Commitment and Confinement  
for the Seal Calculus  

Giuseppe Castagna  
Jan Vitek

Abstract

The Seal calculus is a distributed process calculus in which locations and movement of computational entities are explicit. The calculus is targeted at secure distributed applications over large scale open networks such as the Internet. Security is addressed by a fine-grained access control mechanism. In [14] we motivated the design choices. Here we develop some technical tools, which include both a reduction and a commitment semantics and observational equivalences, to allow us to state and formally verify properties of mobile programs. In particular for studying security, we develop the notion of confinement. Confinement is essential for security; as it expresses that an execution environment has complete control over the communication capabilities of the programs that are running within it.

1 Introduction

The Seal calculus is a calculus of mobile computations. It has been designed to model programming of large scale distributed systems over open networks such as the Internet. The key problems for programming such systems include scalability to very large configurations, evolution of the hardware and software platform over the lifetime of the system, different administration domains that control different parts of the distributed environment, and finally, security issues linked to privacy and integrity of the system and protection of resources.

The Seal calculus is a process calculus which has been designed according meet the following guiding principles for Internet programming [14]:

1. Explicit localities. Locality is crucial to Internet programming. The physical size of the Internet and the number of hosts it connects mean that scalability and explicit locality information are essential to the implementation of general-purpose services.

2. No reliance on global state. Algorithms that require synchronization over sets of machines or up-to-date information about the state of groups of processors can not be part of a general service infrastructure. Rather, these services should be built on top of a common low–level layer.

3. Restricted connectivity. At a given time, a computation may be able to communicate with only a subset of the other entities on the network, and in the extreme case, a computation may have to operate disconnected for arbitrary periods of time.

4. Dynamic reconfiguration. New hosts and communication links may be added with no advance notice, hosts may disappear and even reappear under a new name and address.
The topology both in the physical sense and in the logical sense of services and resources available is thus changing over time. The location of all elements of a distributed computation should be controlled by the computation itself, allowing it to adapt to these changes.

5. **Resource access control.** To meet application specific security requirements, mechanisms for protecting resources must be provided. Resources include information stored in memory or on disk, the network interface, bandwidth usage and both cpu and memory usage.

The Seal calculus can be roughly described as the $\pi$-calculus [9] with hierarchical location mobility and resource access control. Unlike many distributed programming languages, the goal of the Seal calculus is not to provide a high-level programming interface that eases distributed programming by hiding localities, but rather its goal is to expose the network and hand over control of localities and low-level protection to the system programmer. The means to this end are powerful mobility and protection primitives. Another view of the Seal calculus is as a substrate for implementing higher level languages and advanced distributed services. In this light, the Seal calculus takes on the role of lowest common denominator between various Internet applications. Sophisticated services that require higher degrees of coherence and synchronization can be built on top of it. Examples of such services are distributed memory management, location independent secure messaging, and channels with quality of service guarantees.

What the Seal calculus does provide is a model of mobility which subsumes message passing, remote evaluation, process migration and which models user mobility and hardware mobility. Furthermore, the calculus provides a hierarchical protection model, in which each level can implement security policies by mediation. Mediation means that observable actions performed at a lower level in the hierarchy are scrutinized and controlled by higher levels. The hierarchical model guarantees that a policy defined at some level will not be bypassed or amended at lower levels. Another important point is that security policies are dynamic, i.e. they are embodied by expressions within the calculus. This has the advantage of allowing systems to evolve and adapt to changes in operating conditions. The calculus has a notion of protection boundary, and all operations that cross a boundary are syntactically different from local operations.

**Organization** The paper is organizes as follows. Section 2 introduces informally the main features of the Seal calculus. Section 3 and 4 present respectively the syntax and reduction semantics of the calculus. Section 5 gives a commitment semantics which we prove to correspond to the reduction semantics. We also define barbed equivalence and barbed congruence. Section 6 introduces the notion of confinement and proves our main result. We conclude with a short discussion of related work.

2 **Abstractions**

The Seal calculus unifies several concepts from distributed programming into three abstractions: locations, processes and resources. Locations are meant to stand for physical places...
such as those delimited by the boundaries of address spaces, host machines, routers, firewalls, local area networks or wide area networks. Locations also embody logical boundaries such as protection domains, sandboxes and applications. The process abstraction stands for any flow of control such as a thread or operating system process. Finally, resources unify physical resources such as memory locations and peripheral device interfaces with services such as those offered by other applications, the operating system or a runtime system.

Names   Names may denote two different kinds of computational entities, seals and channels; they are values and as such can be exchanged in communication. The calculus provides an operator, written $\nu x$, to create fresh names which can be used without fear of name clash with any other name. The semantics of the calculus ensures that names can not be manufactured thus they may play a role in security protocols.

Processes   In a process calculus every expression denotes a process — a computation — running in parallel with other processes. The simplest Seal calculus expression is the inert process 0, a process with no behavior. A process $\alpha \cdot P$ is composed of an action $\alpha$ and a continuation process $P$, this expression denotes a process waiting to perform $\alpha$ and then to behave like $P$. Actions consist of communication and moves and are explained later on. $P | Q$ denotes a process composed of two subprocesses, $P$ and $Q$ running in parallel. The replicated process $! P$ can be considered as a process that creates an infinite number of copies of $P$ running in parallel. Finally a process can also be a location with a process body, that is a seal, as we will see next.

