$

(OPEN FREEZE) $z \notin u; u \in \mathrm{fn}(S)$ $\dfrac{A, u \vdash P \xrightarrow{S_z} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{S_z} P'}$

(SEAL TAU) $\dfrac{A \vdash P \xrightarrow{\tau} P'}{A \vdash x[P] \xrightarrow{\tau} x[P']}$

(OPEN CAPSULE) $y, \eta, \gamma \notin \vec{u}; u \in \mathrm{fn}(S)$ $\dfrac{A, u \vdash P \xrightarrow{\gamma[\overline{y}^{\eta}\{S\}]} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{\gamma[\overline{y}^{\eta}\{S\}]} P'}$

(SEAL LABEL) $\dfrac{A \vdash P \xrightarrow{*[a]} P' \quad a \in \{y^{\uparrow}(\vec{z}), \overline{y}^{\uparrow}(\vec{z}), y^{\uparrow}\{Q\}, \overline{y}^{\uparrow}\{Q\}\}}{A \vdash x[P] \xrightarrow{x[a]} x[P']}$

Communication

(OUT) $\dfrac{}{A \vdash \overline{x}^{\eta}(\vec{y}).P \xrightarrow{*[\overline{x}^{\eta}(\vec{y})]} P}$

(IN) $\dfrac{}{A \vdash x^{\eta}(\vec{y}).P \xrightarrow{*[x^{\eta}(\vec{v})]} P[\vec{v}/\vec{y}]}$

Mobility

(SND) $\dfrac{}{A \vdash \overline{x}^{\eta}\{y\}.P \xrightarrow{*[\overline{x}^{\eta}\{y\}]} P}$

(RCV) $\dfrac{}{A \vdash x^{\eta}\{\vec{y}\}.P \xrightarrow{*[x^{\eta}\{Q\}]} P \mid y_1[Q] \mid \cdots \mid y_n[Q]}$

(CAPSULE) $\dfrac{A \vdash P \xrightarrow{S_z} P' \quad A \vdash Q \xrightarrow{*[\overline{x}^{\eta}\{z\}]} Q'}{A \vdash P \mid Q \xrightarrow{*[\overline{x}^{\eta}\{S\}]} P' \mid Q'}$

(LOCK) $\gamma = \eta = *$ *or* $(\gamma \neq * \wedge \eta = \uparrow)$ $\dfrac{A \vdash P \xrightarrow{S_z} P' \quad A \vdash Q \xrightarrow{\gamma[x^{\eta}\{S\}]} Q'}{A \vdash P \mid Q \xrightarrow{\gamma[x^{\eta}_z]} (\boldsymbol{\nu}\, \mathrm{fn}(S) \setminus A)(P' \mid Q')}$

(FREEZE) $\dfrac{}{A \vdash x[P] \xrightarrow{P_x} \mathbf{0}}$

(CHAIN) $\gamma = \eta_1 = \eta_2 = *$ or $(\eta_1 = \gamma \wedge \eta_2 = \uparrow)$ $\dfrac{A \vdash P \xrightarrow{*[\overline{x}^{\eta_1}\{y\}]} P' \quad A \vdash Q \xrightarrow{\gamma[x^{\eta_2}\{S\}]} Q'}{A \vdash P \mid Q \xrightarrow{S^y} P' \mid Q'}$

Synchronisation

(SYNC) $\ell_1 \curlyvee \ell_2$ $\dfrac{A \vdash P \xrightarrow{\ell_1} P' \qquad A \vdash Q \xrightarrow{\ell_2} Q'}{A \vdash P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}(\mathrm{fn}(\ell_1) \cup \mathrm{fn}(\ell_2)) \setminus A)(P' \mid Q')}$

The symmetric rules for (PAR), (CAPSULE), (LOCK), and (CHAIN) are omitted.

Table 4: A labelled transition system for the Seal Calculus.

$\ell = *[\overline{x}^{\eta}(\vec{z})]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(\overline{x}^{\eta}(\vec{z}).R_1 \mid R_2)$, *with* $x, \eta \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \vec{z})(R_1 \mid R_2)$;

$\ell = a[\overline{\boldsymbol{x}}^{\uparrow}(\vec{z})]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(\overline{x}^{\uparrow}(\vec{z}).R_1 \mid R_2)] \mid R_3)$, *with* $x \notin (\vec{u} \cup \vec{v})$ *and* $a \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \vec{z})(a[(\boldsymbol{\nu}\vec{v} \setminus \vec{z})(R_1 \mid R_2)] \mid R_3)$;

$\ell = *[x^{\eta}(\vec{z})]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(x^{\eta}(\vec{y}).R_1 \mid R_2)$, *with* $x, \eta, \vec{y}, \vec{z} \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1[\vec{z}/\vec{y}] \mid R_2)$;

$\ell = a[x^{\uparrow}(\vec{z})]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(x^{\uparrow}(\vec{y}).R_1 \mid R_2)] \mid R_3)$, *with* $x, \vec{y}, \vec{z} \notin (\vec{u} \cup \vec{v})$, *and* $a \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \{a\})(a[(\boldsymbol{\nu}\vec{v})(R_1[\vec{z}/\vec{y}] \mid R_2)] \mid R_3)$;

$\ell = S_z$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(z[S] \mid R_1)$, *with* $z \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(S))R_1$;

$\ell = S^z$ *implies either*

$P \equiv (\boldsymbol{\nu}\vec{u})(\overline{x}^{*}\{z\}.R_1 \mid x^{*}\{\vec{y}\}.R_2 \mid R_3)$ *with* $z, \mathrm{fn}(S) \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid R_2 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_3)$;

or

$P \equiv (\boldsymbol{\nu}\vec{u})(\overline{x}^{a}\{z\}.R_1 \mid R_2 \mid a[x^{\uparrow}\{\vec{y}\}.R_3 \mid R_4])$ *with* $z \notin \vec{u}$, $\mathrm{fn}(S) \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid R_2 \mid a[y_1[S] \mid \cdots \mid y_n[S] \mid R_3 \mid R_4])$;

$\ell = *[\overline{x}^{\eta}\{y\}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(\overline{x}^{\eta}\{y\}.R_1 \mid R_2)$, *with* $x, \eta, y \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid R_2)$;

$\ell = *[\overline{x}^{\eta}\{S\}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(\overline{x}^{\eta}\{y\}.R_1 \mid y[S] \mid R_2)$, *with* $x, \eta \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(S))(R_1 \mid R_2)$;

$\ell = a[\overline{x}^{\uparrow}\{S\}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(\overline{x}^{\uparrow}\{y\}.R_1 \mid y[S] \mid R_2)] \mid R_3)$, *with* $x \notin (\vec{u} \cup \vec{v})$, *and* $a \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(S))(a[(\boldsymbol{\nu}\vec{v} \setminus \mathrm{fn}(S))(R_1 \mid R_2)] \mid R_3)$;

$\ell = *[x^{\eta}\{S\}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(x^{\eta}\{\vec{y}\}.R_1 \mid R_2)$, *with* $x, \eta, \mathrm{fn}(S) \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_2)$;

$\ell = a[x^{\uparrow}\{S\}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(x^{\uparrow}\{\vec{y}\}.R_1 \mid R_2)] \mid R_3)$, *with* $x, y, \mathrm{fn}(S) \notin \vec{u} \cup \vec{v}$, *and* $a \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \{a\})(a[(\boldsymbol{\nu}\vec{v})(R_1 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_2)] \mid R_3)$;

$\ell = *[x^*_z]$ *implies*

$$P \equiv (\boldsymbol{\nu}\vec{u})(z[S] \mid x^*\{\!|\vec{y}|\!\}.R_1 \mid R_2),\ \textit{with}\ x, z \notin \vec{u};$$
$$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_2);$$

$\ell = a[x^{\uparrow}_z]$ *implies*

$$P \equiv (\boldsymbol{\nu}\vec{u})(z[S] \mid R_1 \mid a[(\boldsymbol{\nu}\vec{v})(x^{\uparrow}\{\!|\vec{y}|\!\}.R_2 \mid R_3)]),\ \textit{with}\ z \notin \vec{u},\ x \notin \vec{u} \cup \vec{v},\ \textit{and}\ a \notin \vec{u};$$
$$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid a[(\boldsymbol{\nu}\vec{v})(R_2 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_3)]);$$

where $\vec{u}$ and $\vec{v}$ are disjoint.

Proof Induction on derivations of transitions. To avoid unnecessary complications in the grammar of the labels, we included the label $[\overline{x}^{\eta}\{\!|y|\!\}]$ even if no process can emit it. □

We now show how injective renaming preserves and reflects transitions. These theorems are fundamental to prove that the bisimulation based equivalence that we will build on top of the LTS is preserved by injective substitutions. It is convenient to introduce some special notations for injective substitutions. Given an injective function $f : A \to B$ between two finite sets of names $A = \{a_1, \ldots, a_n\}$ and B, and a process P such that $\mathrm{fn}(P) \subseteq A$, we write fP for $P[f(a_1), \ldots, f(a_n)/a_1, \ldots, a_n]$. We denote $\mathrm{dom}(f)$ and $\mathrm{img}(f)$ respectively the domain and the image of f. We also use the convention that $f(*) = (*)$ and $f(\uparrow) = \uparrow$. Given $f : A \to B$ and $A' \subseteq A$, we write $f \downharpoonright A'$ for f restricted to A'. Given $f : A_1 \to B_1$ and $g : A_2 \to B_2$ with $A_1 \cap A_2 = \emptyset$, we write $f + g : A_1 \dot{\cup} A_2 \to B_1 \cup B_2$ for the function obtained by the set theoretic union of the graphs.

Lemma 3.9 (Injective substitution) *If $A \vdash P \xrightarrow{\ell} P'$, and $f : A \to B$ and $g : (\mathrm{fn}(\ell) \setminus A) \to (\mathbf{N} \setminus B)$ are injective, then $B \vdash fP \xrightarrow{(f+g)\ell} (f+g)P'$.*

Proof Induction on derivations of transitions.

(IN) Consider $A \vdash x^{\eta}(\vec{y}).P \xrightarrow{*[x^{\eta}(\vec{v})]} P[\vec{v}/\vec{y}]$. By alpha conversion, we suppose $\vec{y} \notin (A \cup B \cup (\mathrm{fn}(\ell) \setminus A) \cup \mathrm{img}(g))$. Then $f(x^{\eta}(\vec{y}).P) = fx^{f\eta}(\vec{y}).fP$ and $\mathrm{fn}(fx^{f\eta}(\vec{y}).fP) \subseteq B$. By (IN) $B \vdash fx^{f\eta}(\vec{y}).fP \xrightarrow{*[fx^{f\eta}((f+g)\vec{v})]} fP[(f+g)\vec{v}/\vec{y}]$. Now $\mathrm{fn}(P) \subseteq A, \vec{y}$ so $\mathrm{fn}(P) \cap \mathrm{dom}(g) = \emptyset$, so $fP = (f+g)P$. Hence $fP[(f+g)\vec{v}/\vec{y}] = (f+g)P[(f+g)\vec{v}/\vec{y}] = (f+g)(P[\vec{v}/\vec{y}])$, so we have the transition $B \vdash fx^{f\eta}(\vec{y}).fP \xrightarrow{(f+g)*[x^{\eta}(\vec{v})]} (f+g)P[\vec{v}/\vec{y}]$.

(RCV) Consider $A \vdash x^{\eta}\{\!|\vec{y}|\!\}.P \xrightarrow{*[x^{\eta}\{\!|Q|\!\}]} P \mid y_1[Q] \mid \cdots \mid y_n[Q]$. By (RCV) $B \vdash fx^{f\eta}\{\!|f\vec{y}|\!\}.fP \xrightarrow{*[fx^{f\eta}\{\!|Q'|\!\}]} fP \mid fy_1[Q'] \mid \cdots \mid fy_n[Q']$ for all processes Q'. In particular, taking $Q' = (f+g)Q$, $B \vdash fx^{f\eta}\{\!|f\vec{y}|\!\}.fP \xrightarrow{*[fx^{f\eta}\{\!|(f+g)Q|\!\}]} fP \mid fy_1[(f+g)Q] \mid \cdots \mid fy_n[(f+g)Q]$. As $\mathrm{fn}(P) \subseteq A$ and $\vec{y} \subseteq A$, we have the transition $B \vdash f(x^{\eta}\{\!|\vec{y}|\!\}.P) \xrightarrow{(f+g)(*[x^{\eta}\{\!|Q|\!\}])} (f+g)(P \mid y_1[Q] \mid \cdots \mid y_n[Q])$.

(OPEN COM) Define $f' : (A, u) \to (B, g(u))$ as $f + (u \mapsto g(u))$, and $g' = g \downharpoonright (\mathrm{fn}(\gamma[\overline{y}^{\eta}(\vec{x})]) \setminus \{u\})$. By the induction hypothesis $B, g(u) \vdash f'P \xrightarrow{(f'+g')\gamma[\overline{y}^{\eta}(\vec{x})]} (f'+g')P'$, so by (OPEN COM) $B \vdash (\boldsymbol{\nu}g(u))f'P \xrightarrow{(f'+g')\gamma[\overline{y}^{\eta}(\vec{x})]} (f'+g')P'$. Notice that by α-conversion and the definition of f' we have $(\boldsymbol{\nu}g(u))f'P = (\boldsymbol{\nu}u)fP$. Therefore, since $f' + g' = f + g$ we have $B \vdash f(\boldsymbol{\nu}u)P \xrightarrow{(f+g)\gamma[\overline{y}^{\eta}(\vec{x})]} (f+g)P'$.

(OPEN FREEZE),(OPEN CAPSULE) Similar to (OPEN COM).

(SYNC) The argument depends on the shape of the matching labels; we detail the case when synchronisation is communication. Other cases are similar. We also suppose $\ell_2 = \gamma[x^{\eta_2}(\vec{y})]$. $\mathrm{fn}(\tau) = \emptyset$, so we have $f : A \to B$ and $g : \emptyset \to (\mathbf{N} \setminus B)$. Take some $g' : (\mathrm{fn}(\gamma_2[x^{\eta_2}(\vec{y})]) \setminus A) \to (\mathbf{N} \setminus B)$ injective. By the induction hypothesis and (COM) we have

$$\frac{B \vdash fP \xrightarrow{(f+g')\gamma_1[\overline{x}^{\eta_1}(\vec{y})]} (f+g')P' \qquad B \vdash fQ \xrightarrow{(f+g')\gamma_2[x^{\eta_2}(\vec{y})]} (f+g')Q'}{B \vdash f(P \mid Q) \xrightarrow{\tau} (\boldsymbol{\nu}((f+g')\vec{y}) \setminus B)(f+g')(P' \mid Q')}$$

Now $(f+g')\vec{y}\backslash B = \mathrm{img}(g')$, so $B \vdash f(P \mid Q) \xrightarrow{\tau} (\boldsymbol{\nu}\,\mathrm{img}(g'))(f+g')(P' \mid Q')$. We have $f((\boldsymbol{\nu}\,\mathrm{dom}(g'))(P' \mid Q')) = (\boldsymbol{\nu}\,\mathrm{img}(g'))(f+g')(P' \mid Q')$, so $B \vdash f(P \mid Q) \xrightarrow{\tau} f((\boldsymbol{\nu}(\vec{y} \setminus B))(P' \mid Q'))$.

(LOCK) Similar to (SYNC).

(OUT),(SND),(FREEZE) Immediate.

(CAPSULE), (CHAIN), (PAR), (BANG), (SEAL TAU), (SEAL LABEL) Straightforward use of the induction hypothesis. □

Lemma 3.10 (Shifting)

1. $(A \vdash P \xrightarrow{\gamma[x^\eta(\vec{u})]} P' \wedge v \in (\vec{u} \setminus A))$ *if and only if* $(A, v \vdash P \xrightarrow{\gamma[x^\eta(\vec{u})]} P' \wedge v \in \vec{u} \setminus \mathrm{fn}(P))$;
2. $(A \vdash P \xrightarrow{\gamma[x^\eta\{Q\}]} P' \wedge v \in (\mathrm{fn}(Q) \setminus A))$ *if and only if* $(A, v \vdash P \xrightarrow{\gamma[x^\eta\{Q\}]} P' \wedge v \in \mathrm{fn}(Q) \setminus \mathrm{fn}(P))$;
3. $(A \vdash P \xrightarrow{Q^y} P' \wedge v \in (\mathrm{fn}(Q) \setminus A))$ *if and only if* $(A, v \vdash P \xrightarrow{Q^y} P' \wedge v \in \mathrm{fn}(Q) \setminus \mathrm{fn}(P))$.

Proof We detail the second part, others are similar (Part 3. depends on Part 2.).

We want to show that $A \vdash P \xrightarrow{\gamma[x^\eta\{Q\}]} P'$ if and only if $A, v \vdash P \xrightarrow{\gamma[x^\eta\{Q\}]} P'$, where $v \in \mathrm{fn}(Q) \setminus A$. We perform two inductions on derivations of transitions.

(RCV) Straightforward.

(PAR),(BANG) Straightforward use of the induction hypothesis.

(RES) Consider

$$\frac{A, y \vdash P \xrightarrow{\gamma[x^\eta\{Q\}]} P'}{A \vdash (\boldsymbol{\nu}y)P \xrightarrow{\gamma[x^\eta\{Q\}]} (\boldsymbol{\nu}y)P'} \qquad \frac{A, v, y \vdash P \xrightarrow{\gamma[x^\eta\{Q\}]} P'}{A, v \vdash (\boldsymbol{\nu}y)P \xrightarrow{\gamma[x^\eta\{Q\}]} (\boldsymbol{\nu}y)P'}$$

$$v \in (\mathrm{fn}(Q) \setminus A) \qquad v \in (\mathrm{fn}(Q) \setminus \mathrm{fn}((\boldsymbol{\nu}y)P))$$
$$y \notin \gamma, x, \eta \qquad y \notin \gamma, x, \eta$$

For the left-to-right implication, note that $v \in \mathrm{fn}(Q) \setminus (A, y)$, so by the induction hypothesis $A, v, y \vdash P \xrightarrow{\gamma[x^\eta\{Q\}]} P'$ and $v \in (\mathrm{fn}(Q) \setminus \mathrm{fn}((\boldsymbol{\nu}y)P))$. For the right-to-left implication, note that as A, v, y is well-formed we have $x \in \mathrm{fn}(Q) \setminus \mathrm{fn}(P)$, so by induction hypothesis $A, y \vdash P \xrightarrow{\gamma[x^\eta\{Q\}]} P'$ and $v \in \mathrm{fn}(Q) \setminus (A, y)$.

The other cases are vacuous. □

The transitions of an injectively substituted process fP are determined by the transitions of P as follows.