Locations   Seals are named, hierarchically-structured, locations. The expression $n[P]$ denotes a seal named $n$ running process $P$. Since a seal is also a process, then a seal can contain a hierarchy of subseals of arbitrary depth. If $P$ contains one or more seals, say $\tilde{m} = m_1 \ldots m_n$, then $n$ is the parent of $\tilde{m}$ and $\tilde{m}$ are the children of $n$. The transitive closure of the set of children is the set subseals of $n$. A configuration is depicted in Figure 1 which shows an outermost seal that represents the network its children are hosts, and their subseals are instances of application programs. The processes $P$, $P'$ and $P''$ denote behaviors of the sandbox and the two

![Figure 1: Seal calculus term and configuration tree.](attachment:image.png)
hosts, and $Q$, $Q'$ and $Q''$ denote the behaviors of the applications. An alternate graphical representation is the configuration tree in the same figure. In a configuration tree process-labeled vertices represent seals while edges represent seal inclusion. The position of seal names (on edges) emphasizes the weak association between names and seals: they are merely tags used by parents to tell their children apart.

**Resources** The only resources in Seal calculus are *channels*. Channels are named computational structures used to synchronize concurrent processes. Just as processes, channels are located. Channel denotations are used to specify in which seal a channel is located. Thus, a channel $x$ is denoted by $x^\eta$, where $\eta$ is $\star$ when the channel is local, is $\uparrow$ when the channel is in the parent, and is $n$ when the channel is in a child seal $n$. Processes are restricted to use only the channels for which they know the names; it is not possible to guess names. In that respect names are simple-minded capabilities.

**Interactions** Since processes (and resources) are located, process interactions can be either *remote* or *local*. The Seal calculus allows only three distinct patterns of interaction. Two of them are *remote* interactions: a process located in the parent synchronizes with a process located in a child on (1) a channel of the child or (2) a channel of the parent. The third one is the *local* interaction: two co-located processes synchronize over a local channel.

These interaction patterns are restrictive. For example, they do not allow processes located in sibling seals, and even less those located in arbitrary seals, to communicate directly. Communication across a seal configuration must be encoded; in other words every distributed interaction up to packet routing must be programmed.

Channel synchronization is used both for communication (the channel is used to pass a name) and for mobility (the channel is used to move a seal), and each of these two forms of interaction corresponds to different actions pairs. *Communication*: $x^\eta(\lambda y).P$ denotes a process waiting to read channel $x$ located in $\eta$ and then behave like $P$; $\pi^\eta(y).P$ denotes a process waiting to output $y$ on channel $x$ locate in $\eta$ and then behave like $P$. *Mobility*: $\pi^\eta(y).P$, denotes a process waiting to send a child seal $y$ along channel $x$ located in $\eta$ and then behave like $P$; $x^\eta(z).P$ denotes a process waiting to receive a seal along channel $x$ located in $\eta$, name it $z$ and behave like $P$.

**Mobility** The specificity of the Seal calculus over other process calculi is that Seals along with their body process and subseals may be moved over channels and even copied, by means of the send/receive actions described above.

On the configuration tree, mobility corresponds to a tree rewriting operation. A move disconnects a subtree rooted at some seal $y$ and grafts it either onto the parent of $y$, onto one of $y$ children, or back onto $y$ itself. The rewriting operation relabel the edge associated to the moved seal, and can create a finite number of copies of the subtree rooted at the moved seal.

The diagrams below show an initial configuration (a) and all three possible configurations obtained after a move. (b) is obtained by moving $n$ into the parent and renaming it to $m$. (c) is obtained by moving $n$ in $x$ and renaming it to $m$. (d) is obtained by renaming $n$ to $m$ (local
move), note that each of these moves can also introduce copies.

As mentioned, it is possible to copy running seals. This is also done as a special case of mobility
where a receive action creates several copies of the received seal. The full form of a receive
action is $x^n\langle y_1 \ldots y_n \rangle \cdot P$ it denotes a process waiting to receive one seal and create $n$ identical
copies of that seal under the names $\vec{y}$.

A simple use of mobility is the implementation of a copy operation. Typically this operation
is useful for fault tolerance to dynamically replicate running applications as new servers come
online, or to improve availability by replicating computations.

Intuitively the result of $\text{COPY } x \text{ AS } z \equiv (\nu y) (y^* \langle x \rangle \mid y^* \langle x z \rangle \cdot P)$

Protection Use of non-local channels implies a threat to security, as an unknown process that
may have migrated from an untrusted location has a way to access resources of its host. As a
first line of defense, seals are not allowed to move about autonomously or arbitrarily. Migration
is always under the control of a seal’s environment which decide when it occurs. Furthermore
a seal must actually perform a receive action in order to allow a new seal to migrate from the
outside and since it choose the name for the new seal it can choose this name fresh and thus
arbitrarily isolate the newcomer from the other processes. The second line of defense is keeping
tight control of names of channels. But, once a name has been given out it may be difficult to
control its propagation in the system, it is safe to assume that it quickly become public.

The third security ingredient is tight control over local resources. We have already said
that a local action may synchronize with a remote action. This constitutes an external access
to a local resource. We want to strictly monitor these access by allowing the synchronization
with a remote action only in presence of an explicit permission. The protection mechanism we
propose to control inter-seal interactions is called the *portal*. 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. As soon as synchronization takes place the portal is closed
again. In the calculus the action to open a portal is either \( \text{open}_\eta x \) to allow seal \( \eta \) to read local channel \( x \) once or \( \text{open}_\eta x \) to allow \( \eta \) to write once on local channel \( x \).