Lemma 3.11 (Converse to injective substitution) *For $f : A \to B$ injective, if $B \vdash fP \xrightarrow{\ell'} Q'$ then*

1. *there exist a label ℓ and a process Q,*
2. *there exist two sets of names H, I with $H \cap I = \emptyset$, $H \cup I = \mathrm{fn}(\ell) \setminus A$,*
3. *there exists $g : I \to B'$ bijective with $B' \cap B = \emptyset$,*
4. *there exists $h : H \to (B \setminus \mathrm{img}(f))$ injective,*

such that

a. *$A \vdash P \xrightarrow{\ell} Q$, and $\ell' = (f+g+h)\ell$, and $Q' = (f+g+h)Q$; and*

b. if $\ell' \notin \{\gamma[x^\eta(\vec{y})], \gamma[x^\eta\{R\}], R^y \mid \textit{for some } \gamma, x, \eta, \vec{y}, R\}$ *then* $H = \emptyset$.

Proof Induction on derivations of transitions.

(OUT) We have $B \vdash f(\overline{x}^\eta(\vec{y}).P) \xrightarrow{f(*[\overline{x}^\eta(\vec{y})])} fP$. Take $Q = P$, $\ell = *[\overline{x}^\eta(\vec{y})]$. Also take $H = I = \emptyset$, $g : \emptyset \to \emptyset$, $h : \emptyset \to B \setminus \mathrm{img}(f)$. Then $A \vdash \overline{x}^\eta(\vec{y}).P \xrightarrow{*[\overline{x}^\eta(\vec{y})]} P$.

(IN) We can suppose without loss of generality that $\vec{y} \notin A \cup B$. Then we have $B \vdash f(x^\eta(\vec{y}).P) \xrightarrow{*[fx^{f\eta}(\vec{z})]} (fP)[\vec{z}/\vec{y}]$. We scan $\vec{z}$ according to the rules below, and in doing so we construct a new set of names $\vec{v}$, and H, I, B', g, h. Start with $I = H = B' = \emptyset$, and g, h always undefined functions. Then, for all i:

- if $z_i \in \mathrm{img}(f)$ then $v_i = u$ for $u \in A$ such that $f(u) = z_i$;
- if $z_i \in B \setminus \mathrm{img}(f)$ and for all $j < i$, $z_j \neq z_i$, then take some fresh o_i such that $o_i \notin A$. Add o_i to H, add $h : o_i \mapsto z_i$ to the current graph of h, and let $v_i = o_i$; if $z_j = z_i$ for some $j < i$, then let $v_i = v_j$;
- if $z_i \notin B$ and for all $j < i$, $z_j \neq z_i$, then take some new o_i such that $o_i \notin A$. Add o_i to I, add z_i to B', add $g : o_i \mapsto z_i$ to the current graph of g, and let $v_i = o_i$; if $z_j = z_i$ for some $j < i$, then let $v_i = v_j$.

Take $\ell = *[x^\eta(\vec{v})]$, $Q = P[\vec{v}/\vec{y}]$. Also take the H, I, g, h constructed while generating $\vec{v}$. Then $H \cap I = \emptyset$, $H \cup I = \mathrm{fn}(\ell) \setminus A$, $g : I \to B'$ is a bijection, $h : H \to (B \setminus \mathrm{img}(f))$ is injective. Moreover, $\ell' = (f + g + h)\ell$, $Q' = (f + g + h)Q$, and $A \vdash x^\eta(\vec{y}).P \xrightarrow{*[x^\eta(\vec{v})]} P[\vec{v}/\vec{y}]$.

(SND), (FREEZE) Similar to (OUT).

(RCV) We have $B \vdash f(x^\eta\{\vec{y}\}.P) \xrightarrow{fx^{f\eta}\{R'\}} fP \mid fy_1[R'] \mid \ldots \mid fy_n[R']$. We construct $\vec{v}, H, I, B', g, h$ as we have done in (INP), but where $\vec{z}$ is replaced by $\overrightarrow{\mathrm{fn}(R')}$ (the ordering is not important). Then let $R = R'[\vec{v}/\overrightarrow{\mathrm{fn}(R')}]$, and take $\ell = *[x^\eta\{R\}]$, and $Q = P \mid y_1[R] \mid \ldots \mid y_n[R]$.

(LOCK) Consider an instance

$$\frac{B \vdash fP \xrightarrow{S_z} P' \quad B \vdash fQ \xrightarrow{\gamma[x^\eta\{S\}]} Q'}{B \vdash f(P \mid Q) \xrightarrow{\gamma[x^\eta_z]} (\boldsymbol{\nu}\, \mathrm{fn}(S) \setminus B)(P' \mid Q')}$$

By induction hypothesis, there exist P'_0, S_0, z_0 and $g : (\mathrm{fn}(S_0) \setminus A) \to B'$ such that $A \vdash P \xrightarrow{S_0 z_0} P'_0$ and $(f + g)S_0 = S$, $f(z_0) = z$, $(f + g)P'_0 = P'$. This also forces H to be empty. Again, by induction hypothesis, there exist $Q'_1, \gamma_1, x_1, \eta_1, S_1$ and $g' : (\mathrm{fn}(S_1) \setminus A) \to B'$ such that $A \vdash Q \xrightarrow{\gamma_1[x_1{}^{\eta_1}\{S_1\}]} Q'_1$ and $(f + g')S_1 = S$, $(f + g')Q'_1 = Q'$, $f\gamma_1 = \gamma$, $f\eta_1 = \eta$. By Lemma 3.9, as g and g' are bijection, $A \vdash Q \xrightarrow{\gamma_1[x_1{}^{\eta_1}\{S_0\}]} g^{-1}(g'(Q'_1))$. By (LINK), $A \vdash P \mid Q \xrightarrow{\gamma_1[x_1{}^{\eta_1}_{z_1}]} (\boldsymbol{\nu}\, \mathrm{fn}(S_0) \setminus A)(P' \mid Q')$. It remains only to note that $(f + g)((\boldsymbol{\nu}\, \mathrm{fn}(S_0) \setminus A)(P'_0 \mid g^{-1}(g'(Q'_1)))) = (\boldsymbol{\nu}\, \mathrm{fn}(S) \setminus B)(P' \mid Q')$.

(CHAIN),(CAPSULE), (SYNC) These cases are similar to (LOCK).

(PAR), (BANG), (SEAL TAU), (SEAL LABEL) By the induction hypothesis.

(RES) Consider an instance

$$\frac{A, u' \vdash P' \xrightarrow{\ell'} Q'}{A \vdash (\boldsymbol{\nu}u')P' \xrightarrow{\ell'} (\boldsymbol{\nu}u')Q'}\ u' \notin \mathrm{fn}(\ell)$$

where $fP = (\boldsymbol{\nu}u')P'$. There exists $u \notin A$ such that $P = (\boldsymbol{\nu}u)P_0$ and $P' = (f + [u'/u])P_0$. By the induction hypothesis there exist ℓ, Q_0, H, I and $g' : I \to B'$, $h : H \to (B \setminus \mathrm{img}(f)$, where $H \cup I = \mathrm{fn}(\ell) \setminus (A \cup \{u\})$, such that $A, u \vdash P_0 \xrightarrow{\ell} Q_0$, and $\ell' = (f + [u'/u] + g' + h)\ell$, and $Q' = (f + [u'/u] + g' + h)Q_0$. Since $u' \notin \mathrm{fn}(\ell')$, it follows that $u \not\in \mathrm{fn}(\ell)$. By (RES), $A \vdash P \xrightarrow{\ell} (\boldsymbol{\nu}u)Q_0$. The case follows taking $g = g' + [u'/u]$ and $Q = (\boldsymbol{\nu}u)Q_0$.

(OPEN COM) Consider an instance

$$\frac{B, u' \vdash P' \xrightarrow{\gamma[\overline{y}^{\eta}(\vec{x}')]} Q'}{B \vdash (\boldsymbol{\nu} u')P' \xrightarrow{\gamma[\overline{y}^{\eta}(\vec{x}')]} Q'} y, \eta, \gamma \neq u'; u' \in \vec{x}'$$

where $fP = (\boldsymbol{\nu} u')P'$. There exist $u \notin A$ and P_0 such that $P = (\boldsymbol{\nu} u)P_0$ and $P' = (f + [u'/u])P_0$. By induction hypothesis, there exist $\ell, Q, g' : \mathrm{fn}(\ell \setminus (A, u)) \to B'$ such that $A, u \vdash P_0 \xrightarrow{\ell} Q$, where $\ell = \gamma_0[\overline{y_0}^{\eta_0}(\vec{x}_0)]$, and $f\gamma_0 = \gamma$, $fy_0 = y$, $f\eta_0 = \eta$ and $(f + g' + [u'/u])\vec{x}_0 = \vec{x}$. By (OPEN COM) $A \vdash P \xrightarrow{\gamma_0[\overline{y_0}^{\eta_0}(\vec{x}_0)]} Q$, and the case follows taking $g = g' + [u'/u]$.

(OPEN FREEZE), (OPEN CAPSULE) Similar to (OPEN COM). □

Corollary 3.12 *Let $f : A \to B$ injective, and $\mathrm{fn}(P) \subseteq A$. $A \vdash P \xrightarrow{\tau} P'$ if and only if $B \vdash fP \xrightarrow{\tau} fP'$.*

To conclude this section, we state and prove the equivalence between the LTS and the semantics in chemical style.

Lemma 3.13 *If $A \vdash P \xrightarrow{\ell} Q$ and $P' \equiv P$ then $A \vdash P' \xrightarrow{\ell} Q'$ for some $Q' \equiv Q$.*

Proof Induction on the size of the derivation of $P' \equiv P$. □

Theorem 3.14 *Let P be a process:*

1. *if $\mathrm{fn}(P) \subseteq A$ and $A \vdash P \xrightarrow{\tau} Q$, then $P \twoheadrightarrow Q$;*
2. *if $P \twoheadrightarrow Q$ then there exists $A \supseteq \mathrm{fn}(P)$ such that $A \vdash P \xrightarrow{\tau} Q'$, where $Q' \equiv Q$.*

Proof Both parts are proved by transition induction.

Part 1. If there is a derivation of $A \vdash P \xrightarrow{\tau} Q$, then one of the following applies:

1. the last rule applied to derive $A \vdash P \xrightarrow{\tau} Q$ is (SYNC);
2. there are processes P', Q' and a context $C[-]$ such that $P = C[P']$, $Q = C[Q']$, and $A \vdash P' \xrightarrow{\tau} Q'$.

We proceed by induction on the structure of the context $C[-]$.

Base Case There are two processes P_1 and P_2 such that $P = P_1 \mid P_2$, $A \vdash P_1 \xrightarrow{\ell_1} P_1'$, $A \vdash P_2 \xrightarrow{\ell_2} P_2'$, and $\ell_1 \curlyvee \ell_2$. Lemma 3.8 describes the structure of a process P and the outcome O for each label ℓ after each transition. It remains to verify that the LTS and the $\curlyvee$ relation characterise matching processes in the reduction relation. This requires checking all the possible matching labels in the $\curlyvee$ relation. We detail the case for $\ell_1 = *[\overline{x}^a\{\!|\, S \,|\!\}]$, $\ell_2 = a[x^{\uparrow}\{\!|\, S \,|\!\}]$. Other cases are similar.

The derivation $P_1 \xrightarrow{*[\overline{x}^a\{\!|\, S \,|\!\}]} P_1'$ implies that $P_1 \equiv (\boldsymbol{\nu}\vec{z})(\overline{x}^a\{\!|\, y \,|\!\}.R_1 \mid y[S] \mid R_2)$ and $P_1' \equiv (\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(S))(R_1 \mid R_2)$, with $x, a \notin \vec{z}$.

The derivation $P_2 \xrightarrow{a[x^{\uparrow}\{\!|\, S \,|\!\}]} P_2'$ implies that $P_2 \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(x^{\uparrow}\{\!|\, \vec{y} \,|\!\}.R_1 \mid R_2)] \mid R_3)$ and $P_2' \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(R_1 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_2)] \mid R_3)$, with $x, y, \mathrm{fn}(S) \notin (\vec{u} \cup \vec{v})$ and $a \notin \vec{u}$.

Observe that $\mathrm{fn}(S) \setminus A = \mathrm{fn}(S) \cap \vec{z}$. It holds

$$\begin{aligned} P_1 \mid P_2 \;&\equiv\; (\boldsymbol{\nu}\vec{z})(\overline{x}^a\{\!|\, y \,|\!\}.R_1 \mid y[S] \mid R_2) \mid (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(x^{\uparrow}\{\!|\, \vec{y} \,|\!\}.R_1 \mid R_2)] \mid R_3) \\ &\twoheadrightarrow\; (\boldsymbol{\nu}(\mathrm{fn}(S) \cup \{a\}) \setminus A)((\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(S))(R_1 \mid R_2) \\ &\qquad\qquad \mid (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(R_1 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_2)] \mid R_3)) \\ &\equiv\; P_1' \mid P_2' \,. \end{aligned}$$

Inductive step Follows easily from the congruence laws of reduction.

Part 2. We detail one of the base cases, the others are similar. Rule (move out) says:

$$x^a\{\!|\vec{u}|\!\}.P \mid a[(\boldsymbol{\nu}\vec{z})(\overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \mid v[R] \mid Q_2)] \twoheadrightarrow (\boldsymbol{\nu}\,\mathrm{fn}(R)\cap\vec{z})(P \mid u_1[R] \mid \cdots \mid u_n[R] \mid a[(\boldsymbol{\nu}\vec{z}\setminus\mathrm{fn}(R))(Q_1 \mid Q_2)])$$

where $x \notin \vec{z}$. Let $A = \mathrm{fn}(P)$. From the LTS, we can derive:

1. $A \vdash x^a\{\!|\vec{u}|\!\}.P \xrightarrow{*[x^a\{\!|R|\!\}]} P \mid u_1[R] \mid \cdots \mid u_n[R]$
2. As $x \notin \vec{z}$ the following is a valid derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{A \vdash \overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \xrightarrow{*[\overline{x}^{\uparrow}\{\!|v|\!\}]} Q_1 \qquad A \vdash v[R] \xrightarrow{R_v} \mathbf{0}}{A \vdash \overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \mid v[R] \xrightarrow{*[\overline{x}^{\uparrow}\{\!|R|\!\}]} Q_1 \mid \mathbf{0}}}{A \vdash \overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \mid v[R] \mid Q_2 \xrightarrow{*[\overline{x}^{\uparrow}\{\!|R|\!\}]} Q_1 \mid \mathbf{0} \mid Q_2}}{A \vdash (\boldsymbol{\nu}\vec{z})(\overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \mid v[R] \mid Q_2) \xrightarrow{*[\overline{x}^{\uparrow}\{\!|R|\!\}]} (\boldsymbol{\nu}\vec{z}\setminus\mathrm{fn}(R))(Q_1 \mid \mathbf{0} \mid Q_2)}}{A \vdash a[(\boldsymbol{\nu}\vec{z}\setminus\mathrm{fn}(R))(\overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \mid v[R] \mid Q_2)] \xrightarrow{a[\overline{x}^{\uparrow}\{\!|R|\!\}]} a[(\boldsymbol{\nu}\vec{z}\setminus\mathrm{fn}(R))(Q_1 \mid \mathbf{0} \mid Q_2)]}$$

As $*[x^a\{\!|R|\!\}] \curlyvee a[\overline{x}^{\uparrow}\{\!|R|\!\}]$, we can apply the rule (SYNC) to derive

$$x^a\{\!|\vec{u}|\!\}.P \mid a[(\boldsymbol{\nu}\vec{z})(\overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \mid v[R] \mid Q_2)] \xrightarrow{\tau} (\boldsymbol{\nu}\,\mathrm{fn}(R)\cap\vec{z})(P \mid u_1[R] \mid \cdots \mid u_n[R] \mid a[(\boldsymbol{\nu}\vec{z}\setminus\mathrm{fn}(R))(Q_1 \mid \mathbf{0} \mid Q_2)])$$

as desired.

To complete the inductive step, observe that the congruence cases follow because the label τ can cross every context (rules (PAR), (RES), (SEAL TAU)), and the insensitivity to structural congruence follows from Lemma 3.13. □

3.5 A proof method

In this section we use the LTS defined in the previous section to define a bisimulation based relation contained in reduction barbed congruence. This will be an useful proof method to ensure equivalence between terms, and will be subsequently used to explore the behavioural theory of S-Seal. At the same time, its development will point out some peculiarities of Seal semantics.

3.5.1 LTS and observable actions

We first reexamine the definition of reduction barbed congruence, and we show that it is very robust under changes to the definition of barbs.

The predicate $P \downarrow n$ detects the ability of a process P to interact with the environment via the seal n. In other process calculi, like the π-calculus, barbs are defined using visible actions. We show that reduction barbed congruence is not affected if we change our definition of barb with barbs inherited from the *visible* actions of the LTS. We refer to barbs as of Definition 3.3 as *natural barbs*.

First, we remark that the exposure of a natural barb corresponds to the emission of a 'freeze' label in the labelled transition system:

Lemma 3.15 *$P \downarrow n$ if and only if $A \vdash P \xrightarrow{Q_n} P'$ for some P', Q, and A, with $\mathrm{fn}(P) \subseteq A$.*

Proof The 'if' part is a consequence of Lemma 3.8, while the 'only if' part follows straightforwardly from Definition 3.5. □

$$C_{*[\overline{x}^*\{y\}]}[-] = y[\overline{a}^\uparrow()] \mid x^*\{i\}.a^i().o[\,] \mid - \qquad C_{*[x_z^*]}[-] = \overline{x}^*\{z\}.o[\,] \mid -$$
$$C_{*[\overline{x}^\uparrow\{y\}]}[-] = y[\overline{a}^\uparrow()] \mid x^u\{i\}.a^i().o[\,] \mid u[-] \qquad C_{k[x_z^\uparrow]}[-] = \overline{x}^k\{z\}.o[\,] \mid -$$

$$C_{*[x^*\{S\}]}[-] = a[S] \mid \overline{x}^*\{a\}.o[\,] \mid - \qquad C_{*[\overline{x}^*\{\star\}]}[-] = x^*\{a\}.o[\,] \mid -$$
$$C_{*[x^\uparrow\{S\}]}[-] = a[S] \mid \overline{x}^u\{a\}.o[\,] \mid u[-] \qquad C_{*[\overline{x}^\uparrow\{\star\}]}[-] = x^u\{a\}.o[\,] \mid u[-]$$
$$C_{k[x^\uparrow\{S\}]}[-] = a[S] \mid \overline{x}^k\{a\}.o[\,] \mid - \qquad C_{k[\overline{x}^\uparrow\{\star\}]}[-] = x^k\{a\}.o[\,] \mid -$$

where a, i, o, u are fresh names.

Table 5: Contexts that transform a λ-barb into a natural barb.

We now prove that for all classes of visible actions of our LTS the resulting definitions of barbed congruence collapse and coincide with $\cong$. For that we classify the actions of the LTS according to the axiom or rule responsible for generating them (that is, according to their 'shape'):

Definition 3.16 (λ-barbs) *We define the following sets of actions*

$$\mathcal{L}_{*SND^*} = \{\ *[\overline{x}^*\{y\}] \mid \textit{for all } x, y\ \} \qquad \mathcal{L}_{*LOCK^*} = \{\ *[x_z^*] \mid \textit{for all } x, z\ \}$$
$$\mathcal{L}_{*SND^\uparrow} = \{\ *[\overline{x}^\uparrow\{y\}] \mid \textit{for all } x, y\ \} \qquad \mathcal{L}_{*LOCK^\uparrow} = \{\ *[x_z^\uparrow] \mid \textit{for all } x, z\ \}$$

$$\mathcal{L}_{*RCV^*} = \{\ *[x^*\{S\}] \mid \textit{for all } x, S\ \} \qquad \mathcal{W}_{*CAPS^*} = \{\ *[\overline{x}^*\{S\}] \mid \textit{for all } x, S\ \}$$
$$\mathcal{L}_{*RCV^\uparrow} = \{\ *[x^\uparrow\{S\}] \mid \textit{for all } x, S\ \} \qquad \mathcal{W}_{*CAPS^\uparrow} = \{\ *[\overline{x}^\uparrow\{S\}] \mid \textit{for all } x, S\ \}$$
$$\mathcal{L}_{kRCV^\uparrow} = \{\ k[x^\uparrow\{S\}] \mid \textit{for all } x, S\ \} \qquad \mathcal{W}_{kCAPS^\uparrow} = \{\ k[\overline{x}^\uparrow\{S\}] \mid \textit{for all } x, S\ \}$$

and we refer to each of them as a class of actions.

Let $\mathcal{L} = \bigcup_\lambda \mathcal{L}_\lambda$. For $\omega \in \mathcal{L}$, we write $P \downarrow \omega$ if $P \xrightarrow{\omega}$, and $P \Downarrow \omega$ if $P \Rightarrow \xrightarrow{\omega}$.

Let $\mathcal{W} = \bigcup_\lambda \mathcal{W}_\lambda$. We write $P \downarrow \gamma[\overline{x}^\eta\{\star\}]$ if there exists a process S such that $P \xrightarrow{\gamma[\overline{x}^\eta\{S\}]}$. We write $P \Downarrow \gamma[\overline{x}^\eta\{\star\}]$ if there exists a process S such that $P \Rightarrow \xrightarrow{\gamma[\overline{x}^\eta\{S\}]}$.

Let $\mathcal{A} = \mathcal{L} \cup \mathcal{W}$, and let λ range over the classes of actions. We call λ-barbs *all the barbs that belong to the class λ.*

Definition 3.17 *For each class of actions λ, let $\cong_\lambda$ be the largest congruence over processes that is* reduction closed *and* preserves λ-barbs.

For example, if $P \cong_{*SND^*} Q$ and $P \downarrow *[\overline{a}^*\{z\}]$ (that is, $P \xrightarrow{*[\overline{a}^*\{z\}]}$), then $Q \Downarrow *[\overline{a}^*\{z\}]$ (that is, $Q \Rightarrow \xrightarrow{*[\overline{a}^*\{z\}]}$). And if $P \cong_{*CAPS^*} Q$ and $P \downarrow *[\overline{a}^*\{\star\}]$ (that is, $P \xrightarrow{*[\overline{a}^*\{R\}]}$ for some R), then $Q \Downarrow *[\overline{a}^*\{\star\}]$ (that is, $Q \Rightarrow \xrightarrow{*[\overline{a}^*\{S\}]}$ for some S).

Lemma 3.18 *Consider the contexts defined in Table 5. Then $C_\omega[P] \Downarrow o$ if and only if $P \Downarrow \omega$.*

Proof The *if* direction is a direct consequence of Lemma 3.8.

For the *only if* direction, the proof depends on the action ω. The main argument is that the process $C_\omega[P]$ can unleash the seal named o only if P performs the action ω.

The most interesting case is for $\omega = *[\overline{x}^*\{y\}]$. The term

$$C_{*[\overline{x}^*\{y\}]}[P] \;=\; y[\overline{a}^\uparrow()] \mid x^*\{i\}.a^i().o[\,] \mid P$$

must consume the two actions $x^*\{i\}$ and $a^i()$ to activate the seal named o. The first one synchronises either with $\overline{x}^*\{y\}$ (as we desire) or $\overline{x}^*\{S\}$. In this last case the moved seal is provided by P and it cannot

$$D_n^{*[\overline{x}^*\{y\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.\overline{x}^*\{y\} \qquad D_n^{*[x_z^*]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.(x^*\{z\} \mid z[\mathbf{0}])$$

$$D_n^{*[\overline{x}^\uparrow\{y\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.\overline{x}^\uparrow\{y\} \qquad D_n^{k[x_z^\uparrow]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.(k[x^*\{z\}] \mid z[\mathbf{0}])$$

$$D_n^{*[x^*\{S\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.x^*\{\} \qquad D_n^{*[\overline{x}^*\{\star\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.\overline{x}^*\{\mathbf{0}\}$$

$$D_n^{*[x^\uparrow\{S\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.x^\uparrow\{\} \qquad D_n^{*[\overline{x}^\uparrow\{\star\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.\overline{x}^\uparrow\{\mathbf{0}\}$$

$$D_n^{k[x^\uparrow\{S\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.k[x^\uparrow\{\}] \qquad D_n^{k[\overline{x}^\uparrow\{\star\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.k[\overline{x}^\uparrow\{\mathbf{0}\}]$$

where a is a fresh name.

Table 6: Contexts that transform a natural barb into a λ-barb.

perform an interaction over the channel a, that has been chosen fresh for P. In turn, the second action can be consumed only if the seal y provided by the context is moved and renamed into i. This guarantees that $P \Downarrow \overline{x}^*\{y\}$.

The other cases are straightforward. □

Lemma 3.19 *Consider the contexts defined in Table 6. Then $D_n^\omega[P] \Downarrow \omega$ if and only if $P \Downarrow n$.*

Proof Each context can be rewritten as $- \mid \overline{a}^*\{n\} \mid a^*\{n\}.X$ where X is depends on the particular context. As the name a is fresh, the prefix $a^*\{n\}$ is consumed only if the process that fills the hole offers a seal named n at top-level, that is, it emits $\downarrow n$. The continuation X is then responsible for generating the appropriate λ-barb. □

Theorem 3.20 *Let P and Q be two processes, and let λ be a class of actions. Then $P \cong Q$ if and only if $P \cong_\lambda Q$.*

Proof Since $\cong$ and $\cong_\lambda$ differ only in the barb which is used, it suffices to show $\downarrow n$ and $\downarrow \lambda$ imply each other. First we prove that $P \cong_\lambda Q$ implies $P \cong Q$. Let $P \cong Q$ and $P \downarrow \omega$ for $\omega \in \lambda$. We want to conclude that $Q \Downarrow \omega$. As $\cong$ is contextual, it holds $C_\omega[P] \cong C_\omega[Q]$. Lemma 3.18 guarantees that $C_\omega[P] \Downarrow o$. As $\cong$ preserves natural barbs, it must hold $C_\omega[Q] \Downarrow o$. Lemma 3.18 allows us to conclude $Q \Downarrow \omega$.

For the other implication, suppose $P \cong_\lambda Q$ and $P \downarrow n$. We want to conclude that $Q \Downarrow n$. Let ω be a label in λ. As $\cong_\lambda$ is contextual, it holds $D_n^\omega[P] \cong D_n^\omega[Q]$. Lemma 3.19 guarantees that $D_n^\omega[P] \Downarrow \omega$. As $\cong$ preserves λ-barbs, it must hold $D_n^\omega[Q] \Downarrow \omega$. Lemma 3.19 allows us to conclude $Q \Downarrow n$. □

To shorten the presentation, in the Definition of λ-barbs we did not include actions related to communication. Unsurprisingly, the theorem above can easily be expanded to take into account also barbs exposing communication actions.

Theorem 3.20 brings out two peculiarities of the LTS. First, the class of action generated by the (Chain) rule does not appear among the classes for which reduction λ-barbed congruence coincides with reduction barbed congruence. As we will discuss in Section 3.6, these actions cannot be observed by a context. Second, as with natural barbs, the classes of action $\gamma CAPS^\eta$ do not report the body of the moved seal in their prototypical action. This is typical of our treatment of some of the higher order actions, where weak matching requirements on the transmitted processes are imposed.

3.5.2 Higher-order actions and bisimulation

Lemma 3.15 shows that the observation used in the contextual equivalence is insensitive to the particular process Q occurring in the label of the corresponding 'freeze' transition. Thus we expect an LTS-based characterisation of this equivalence not to be strict in matching higher-order labels.

The difficulties of defining equivalences based on LTS whose labels may contain processes are well-known. To avoid them we resort to the intuition underlying the definition of Sangiorgi's *context bisimulation* for HO-π [San92, San96], and require that the outcomes of two bisimilar processes emitting higher order transitions must be equivalent with respect to every possible interaction with a context.

Processes appear in four kinds of higher-order labels: $\gamma[x^\eta\{\!|S|\!\}]$, S^z, R_z, and $\gamma[\overline{x}^\eta\{\!|R|\!\}]$. We focus on the contexts with which a process emitting a higher-order label can interact, with the aim of finding out how to apply context bisimulation to our framework.

Receiving a seal body Suppose that $A \vdash P \xrightarrow{\gamma[x^\eta\{\!|S|\!\}]} P'$, and let Q be a process supposed equivalent to P. The process P' is the outcome of the interaction between P and a context $C[-]$ sending a seal body S over the channel x^η. In the bisimulation game, if the process Q can receive an arbitrary seal body over the channel x^η, then a context cannot easily separate P from Q. At the same time, it should be stressed that the process S appearing in the label is offered by the context, and as such the processes P and Q must both be ready to receive it. Therefore when comparing processes that receive a seal body, it is not restrictive to require syntactically identical processes in higher-order labels.

The case for S^z is analogous since, again, the process S is offered by the environment.

Emitting a seal body Suppose that $A \vdash P \xrightarrow{R_z} P'$. This can be seen as the offer of an interaction with a context that can move and/or duplicate the seal z. We must at least require that there exists a process Q' such that $A \vdash Q \xrightarrow{S_z} Q'$ for some seal body S, otherwise a context can easily separate P from Q. At the same time, imposing S to be syntactically the same as R is overly restrictive. This is a consequence of the fact that the context can not directly investigate the seal body S, but it must move and/or duplicate it before being able to interact with it. Thus, we do not impose directly any condition on S. Rather, we ask that the outcomes obtained after interacting with an arbitrary context that receives (that is, moves and/or copies) the seal are equivalent.

The case for $\overline{x}^\eta\{\!|R|\!\}$ is similar: the only advantage is that it is known where the interacting context is going to reactivate the copies of the seal.

To state the equivalence of processes emitting a seal body, we must characterise all the possible outcomes obtained after an interaction between a mobility offer and a context willing to receive the seal body being sent. This role is played by the *receiving contexts*.

Receiving contexts represent all the processes that may result from the migration of a seal. An example is probably helpful here. Let A be a set of names. In this 'universe', consider a process P that emits the label $z[\overline{x}^\uparrow\{\!|R|\!\}]$ and becomes P':

$$A \vdash P \xrightarrow{z[\overline{x}^\uparrow\{\!|R|\!\}]} P' \ .$$

A context that synchronises with P over this label must have the shape $C'[-] \equiv C[\ - \mid x^z\{\!|\vec{y}|\!\}.V\ ]$, for some $C[-]$ and V. The resulting reduction is

$$C'[P] \ \equiv\ C[\ P \mid x^z\{\!|\vec{y}|\!\}.V\ ] \quad \twoheadrightarrow \quad C[\ (\boldsymbol{\nu}\, \mathrm{fn}(R) \setminus A)(P' \mid y_1[R] \mid \cdots \mid y_n[R] \mid V)\ ] \ .$$

The process V cannot know any name in $\mathrm{fn}(R) \setminus A$, so we rewrite the above reduction as

$$C[\ P \mid x^z\{\!|\vec{y}|\!\}.V\ ] \quad \twoheadrightarrow \quad C[\ (\boldsymbol{\nu}\, \mathrm{fn}(R) \setminus A)(P' \mid y_1[R] \mid \cdots \mid y_n[R]) \mid V\ ] \ . \tag{3}$$

Our aim is to define a bisimulation based equivalence, $\approx$, such that contexts like $C'[-]$ cannot distinguish equivalent processes. In the example above, this means that the process $C'[Q]$ must reduce to a process equivalent to the outcome of (3). An easy way to achieve this is to require that in the bisimulation game Q emits a label that consumes the action $x^z\{\!|\vec{y}|\!\}.V$, that is

$$A \vdash Q \xrightarrow{z[\overline{x}^\uparrow\{\!|S|\!\}]} Q' \quad \text{for some } S, Q',$$

and that the outcome of $C'[P]$ is equivalent to the outcome of $C'[Q]$:

$$C[\ (\boldsymbol{\nu}\, \mathrm{fn}(R) \setminus A)(P' \mid y_1[R] \mid \cdots \mid y_n[R]) \mid V\ ] \approx C[\ (\boldsymbol{\nu}\, \mathrm{fn}(S) \setminus A)(Q' \mid y_1[S] \mid \cdots \mid y_n[S]) \mid V\ ]$$

Some housekeeping now: we factor out the context common to both members of the equation, $C[[-] \mid V]$, and we check that the following equivalence holds:

$$(\boldsymbol{\nu}\,\mathrm{fn}(R) \setminus A)(P' \mid y_1[R] \mid \cdots \mid y_n[R]) \quad \approx \quad (\boldsymbol{\nu}\,\mathrm{fn}(S) \setminus A)(Q' \mid y_1[S] \mid \cdots \mid y_n[S])\ .$$

This idea can be generalised. Consider two processes (like P and Q) that in an universe A send two (possibly different) seal bodies from a location z toward their parent $\uparrow$. We consider these two processes as equivalent only if the substitutions of the sent seal bodies for Y and of the outcomes for X in the following pattern

$$(\boldsymbol{\nu}\,\mathrm{fn}(Y) \setminus A)(X \mid y_1[Y] \mid \cdots \mid y_n[Y])$$

yield equivalent processes. The pattern process above is called a *receiving context*, from z toward $\uparrow$ in A, and is noted as $\mathcal{D}^A_{z,\uparrow}[X,Y]$. Its *associated environment* is the set of free variables of the process obtained after instantiating X and Y.

Definition 3.21 (Receiving Context) *Let A be a set of variables. Given two processes X and Y such that* $\mathrm{fn}(X) \subseteq A$, *a* receiving context $\mathcal{D}^A_{\gamma,\eta}[X,Y]$ *and its* associated environment $A_{\mathcal{D}^A_{\gamma,\eta}[X,Y]}$ *are respectively a process and a set of names defined as:*

if $\gamma,\eta = *,*$, or $\gamma,\eta = z,\uparrow$, then

$$\mathcal{D}^A_{\gamma,\eta}[X,Y] \;=\; (\boldsymbol{\nu}\,\mathrm{fn}(Y) \setminus A)(\ X \mid z_1[Y] \mid \cdots \mid z_n[Y]\ )$$

where the names $\vec{z}$ satisfy $\mathrm{fn}(\mathcal{D}^A_{\gamma,\eta}[X,Y]) \subseteq A \cup \vec{z}$.

Its associated environment $A_{\mathcal{D}^A_{\gamma,\eta}[X,Y]}$ is $A \cup \vec{z}$;

if $\gamma,\eta = *,z$, then

$$\mathcal{D}^A_{*,z}[X,Y] \;=\; (\boldsymbol{\nu}\,\mathrm{fn}(Y) \setminus A)(\ X \mid z[(\ z_1[Y] \mid \cdots \mid z_n[Y] \mid U\ )]\ )$$

where the names $\vec{z}$ and the process U satisfy $\mathrm{fn}(\mathcal{D}^A_{*,z}[X,Y]) \subseteq A \cup \vec{z}$.

Its associated environment $A_{\mathcal{D}^A_{*,z}[X,Y]}$ is $A \cup \vec{z}$;

if $\gamma,\eta = *,\uparrow$, then

$$\mathcal{D}^A_{*,\uparrow}[X,Y] \;=\; z[X \mid U] \mid z_1[Y] \mid \cdots \mid z_n[Y]$$

where the names $z,\vec{z}$ and the process U satisfy $\mathrm{fn}(\mathcal{D}^A_{*,\uparrow}[X,Y]) \subseteq A \cup \vec{z} \cup \{z\}$.

Its associated environment $A_{\mathcal{D}^A_{*,\uparrow}[X,Y]}$ is $A \cup \vec{z} \cup \{z\}$.

We write $\mathcal{D}^A[X,Y]$ when we quantify over all γ,η and abbreviate $A_{\mathcal{D}^A_{\gamma,\eta}[X,Y]}$ by $A_{\mathcal{D}}$ when no ambiguity arises.

3.5.3 Bisimulation for S-Seal

According to our analysis, we are now ready to define a bisimulation based equivalence relation over seal processes.

Definition 3.22 (Bisimilarity) *Let bisimilarity, denoted $\approx$, be the family of largest relations indexed by finite sets of names such that each $\approx_A$ is a symmetric relation over $\{P \mid \mathrm{fn}(P) \subseteq A\}$ and for all $P \approx_A Q$ the following conditions hold:*

0. $\mathrm{fn}(P) = \mathrm{fn}(Q)$;

1. if $A \vdash P \xrightarrow{\tau} P'$ then there exists a Q' such that $A \vdash Q \Rightarrow Q'$ and $P' \approx_A Q'$;

2. if $A \vdash P \xrightarrow{\ell} P'$ and $\ell \in \{\ \gamma[x^\eta(\vec{y})], \gamma[\overline{x}^\eta(\vec{y})], *[\overline{x}^\eta\{\!|\, y \,|\!\}], \gamma[x^\eta\{\!|\, S \,|\!\}], S^z, \gamma[x^\eta_z]\ \}$, then there exists a Q' such that $A \vdash Q \Rightarrow \xrightarrow{\ell} Q'$ and $P' \approx_{A \cup \mathrm{fn}(\ell)} Q'$;

3. *if* $A \vdash P \xrightarrow{R_z} P'$ *then there exist* Q', S *such that* $A \vdash Q \Rightarrow \xrightarrow{S_z} Q'$ *and for all admissible contexts* $\mathcal{D}^A[-,-]$ *it holds* $\mathcal{D}^A[P', R] \approx_{A_\mathcal{D}} \mathcal{D}^A[Q', S]$;

4. *if* $A \vdash P \xrightarrow{\gamma[\overline{x}^\eta\{R\}]} P'$ *then there exist* Q', S *such that* $A \vdash Q \Rightarrow \xrightarrow{\gamma[\overline{x}^\eta\{S\}]} Q'$ *and for all admissible contexts* $\mathcal{D}^A_{\gamma,\eta}[-,-]$ *it holds* $\mathcal{D}^A_{\gamma,\eta}[P', R] \approx_{A_\mathcal{D}} \mathcal{D}^A_{\gamma,\eta}[Q', S]$;

5. *for all substitutions* σ *such that* $\mathrm{dom}(\sigma) \subseteq A$ *and* $\mathrm{img}(\sigma) \subseteq A$, $P\sigma \approx_A Q\sigma$;

where a context $\mathcal{D}^A[-,-]$ *is* admissible *if both process substitutions* $\mathcal{D}^A[P', R]$ *and* $\mathcal{D}^A[Q', S]$ *are well-formed (i.e. no name capture arises).*

Intuitively, the first two cases of Definition 3.22 handle all first-order labels, as well as labels originating from receive actions: these do not deserve a special treatment because our early semantics implicitly tests all possible interactions. The cases 3. and 4. check processes that offer a seal body. Condition 5. ensures that $\approx$ is preserved by contexts where the hole in under a input prefix.