3 Seal calculus syntax

We assume infinite sets \( \mathcal{N} \) of names and \( \overline{\mathcal{N}} \) of co-names disjoint and in bijection via \( (\cdot) \); we declare \( \overline{x} = x \). The set of location denotations extends names with two symbols \( (\cdot, \uparrow) \). Bold font variables denote either a name or the corresponding co-name, thus \( x \) may be either \( x \) or \( \overline{x} \).

\[
\begin{array}{c|c}
\mathcal{N} & m, n, \ldots, x, y, z \\
\overline{\mathcal{N}} & \overline{m}, \overline{n}, \ldots, \overline{x}, \overline{y}, \overline{z} \\
x & := x \mid \overline{x}
\end{array}
\]

The set of processes, ranged over by \( P, Q, R, S \) and actions, ranged over by \( \alpha \), are defined by the following grammars:

**Table 1: Processes and Actions**

<table>
<thead>
<tr>
<th>Processes</th>
<th>Actions</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P ::= 0 )</td>
<td>( \alpha ::= \overline{x}^\eta(\overline{y}) )</td>
<td>name output</td>
</tr>
<tr>
<td>( P</td>
<td>Q )</td>
<td>( \alpha ::= x^\eta(\lambda \overline{y}) )</td>
</tr>
<tr>
<td>( (\nu x)P )</td>
<td>( \alpha ::= \overline{x}^\eta(y) )</td>
<td>seal send</td>
</tr>
<tr>
<td>( \alpha . P )</td>
<td>( \alpha ::= x^\eta(\overline{y}) )</td>
<td>seal receive</td>
</tr>
<tr>
<td>( ! P )</td>
<td>( \alpha ::= \text{open}_\eta x )</td>
<td>portal open</td>
</tr>
<tr>
<td>( x[P] )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x[X] )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0 denotes the inert process. \( P | Q \) denotes parallel composition. \( (\nu x)P \) denotes restriction. \( \alpha . P \) denotes an action \( \alpha \) and a continuation \( P \). \( ! P \) denotes replication. Finally, \( x[P] \) and \( x[X] \) denote, respectively, a seal named \( x \) with body process \( P \) and a seal \( x \) with body a process variable \( X \).

**Definition 1** A process \( P \) is well-formed if and only if it contains no abstract seal \( x[X] \) and no action of the form \( \text{open}_\eta x \) (portal local open).

Subsequently, we shall deal only with well-formed processes.

Following accepted terminology, the polarity of actions on names is positive and the polarity of actions on co-names is negative. \( \overline{x}^\eta(\overline{y}) \) \( P \) denotes a process offering \( \overline{y} \) at channel \( x \) located in seal \( \eta \) with a continuation \( P \). The process \( \overline{x}^\eta(\lambda \overline{y}) \) \( P \) denotes a process ready to input distinct names \( \overline{y} \) at \( x \) in \( \eta \). The \( \lambda \) is a visual cue to remind the reader the \( \overline{y} \) are bound in \( P \). \( \overline{x}^\eta(y) \) \( P \) denotes the sender process offering at \( x \) in \( \eta \) a seal named \( y \). The process \( \overline{x}^\eta(\overline{y}) \) \( P \) denotes the
receiver process waiting to read a seal at \( x \) in \( \eta \) and start \( n \) copies of it under names \( y_1 \ldots y_n \). Note that this action is not binding. Finally, \( \text{open}_\eta \ x \ . \ P \) (respectively \( \text{open}_\eta \ \pi \ . \ P \)) denotes a process offering to open a portal for seal \( \eta \) to perform a positive (respectively negative) action on local channel \( x \) and then behave as \( P \).

\( \{ y/x \} \) and \( \{ Q/x \} \) are meta-notations for substitutions. Thus \( P\{ y/x \} \) denotes the term obtained from \( P \) by simultaneous substitution of \( y_1 \ldots y_n \) for the free occurrences of distinct names \( x_1 \ldots x_n \), and \( P\{ Q/x \} \) denotes the term obtained from \( P \) by substituting process \( Q \) for all free occurrences of process variable \( X \). Substitutions are ranged over by \( \sigma \).

Location denotations \( \star \), \( \uparrow \) and \( n \) denote respectively the current seal, the parent seal and a subseal bearing name \( n \). The location denotations refer to the seal in which synchronization occurs. The simple case, which reduces to the \( \pi \)-calculus, is local synchronization, thus \( P = \pi^* (y) \ . \ 0 \) is willing to emit name \( y \) along \( x \). \( Q = x^*(\lambda z) \ . \ pi^*(z) \ . \ 0 \) is a repeater which reads a name form \( x \) and emits it on the same channel. Local communication is always allowed, so the composition of the above mentioned processes reduces in one step:

\[
P | Q \rightarrow 0 | \pi^*(y) \ . \ 0
\]

Consider now the processes \( P = \pi^* (y) \ . \ P' \), \( Q = x^*(\lambda z) \ . \ Q' \) and \( S = \text{open}_n \pi \ . \ S' \). The following configuration also reduces in one step:

\[
n[P] | Q | S \rightarrow n[P'] | Q' \{ y/x \} | S'
\]

The case above involves a subseal trying to use a resource located in its environment; the symmetric case occurs when a process tries to access resources located in a subseal. Here for example, let \( P = x^n (\lambda y) \ . \ P' \), \( Q = \pi^* (z) \ . \ Q' \) and \( S = \text{open}_n x \ . \ S' \). The following configuration reduces in one step:

\[
P | n[Q | S] \rightarrow P' \{ z/y \} | n[Q' | S]
\]

**Notation** We abbreviate possibly empty sequences \( x_1 \ldots x_n \) to \( \overline{x} \), and \( (\nu x_1) \ldots (\nu x_n) \), are abbreviated to \( (\nu \overline{x}) \). We often omit the \( * \) at the index in local communication and trailing \( 0 \) processes are elided, thus \( x^*(\lambda y) \ . \ 0 \) becomes \( x(\lambda y) \). When communication is used purely to synchronize processes, we abbreviate \( \pi^n (y) \) to \( \pi^n () \) and input \( x^n (\lambda y) \) to \( x^n () \). Actions bind tighter than composition and composition bind tighter than restrictions, so that for instance \( (\nu \ x \ x \ () \ . \ \overline{y} () | \overline{x} () \) means \( (\nu \ x \)((x () \ . \ \overline{y} ()) | \overline{x} ()) \).

## 4 Reduction semantics

The reduction relation \( \rightarrow \) is defined over processes and represents one step of computation. Reduction is defined by means of two auxiliary notions: structural congruence and heating.

### 4.1 Structural congruence

Structural congruence, \( \equiv \), is the least congruence on processes satisfying the following axioms and rules:
### Table 2: Structural congruence.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Congruence Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \equiv 0 \equiv P$</td>
<td>(Struct Dead Par)</td>
</tr>
<tr>
<td>$P \equiv Q \equiv Q \equiv P$</td>
<td>(Struct Par Comm)</td>
</tr>
<tr>
<td>$(P \equiv Q) \equiv R \equiv (Q \equiv R)$</td>
<td>(Struct Par Assoc)</td>
</tr>
<tr>
<td>$!P \equiv !P$</td>
<td>(Struct Repl Par)</td>
</tr>
<tr>
<td>$(\nu x)0 \equiv 0$</td>
<td>(Struct Dead Res)</td>
</tr>
<tr>
<td>$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$</td>
<td>(Struct Res Res)</td>
</tr>
<tr>
<td>$x \notin fn(P) \Rightarrow (\nu x)(P \equiv Q) \equiv P \equiv (\nu x)Q$</td>
<td>(Struct Res Par)</td>
</tr>
</tbody>
</table>

The set $fn(P)$ of free names of a process $P$ is defined in a standard way (e.g., see [9]).

Intuitively, this relation does not correspond to any step of computation, instead it allows processes to be rearranged so that reduction can take place.

For example, we already saw that local synchronization is enabled when two complementary actions on a channel appear at the same level:

$$x(\lambda y) \cdot \bar{y}(z) \mid \bar{x}(0) \rightarrow \bar{y}(0) \mid 0$$

But, if the emitting process was $(\nu z)\bar{x}(z)$, then the $\nu$-abstraction would hide the output action and thus prevent synchronization. Structural congruence rearranges the term so that can reduce:

$$x(\lambda y) \cdot \bar{y}(z) \equiv (\nu z)(x(\lambda y) \cdot \bar{y}(z)) \rightarrow (\nu z)(\bar{x}(\bar{0}) \mid 0)$$

This result is obtained by alternating $\rightarrow$ and $\equiv$ rewritings, that is, by allowing replacement of terms by structurally equivalent terms. Structural congruence also handles the semantics of replicated actions and performs some house-keeping by sweeping out dead processes.

Structural congruence does not handle renaming of bound variables. Instead, we consider that alpha-conversions are silently performed whenever needed.

### 4.2 Heating

Although structural congruence provides a convenient way of rearranging terms to enable local synchronization, it does not suffice for non-local synchronization. Communication across seal boundaries requires special treatment. To illustrate this point let’s modify example (1) so that the emitting process is located in a subseal:

$$\text{open}_n \bar{x} \mid x^*(\lambda y) \cdot \bar{y}(z) \mid n[\bar{x}(\bar{z})] \rightarrow 0 \mid \bar{z}(\bar{0}) \mid n[0]$$

Now, if like in (2) we $\nu$-abstract the argument of the output, we expect the $\nu$-abstraction to extrude the seal boundary and wrap around the input process, that is, informally the following should hold:

$$\text{open}_n \bar{x} \mid x^*(\lambda y) \cdot \bar{y}(z) \mid n[\nu z]\bar{x}(\bar{z}) \rightarrow (\nu z)(0 \mid \bar{z}(\bar{0}) \mid n[0])$$
This would require an equivalence such as \( n[(\nu x)P] \equiv (\nu x)n[P] \). However it would be an error to define these terms as equivalent, because we allow seal duplication \(^1\). Indeed, if we compose both terms with the copier process defined earlier \( Q = \text{COPY} \) we obtain:

\[
 n[(\nu x)P] \mid Q \rightarrow n[(\nu x)P] \mid m[(\nu x)P] \quad \text{and} \quad (\nu x)n[P] \mid Q \rightarrow (\nu x)n[P] \mid m[P]
\]

The first term yields a configuration where seals \( n \) and \( m \) have each a private channel \( x \), while in the other case, they share a common channel \( x \). Our solution is to forego structural congruence at this point and perform the extrusion together with synchronization. To this end we define a \textit{heating} relation on terms. A term is “heated” to allow synchronization. Heating singles out all the \( \nu \)-abstractions that must be extruded, that is, those that bind arguments of the negative action about to be performed. Heating will extrude as few \( \nu \)-abstractions as possible. So for example the term

\[
 \text{open}_n x \mid x^n (\lambda y).y() \mid n[(\nu w)(\nu z)x^i(z)]
\]

reduces to \( 0 \mid (\nu z)(\bar z()) \mid n[(\nu w)0] \) rather than to \( 0 \mid (\nu w)(\nu z)(\bar z()) \mid n[0] \).

More precisely, consider the term \( (\nu w)(\nu z)x^i(z) \). Heating will tell us that to synchronize on \( x^i \) it is necessary to extrude \( (\nu z) \). This is expressed by the following heating relation pair:

\[
(\nu w)(\nu z)x^i(z) \quad \sim \quad x^i.(\nu z)[z][(\nu w)0]
\]

T

The heated form actually mimic Milner’s \textit{agents} \([8]\). The channel name comes first, it is followed by a list of extruded names, \( (\nu x) \) in this case, the arguments, \( z \), and the residual process, \( (\nu w)0 \). The argument values include both names and processes.

A term in heated form is called an \textit{agent}. Agents are written \( \omega P \) where \( \omega \) is an agent prefix and \( P \) is a process. The set of \textit{agent prefixes} ranged over by \( \omega \) is defined by the following grammar:

\[
\begin{align*}
\omega &::= \epsilon & \text{empty prefix} \\
& \mid (\nu \bar x)\bar y & \text{name concretion} \\
& \mid (\nu \bar x)\bar P & \text{process concretion} \\
& \mid \lambda \bar y & \text{name abstraction} \\
& \mid \lambda X & \text{process abstraction}
\end{align*}
\]

The set \( fn(P) \) of \textit{free names} of a process \( P \), \( fn(\omega) \) of \textit{free names} of an agent prefix and \( bn(\omega) \) \textit{bound names} of an agent prefix have standard definitions.