Standard reasoning, albeit here for an indexed relation, guarantees that the following proposition holds.

Proposition 3.23 *The relation* $\approx$ *exists uniquely and is an equivalence.*

Bisimilarity is preserved by all the constructors of the calculus, albeit working with an indexed relation requires a little care even to state the congruence property, to keep the A-indexing straight. Following [Sew00], we resort to the notion of *indexed congruence*.

Definition 3.24 (Indexed congruence) *Consider a family* $\mathcal{R}$ *of relations indexed by finite sets of names such that each* $\mathcal{R}_A$ *is a relation over* $\{P \mid \mathrm{fn}(P) \subseteq A\}$. $\mathcal{R}$ *is an* indexed congruence *if each* $\mathcal{R}_A$ *is an equivalence relation and the following hold:*

$$\frac{P\ \mathcal{R}_{A,\vec{y}}\ Q \quad x, \mathrm{fn}(\eta) \subseteq A}{x^\eta(\vec{y}).P\ \mathcal{R}_A\ x^\eta(\vec{y}).Q} \qquad \frac{P\ \mathcal{R}_A\ Q \quad \alpha \neq x^\eta(y), \mathrm{fn}(\alpha) \subseteq A}{\alpha.P\ \mathcal{R}_A\ \alpha.Q}$$

$$\frac{P\ \mathcal{R}_{A,\vec{y}}\ Q \quad x, \mathrm{fn}(\eta) \subseteq A}{!x^\eta(\vec{y}).P\ \mathcal{R}_A\ !x^\eta(\vec{y}).Q} \qquad \frac{P\ \mathcal{R}_A\ Q \quad \alpha \neq x^\eta(y), \mathrm{fn}(\alpha) \subseteq A}{!\alpha.P\ \mathcal{R}_A\ !\alpha.Q}\ .$$

$$\frac{P\ \mathcal{R}_A\ Q \quad x \in A}{x[P]\ \mathcal{R}_A\ x[Q]} \qquad \frac{P\ \mathcal{R}_A\ Q \quad \mathrm{fn}(R) \subseteq A}{\begin{array}{c} P \mid R\ \mathcal{R}_A\ Q \mid R \\ R \mid P\ \mathcal{R}_A\ R \mid Q \end{array}} \qquad \frac{P\ \mathcal{R}_{A,x}\ Q}{(\boldsymbol{\nu} x)P\ \mathcal{R}_A\ (\boldsymbol{\nu} x)Q}$$

Our aim here is to prove that bisimilarity is an indexed congruence. It is easier to prove separately that bisimilarity is preserved by the different constructs of the language. Before proceeding, we recall an elementary but useful up-to proof technique.

Lemma 3.25 *If* $\mathcal{R}$ *is a bisimulation up to structural congruence, then* $\mathcal{R} \subseteq \approx$.

Proof Let $\mathcal{S}_A = \{(P, Q) \mid (P_1, Q_1) \in \mathcal{R}_A, P_1 \equiv P \text{ and } Q_1 \equiv Q\}$, where A is a set of names. Clearly, for every A, it holds $\mathcal{R}_A \subseteq \mathcal{S}_A$, and $\bigcup_A \mathcal{R}_A \subseteq \bigcup_A \mathcal{S}_A$. Because of Lemma 3.13, a diagram chasing argument allows us to show that for every set of names A, the relation $\mathcal{S}_A$ is closed under the conditions to be a bisimulation, as required. □

Lemma 3.26 *If* $P \approx_A Q$, *then* $x[P] \approx_A x[Q]$ *for all* $x \in A$.

Proof Define $\mathcal{S}_A = \{(x[P], x[Q]) \mid P \approx_A Q, x \in A\} \cup \approx_A$ and let $\mathcal{S} = \bigcup_A \mathcal{S}_A$. We show that $\mathcal{S}$ is a bisimulation. Consider $(R, S) \in \mathcal{S}$. Then there exists A such that either $R \approx_A S$ holds, or $R = x[P]$ and $S = x[Q]$ for $x \in A$ and $P \approx_A Q$. In the former case the result follows straightforwardly. We next consider the possible transitions in the latter case. A simple inspection of the rules of LTS shows that if $x[P]$ emits a label, then one of the following cases holds:

- $A \vdash x[P] \xrightarrow{x[a]} x[P']$ for some P', where $a \in \{\overline{y}^{\uparrow}(\vec{z}), y^{\uparrow}(\vec{z}), y^{\uparrow}\{\!|Q|\!\}\}$ and $A \vdash P \xrightarrow{*[a]} P'$. We have $A \vdash Q \Rightarrow \xrightarrow{*[a]} Q'$ and $P' \approx_{A \cup \mathrm{fn}(a)} Q'$ because of the definition of bisimulation. Since $x \in A$, we have $A \vdash x[Q] \Rightarrow \xrightarrow{x[a]} x[Q']$, and $x[P']\ \mathcal{S}_{A \cup \mathrm{fn}(a)}\ x[Q']$ follows from the definition of $\mathcal{S}$. The case $A \vdash x[P] \xrightarrow{\tau} x[P']$ is similar.

- $A \vdash x[P] \xrightarrow{x[\overline{y}^{\uparrow}\{\!|R|\!\}]} x[P']$ for some P', where $A \vdash P \xrightarrow{*[\overline{y}^{\uparrow}\{\!|R|\!\}]} P'$. The definition of bisimulation ensures that (i) $A \vdash Q \Rightarrow \xrightarrow{*[\overline{y}^{\uparrow}\{\!|S|\!\}]} Q'$ for some Q', S; and (ii) $\mathcal{D}^A_{*,\uparrow}[P', R] \approx_{A_{\mathcal{D}}} \mathcal{D}^A_{*,\uparrow}[Q', S]$ for every possible such a pair of receiving contexts. From (i) we deduce that $A \vdash x[Q] \Rightarrow \xrightarrow{x[\overline{y}^{\uparrow}\{\!|S|\!\}]} x[Q']$. Now use (ii) and notice that every process that is a receiving context $\mathcal{D}^A_{x,\uparrow}[x[O], -]$ is also a receiving context $\mathcal{D}^A_{*,\uparrow}[O, -]$. Therefore (ii) implies that $\mathcal{D}^A_{x,\uparrow}[x[P'], R] \approx_{A_{\mathcal{D}}} \mathcal{D}^A_{x,\uparrow}[x[Q'], S]$, which, by definition of $\mathcal{S}$ implies
$$\mathcal{D}^A_{x,\uparrow}[x[P'], R]\ \mathcal{S}_{A_{\mathcal{D}}}\ \mathcal{D}^A_{x,\uparrow}[x[Q'], S]\ .$$

The last property we have to check for is closure under substitutions. Let σ be a substitution defined on A. Let $(R, S) \in \mathcal{S}$. Then there exists A such that either $R \approx_A S$ holds, or $R = x[P]$ and $S = x[Q]$ for $x \in A$ and $P \approx_A Q$. In the first case $P\sigma \approx_{A\sigma} Q\sigma$ holds because $\approx$ is closed under arbitrary substitution. Then $P\sigma\ \mathcal{S}_{A\sigma}\ Q\sigma$ follows from the construction of $\mathcal{S}$. In the latter, observe that $x\sigma \in A\sigma$ because $x \in A$, and $R\sigma \approx_{A\sigma} S\sigma$ holds because $\approx$ is closed under arbitrary substitution. The definition of substitution assures that $x\sigma[R\sigma] = (x[R])\sigma$ and $x\sigma[S\sigma] = (x[S])\sigma$. Then $(x[R])\sigma\ \mathcal{S}_{A\sigma}\ (x[S])\sigma$ follows because of the construction of $\mathcal{S}$. □

Lemma 3.27 *If $P \approx_A Q$, then $P \mid R \approx_A Q \mid R$ and $(\boldsymbol{\nu} x)P \approx_{A \setminus x} (\boldsymbol{\nu} x)Q$, for all $x \in A$, and for all R such that* $\mathrm{fn}(R) \subseteq A$.

Proof Define $\mathcal{S}_A = \{(\,(\boldsymbol{\nu}\vec{u})(P \mid R)\,, (\boldsymbol{\nu}\vec{u})(Q \mid R)\,) \;\big|\; P \approx_{A,\vec{u}} Q,\ (\mathrm{fn}(R) \setminus \vec{u}) \subseteq A\}$, and let $\mathcal{S} = \bigcup_A \mathcal{S}_A$. We show that $\mathcal{S}$ is a bisimulation up to structural congruence. Suppose $P \approx_A Q$. As a useful shorthand, we write $\bar{P}$ for $(\boldsymbol{\nu}\vec{u})(P \mid R)$, and $\bar{Q}$ for $(\boldsymbol{\nu}\vec{u})(Q \mid R)$.

We perform a case analysis on the transitions performed by $\bar{P}$. We detail all the most difficult and interesting situations (namely higher-order synchronisation and scope extrusion): the others follow accordingly.

- Suppose $A \vdash \bar{P} \xrightarrow{\gamma[x^{\eta}_z]} O_1$ because $A \cdot \vec{u} \vdash P \xrightarrow{\gamma[x^{\eta}\{\!|F|\!\}]} P'$ and $A \cdot \vec{u} \vdash R \xrightarrow{F_z} R'$. From the definition of bisimulation it follows that $A \cdot \vec{u} \vdash Q \Rightarrow \xrightarrow{\gamma[x^{\eta}\{\!|F|\!\}]} Q'$ and
$$P' \approx_{A \cup \vec{u} \cup \mathrm{fn}(F)} Q'\ . \tag{4}$$
Remark that $\mathrm{fn}(R') \subseteq A \cup \vec{u} \cup \mathrm{fn}(F_z) = A \cup \vec{u} \cup \mathrm{fn}(F)$ since $z \in A$. Thus, $A \vdash \bar{Q} \Rightarrow \xrightarrow{\gamma[x^{\eta}_z]} O_2$. The outcomes are respectively of the form $O_1 \equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus (A \cup \vec{u}))(P' \mid R')$ and $O_2 \equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus (A \cup \vec{u}))(Q' \mid R')$. Thus, from (4) we can conclude $O_1\ \mathcal{S}_A\ O_2$ because of the construction of $\mathcal{S}$.

- Suppose $A \vdash \bar{P} \xrightarrow{*[\overline{x}^{\eta}\{\!|F|\!\}]} O_1$ because $A \cdot \vec{u} \vdash P \xrightarrow{F_z} P'$ and $A \cdot \vec{u} \vdash R \xrightarrow{*[\overline{x}^{\eta}\{\!|z|\!\}]} R'$. From the definition of bisimulation it follows that $A \vdash Q \Rightarrow \xrightarrow{G_z} Q'$ and
$$\mathcal{D}^{A,\vec{u}}[P', F] \approx_{A_{\mathcal{D}}} \mathcal{D}^{A,\vec{u}}[Q', G]\ . \tag{5}$$
Thus, $A \vdash \bar{Q} \Rightarrow \xrightarrow{*[\overline{x}^{\eta}\{\!|G|\!\}]} O_2$. We must prove that for all admissible contexts $\mathcal{E}^A_{*,\eta}[-,-]$ it holds $\mathcal{E}^A_{*,\eta}[O_1, F]\ \mathcal{S}_{A_{\mathcal{E}}}\ \mathcal{E}^A_{*,\eta}[O_2, G]$. We have to consider different cases according to whether the names in $\vec{u}$ are extruded or not.

  Suppose $\mathrm{fn}(F) \subseteq A$ and $\mathrm{fn}(G) \subseteq A$. In this case, the outcomes are respectively of the form $O_1 \equiv (\boldsymbol{\nu}\vec{u})(P' \mid R')$ and $O_2 \equiv (\boldsymbol{\nu}\vec{u})(Q' \mid R')$ (no extrusion occurs). The structure of the context $\mathcal{E}$ depends on the localisation η of the mobility offer:

– If $\eta = *$, then (5) states that for all names $\vec{z}$ it holds:

$$(\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus (A, \vec{u}))(P' \mid z_1[F] \mid \cdots \mid z_n[F]) \\ \approx_{A_{\mathcal{D}}} (\boldsymbol{\nu}\,\mathrm{fn}(G) \setminus (A, \vec{u}))(Q' \mid z_1[G] \mid \cdots \mid z_n[G])$$

Since $\mathrm{fn}(F),\ \mathrm{fn}(G) \subseteq A$, then the leading restrictions are empty. Combining this with the observation that $\approx_{A_{\mathcal{D}}} \subseteq \mathcal{S}_{A_{\mathcal{D}}}$, we obtain:

$$P' \mid z_1[F] \mid \cdots \mid z_n[F]\ \mathcal{S}_{A_{\mathcal{D}}}\ Q' \mid z_1[G] \mid \cdots \mid z_n[G] \tag{6}$$

Remark that $A_{\mathcal{D}} = A \cup \vec{z} \cup \vec{u}$ and recall that $A \cap \vec{u} = \varnothing$. Since $\mathcal{S}$ is closed under parallel composition and restriction we can write:

$$(\boldsymbol{\nu}\vec{u})(P' \mid R' \mid z_1[F] \mid \cdots \mid z_n[F])\ \mathcal{S}_{A,\vec{z}}\ (\boldsymbol{\nu}\vec{u})(Q' \mid R' \mid z_1[G] \mid \cdots \mid z_n[G])$$

Since (5) holds for all admissible receiving contexts, in particular it holds for those receiving contexts of form (6) where $\vec{z} \cap \vec{u} = \emptyset$. For these particular receiving contexts we can rearrange both sides of the equation (6) by structural congruence, obtaining:

$$(\boldsymbol{\nu}\vec{u})(P' \mid R') \mid z_1[F] \mid \cdots \mid z_n[F]\ \mathcal{S}_{A,\vec{z}}\ (\boldsymbol{\nu}\vec{u})(Q' \mid R') \mid z_1[G] \mid \cdots \mid z_n[G]$$

that is $\mathcal{E}^{A}_{*,*}[O_1, F]\ \mathcal{S}_{A,\vec{z}}\ \mathcal{E}^{A}_{*,*}[O_2, G]$. The result $\mathcal{E}^{A,\vec{u}}_{*,*}[O_1, F]\ \mathcal{S}_{A_{\mathcal{E}}}\ \mathcal{E}^{A,\vec{u}}_{*,*}[O_2, G]$ follows because the receiving context is generated by $\gamma = *, \eta = *$, and so $A_{\mathcal{E}} = A, \vec{z}$.

– The case $\eta = z$ is analogous to the previous one.

– If $\eta = \uparrow$ then from (5), case $\gamma, \eta = *, \uparrow$ we can deduce that for all U' it holds

$$z[P' \mid R' \mid U'] \mid z_1[F] \mid \cdots \mid z_n[F]\ \approx_{A \cup \vec{z} \cup \vec{u}}\ z[Q' \mid R' \mid U'] \mid z_1[G] \mid \cdots \mid z_n[G]$$

(it suffices to take $U = U' \mid R'$). The construction of $\mathcal{S}$ then guarantees that

$$(\boldsymbol{\nu}\vec{u})z[P' \mid R' \mid U'] \mid z_1[F] \mid \cdots \mid z_n[F] \\ \approx_{A \cup \vec{z}}\ (\boldsymbol{\nu}\vec{u})z[Q' \mid R' \mid U'] \mid z_1[G] \mid \cdots \mid z_n[G]$$

and up to structural congruence we obtain

$$z[(\boldsymbol{\nu}\vec{u})(P' \mid R' \mid U')] \mid z_1[F] \mid \cdots \mid z_n[F] \\ \approx_{A \cup \vec{z}}\ z[(\boldsymbol{\nu}\vec{u})(Q' \mid R' \mid U')] \mid z_1[G] \mid \cdots \mid z_n[G]\ .$$

This can be written using receiving contexts as $\mathcal{E}^{A}_{*,\uparrow}[O_1, F]\ \mathcal{S}_{A_{\mathcal{E}}}\ \mathcal{E}^{A}_{*,\uparrow}[O_2, G]$, as required.

Suppose now that $\mathrm{fn}(F) \not\subseteq A$ and $\mathrm{fn}(G) \subseteq A$. This means that in $\bar{P}$ some of the names in $\vec{u}$ are extruded by the mobility offer. condition As before, we perform a case analysis on the localisation η. We detail only the case $\eta = *$: the others are similar. If $\eta = *$, then the outcomes are respectively of the form $O_1 \equiv (\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(F))(P' \mid R')$ and $O_2 \equiv (\boldsymbol{\nu}\vec{u})(Q' \mid R')$. We spell out (5) for $\gamma, \eta = *, *$. It holds $P' \mid z_1[F] \mid \cdots \mid z_n[F] \approx_{A \cup \vec{u} \cup \vec{z}} Q' \mid z_1[G] \mid \cdots \mid z_n[G]$. This implies that for all $\vec{z}$ it holds

$$(\boldsymbol{\nu}\vec{u})(P' \mid R' \mid z_1[F] \mid \cdots \mid z_n[F])\ \mathcal{S}_{A \cup \vec{z}}\ (\boldsymbol{\nu}\vec{u})(Q' \mid R' \mid z_1[G] \mid \cdots \mid z_n[G])$$

Now notice that since $A \vdash \bar{P} = (\boldsymbol{\nu}\vec{u})(P \mid R) \xrightarrow{*[\overline{x}^{\eta}\{F\}]}$ then F occurs in $\bar{P}$ and therefore $\mathrm{fn}(F) \subseteq A \cup \vec{u}$. From the last formula we obtain $\mathrm{fn}(F) \setminus A \subseteq \vec{u}$ from which it is possible to deduce that $\vec{u} = (\mathrm{fn}(F) \setminus A) \cup (\vec{u} \setminus \mathrm{fn}(F))$. Therefore we have:

$$(\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus A)(\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(F))(P' \mid R' \mid z_1[F] \mid \cdots \mid z_n[F]\ ) \\ \mathcal{S}_{A \cup \vec{z}}\ (\boldsymbol{\nu}\vec{u})(Q' \mid R' \mid z_1[G] \mid \cdots \mid z_n[G]\ )$$

Since the equation above holds for all $\vec{z}$, then in particular it holds for those $\vec{z}$ such that $\vec{u} \cap \vec{z} = \varnothing$. In this case then we can extrude the z_i's from the scope of the inner restriction in the left hand-side of the above equation above. Furthermore recall that $A \cap \vec{u} = \varnothing$ and $\mathrm{fn}(G) \subseteq A$, which implies $\mathrm{fn}(G) \cap \vec{u} = \varnothing$. Then we can extrude these seals in the right hand-side. Up to structural congruence we have:

$$(\boldsymbol{\nu}\, \mathrm{fn}(F) \setminus A)(\ ((\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(F))(P' \mid R')) \mid z_1[F] \mid \cdots \mid z_n[F]\ ) \\ \mathcal{S}_{A \cup \vec{z}} \quad (\boldsymbol{\nu}\vec{u})(Q' \mid R') \mid z_1[G] \mid \cdots \mid z_n[G]$$

that is $\mathcal{E}^A_{*,*}[O_1, F]\ \mathcal{S}_{A_\mathcal{E}}\ \mathcal{E}^A_{*,*}[O_2, G]$.