In order to simplify the presentation of the reduction rules we introduce the set \( \mathcal{L} \) of \textit{co-locations} (that is in bijection with \( \mathcal{L} \) via \( (\cdot) \)) and the set of \textit{sync-locations}. Their use is explained later on.

\(^1\)Ambient Calculus does not allow duplication and therefore it can permit such equivalence to hold.
The heating relation \( \prec \) relates a well-formed process to a term of the form \( x^q \cdot \omega P \) and is defined as the least relation respecting the following axioms and rules (where \( \eta \) denotes either \( \eta \) or \( \overline{\eta} \)):

<table>
<thead>
<tr>
<th>Table 4: Heating.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x^n(y) \cdot P \prec x^n(\langle y \rangle P ) (Heat Out)</td>
</tr>
<tr>
<td>( x^n(\lambda y) \cdot P \prec x^n(\langle \lambda y \rangle P ) (Heat In)</td>
</tr>
<tr>
<td>( x^n(y) \cdot P</td>
</tr>
<tr>
<td>( x^n(y_1, \ldots, y_n) \cdot P \prec x^n(\langle \lambda X \rangle (P</td>
</tr>
<tr>
<td>( y \notin fn(\omega), y \notin {x, \eta}, P \prec x^q \cdot \omega P' \Rightarrow (\nu y)P \prec x^q \cdot \omega(\nu y)P' ) (Heat Res-1)</td>
</tr>
<tr>
<td>( y \in fn(\omega), y \notin {x, \eta}, P \prec x^q \cdot \omega P' \Rightarrow (\nu y)P \prec x^q \cdot (\nu y)\omega P' ) (Heat Res-2)</td>
</tr>
<tr>
<td>( bn(\omega) \cap fn(Q) = \emptyset, P \prec x^q \cdot \omega P' \Rightarrow P</td>
</tr>
<tr>
<td>( bn(\omega) \cap fn(Q) = \emptyset, P \prec x^t \cdot \omega P' \Rightarrow P</td>
</tr>
<tr>
<td>( y \notin bn(\omega), P \prec x^t \cdot \omega P' \Rightarrow y[P] \prec x^{\eta_1} \cdot \omega y[P] ) (Heat Seal-1)</td>
</tr>
<tr>
<td>( y \notin bn(\omega), P \prec x^t \cdot \omega P' \Rightarrow y[P] \prec x^{\overline{\eta_1}} \cdot \omega y[P] ) (Heat Seal-2)</td>
</tr>
</tbody>
</table>

The first two axioms handle synchronization for communication. In particular the first axiom states that an output process does not need to extrude any name. The (Heat Send) and (Heat Recv) axioms deal with synchronization for mobility. (Heat Send) states that a negative action offers as argument the body of a seal. The fourth axiom says that a positive action heats into an abstraction where the process variable \( X \) stands for the body of the seals specified by \( y_1 \). after synchronization the residual consists of the continuation \( P \) in parallel with the seals where \( X \) has been substituted by some process \( Q \).

The following two rules select the names that will be extruded. If a \( \nu \)-abstracted name does not occur in the free names of the agent prefix \( \omega \) then (Heat Res-1) applies and the name is not extruded. Instead, if a \( \nu \)-abstracted name does occur in the free names of the agent prefix \( \omega \) then (Heat Res-2) applies and the name is extruded. The rule (Heat Par) simply propagates restrictions taking care of name conflicts. Note that it is always possible to alpha-convert bound variables so that name clashes are avoided.

The rule (Heat Open) combines a local action on some channel \( x \) and a permission to interact with a matching action from a process located in seal \( \eta \). This is represented by changing the action label from \( x^s \) to \( x^s \).

Finally the last two rules allow actions originating from a seal \( y \) to synchronize with matching actions in the parent. When they flow through the boundaries of seal \( y \) the action labels are changed from \( x^t \) to \( x^{\eta_1} \) and from \( x^t \) to \( x^{\overline{\eta_1}} \) in the process to prevent accidental matches and

| \( \overline{C} \) \( \overline{\eta} := x | \overline{\eta} ) \overline{\eta} \) co-locations |
|-------------------|
| \( \eta := \eta | \overline{\eta} | x[| \) sync-locations |
further propagation. In summary, $x^\forall y$ means that the seal $y$ is ready to synchronize on its own channel $x$ with the environment (it opened the channel to the environment and committed — see Section 5 — to perform an action on it), while $x^\exists$ means that the environment is ready to synchronize $y$ on the local channel $x$ with the seal $y$ (it opened the channel to $y$ and committed to perform an action on it).

4.3 Reduction

We define the reduction relation $\rightarrow$ as the least relation on well-formed processes that satisfies:

$$
\begin{align*}
\text{(Red Res)} & \quad P \rightarrow Q \\
& \quad (\nu x)P \rightarrow (\nu x)Q \\
\text{(Red Par)} & \quad P \rightarrow Q \\
& \quad R \rightarrow Q \quad P \mid R \rightarrow Q \\
\text{(Red Seal)} & \quad P \rightarrow Q \\
& \quad x[P] \rightarrow x[Q] \\
\text{(Red Local)} & \quad P \rightarrow Q \\
& \quad x^\forall y \cdot P \rightarrow (\omega_1 P') \cdot (\omega_2 Q') \\
& \quad P \mid Q \rightarrow (\omega_1 P') \cdot (\omega_2 Q') \\
\text{(Red Remote)} & \quad P \rightarrow Q \\
& \quad x \cdot P \rightarrow x\cdot (\omega_1 P') \cdot (\omega_2 Q') \\
& \quad P \mid Q \rightarrow (\omega_1 P') \cdot (\omega_2 Q') \\
\end{align*}
$$

The pseudoapplication relation $(\cdot) \cdot (\cdot)$ used in the definition of synchronization is a partial commutative binary function from agents to processes. Let $\vec{y}, \vec{x}$ be vectors of the same arity and $\vec{x} \not\in fn(P)$, then we define pseudoapplication as:

1. $(\langle \lambda \vec{y} \rangle P) \cdot (\langle \nu \vec{x} \rangle \langle \vec{z} \rangle Q) = (\nu \vec{x})(P\{\vec{z}/\vec{y}\} \mid Q)$
2. $(\langle \lambda \vec{X} \rangle P) \cdot (\langle \nu \vec{x} \rangle \langle \vec{R} \rangle Q) = (\nu \vec{x})(P\{\vec{R}/\vec{X}\} \mid Q)$
3. Undefined otherwise.

Of course, (Red Local) and (Red Remote) apply only provided that the inferred pseudoapplication is defined.

The first three rules perform reduction within restrictions, seals and parallel composition. The rule (Red $\equiv$) allows structural rearrangements to take place around a step of reduction.

The core of the semantics is given by the last two rules. They describe synchronization on a channel that is respectively local or remote. The combined use of heating and pseudoapplication allows us to compact several rules into a single one. For example, every single rule describes the synchronization in case of both communication and mobility. Let us show how the rules work by a couple of examples, starting with (Red Local).

Example 1 Consider the process $x(\lambda y) \cdot P \mid \pi(z) \cdot Q$. Without loss of generality we can sup-
pose that \( y \not\in \text{fn}(Q) \), by possibly alpha-converting the input subprocess. By definition of heating we have \( x(\lambda y).P \prec x. \langle \lambda y \rangle P \) and \( \overline{x}(z).Q \prec \overline{x}.\langle z \rangle Q \), thus by (Red Local) the process reduces to \( (\langle \lambda y \rangle P) \bullet (\langle z \rangle Q) \), that is \( P \{z/y\} \mid Q \). In summary, \( x(\lambda y).P \mid \overline{x}(z).Q \rightarrow P \{z/y\} \mid Q \).

In case of local synchronization the notational additions collapse two rules, communication and mobility, into the single (Red Local) rule. In the case of (Red Remote) the advantage is greater since in the absence of such notation we would be obliged to specify different rules for mobility and communication and also rule for positive and negative actions.

In order to understand the (Red Remote) rule let us show a second example of communication.

**Example 2** Consider the process \( x^* (\lambda y).P \mid \text{open}_n \overline{x} \cdot R \mid n[[\nu w](\nu z)\overline{x}(z) \cdot Q] \). By silent alpha conversion, we can consider, without loss of generality, that \( y \not\in \text{fn}(R) \), and \( z \not\in \text{fn}(P \mid R) \). By (Heat In), we have \( x^*(\lambda y).P \prec x^*. \langle \lambda y \rangle P \). By (Heat Open) and the fact that \( y \not\in \text{fn}(R) \), we obtain \( x^*(\lambda y).P \mid \text{open}_n \overline{x} \cdot R \prec x\overline{x}. \langle \lambda y \rangle (P \mid R) \). Turning now to the seal, (Heat Out), (Heat Res-2) and (Heat Res-1) give us \( \nu w[\nu z] \overline{x}(z) \cdot Q \prec \overline{x}. \langle z \rangle \langle \nu z \rangle (\nu w)Q \). Therefore, by (Heat Seal), we get \( n[[\nu w](\nu z)\overline{x}(z) \cdot Q] \prec \overline{x}^n. \langle z \rangle n[[\nu w]Q] \). The side conditions of pseudoapplication in (Red Remote) being satisfied we obtain \( (\langle \lambda y \rangle (P \mid R) \bullet (\nu z) \langle z \rangle n[[\nu w]Q] = \langle \nu z \rangle \langle P \mid R \rangle \{z/y\} \mid n[[\nu w]Q] \). Finally, since \( y \not\in \text{fn}(R) \) we have \( (P \mid R) \{z/y\} = P \{z/y\} \mid R \). In summary we have

\[
x^*(\lambda y).P \mid \text{open}_n \overline{x} \cdot R \mid n[[\nu w](\nu z)\overline{x}(z) \cdot Q] \rightarrow (\nu z)(P \{z/y\} \mid R \mid n[[\nu w]Q])
\]

The Seal calculus, like CHOCS [12], sends processes as values. But, for safety (and implementation) reasons, we restrict the occurrences of process variables to be encapsulated within seals. In fact our syntax ensures that a seal abstraction will always have the form \( x^n. \langle \lambda X \rangle (P \mid y_1[X] \mid \ldots \mid y_n[X]) \) where the only occurrences of a process variable are the bodies of seals \( y_1 \) through \( y_n \). This guarantees that migrating processes will always be protected by boundaries and that their parent will not be able to compose them with arbitrary processes \( (y[X] \mid P) \) is forbidden as \( P \) may be a virus. One of the side effects of this restriction is that our calculus purposefully disallows primitives such as the Ambient calculus open which would release the contents of a seal in its enclosing environment.

We conclude this section on reduction semantics by two remarks. First, as the examples show, it is always possible to make terms satisfy the side conditions of pseudoapplication and heating rules. In fact, these conditions are on bound variables, that can be always alpha-converted to match the constraints. Secondly, we can show that reduction preserves well-formedness:

**Lemma 1** If \( P \) is well-formed and \( P \prec x^n. \omega Q \) then either \( Q \) is well formed or \( \omega = \langle \lambda X \rangle \) and \( Q \{R/X\} \) is well-formed for any well-formed \( R \).