Finally, the remaining two cases, that is $\mathrm{fn}(F) \subseteq A$, $\mathrm{fn}(G) \not\subseteq A$ and $\mathrm{fn}(F) \not\subseteq A$, $\mathrm{fn}(G) \not\subseteq A$, are similar to the previous one.

- Suppose $A \vdash \bar{P} \xrightarrow{\tau} O_1$, where $A \cdot \vec{u} \vdash P \xrightarrow{F_z} P'$ and $A \cdot \vec{u} \vdash R \xrightarrow{F^z} R_1$.
By Lemma 3.8 either $R_1 \equiv (\boldsymbol{\nu}\vec{w})(R_3 \mid z_1[F] \mid \cdots \mid z_n[F])$ or $R_1 \equiv (\boldsymbol{\nu}\vec{w})(R_3 \mid z[R_4 \mid z_1[F] \mid \cdots \mid z_n[F]])$ hold. From the definition of bisimulation it follows that $A \cdot \vec{u} \vdash Q \Rightarrow \xrightarrow{G_z} Q'$, and for all $\mathcal{D}^{A,\vec{u}}[-,-]$ admissible it holds

$$\mathcal{D}^{A,\vec{u}}[P', F] \approx_{A_\mathcal{D}} \mathcal{D}^{A,\vec{u}}[Q', G]. \tag{7}$$

The early nature of the LTS allows process R to receive G as well: $A \cdot \vec{u} \vdash R \xrightarrow{G^z} R_2$, with R_2 either in the form $R_2 \equiv (\boldsymbol{\nu}\vec{w})(R_3 \mid z_1[G] \mid \cdots \mid z_n[G])$, or $R_2 \equiv (\boldsymbol{\nu}\vec{w})(R_3 \mid z[R_4 \mid z_1[G] \mid \cdots \mid z_n[G]])$. Thus, $A \vdash \bar{Q} \Rightarrow \xrightarrow{\tau} O_2$.

Consider the first case.

Before going on, we point out that as $A \cdot \vec{u} \vdash R \xrightarrow{F^z} \equiv (\boldsymbol{\nu}\vec{w})(R_3 \mid z_1[F] \mid \cdots \mid z_n[F])$, the condition $\vec{z} \subseteq A \cup \vec{u} \cup \vec{w}$ must hold. Also, $\mathrm{fn}(R_3) \subseteq A \cup \vec{u} \cup \vec{w}$. To see why, remember that R_3 is formed by the parallel composition of two residuals: the continuation of the receiving action, and the continuation of the send action. The free variables of the first process must be contained in $A \cup \vec{u} \cup \vec{z} \cup \{x\}$ where x is the channel on which the move takes place; those of the second process are instead contained in $A \cup \vec{u} \cup \{z\} \cup \{x\}$. Thus $\mathrm{fn}(R_3) \subseteq A \cup \vec{u} \cup \vec{z} \cup \{z\} \cup \{x\}$. But $\{z\} \cup \{x\} \subseteq A \cup \vec{u}$ as they obviously occur free in the sending process (cf. rule (SND)). So, $\vec{z} \subseteq A \cup \vec{w}$ and we conclude that $\mathrm{fn}(R_3) \subseteq A \cup \vec{u} \cup \vec{w}$.

The outcomes are respectively of the form

$$\begin{aligned} O_1 &\equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\, \mathrm{fn}(F) \setminus (A \cup \vec{u}))(P' \mid (\boldsymbol{\nu}\vec{w})(R_3 \mid z_1[F] \mid \cdots \mid z_n[F]\ )) \\ O_2 &\equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\, \mathrm{fn}(G) \setminus (A \cup \vec{u}))(Q' \mid (\boldsymbol{\nu}\vec{w})(R_3 \mid z_1[G] \mid \cdots \mid z_n[G]\ ))\ . \end{aligned}$$

Since $\mathrm{fn}(R_3) \subseteq A \cup \vec{u} \cup \vec{w}$ we have $\mathrm{fn}(R_3) \cap (\mathrm{fn}(F) \setminus (A \cup \vec{u})) = \emptyset$, and similarly $\mathrm{fn}(R_3) \cap (\mathrm{fn}(G) \setminus (A \cup \vec{u})) = \emptyset$. We can then extrude R_3 from the scope of the inner restriction as follows:

$$\begin{aligned} O_1 &\equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\vec{w})(R_3 \mid (\boldsymbol{\nu}\, \mathrm{fn}(F) \setminus (A \cup \vec{u}))(P' \mid z_1[F] \mid \cdots \mid z_n[F]\ )) \\ O_2 &\equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\vec{w})(R_3 \mid (\boldsymbol{\nu}\, \mathrm{fn}(G) \setminus (A \cup \vec{u}))(Q' \mid z_1[G] \mid \cdots \mid z_n[G]\ ))\ . \end{aligned}$$

The case $\gamma, \eta = *, *$ of (7) implies that

$$(\boldsymbol{\nu}\, \mathrm{fn}(F) \setminus (A \cup \vec{u}))(P' \mid z_1[F] \mid \cdots z_n[F]\ ) \\ \approx_{A \cup \vec{z} \cup \vec{u}} \ (\boldsymbol{\nu}\, \mathrm{fn}(G) \setminus (A \cup \vec{u}))(Q' \mid z_1[G] \mid \cdots z_n[G]\ ) \tag{8}$$

The construction of $\mathcal{S}$ allows us to derive $O_1\ \mathcal{S}_{(A \cup \vec{u} \cup \vec{z}) \setminus (\vec{u} \cup \vec{w})}\ O_2$ from 8. As $\vec{z} \subseteq A \cup \vec{u} \cup \vec{w}$, $(A \cup \vec{u} \cup \vec{z}) \setminus (\vec{u} \cup \vec{w}) = A$, and we conclude $O_1\ \mathcal{S}_A\ O_2$, as required.

The latter case follows accordingly. □

Theorem 3.28 *Bisimilarity is an indexed congruence.*

Proof It is straightforward to prove that $\approx$ is preserved by prefix and replication. The other cases are covered by Lemma 3.26 and Lemma 3.27. The proof of the symmetric case of congruence with respect to parallel composition is analogous to the proof of Lemma 3.27. □

Lemma 3.29 (Injective substitution — bisimulation) *If $P \approx_A Q$ and $f : A \to B$ is injective then $fP \approx_B fQ$.*

Proof We check

$$\mathcal{R}_B = \{\ (fP, fQ) \mid f : A \to_{\text{inj}} B \text{ and } P \approx_A Q\ \}$$

is a bisimulation.

Suppose $B \vdash fP \xrightarrow{\ell'} P'_1$. We perform a case analysis on the label ℓ'.

Case $\ell = \tau$. By Corollary 3.12, there exists P' such that $A \vdash P \xrightarrow{\tau} P'$ and $P'_1 = fP'$. By bisimulation, there exists Q' such that $A \vdash Q \Rightarrow Q'$ and $P' \approx_A Q'$. By Corollary 3.12, $B \vdash fQ \Rightarrow fQ'$. Finally $fP' \mathcal{R}_B fQ'$.

Case $\ell \in \{\ \gamma[x^\eta(\vec{y})], \gamma[\overline{x}^\eta(\vec{y})], *[\overline{x}^\eta\{y\}], \gamma[x^\eta\{S\}], S^z, \gamma[x^\eta_z]\ \}$. By Lemma 3.11, there exist $\ell, P', H, I, g : I \to_{\text{bij}} B', h : H \to_{\text{inj}} (B \setminus \text{img}(f))$: such that $B' \cap B = \emptyset$, $H \cup I = \text{fn}(\ell) \setminus A$, $H \cap I = \emptyset$, and

$$A \vdash P \xrightarrow{\ell} P' \qquad P'_1 = (f + g + h)P' \qquad \ell' = (f + g + h)\ell\ .$$

By bisimulation, there exists Q' such that $A \vdash Q \Rightarrow \xrightarrow{l} Q'$ and $P' \approx_{A \cup \text{fn}(\ell)} Q'$.

Then, for f, g, h as above, we have $B \vdash fQ \Rightarrow \xrightarrow{(f+g+h)\ell} (f + g + h)Q'$. Finally, $(f + g + h)P'\ \mathcal{R}_B\ (f + g + h)Q'$ follows from the construction of $\mathcal{R}$.

Case $\ell = R_z$. Then, $B \vdash fP \xrightarrow{R_z} P'_1$. By Lemma 3.11, there exist ℓ, P' and $I, g : I \to_{\text{bij}} B'$ with $I \cap A = \emptyset$ and $B' \cap B = \emptyset$, such that

$$A \vdash P \xrightarrow{\ell} P' \qquad (f + g)\ell = R_z \qquad (f + g)P' = P'_1\ .$$

Observe that ℓ must be of the form $R'_{z'}$ for some R', z' such that $(f + g)R' = R$ and $fz' = z$. Then by bisimulation there exists a Q' such that $A \vdash Q \xrightarrow{S'_{z'}} Q'$ and for all admissible contexts $\mathcal{D}^A[-,-]$ it holds $\mathcal{D}^A[P', R'] \approx_{A_\mathcal{D}} \mathcal{D}^A[Q', S']$. By Lemma 3.9 we have $B \vdash fQ \xrightarrow{S_z} Q'_1$, where $S = (f + g)S'$ and $Q'_1 = (f + g)Q'$. To conclude that for all admissible contexts $\mathcal{E}^B[-,-]$ it holds $\mathcal{E}^B[P'_1, R]\ \mathcal{R}_{A_\mathcal{E}}\ \mathcal{E}^B[Q'_1, S]$, observe that every context $\mathcal{E}^B[-,-]$ can be written as $(f + g)\mathcal{D}^A[-,-]$ for a receiving context $\mathcal{D}^A[-,-]$ because $(f + g)$ is injective and because $g(\text{fn}(R') \setminus A) = \text{fn}((f + g)R) \setminus B$.

Case $\ell = \gamma[\overline{x}^\eta\{R\}]$. Similar to $\ell = R_z$.

It remains to prove that $\mathcal{R}$ is closed under substitution. For that, suppose $fP\ \mathcal{R}_B\ fQ$ and let $\sigma : B \to C$ be a substitution. We want to show that $(fP)\sigma\ \mathcal{R}_{B\sigma}\ (fQ)\sigma$. By definition of $\mathcal{R}$, $P\ \mathcal{R}_A\ Q$ for some A, $f : A \to_{\text{inj}} B$. The composition of σ and f is a substitution $\sigma f : A \to C$. As $\approx_A$ is closed under substitution, $P(\sigma f)\ \mathcal{R}_{A(\sigma f)}\ Q(\sigma f)$, that is, $(fP)\sigma\ \mathcal{R}_{(fA)\sigma}\ (fQ)\sigma$. Let $i : (fA)\sigma \to B\sigma$ be the injection of the set $(fA)\sigma$ into $B\sigma$ (the injection exists because $fA \subseteq B$). Then

$$(fP)\sigma = i((fP)\sigma)\ \mathcal{R}_{B\sigma}\ i((fQ)\sigma) = (fQ)\sigma$$

as required. □

The soundness of our bisimulation based proof method is an easy consequence of the two Lemmas above.

Theorem 3.30 (Soundness) *Bisimilarity is sound with respect to barbed congruence: if $P \approx_A Q$ for some A, then $P \cong Q$.*

Proof Suppose $P \approx_A Q$ for some A. By Definition 3.22.1 bisimilarity $\approx$ is reduction closed. bisimilarity is preserved by arbitrary contexts because it is an indexed congruence (Lemma 3.28), and because it is preserved by injective renaming (Lemma 3.29). It remains to prove that if $P \downarrow n$ then $Q \Downarrow n$. For this, note that the first part of condition 3 in Definition 3.22 ensures that whenever $A \vdash P \xrightarrow{F_n}$ for some process F, $A \vdash Q \Rightarrow \xrightarrow{G_n}$ for some process G. Therefore, the thesis follows by Lemma 3.15. □

3.6 On completeness

Bisimilarity is not complete with respect to barbed congruence, for several reasons. First of all, there is a technical cause: bisimilarity is a delay bisimilarity, as such the weak transitions $\Rightarrow \xrightarrow{\ell}$ do not allow τ moves after a visible action. We keep the delay formulation of the bisimulation because it is easier to apply in practice. Also, the delay formulation captures the intuition that a process that performs an higher-order action needs the contribution of a receiving context before continuing.

Reasons also lurk in the design of the calculus, both in the communication and in the mobility subsystems. Seal communication is an extension of the π-calculus. With an LTS analogous to ours, the matching operator is necessary to obtain completeness for the π-calculus, thus the same operator might be required in Seal. Seal mobility raises the most interesting issue. A direct consequence of Seal's model of computation is that a seal can not detect if the environment moves it or not. By exploiting this feature, we construct the two processes:

$$P \;=\; (\boldsymbol{\nu} x)(\overline{x}^*\{\!|y|\!\} \mid x^*\{\!|y|\!\}) \quad \text{and} \quad Q \;=\; \mathbf{0}\;.$$

Let $A \supseteq \mathrm{fn}(P)$. Then for all processes S, it holds $A \vdash P \xrightarrow{S^y}$, while Q does not emit anything. Thus $P \not\approx Q$. At the same time no context can separate P from Q: even $C[-] \;=\; - \mid y[S]$, that is offering a seal to be moved, does not help, because the context cannot determine if the seal y has been moved or not. In technical terms, this example shows that no context can reliably test for a S^z action. As a consequence, reduction barbed congruence is not insensitive to replacing natural barbs with barbs inherited from the S^z action.

This observation might suggest that for the same reasons it should be difficult for a context to distinguish the label $\gamma[\overline{x}^\eta\{\!|y|\!\}]$ from $\gamma[\overline{x}^\eta\{\!|S|\!\}]$, and $*[x^\eta\{\!|S|\!\}]$ from $*[x^\eta_z]$. It is not the case: a careful use of fresh names leads to the definition of contexts that differentiate these actions, as seen in Section 3.5.1.

3.7 Algebraic theory

The following equation, suggestively named *the perfect sandbox equation*, holds in S-Seal:

Lemma 3.31 (Perfect sandbox) *For all processes P and Q such that* $\mathrm{fn}(P) = \mathrm{fn}(Q)$*, it holds:*

$$(\boldsymbol{\nu} n)n[!P] \;\cong\; (\boldsymbol{\nu} n)n[!Q]\;.$$

Proof Let $\mathcal{S}_A = \{\, ((\boldsymbol{\nu} n)n[!P], (\boldsymbol{\nu} n)n[!Q]) \mid \mathrm{fn}(P) = \mathrm{fn}(Q) \subseteq A \,\}^=$, where $\mathcal{R}^=$ denotes the symmetric closure of $\mathcal{R}$. Let $\mathcal{S} = \cup_A \mathcal{S}_A$. The process $(\boldsymbol{\nu} n)n[!P]$ can not emit any label: if $n[!P]$ performs a visible action, then it is of the form $n[a]$ or P'_n, where $P \Rightarrow P'$. In both cases n belongs to the free names of the label, and it can not be observed under $(\boldsymbol{\nu} n)$. The same holds for $(\boldsymbol{\nu} n)n[!Q]$. So $\mathcal{S}$ is a bisimulation. The lemma follows because $\approx \subseteq \cong$. □

This equation formalizes how restriction on a seal's name prevents it from interacting with its environment (the replication guarantees that the *lhs* and the *rhs* of the equation always have the same set of free names). This justifies processes like

$$\begin{array}{lcl} (\mathsf{halt}\ x.P) & \stackrel{def}{=} & (\boldsymbol{\nu} n)(x\ \mathsf{be}\ n.(P \mid x^*(z).n\ \mathsf{be}\ x)) \\ (\mathsf{restart}\ x) & \stackrel{def}{=} & \overline{x}^*(x).P \end{array}$$

where be is the renaming operator defined as

$$(n\ \mathsf{be}\ m).P \;\stackrel{def}{=}\; (\boldsymbol{\nu} x)(\overline{x}^*\{\!|n|\!\} \mid x^*\{\!|m|\!\}.P)\;.$$

These operators do not modify the process running within the seal, rather they affect its communication capabilities. In fact,

$$\mathsf{halt}\ x.P \mid x[Q] \ \twoheadrightarrow\ P \mid (\boldsymbol{\nu} n)(n[Q] \mid x^*().n\ \mathsf{be}\ x)$$

and the previously active seal x is prevented from interacting with the environment because it has been renamed into n, a secret name (the context can still observe the free names of Q, but this is of little use). The same seal can be freed by executing $\mathsf{restart}\ x$:

$$(\boldsymbol{\nu} n)(n[Q] \mid x^*().n\ \mathsf{be}\ x) \mid \mathsf{restart}\ x \ \twoheadrightarrow\ x[Q]\ .$$

Thus, the perfect sandbox equation is an useful aid to the programmer.

When we defined Seal our goal was to be able to observe (and control) agent mobility, since we were interested in modeling programs roaming over untrusted networks. The analysis of the Seal model of computation presented in this section validates this design choice: $(\boldsymbol{\nu} n)n[Q]$ has no interaction whatsoever with the environment, except for possibly originating scope extrusions. In contrast, in Mobile Ambients, terms like $(\boldsymbol{\nu} n)n[P]$, while under the condition that n does not occur free in P cannot be distinguished from the inactive process, can still enter another ambient that runs in parallel or exit the ambient they reside in. In other words, restricting the name of the enclosing ambients, does not prevent ambient mobility.

In the proposition below, we enumerate a collection of algebraic laws that describe the use of restriction to avoid interference in mobility and communication. The same equations hold if seals are duplicated, and vector of names transmitted.