**Proof** By induction on structure of the derivation of \( P \prec x^n. \omega Q. \)

**Lemma 2** Given a well-formed term \( P \), if \( P \rightarrow Q \) then \( Q \) is well formed.
Proof. By induction on the structure of the derivation of $P \rightarrow Q$. For the rules of communication simply note that by (HeatSend) and (Heat Res-2) if $P'$ is well-formed and $P' \sim x^n.(\nu x)(Q')P''$, then also $Q'$ is well-formed; use then Lemma 1. □

5 Commitment semantics

The astute reader will have noticed the similarity between the heating relation and a commitment semantics. Indeed a commitment semantics describes the observable continuations of a given process. It has the form $P \xrightarrow{\ell} A$ where $P$ is a process, $A$ is an agent, and $\ell$ is either an observable signal—the signal is observable in the sense that it can interact with some other process in parallel—or the invisible signal $\tau$ which corresponds to an internal synchronization of process $P$. The relation $P \xrightarrow{\ell} A$ states that process $P$ emit the event $\ell$ and then acts like $A$, we call $\ell.A$ a commitment. The idea is that a process is semantically characterized by the set of signals it commits to. The heating relation fully describes the commitments relating to communication actions, only incompletely characterizes the commitments of seals, and completely overlook commitments of the remaining processes, these commitments being hard-wired in the reduction rules. So for example we want to say that an opening process emits a “portal open” signal, that is

$$\text{open}_y x. P \xrightarrow{\epsilon} \epsilon P$$

The most interesting commitments are those of seals. Besides the invisible $\tau$ signal, which are the observable signals of a seal $x[P]$? There are only two of them: the signal $y^{x[\Gamma]}$ by which the seal asks to synchronize with its parent on a channel $y$ (note the boldface $x$), and the signal $x[]$ by which the seal offers itself for a movement. Any other signal is blocked by the seal’s boundary.

Weak-$\equiv$. The commitment semantics can be defined modulo term rearrangement via $\equiv$ congruence. However, we prove that the full power of $\equiv$ is not necessary to the definition of commitment. A much simpler relation denoted by $\equiv$ (that is not even an equivalence) suffices:

Table 6: Weak-$\equiv$.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$</td>
<td>(Weak Trans)</td>
</tr>
<tr>
<td>$P \mid Q \equiv Q \mid P$</td>
<td>(Weak Par Comm)</td>
</tr>
<tr>
<td>$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$</td>
<td>(Weak Par Assoc)</td>
</tr>
<tr>
<td>$x \not\in ft(P)$</td>
<td>$P \mid (\nu x)Q \equiv (\nu x)(P \mid Q)$</td>
</tr>
<tr>
<td>$!P \equiv P \mid !P$</td>
<td>(Weak Repl)</td>
</tr>
</tbody>
</table>
5.1 Commitment relation

We now introduce the commitment relation. The commitment relation is indexed by a set of labels \( \ell \) ranged over by \( \ell \) defined by the following grammar:

\[
\ell ::= \tau \quad \text{internal action} \\
\mid x[] \quad \text{seal offer} \\
\mid x^y \quad \text{communication action} \\
\mid \overline{x}^y(y) \quad \text{seal send action} \\
\mid \text{open}_\eta x \quad \text{portal offer}
\]

The commitment relation relates a process \( P \) to a label \( \ell \) and an agent \( \omega Q \) and is written:

\[
P \xrightarrow{\ell} \omega Q
\]

and defined as is smallest relation satisfying following groups of rules.

The first group of rules deals with the structure of the processes. (Comm \( \equiv \)) states that if \( P' \) can be obtained from \( P \) by structural rearrangement then both processes have exactly the same observable behaviors. (Comm Par) states that the commitments of a process \( P \) are visible when this process is put in parallel with \( R \), provided that the agent prefix does not bind free names in \( R \). (Comm Res-1) allows commitment to traverse the scope of a restriction provided that the bound name \( x \) does not become free and does not occur in the label. Finally (Comm Res-2) extrudes a bound name \( x \), if this name occurs in the agent prefix.

<table>
<thead>
<tr>
<th>Table 7: Labels.</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ell ::= \tau \quad \text{internal action} )</td>
</tr>
<tr>
<td>\mid x[] \quad \text{seal offer}</td>
</tr>
<tr>
<td>\mid x^y \quad \text{communication action}</td>
</tr>
<tr>
<td>\mid \overline{x}^y(y) \quad \text{seal send action}</td>
</tr>
<tr>
<td>\mid \text{open}_\eta x \quad \text{portal offer}</td>
</tr>
</tbody>
</table>

The second group deals with the signals emitted by a seal. (Comm Seal Up) states that an action trying to synchronize on a channel located in the parent is allowed to pass through the boundary of a seal named \( y \), provided that label is changed from \( x^\top \) to \( x^y\bot \). Similarly (Comm Seal Down) deals with an action that allows a process in the parent to synchronize on one of the seal’s local channels. (Comm Seal Send) states that a seal is willing to be moved. (Comm Seal \( \tau \)) allows silent actions to occur within a seal.

<table>
<thead>
<tr>
<th>Table 8: Structure.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Comm ( \equiv ))</td>
</tr>
</tbody>
</table>
| \[
P \equiv P' \quad P' \xrightarrow{\ell} \omega Q \\
\quad \Rightarrow P \xrightarrow{\ell} \omega Q
\]
| (Comm Par) |
| \[
P \xrightarrow{\ell} \omega Q \\
\quad R \xrightarrow{\ell} \omega Q \\
\quad \text{bn}(\omega) \cap \text{fn}(R) = \emptyset
\]
| (Comm Res-1) |
| \[
(\nu x) P \xrightarrow{\ell} \omega(\nu x)Q \\
\quad x \notin \text{fn}(\ell) \cup \text{fn}(\omega)
\]
| (Comm Res-2) |
| \[
(\nu x) P \xrightarrow{\ell} (\nu x)\omega Q \\
\quad x \notin \text{fn}(\ell) \land x \notin \text{fn}(\omega)
\]
Table 9: Seals.

\[
\begin{align*}
(\text{Comm Seal Up}) & \quad P \xrightarrow{x} \omega Q & y \in b[n(\omega)] \\
(\text{Comm Seal Down}) & \quad P \xrightarrow{x} \omega Q & y \in b[n(\omega)] \\
(\text{Comm Seal Send}) & \quad x[P] \xrightarrow{\mathbf{1}} \langle P \rangle 0 & y \in b[n(\omega)] 
\end{align*}
\]

The next two rules describe the behavior of portals. (Comm Port) states that a portal action $\text{open}_\eta x$ emits a signal with the same label. (Comm Open) states that a portal action combines with a local synchronization provided that names in the continuation of portal action are not captured in the agent prefix. Notice that the label is marked with a co-location.