Proposition 3.32 *For all processes P, Q, and R, if $\mathrm{fn}(lhs) = \mathrm{fn}(rhs)$ then:*

1. $(\boldsymbol{\nu} m, x)(m[P] \mid \overline{x}^*\{m\}.Q \mid x^*\{n\}.R) \ \approx\ (\boldsymbol{\nu} m, x)(n[P] \mid Q \mid R)$
2. $(\boldsymbol{\nu} m, x, o)(m[P] \mid \overline{x}^o\{m\}.Q \mid o[x^\uparrow\{n\}.R]) \ \approx\ (\boldsymbol{\nu} m, x, o)(Q \mid o[n[P] \mid R])$
3. $(\boldsymbol{\nu} x, o)(x^o\{n\}.Q \mid o[m[P] \mid \overline{x}^\uparrow\{m\}.R]) \ \approx\ (\boldsymbol{\nu} x, o)(n[P] \mid Q \mid o[R])$
4. $(\boldsymbol{\nu} x)(\overline{x}^*(v).Q \mid x^*(u).R) \ \approx\ (\boldsymbol{\nu} x)(Q \mid R[v/u])$
5. $(\boldsymbol{\nu} x, o)(\overline{x}^o(v).Q \mid o[x^\uparrow(u).R]) \ \approx\ (\boldsymbol{\nu} x, o)(Q \mid o[R[v/u]])$
6. $(\boldsymbol{\nu} x, o)(x^o(u).R \mid o[\overline{x}^\uparrow(v).R]) \ \approx\ (\boldsymbol{\nu} x, o)(Q[v/u] \mid o[Q])$

Proof The proofs of the above laws are by exhibiting the appropriate bisimulation. In all cases the bisimulation has a similar form:

$$\mathcal{S} = \{(lhs, rhs), (rhs, lhs)\} \cup \mathcal{I}$$

where *lhs* and *rhs* denote respectively the left hand side and the right hand side of the equation, and $\mathcal{I}$ is the identical relation over processes. □

Limits of the proof method In the other dialects of the Seal Calculus, the perfect sandbox equation does not hold. But in all dialects, a safe way to isolate a seal consists in enclosing it in a secret sandbox.

Lemma 3.33 (Universal sandbox) *For all processes P, Q such that $\mathrm{fn}(P) = \mathrm{fn}(Q)$, it holds:*

$$(\boldsymbol{\nu} a)a[n[!P]] \ \cong\ (\boldsymbol{\nu} a)a[n[!Q]]\ .$$

In S-Seal, it is easy to prove the universal sandbox equation using our proof method.

We conjecture that a stronger equation holds.

Conjecture 3.34 (Sandbox) *For all processes P, Q such that $\mathrm{fn}(P) = \mathrm{fn}(Q)$, it holds:*

$$a[n[!P]] \ \cong\ a[n[!Q]]\ .$$

This equation captures the key idea that all interactions a seal can perform are under control of its enclosing seal. Unfortunately, it is far from easy to guess the smallest bisimulation relating $a[n[!P]]$ and $a[n[!Q]]$. In fact, the seal a can be duplicated by an arbitrary context in almost arbitrary locations (receiving contexts can have very complex shapes, especially after few iterations), and the bisimulation candidate quickly becomes intractable.

3.8 Related works

Our approach was influenced by the work of Merro and Hennessy [MH02] on the behavioural theory of a dialect of Mobile Ambients. Their work starts from Sangiorgi's observation in [San01] that the model of computation offered by Mobile Ambients presents several difficulties. The goal of [MH02] is thus to modify Ambients to endow them with an equational theory that is (*i*) richer, (*ii*) reasonable, (*iii*) adequate, and (*iv*) practical. What do these four properties mean? Richer: that it proves equivalences more interesting than the simple structural congruence relation; reasonable: that it is a contextual equivalence that preserves reductions and some simple observational property; adequate: that it is invariant to different choices of observations (technically, of *barbs*); practical: that it can be expressed in terms of bisimulation, whose coinductive nature ensures the existence of powerful proof techniques.

Levi and Sangiorgi [LS00] opened the way by extending Ambients with coactions. In their system a reduction takes place only if an action synchronizes with a corresponding coaction. This yields a more satisfactory equational theory. However the contextual equivalence does not enjoy the last two properties. Merro and Hennessy go further by also adding to Ambients *passwords*: an action and the corresponding coaction synchronize only if they possess the same password (which incidentally is quite similar to channel communication). Then, Merro and Hennessy define a bisimulation-based equivalence that is invariant for a large choice of observations. Their extension enjoys the four above mentioned properties.

It is interesting to notice that all these modifications bring Ambients ever closer to the Seal calculus. [LS00] requires mobility to be the consequence of a process synchronization. [MH02] simply requires that the mobility takes place on (what we can assimilate to) channels. (Merro and Hennessy also modify Levi and Sangiorgi's calculus so that the coaction of an out must be placed exactly as a receive action in Seal.) The very last step that distinguishes these Ambient variations from Seal is that Seal uses objective mobility—the agent is sent by the surrounding environment—while in Ambient-based calculi mobility is subjective—the agent sends itself—(as an aside, note that objective moves have also been added to Ambients by Cardelli, Ghelli and Gordon [CGG00] in order to have more refined typings).

Adapting Merro and Hennessy techniques to Seal turned to be quite difficult, because of the three parties synchronisations, and in general because of the subtle rules governing Seals.

Other related works, include the higher-order LTSs for Mobile Ambients that can be found in [CG00, CG99a, Vig99, FMT01]. But we are not aware of any form of bisimilarity defined using these LTSs. A simple first-order LTS for MA without restriction is proposed by Sangiorgi in [San01]. Using this LTS the author defines an *intensional* bisimilarity for MA that separates terms on the basis of their internal structure. Merro and Zappa Nardelli [MZ03] define a new LTS for Mobile Ambients and associated bisimulation that coincides with the natural contextual equivalence, thus solving the original problem of providing a behavioural theory for Mobile Ambients. In the extended version of [BCMS02] the authors present a sound and complete coinductive characterization of a contextual equivalence for the New Boxed Ambients; an analogous result is presented for the calculus of Mobile Resources in [GHS02]. Other forms of labelled bisimilarity for higher-order distributed calculi, such as Distributed π-calculus [RH98], Safe Dπ [HRY03], and Nomadic Pict [US01] can be found in [HMR03, BCMS02, HRY03, US01], but only the first three prove labelled characterisations of a contextually defined notion of equivalence.

4 Types for Mobility

A goal of the Seal project was to integrate security from the very beginning into the design of the language. Although the calculus allows to define secure systems, security properties cannot, and should not, rely solely on the language dynamic semantics. A well accepted way to provide strong security guarantees is to statically restrict the behavior of processes by a type system. There are many advantages to a type-based approach. By restricting the possible interactions types restrict the search space for possible insecure behaviors. Furthermore types have a descriptive role and as such they constitute an approximation of the behavior of processes. Finally, types can be used in a prescriptive way by giving the programmer the possibility to ban some dangerous interactions by appropriate type definitions.

In this section we develop a type system for each variant of the Seal Calculus. We distinguish two variants (rather than the four dialects), namely located vs. shared channels (the *e*-condition does impact

the type system). Each type system statically ensures the property that channels are used as declared, that is, that they transport messages with the right arity and types. More importantly, each type system provides a description of seals. In particular, types characterize the possible interactions that a seal can engage in with its environment, thus resulting in a partial specification of seals. It is noteworthy that since Seal mobility takes place on channels, then by typing mobility channels we type mobility itself. This allows us to use types to specify, constrain, and enforce the mobility properties and behavior of a seal. This can be framed in a more general use of such type systems, that is to statically define and enforce security properties, for example by restricting the use of private names. Finally, for the located channels version we will show that these types can be used to specify fine-grained resource access control policies.

4.1 Interfaces

The basic idea is to type seals by describing interactions a seal may have with the surrounding environment. We know that in the shared variant such interactions take place over the channels that cross the seal boundary while in the localized variant they may take place on local channels, as well. Thus these channels partially specify the interaction protocol of a seal. Keeping track of the set of upward exchanges (that is, exchanges with the parent) that a seal may establish can be statically achieved by keeping track of the channels that would be employed: this gives rise to a notion of interface of an a seal as a set of *upward channels*, i.e. channels that can synchronize with a process in the parent.

Actually not all the upward channels are interesting for describing the interaction protocol of a seal. Those the seal is listening on suffice:

> *The* interface *of a seal is the set of upward channels that the process local to the seal may be listening on, with the type expected from interactions on them.*

To see why such a definition is sensible one may consider the example of a networked host. For the outer world the interface of the computer –the description of how it is possible to interact with it– is given by the set of ports on which a dæmon is listening, together with the type of messages (i.e., the protocol) that they support. So the interface can be described as a set of pairs (*port*:*type*). For example in our system a typical ftp and mail server would be characterized by an interface of the following form [21:*ftp*; 23:*telnet*; 79:*finger*; 110:*pop3*; 143:*imap*; ...].

A different justification for our definition of interface comes from the analogy with object-oriented programming. If we consider a seal as an object, then a process contained in it that listens on an upward channel m is very similar to a method associated to the message m. In other words sending message m with argument v to an object x can be modeled in Seal by the action $\overline{m}^x(v)$, which would type check in our type system only if the pair $m{:}M$ is present in the type of the seal x (with M the type of v). The set of all possible such pairs constitutes the interface of the seal.

Hence, we consider interfaces such as, say, $[x_1{:}\star; x_2{:}\texttt{Ch}\ T; x_3{:}A; x_4{:}\texttt{Id}\ A]$ which characterizes seals that may: (*i*) synchronize with an input operation on the upward channel x_1 ($\star$ is the type of the empty tuple which is used to denote synchronization signals); (*ii*) read over x_2 a channel name of type $\texttt{Ch}\ T$ (the name of a channel that transports items of type T); (*iii*) receive over x_3 a seal whose interface is A; (*iv*) read over x_4 the name of a seal whose interface is A. It is important to stress the difference between what can be transmitted over x_3 and x_4, respectively seals and seal names: the former requires mobility primitives, the latter communication primitives.

4.2 Type Syntax

Both variants of the Seal Calculus share the following syntax for the types (where $n \geq 1$):

Exchange Types

T	::=	$M_1 \times \cdots \times M_n$	messages
	$\mid$	A	agents

Annotations

Z	::=	$\curvearrowright$	mobile
	$\mid$	$\veebar$	immobile

Message Types

M	::=	$\star$	empty
	$\mid$	$\mathtt{Ch}\ T$	channel names
	$\mid$	$\mathtt{Id}^Z A$	agent names

Interfaces

A	::=	$[x_1{:}T_1; \cdots ; x_n{:}T_n]$	agents

There are four syntactic categories in the type syntax, T, M, Z, and A respectively denoting exchange types, message types, mobility annotations and agent interfaces. We use the term *agent* and *seal* interchangeably. In the previous section we informally described three of them, omitting annotations. We now consider them all:

T: Types T classify *exchangeable* values, that is computational entities that can be either moved or communicated (we speak of *exchange* or *transmission* to denote either communication or mobility) over a channel. While in π-calculus only channel names can be exchanged on channels, in Seal, channels transport both messages –that is base values, channel names, and agent names– and seals.

M: Message types M classify *messages*, that is entities that can be *communicated* over channels. A message can be either a synchronization message (without any content) of type $\star$, or a name. In the syntax there is no distinction between channel names and seal names. This distinction is done at the type level: if $x : \mathtt{Ch}\ T$, then x is the name of a channel that transports values of type T; if $x : \mathtt{Id}^Z A$, then x is the name of a seal with interface A and with mobility attribute Z.

Z: Along the lines of [CGG99] we use mobility attributes to specify elementary mobility properties of seals: a $\curvearrowright$ attribute characterizes a mobile seal, while a $\veebar$ attribute characterizes an immobile one. Being able to discriminate between mobile and immobile agents is one of the simplest properties related to mobility. Contrary to what happens in [CGG99], adding this feature does not require any syntactic modification to the Seal calculus.

A: Interfaces A classify the *agents* of the calculus, keeping track of their interface, that is of the set of their upward channels together with their types. The notation $[x_1{:}T_1; \cdots ; x_n{:}T_n]$ is used to record the information about the interface of an agent and denotes a set of pairs *channel_name* : *type*, defining a functional relation. Interfaces are used in the type expressions $\mathtt{Id}^Z A$ to classify names denoting agents of interface A, and in the type expressions $\mathtt{Ch}\ A$ to classify channels over which agents with interface A can be moved.

The introduction of types requires minimal modifications to the syntax of the untyped calculus of Section 2. We have to add message-type annotations to the two binders of the language: $(\boldsymbol{\nu}\, x{:}M)$ and $x^\eta(y_1{:}M_1, \ldots, y_n{:}M_n)$ (the latter will be often abbreviated as $x^\eta(\vec{y}{:}\vec{M})$), and to redefine the set of free names fn() as follows.

$$\mathrm{fn}(x) = \{x\} \qquad \mathrm{fn}((\boldsymbol{\nu}\, x{:}M)\, P) = (\mathrm{fn}(P) \setminus \{x\}) \cup \mathrm{fn}(M) \qquad \mathrm{fn}(\uparrow) = \mathrm{fn}(*) = \mathrm{fn}(\star) = \varnothing$$
$$\mathrm{fn}([x_1{:}T_1; \ldots ; x_n{:}T_n]) = \{x_1, \ldots, x_n\} \cup \mathrm{fn}(T_1) \cup \cdots \cup \mathrm{fn}(T_n) \qquad \mathrm{fn}(\mathtt{Ch}\ T) = \mathrm{fn}(T) \qquad \mathrm{fn}(\mathtt{Id}^Z A) = \mathrm{fn}(A)$$
$$\mathrm{fn}(x^\eta(y_1{:}M_1 \ldots, y_n{:}M_n).P) = (\mathrm{fn}(P) \setminus \{y_1, \ldots, y_n\}) \cup \mathrm{fn}(M_1) \cup \ldots \cup \mathrm{fn}(M_n) \cup \mathrm{fn}(\eta) \cup \{x\}$$

The rule of structural congruence that swaps binders has to be modified as well:

$$(\boldsymbol{\nu}\, x{:}M)\, (\boldsymbol{\nu}\, y{:}M')\, P \equiv (\boldsymbol{\nu}\, y{:}M')\, (\boldsymbol{\nu}\, x{:}M)\, P \qquad \text{for } x \notin \mathrm{fn}(M') \wedge y \notin \mathrm{fn}(M) \wedge x \neq y$$

Consequently, the set-theoretic shorthands used in the reduction rules must be seen as operations on lists.

4.3 Type dependencies

The notion of interface introduces names of the calculus at the type level. Since names are first order terms, type dependencies may arise. Consider for example the following terms in S-Seal.

$$P' = x^*(y{:}\mathtt{Ch}\ M).y^\uparrow(z{:}M) \qquad P = \overline{x}^*(w) \mid P'$$

P' offers upward input on channel y. Hence, a naive syntax based analysis would allow P' (and thus P) to be the body of some seal u only if the interface of u declares at least the channel y with type M. More precisely, assume that when verifying the well-formation of a process P we record as an index of the turnstile symbol the name u of the current ambient, that is of the innermost ambient that contains P (in object-orientation we would use the reserved keyword `self` instead of a generic variable u). Then the least type environment in which the well-formation of P can be deduced is:

$$u{:}[y{:}M]\,,\, x{:}\mathtt{Ch}(\mathtt{Ch}\ M)\,,\, y{:}\mathtt{Ch}\ M\,,\, w{:}\mathtt{Ch}\ M \vdash_u P$$

However, the process P may perform an internal reduction on the channel x, and then it would offer upward input on channel w, hence changing the requirement on the interface type of u:

$$\underbrace{\overline{x}^*(w)\ \mid\ x^*(y{:}\mathtt{Ch}\ M).y^{\uparrow}(z{:}M)}_{u:[y:M]} \quad \twoheadrightarrow \quad \underbrace{w^{\uparrow}(z{:}M)}_{u:[w:M]}$$

This is the recurrent problem when trying to define non-trivial channel-based types for processes: to solve it one may consider using dependent types and deal explicitly with types that change during computation. Dependent types work fine for calculi where the notion of interaction is syntactically well-determined, as in λ-calculus. Unfortunately in process calculi, where interaction is a consequence of parallel composition (which admits arbitrary rearrangements of sub-terms), all known attempts were somewhat unsatisfactory: they are usually restricted to a subset of the calculus, allowing dependent types only in particular, well-determined constructions (e.g., [YH00]).

Following a suggestion of Davide Sangiorgi, we decide to disallow input on channel names bound by an input action. In this way interfaces cannot change during reduction: for example the process P above is not well-typed, since y is first bound by an input on x and then used to perform an input from $\uparrow$. This restriction is obtained by a subtle blend of the formation rules of type environments and of interfaces and the typing rules for the input and restriction binders, as explained later on.

This restriction does not seem to limit the expressive power of the calculus: besides being theoretically well studied (see for example [Mer00]), nearly all programming languages based on π-calculus impose this constraint, while programs written in concurrent languages that do not, mostly seem to use a programming pattern that obeys to the same condition.

4.4 Typing rules

To complete the presentation of the type system it remains to define the typing rules. The rules describe a system that is close to the one of [CGZ01]. But the present version is a much simpler, more understandable, and, more expressive type system. In particular in [CGZ01], interfaces are built up during the checking of the processes, and this requires the use of two different typing environments and a subtyping relation. All of this is avoided in the system presented here as all the necessary information can be deduced by recording in the index of the turnstile the name of the "current" seal as described in the previous section. The result is a type system that is more flexible insofar as it makes it possible to type "agent generators", that is, agents that are parametric (also) in their interfaces (see later on). A further advantage of the technique used here is that the resulting system is syntax-directed and, as such, describes a deterministic typing algorithm. Formally, the rules we present next are used to derive four different judgments:

$\Gamma \vdash \diamond$ stating that Γ is a well formed environment;

$\Gamma \vdash T$ stating that T is well-formed in Γ;

$\Gamma \vdash x : M$ stating that x has type M in Γ;

$\Gamma \vdash_u P$ stating that process P is well-typed when contained in the seal u.

The typing rules are mostly the same in the two seal variants but, of course, there are small differences when dealing with the typing of channels. We start by defining the type system for shared channels. This is the easiest case as the upward channels characterizing the interface of a seal are univocally identified

by the $\uparrow$ location. We discuss the rules in detail and then show the few simple modifications needed for the shared variant for which we show that a different interesting interpretation of the type system as resource access control is possible.

4.4.1 Shared Channels

In the case of shared channels we want the interface of a seal to record all input and receive channels occurring with a $\uparrow$ location. The type system will forbid a write or send operation on x^z whenever x does not appear with the right type in the interface of z. This is obtained by the following rules:

Environment

(Env Empty) $\dfrac{}{\varnothing \vdash \diamond}$ (Env Add) $x \notin \mathrm{dom}(\Gamma)$ $\dfrac{\Gamma \vdash M}{\Gamma \cdot x{:}M \vdash \diamond}$ (Var) $\dfrac{\Gamma \vdash \diamond}{\Gamma \vdash x : \Gamma(x)}$

Well-formed types

(Type $\star$) $\dfrac{\Gamma \vdash \diamond}{\Gamma \vdash \star}$ (Type $\mathtt{Id}$) $\dfrac{\Gamma \vdash A}{\Gamma \vdash \mathtt{Id}^Z A}$ (Type $\mathtt{Ch}$) $\dfrac{\Gamma \vdash T}{\Gamma \vdash \mathtt{Ch}\ T}$ (Type Tuple) $\dfrac{\Gamma \vdash M_1 \ldots \Gamma \vdash M_n}{\Gamma \vdash M_1 \times \cdots \times M_n}$ (Type Interface) $\dfrac{\Gamma \vdash \diamond \quad \forall i \in 1..n \quad \Gamma \vdash x_i{:}\mathtt{Ch}\ T_i}{\Gamma \vdash [x_1{:}T_1, \ldots, x_n{:}T_n]}$

Processes

(Dead) $\dfrac{\Gamma \vdash \diamond}{\Gamma \vdash_u \mathbf{0}}$ (Par) $\dfrac{\Gamma \vdash_u P_1 \quad \Gamma \vdash_u P_2}{\Gamma \vdash_u P_1 \mid P_2}$ (Bang) $\dfrac{\Gamma \vdash_u P}{\Gamma \vdash_u !P}$ (Res) $\dfrac{\Gamma \cdot x{:}M \vdash_u P}{\Gamma \vdash_u (\boldsymbol{\nu}\, x{:}M)\, P}$ (Seal) $\dfrac{\Gamma \vdash x : \mathtt{Id}^Z A \quad \Gamma \vdash_x P}{\Gamma \vdash_u x[P]}$

(Output Local) $\dfrac{\Gamma \vdash x : \mathtt{Ch}\ \vec{M} \quad \Gamma \vdash \vec{y} : \vec{M} \quad \Gamma \vdash_u P}{\Gamma \vdash_u \overline{x}^*(\vec{y}).P}$

(Input Local) $\dfrac{\Gamma \vdash x : \mathtt{Ch}\ \vec{M} \quad \Gamma \cdot \vec{y}{:}\vec{M} \vdash_u P}{\Gamma \vdash_u x^*(\vec{y}{:}\vec{M}).P}$

(Output Up) $\dfrac{\Gamma \vdash x : \mathtt{Ch}\ \vec{M} \quad \Gamma \vdash \vec{y} : \vec{M} \quad \Gamma \vdash_u P}{\Gamma \vdash_u \overline{x}^{\uparrow}(\vec{y}).P}$

(Input Up) $(x{:}\vec{M}) \in A$ $\dfrac{\Gamma \vdash u : \mathtt{Id}^Z A \quad \Gamma \vdash x : \mathtt{Ch}\ \vec{M} \quad \Gamma \cdot \vec{y}{:}\vec{M} \vdash_u P}{\Gamma \vdash_u x^{\uparrow}(\vec{y}{:}\vec{M}).P}$

(Output Down) $(x{:}\vec{M}) \in A$ $\dfrac{\Gamma \vdash z : \mathtt{Id}^Z A \quad \Gamma \vdash \vec{y} : \vec{M} \quad \Gamma \vdash_u P}{\Gamma \vdash_u \overline{x}^z(\vec{y}).P}$

(Input Down) $\dfrac{\Gamma \vdash_u z : \mathtt{Id}^Z A \quad \Gamma \vdash x : \mathtt{Ch}\ \vec{M} \quad \Gamma \cdot \vec{y}{:}\vec{M} \vdash_u P}{\Gamma \vdash_u x^z(\vec{y}{:}\vec{M}).P}$

(Snd Local) $\dfrac{\Gamma \vdash x : \mathtt{Ch}\ A \quad \Gamma \vdash y : \mathtt{Id}^{\frown} A \quad \Gamma \vdash_u P}{\Gamma \vdash_u \overline{x}^*\{y\}.P}$

(Rcv Local) $\dfrac{\Gamma \vdash x : \mathtt{Ch}\ A \quad \Gamma \vdash y_i : \mathtt{Id}^{Z_i} A\ {\scriptstyle (i=1..n)} \quad \Gamma \vdash_u P}{\Gamma \vdash_u x^*\{\vec{y}\}.P}$

(Rcv Up) $(x{:}A) \in B$ $\dfrac{\Gamma \vdash u : \mathtt{Id}^Y B \quad \Gamma \vdash x : \mathtt{Ch}\ A \quad \Gamma \vdash y_i : \mathtt{Id}^{Z_i} A\ {\scriptstyle (i=1..n)} \quad \Gamma \vdash_u P}{\Gamma \vdash_u x^{\uparrow}\{\vec{y}\}.P}$

(Snd Up) $\dfrac{\Gamma \vdash x : \mathtt{Ch}\ A \quad \Gamma \vdash y : \mathtt{Id}^{\frown} A \quad \Gamma \vdash_u P}{\Gamma \vdash_u \overline{x}^{\uparrow}\{y\}.P}$

(Rcv Down) $\dfrac{\Gamma \vdash z : \mathtt{Id}^Y B \quad \Gamma \vdash x : \mathtt{Ch}\, A \quad \Gamma \vdash y_i : \mathtt{Id}^{Z_i} A\ {\scriptstyle (i=1..n)} \quad \Gamma \vdash_u P}{\Gamma \vdash_u x^z\{\vec{y}\}.P}$

(Snd Down) $(x{:}B) \in A$ $\dfrac{\Gamma \vdash z : \mathtt{Id}^Z A \quad \Gamma \vdash y : \mathtt{Id}^{\frown} B \quad \Gamma \vdash_u P}{\Gamma \vdash_u \overline{x}^z\{y\}.P}$

where $\Gamma \vdash \vec{y}{:}\vec{M}$ stands for $\Gamma \vdash y_1{:}M_1 \cdots \Gamma \vdash y_n{:}M_n$, and $\Gamma \cdot \vec{x}{:}\vec{M}$ for $\Gamma \cdot x_1{:}M_1 \cdot \ldots \cdot x_n{:}M_n$. We discuss the most important rules:

(Env Add): Environments are ordered lists of name and message-type pairs. A new declaration can be added to an environment Γ provided that the name is not already declared in Γ and that the type is well-formed under Γ.

(Type Interface): An interface type is well-formed if every name in it has been previously declared with the correct type in Γ. The premise $\Gamma \vdash \diamond$ ensures the well formation of the type environment for the case of the empty interface.

(Dead): The main interest of this rule is that it starts the deduction of a judgment indexed by a variable u (the other rule doing it is the (Seal) rule). This variable stores the name of the *current seal*, that is the innermost seal in which the process being typed is enclosed. Note that this rule does not impose any restriction on u; in particular u can even be not declared in Γ (this is typically the case when we check a top-level process). Restrictions on the *current seal* are imposed later in the deduction by the rules (Seal), (Receive Up), and (Input Up): they all require the current seal to be declared with a `Id` type in Γ, and the last two rules further require the presence of a given channel-type pair in its interface.

(Res): Despite of its apparent simplicity, the (Res) rule is by far the most complex rule of the system since through it the type system imposes subtle conditions on names appearing in an interface.

First of all, it is easy to see that if $\Gamma \cdot x\colon T \vdash_u P$ is provable then $\Gamma \cdot x\colon T$ is well formed and therefore $x \notin \mathrm{dom}(\Gamma)$ (see Property 4.1 in Section 4.5). This is quite important as it means that no confusion is possible when typing two nested bindings of the same variable as in that case type deductions have to resort to alpha conversion to proceed.

Consider now the processes

$$(\boldsymbol{\nu}\, x{:}\mathtt{Ch}\ M)\,(\boldsymbol{\nu}\, y{:}\mathtt{Id}\ [x{:}M])\, y[x^{\uparrow}(u{:}M)]$$

and

$$(\boldsymbol{\nu}\, x{:}\mathtt{Ch}\ M)\,(\boldsymbol{\nu}\, y{:}\mathtt{Id}\ [x{:}M])\, y[(\boldsymbol{\nu}\, x{:}\mathtt{Ch}\ M)\, x^{\uparrow}(u{:}M)]$$

which differ only for the restriction in the y seal present in the latter but not in the former. The first is well-typed while the second is not: in the latter the input operation refers to an input channel x that is different from the one appearing in the interface. A channel name that is used by a seal to read from its environment must already exist in the environment where the seal is declared. This is a very desirable feature of the type system: the interface's names must be public.

In terms of the example in Section 4.1, this means that we can declare that a machine x has interface $[23 : \mathit{telnet}]$ only if the channel named 23 and the type *telnet* are both already known (that is, declared) in the environment.

Another facet of the same phenomena appears when studying whether and when channels appearing in the interface of a seal can be renamed. For instance, the process

$$(\boldsymbol{\nu}\, y{:}\mathtt{Id}\ [u{:}M])\, y[\overline{x}^{*}(z)\ \mid\ x^{*}(u).u^{\uparrow}(v{:}M)]$$

is not well-typed because the channel u over which the input is performed has nothing to do with the channel u declared in the interface (just apply an alpha-conversion). This means that, as we discussed at length in Section 4.3, channels appearing in the interface of an *active seal*, that is, a seal not prefixed by any action, can not be bound in an input operation. This implies that seal interfaces, and thus types, do not change during reduction. On the other hand, consider the process

$$x^{*}(u{:}M).((\boldsymbol{\nu}\, y{:}\mathtt{Id}\ [u{:}M])\, y[u^{\uparrow}(v{:}M)])$$

which is similar to the previous one, with the only difference that the input operation now prefixes the whole seal (which, thus, is no longer active). This process is well-typed and models a seal "generator"

parametric in the channel of their interface. When it is put in parallel with, say, $(\boldsymbol{\nu}\, z{:}M)\, \overline{x}^*(z)$ it reduces to

$$(\boldsymbol{\nu}\, z{:}M)\, (\boldsymbol{\nu}\, y{:}\mathtt{Id}\ [z{:}M])\, y[z^{\uparrow}(v{:}M)])$$

that corresponds to an instance of the generator where, in particular, the interface is $[z{:}M]$.

In this way the type system allows the definition of inactive agents whose interface can be decided dynamically at the moment of their initialization (for example, an agent may get its definitive shape only once it is arrived to its destination). This was not possible in the system of [CGZ01].

(Input _): The action $x^{\eta}(\vec{y}{:}\vec{M}).P$ binds $\vec{y}$ in P. Thus $\vec{y}$ must be added to the environment to type P, provided that its type matches the type of x. In *(Input Local)*, the input operation is local and nothing more has to be done. In *(Input Down)* we also check that the name of the seal from which the process wants to read is indeed declared in Γ as a name of seal. In *(Input Up)* the input is from $\uparrow$, therefore the channel the process wants to read from must appear in the interface that has been declared for the enclosing seal u.

(Output _): In the case of local and upward output actions the rules *(Output Local)* and *(Output Up)* check that the types of the channel and of the argument match. The rule *(Output Down)* further checks that the channel appears with the right type in the interface of the target seal. This enforces the interpretation of the interfaces: a process can write down to a seal only if the processes local to the seal has declared in its interface that it is going to read it.

(Rcv _): The typing rules for mobility actions do not differ from the respective communication actions. The main point is that since a receive operation does not bind the seal names it specifies, they are not added to the environment used to type the continuation. Remark that in order to send a seal on a channel, it must be declared to be mobile (attribute $\curvearrowright$). In the Seal's model of mobility, when a seal is received it gets a name chosen by the receiver process. We use this feature, together with the fact that the mobility attribute is tied to seals names, to turn a mobile seal into an immobile one. For instance,

$$(\boldsymbol{\nu}\, x{:}\mathtt{Ch}\ A)\, (\boldsymbol{\nu}\, a{:}\mathtt{Id}^{\curvearrowright}A)\, (\boldsymbol{\nu}\, b{:}\mathtt{Id}^{\not\curvearrowright}A)\ \ \overline{x}^*\{a\}\ \mid\ x^*\{b\}\ \mid\ a[P] \quad \twoheadrightarrow \quad (\boldsymbol{\nu}\, b{:}\mathtt{Id}^{\not\curvearrowright}A)\ \ b[P]$$

turns the mobile seal named a into an immobile seal named b. This is achieved by imposing no constraints on the mobility attribute of the receiving name in the receive typing rule. Neither this nor the opposite is possible in Ambients with mobility attributes [CGG99]. Note also that the different instances of a received seal may have different mobility attributes.

4.4.2 Located Channels

In the case of located channels, collecting in the interface of a seal all the channels occurring with a $\uparrow$ location is not very useful: downward transmissions synchronize with channels local to the target seal, thus it is more interesting to include in the interface local channels rather than the $\uparrow$-labeled ones. Furthermore, in the previous systems it was interesting to collect in the interface of a seal *all* the $\uparrow$-located channels occurring in it, since any $\uparrow$-located channel not included in the interface would have been useless. Here instead a local channel that does not appear in the interface can still synchronize locally. Therefore in the located-channels variant the interface of a seal contains just *some* of the local channels on which the seal commits to listen. This induces a nice interpretation of the resulting type system as a system for resource access control: a seal declares in its interface the local resources it makes available to the surrounding environment, all the others being reserved to private use.

The rules that characterize such a type system are easily defined: they are the same rules as those of Section 4.4.1 with the only difference that in the rules (Input Up) and (Rcv Up) each side condition $(x{:}T) \in T'$ is replaced by the extra premise $\Gamma \vdash x{:}\mathtt{Ch}\ T$. Note that in this case the local resources declared in an interface, that is, the 'public' resources, must comply with the conditions discussed for the rule (Res) and in particular they cannot be renamed in active seals.

4.5 Properties

A first property that straightforwardly holds is the decidability of all the previous type systems. The rules are syntax-directed and satisfy the subformula property; they univocally describe a deterministic algorithm that for given Γ, P, u determines whether $\Gamma \vdash_u P$ is provable. A simple induction shows that it always terminates.

Other interesting properties that follow by induction on the depth of the derivation are the following

Property 4.1

1. *if either* $\Gamma \vdash_u P$, *or* $\Gamma \vdash x{:}M$, *or* $\Gamma \vdash T$ *is provable, then also* $\Gamma \vdash \diamond$ *is provable;*
2. *if* $\Gamma \cdot x{:}M \vdash \diamond$ *then* $\mathrm{fn}(M) \subseteq \mathrm{dom}(\Gamma)$.

These properties give some insight on the way the typing rules work. In particular they show that in the rules in which the type environment is extended (that is, the three (Input _) rules and (Res)) the new variables cannot have been declared in the old environment. Actually, 4.1:2 tells us more, as the new variables cannot *occur* in the old environment. This rules out environments such as $\Gamma \cdot y{:}\mathtt{Id}[x{:}M'] \cdot x{:}M$, and as we discussed, it is the key property that forbids the channels of the interface of an active seal to be bound in an input action.

The substitution lemma below captures a key feature of the type system, namely that names that get substituted can not appear in interfaces of seals.

Lemma 4.2 *If* $\Gamma, x{:}M \vdash_u P$ *and* $\Gamma \vdash y : M$, *then* $\Gamma \vdash_u P\{y/x\}$.

Proof [Sketch] The proof is long but standard. The key observation that deserves to be highlighted is that if $\Gamma \cdot \vec{z}{:}\vec{M} \vdash u : [x_1{:}M_1, \ldots, x_n : M_n]$ is a valid derivation, then $\Gamma \cdot x{:}M \vdash \diamond$. This, by construction of $\Gamma \cdot x{:}M$, that for all i it holds $z \neq x_i$. □

Finally, the soundness of the various systems is obtained by proving the subject reduction property, which for both variants is stated as follows:

Theorem 4.3 (Subject Reduction) *For all u,P, Q, if* $\Gamma \vdash_u P$ *and* $P \twoheadrightarrow Q$, *then* $\Gamma \vdash_u Q$.

Proof Induction on the derivation of $P \twoheadrightarrow Q$.

We detail two base cases, showing how types are preserved by communication and mobility reductions. The other base cases follow along the same lines.

- Suppose $\overline{x}^*(\vec{y}).P \mid x^*(\vec{z} : \vec{M}).Q \twoheadrightarrow P \mid Q\{\vec{y}/\vec{z}\}$. By hypothesis, we know that $\Gamma \vdash_u \overline{x}^*(\vec{y}).P \mid x^*(\vec{z} : \vec{M}).Q$. This implies $\Gamma \vdash x : \mathtt{Ch}\ \vec{M}$, $\Gamma \vdash \vec{y} : \vec{M}$, $\Gamma \vdash_u P$, and $\Gamma \cdot \vec{z}{:}\vec{M} \vdash_u Q$. Using the substitution lemma we derive $\Gamma \vdash_u Q\{\vec{y}/\vec{z}\}$. The type rule of parallel composition allows us to conclude $\Gamma \vdash_u P \mid Q$.
- Suppose $n[(\boldsymbol{\nu}\vec{r} : \vec{M})(x^{\uparrow}\{\vec{y}\}.P \mid Q)] \mid \overline{x}^n\{z\}.R \mid z[S] \twoheadrightarrow n[(\boldsymbol{\nu}\vec{r} : \vec{M})(P \mid Q \mid y_1[S] \mid \cdots \mid y_n[S])] \mid R$. By hypothesis, we know that $\Gamma \vdash_u n[(\boldsymbol{\nu}\vec{r} : \vec{M})(x^{\uparrow}\{\vec{y}\}.P \mid Q)] \mid \overline{x}^n\{z\}.R \mid z[S]$. This implies $\Gamma \vdash_u R$, $\Gamma \vdash n : [\cdots x{:}A \cdots]$, $\Gamma \vdash z : \mathtt{Id}\ A$, $\Gamma \vdash_z S$, and that for all i we have $\Gamma \cdot \vec{r}{:}\vec{M} \vdash y_i : \mathtt{Id}\ A$. These three last judgements allow us to derive $\Gamma \cdot (\boldsymbol{\nu}\vec{r}{:}\vec{M}) \vdash_n y_i[S]$, for all i. Then we can construct a derivation for $\Gamma \vdash_n (\boldsymbol{\nu}\vec{r}{:}\vec{M})(P \mid Q \mid y_1[S] \mid \cdots \mid y_n[S])$, and it is easy to construct a valid derivation for $\Gamma \vdash_u n[(\boldsymbol{\nu}\vec{r} : \vec{M})(P \mid Q \mid y_1[S] \mid \cdots \mid y_n[S])] \mid R$.

Induction steps are standard. □

4.6 Related work

Yoshida and Hennessy propose in [YH00] a type system for a higher-order π-calculus that can be used to control the effects of migrating code on local environments. The type of a process takes the form of an interface limiting the resources to which it has access, and the type at which they may be used. In their type system both input and output channels can appear in the interface, appearing strictly more expressive than the system we propose here, where input channels are only considered. However, they do

not allow active agents, but only pieces of code, to be sent over a channel. When code is received it can be activated, possibly after parameter instantiation. Besides, their type system limits the application of dependent types to the instantiation of parameters, resulting in the impossibility of giving an informative type to processes in which an output operation depends on an input one.

In [HR02] Hennessy and Riely define Dπ, a distributed variant of the π-calculus where agents are "located" (i.e., "named") threads. The main difference with respect to our work is that locations cannot be nested (that is, locations are not threads), and therefore mobility in [HR02] consist in spawning passive code rather than migrating active agents. In [HR02] locations types have the form $\mathsf{loc}\{x_1 : T_1, \ldots x_n : T_n\}$ where x_i's are channels belonging to the location. They are located resources and as such Dπ is much closer to the located Seal Calculus. The same holds true for the type systems.

Types for locations have been extensively studied in a series of papers [CG99b, CGG00, CGG99] on the Ambient Calculus. We already stressed that Seal Calculus differs from Ambient Calculus in many aspects, indissoluble boundaries, tight connection of agents with the names and objective vs. subjective mobility being the main ones. From the viewpoint of types the most prominent difference between Ambient Calculus and Seal Calculus is that the former does not provide an explicit "physical" support for mobility, while in the latter this support is provided by channels. In other words while in Ambients mobility take place on some unmaterialized *ethereal* transport medium, in Seal the medium is materialized by channels. Therefore the main novelty of seal types is that not only we type locations (agents or seals), but we also type mobility (more precisely, its support). In some sense we introduce higher-order typing: while in Ambient Calculus an agent can not discriminate which agents can traverse its boundaries, this is possible in our type system. For the same reason we can make a mobile location become immobile, while this is not possible in the cited works on Ambient Calculus.

5 Conclusion

To conclude this paper, we review the evolution of the Seal Calculus and we briefly discuss the Seal experience in the broader context of process languages for mobile computation.

The evolution of Seal The original presentation of the Seal calculus [VC99] was tailored with an implementation in mind, and offered features that were considered important from a practical viewpoint but turned out to be inessential for its theoretical study. The most visible difference is that [VC99] included a further security ingredient called *portals*. The aim of portals was to specify access control policies independently from process behavior. In a similar manner as access rights are specified in modern operating systems, portals allow a process to count (and limit) the number of times a channel can be used. The idea is that if a seal a wants to use seal b's channel x, then b must "open" a portal for a at x. A portal is best viewed as an linear channel access permission, a temporary capability since as soon as synchronization takes place the portal is closed again. The action to open a portal is either x^η to allow seal η to read the local channel x once, or $\overline{x}^\eta$ to allow η to write once on local channel x. Portals are akin to capabilities; they are managed by the environment and can be used for controlling access to resources, and even for revocation. Portals were also needed to limit the scope of multicast operations, for example, $\overline{x}\langle * \rangle y \mid x^a \mid x^b$ limits the scope of the out operation to local processes and processes in subseals a and b. Although such a construction is useful, it was not retained in later version of the calculus as it introduces undue complexity in the formal treatment (e.g., portals induce a cumbersome four parties synchronization).

Another point of difference is that the calculus in [VC99] considered channels that were localized inside seals as this gave a clear way to structure the implementation. The shared channels variant of Seal was introduced in [CGZ01] to have a clear definition of the interface of a Seal. This variant is also easier to study, because it halves the number of cases of remote interaction. The generalization we do here is essentially the one in [CZ02] where a reduction semantics parametric on the localization of the channels was given. There are also many minor differences with respect to the calculus of [VC99]. Among these it is worth mention the handling of extrusions and of free names of moving seals (such as the e-condition), as these have non-trivial consequences on the theoretical study of the calculus. These modifications were

coupled to a general clean-up of the definition of Seal, and in particular of its semantics that contrary to what happens in [VC99] is given in pure chemical style without resorting to auxiliary definitions (we think of the heating relation of [VC99]).

Seal in Perspective We now review the place of Seal in the landscapes of languages for mobile computations. The difficulties of modeling some key aspect of distributed computing, in particular failures, within the π-calculus motivated a number of researchers to specify distributed extensions [Ama97, RH97]. But these did not take into account security and disconnected operation. In another research direction distributed shared memory systems such as Linda were extended with explicit localities and the ability to evaluate dynamically scoped processes at a given locality [DFP97b]. But security remained a significant problem in this approach. Process mobility was first addressed by Thomsen in his calculus of higher order processes (or CHOCS) [Tho93]. Sangiorgi's elegant encoding [San92] of higher order process demonstrated that it was not necessary to transmit processes on labels. One main difference between these treatments of mobility and the one advocated here is that the processes being exchanged are inactive ones, while in Seal or Ambients, active processes can move. Fournet and Gonthier proposed the distributed join-calculus [FG96] which supports agent mobility [FGL$^+$96, FLS00]), and failures. The differences with Seal are mostly in the treatment of communication. Join assumes channels are rooted at locations. Thus possession of a channel name entails the ability to communicate with the owner of the channel. In a wide area network a global location service is needed which represents a non-trivial challenge as shown by Nomadic Pict [SWP98]. This choice also implies that the mediation property can not be easily achieved. In practical terms this means that isolation policies can not be straightforwardly implemented. More subtly, the lack of syntactic difference between local and remote resources promotes a programming style in which computations are spread over a number of different nodes, thus increasing the degree of interdependency and making computation potentially more sensitive to failures. Several researchers have proposed to rely on types for resource access control [HR99, DFP97a, Sew98]. The work of Hennessy and Riely is innovative as it deals with open networks where a subset of hosts may be malicious. This raises challenging problems which include, but are not restricted to, handing out an ill-typed value to a mobile application which is detected by the type system before the value is used, but this remains a genuine attack as the process holding the value will experience a runtime type violation.

The Ambient Calculus of Cardelli and Gordon [CG98] is of course the most closely related research effort. Ambients resemble seals as they are named hierarchically structured locations. The calculi differ in their mobility models. Ambient can trigger a move action by exercising a capability, this is termed subjective mobility. Moreover, with the proper capability an ambient may enter another ambient at any time. Our model allows the parent to control mobility of its subseals. While trapping a migrating ambient is not entirely straightforward, a parent seal can enforce confinement on any subseal. Another significant difference is that the boundary around an ambient can be dissolved, thus releasing the ambient's content in the current environment.

Seal influenced other works in the field. Location mobility as a result of process synchronization was first introduced in Seal as a natural extension of π-calculus communication primitives. It was later introduced for ambients by Levi and Sangiorgi in [LS00], and then in several other Ambient variants. The Seal calculus also had a direct influence on the design of Boxed Ambients [BCC01a, BCC01b], a variant of the Ambient calculus obtained by suppressing the open capability and where non local communication are made possible by enriching the calculus with the communication primitives of the *located* Seal calculus. The communication primitives of the *shared* Seal calculus, first defined in [CGZ01], were adapted to Boxed Ambients in [CBC02] in order to ease static detection of insecure information flows. The difference in expressiveness of the communication primitives of the located and shared variant of the primitives is at the origin of the NBA (New Boxed Ambients) calculus defined in [BCMS02], which enriches the shared channel communication of Seal they borrow from [CGZ01] with name capturing receiving actions (however these are useful only because they conform to the ambient calculus model where names are tightly bound to agents and are not modified by mobility: since in Seal the receiving agents decide the name of incoming seals this feature would be useless in the Seal model). The interaction pattern introduced in Seal for mobility of exiting agents is first used for ambients in [MH02]. Seal has inspired the calculus of mobile resources of [GHS02] which inherits the interaction pattern for exiting agents and the fact that mobility

takes place on the anonymous content of locations. It has also played a role in the design of the crypto-loc calculus [BA03] which has a similar communication model. Finally Seal primitives are also at the basis of the definition of Boxed-π, an extension of the π-calculus for securely integrating trusted and untrusted off-the-shelf software components [SV99, SV03].

Acknowledgments. The authors are grateful to the anonymous referees for their detailed comments. The authors thank Luca Cardelli for his kind advice and encouragement; Peter Sewell for insights into process calculi and language design; Christian Tschudin for many discussions on mobile code; Doug Lea for patiently explaining distributed object systems; Walter Binder, Ciaran Bryce, Andreas Krall and Manuel Serrano for the Java implementation of Seal; Dimtri Konstantas and Jean-Henri Morin for using seal in practice. Francesco Zappa Nardelli is grateful to the *S3* group of Purdue University for their hospitality. This work was partially supported by the European FET contract *MyThS*, IST-2001-32617 and NSF under grant CCR–0093282 "CAREER: Foundations and Implementation of Mobile Object Systems" and CCR-0209083 "Distributed Access Control for Accountable Systems".

References

[Ama97] Roberto M. Amadio. An asynchronous model of locality, failure, and process mobility. In *Proceedings of COORDINATION '97*. Springer-Verlag, 1997. Full version as Rapport Interne, LIM Marseille, and Rapport de Recherche RR-3109, INRIA Sophia-Antipolis, 1997.

[BA03] Bruno Blanchet and Benjamin Aziz. A calculus for secure mobility. In *Eighth Asian Computing Science Conference (ASIAN'03)*, pages 188–204, Mumbai, India, December 2003.

[BC01] M. Bugliesi and G. Castagna. Secure safe ambients. In *Proc. of POPL'01*, pages 222–235. ACM Press, 2001.

[BCC01a] M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *TACS'01*, number 2215 in Lecture Notes in Computer Science, pages 38–63. Springer-Verlag, 2001.

[BCC01b] M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in Mobile Ambients. In *CONCUR'01*, number 2154 in Lecture Notes in Computer Science, pages 102–120. Springer-Verlag, 2001.

[BCMS02] M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication interference in mobile boxed ambients. In *FST&TCS '02, 22th Conference on the Foundations of Software Technology and Theoretical Computer Science*, number 2556 in LNCS, pages 85–96. Springer, December 2002. Extended version available at http://www.cogs.susx.ac.uk/users/vs/research/paps/nba-subm.ps.gz.

[Bin01] W. Binder. J-SEAL2: a secure high-performance mobile agent system. *Electronic Commerce Research*, 1(1/2):131–148, February 2001.

[BV01] C. Bryce and J. Vitek. The JavaSeal mobile agent kernel. *Autonomous Agents and Multi-Agent Systems*, 4(4):359–384, 2001.

[Car95] L. Cardelli. A language with distributed scope. *Computing Systems*, 8(1):27–59, 1995.

[CBC02] S. Crafa, M. Bugliesi, and G. Castagna. Information flow security for Boxed Ambients. In *Foundations of Wide Area Network Computing (F-WAN)*, number 66(3) in Electronic Notes in Theoretical Computer Science. Elsevier Science B.V., 2002.

[CG98] L. Cardelli and A. Gordon. Mobile Ambients. In *Proceedings of F0SSaCS'98*, number 1378 in Lecture Notes in Computer Science, pages 140–155. Springer-Verlag, 1998.

[CG99a] L. Cardelli and A. Gordon. Equational properties for Mobile Ambients. In *Proceedings FoSSaCS '99*. Springer-Verlag LNCS, 1999.

[CG99b] L. Cardelli and A. Gordon. Types for Mobile Ambients. In *Proceedings of POPL '99*, pages 79–92. ACM Press, 1999.

[CG00] L. Cardelli and A. Gordon. A commitment relation for the Ambient Calculus. Available at `http://research.microsoft.com/~adg/Publications/ambient-commitment.pdf`, October 2000.

[CGG99] L. Cardelli, G. Ghelli, and A. Gordon. Mobility types for Mobile Ambients. In *Proceedings of ICALP '99*, number 1644 in Lecture Notes in Computer Science, pages 230–239. Springer-Verlag, 1999.

[CGG00] L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient groups and mobility types. In *International Conference IFIP TCS*, number 1872 in Lecture Notes in Computer Science, pages 333–347. Springer-Verlag, 2000.

[CGZ01] G. Castagna, G. Ghelli, and F. Zappa Nardelli. Typing mobility in the Seal Calculus. In *CONCUR'01*, number 2154 in Lecture Notes in Computer Science, pages 82–101. Springer-Verlag, 2001.

[CZ02] G. Castagna and F. Zappa Nardelli. The Seal Calculus revisited: contextual equivalence and bisimilarity. In *FST&TCS '02, 22th Conference on the Foundations of Software Technology and Theoretical Computer Science*, number 2556 in LNCS, pages 85–96. Springer, December 2002.

[DFP97a] Rocco De Nicola, GianLuigi Ferrari, and Rosario Pugliese. Coordinating mobile agents via blackboards and access rights. In *Proceedings of COORDINATION'97*. Springer-Verlag, 1997.

[DFP97b] Rocco De Nicola, Gianluigi Ferrari, and Rosario Pugliese. Locality based Linda: programming with explicit localities. In *Proceedings of FASE–TAPSOFT'97*. Springer-Verlag, 1997.

[FG96] Cédric Fournet and Georges Gonthier. The reflexive chemical abstract machine and the join-calculus. In *Proceedings of POPL'96*, pages 372–385. ACM, January 1996.

[FGL+96] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *7th International Conference on Concurrency Theory (CONCUR '96)*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421. Springer-Verlag, 1996.

[FLS00] C. Fournet, J-J. Levy, and A. Schmitt. An asynchronous, distributed implementation of Mobile Ambients. In *International Conference IFIP TCS*, number 1872 in Lecture Notes in Computer Science. Springer-Verlag, 2000.

[FMT01] G. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *Proc. ICTCS*, volume 2202 of *Lecture Notes in Computer Science*, 2001.

[Gel85] David Gelernter. Generative communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, January 1985.

[GHS02] J.C. Godskesen, T. Hildebrandt, and V. Sassone. A calculus of mobile resources. In *CONCUR 2002 (13th. International Conference on Concurrency Theory)*, Lecture Notes in Computer Science. Springer, 2002.

[Gra98] Robert S. Gray. Agent Tcl: A flexible and secure mobile-agent system. Technical Report PCS-TR98-327, Dartmouth College, Computer Science, Hanover, NH, January 1998.

[HKM+98] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A packet language for active networks. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages*, pages 86–93. ACM, 1998. Available at `www.cis.upenn.edu/ ~switchware/ papers/ plan.ps`.

[HMR03] M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed system. In *Proc. 5th FoSSaCS '03*, Lecture Notes in Computer Science. Springer Verlag, 2003.

[HR99] Matthew Hennessy and James Riely. Type-safe execution of mobile agents in anonymous networks. In *Internet Programming Languages*, number 1686 in Lecture Notes in Computer Science. Springer-Verlag, 1999.

[HR02] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173:82–120, 2002.

[HRY03] M. Hennessy, J. Rathke, and N Yoshida. Safedpi: a language for controlling mobile code. In *Proc. FOSSACS 03*, volume 2620. Lecture Notes in Computer Science, 2003.

[HY95a] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[HY95b] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[JLHB87] E. Jul, H. Levy, N. Hutchinson, and A. Black. Fine-grained mobility in the emerald system. *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, pages 62–74, November 1987.

[LS00] F. Levi and D. Sangiorgi. Controlling interference in Ambients. In *POPL '00*, pages 352–364. ACM Press, 2000.

[Mer00] M. Merro. *Locality in the π-calculus and applications to distributed objects*. PhD thesis, École de Mines de Paris, October 2000.

[MH02] M. Merro and M. Hennessy. Bisimulation conguences in Safe Ambients. In *POPL'02*, pages 71–80. ACM Press, 2002.

[Mil91] Robin Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, UK, October 1991. Appeared in *Proceedings of the International Summer School on Logic and Algebra of Specification*, Marktoberdorf, August 1991. Reprinted in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer, and H. Schwichtenberg, Springer-Verlag, 1993.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Parts I and II. *Information and Computation*, 100:1–77, September 1992.

[MS92] R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. ICALP 92*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer Verlag, 1992.

[MZ03] M. Merro and F. Zappa Nardelli. Bisimulation proof methods for mobile ambients. In *ICALP '03*, LNCS, 2003. An extended version is available as COGS Computer Science Report 2003:01.

[NNHJ99] F. Nielson, H. Riis Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in mobile ambients. In *Proc. CONCUR '99*, number 1664 in Lecture Notes in Computer Science, pages 463–477. Springer-Verlag, 1999.

[PSB95] Przemyslaw Pardyak, Stefan Savage, and Brian N. Bershad. Language and runtime support for dynamic interposition of system code. Unpublished manuscript , November 1995.

[RH97] James Riely and Matthew Hennessy. Distributed processes and location failures. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Proceedings of ICALP '97*, volume 1256 of *Lecture Notes in Computer Science*, pages 471–481. Springer-Verlag, 1997. Full version as Report 2/97, University of Sussex, Brighton.

[RH98] J. Riely and M. Hennessy. A typed language for distributed mobile processes. In *Proceedings of POPL'98*, pages 378–390. ACM Press, 1998.

[San92] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.

[San96] D. Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Information and Computation*, 131(2):141–178, 1996.

[San01] D. Sangiorgi. Extensionality and intensionality of the ambient logic. In *Proc. of the 28th ACM Symposium on Principles of Programming Languages*, London, 2001. ACM Press.

[Sew98] Peter Sewell. Global/local subtyping and capability inference for a distributed π-calculus. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Proceedings of ICALP '98*, volume 1443 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[Sew00] P. Sewell. Applied π – a brief tutorial. Technical Report 498, Computer Laboratory, University of Cambridge, 2000.

[SV99] P. Sewell and J. Vitek. Secure composition of insecure components. In *12th IEEE Computer Security Foundations Workshop*, 1999.

[SV03] Peter Sewell and Jan Vitek. Composition of untrusted code: Wrappers and causality types. *Journal of Computer Security, to appear in* 2003.

[SW02] D. Sangiorgi and D. Walker. *The π-calculus*. Cambridge University Press, 2002.

[SWP98] Peter Sewell, Pawel Wojciechowski, and Benjamin Pierce. Location independence for mobile agents. In *Proceedings of the 1998 Workshop on Internet Programming Languages*, Chicago, Ill., May 1998.

[Ten96] David Tennenhouse. Active networks. In USENIX, editor, *2nd Symposium on Operating Systems Design and Implementation (OSDI '96), October 28–31, 1996. Seattle, WA*, Berkeley, CA, USA, October 1996. USENIX.

[Tho93] Bent Thomsen. Plain CHOCS. A second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, 1993.

[US01] A. Unyapoth and P. Sewell. Nomadic Pict: Correct communication infrastructures for mobile computation. In *Proc. 28th POPL*. ACM Press, 2001.

[VC99] J. Vitek and G. Castagna. Seal: A framework for secure mobile computations. In *Internet Programming Languages*, number 1686 in Lecture Notes in Computer Science, pages 47–77. Springer-Verlag, 1999.

[Vig99] M. G. Vigliotti. Transition systems for the ambient calculus. Master thesis, Imperial College of Science, Technology and Medicine (University of London), September 1999.

[Vit99] Jan Vitek. *The Seal model of Mobile Computations*. PhD thesis, University of Geneva, 1999.

[Whi96] Jim White. Mobile agents white paper. General Magic, 1996. Available from `http://www.generalmagic.com`.

[WWWK97] Jim Waldo, Geoff Wyant, Ann Wollrath, and Sam Kendall. A note on distributed computing. In Jan Vitek and Christian Tschudin, editors, *Mobile Object Systems: Towards the programmable Internet*. Springer-Verlag, 1997.

[YH00] N. Yoshida and M. Hennessy. Assigning types to processes. In *Proceedings, Fifteenth Annual IEEE Symposium on Logic in Computer Science*, pages 334–348, 2000.

[Zap03] F. Zappa Nardelli. *De la sémantique des processus d'ordre supérieur*. PhD thesis, U. Paris 7, 2003.