Table 10: Portals.

\[
\begin{align*}
(\text{Comm Port}) & \quad \text{open}_\eta x \cdot P \xrightarrow{\text{open}_\eta x} \epsilon P \\
(\text{Comm Open}) & \quad P \xrightarrow{x} \omega P' & Q \xrightarrow{\text{open}_\eta x} \epsilon Q' \\
 & \quad P | Q \xrightarrow{\text{open}_\eta x} \omega (P' | Q') & b[n(\omega)] \cap f[n(Q')] = \emptyset
\end{align*}
\]

The next four rules describe the behavior of communication actions. (Comm Out) and (Comm In) state that communication actions commit to name concretions and name abstractions respectively. (Comm Local) and (Comm Remote) state that two matching communication actions synchronize.

Table 11: Communication.

\[
\begin{align*}
(\text{Comm Out}) & \quad \pi^\eta(\bar{g}) \cdot P \xrightarrow{\bar{g}'} \langle \bar{g} \rangle P \\
(\text{Comm In}) & \quad x^\eta(\lambda \bar{g}) \cdot P \xrightarrow{x^n} \langle \lambda \bar{g} \rangle P \\
(\text{Comm Local}) & \quad P \xrightarrow{x} \omega_1 P' & Q \xrightarrow{\bar{g}} \omega_2 Q' \\
 & \quad P | Q \xrightarrow{\bar{g}} \epsilon (\omega_1 P') \bullet (\omega_2 Q') \\
(\text{Comm Remote}) & \quad P \xrightarrow{\bar{g}} \omega_1 P' & Q \xrightarrow{\bar{g}'} \omega_2 Q' \\
 & \quad P | Q \xrightarrow{\bar{g}} \epsilon (\omega_1 P') \bullet (\omega_2 Q')
\end{align*}
\]

The last group of rules describes the behavior of mobility. (Comm PreSend) states that a send action emits the action as a label. (Comm Send) combines a send commitment with a seal offer, provided that free names are not captured, to a process concretion. (Comm Recv) states that a
receive action commits to process abstraction.

### Table 12: Mobility.

<table>
<thead>
<tr>
<th>(Comm PreSend)</th>
<th>(Comm Send)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi^y \langle y \rangle \cdot P \xrightarrow{\pi^y} \epsilon P )</td>
<td>( P \xrightarrow{\pi^y} \epsilon P' )</td>
</tr>
<tr>
<td>( P \mid Q \xrightarrow{\pi^y} \omega (P' \mid Q') )</td>
<td>bn((\omega))(\cap fn(P')=\emptyset)</td>
</tr>
</tbody>
</table>

(Comm Recv)

\[
\pi^y \langle y_1 \ldots y_n \rangle \cdot P \xrightarrow{\pi^y} \langle \lambda X \rangle (P \mid y_1[X] \mid \ldots \mid y_n[X])
\]

The main result of this section is the following theorem that relates reduction and commitment semantics. The proof of this theorem is rather long, the details are given in the appendices.

**Theorem 3** \( P \xrightarrow{\tau} Q \) if and only if there is \( R \equiv Q \) with \( P \xrightarrow{\tau} \epsilon R \).

We need the following results for the proof of the confinement property of the next section.

**Lemma 4** For every process \( P \) and substitution \( \sigma \), if \( P \xrightarrow{e} \omega Q \) then \( P\sigma \xrightarrow{e}\sigma (\omega Q)\sigma \)

**Proof** Straightforward induction on the depth of the derivation of \( P \xrightarrow{e} \omega Q \). For the cases of communication simply note that \( (\omega_1 P_1)\sigma \bullet ((\omega_2 P_2)\sigma) = ((\omega_1 P_1) \bullet (\omega_2 P_2))\sigma \) \(\square\)

As customary we denote by \( \xrightarrow{\cdot} \) the reflexive and transitive closure of the reduction relation \( \rightarrow \).

**Corollary 5** If \( P \xrightarrow{\cdot} Q \) then for all substitution \( \sigma \), \( P\sigma \xrightarrow{\cdot} Q\sigma \).

**Proof** By induction on the number of steps of \( \xrightarrow{\cdot} \), from Lemma 4 and Theorem 3. \(\square\)

### 5.2 Barbed Equivalence

In our case we choose to look at all interaction offers. We define a barb \( \beta \) to be any label different from \( \tau \) and we define that process \( P \) exhibits barb \( \beta \), written \( P \downarrow \beta \), if and only if \( P \xrightarrow{\beta} \omega Q \) for some agent \( \omega Q \). We also say that a process \( P \) exhibits a weak barb \( \beta \), written \( P \downarrow \beta \), if and only if either \( P \downarrow \beta \) or \( P \rightarrow Q \) and \( Q \downarrow \beta \) for some process \( Q \).

**Definition 2 (Barbed simulation)** A relation \( R \) on well-formed processes is a barbed simulation if and only if

\[
\begin{align*}
(1) & \quad P \mathcal{R} Q \text{ and } P \downarrow \beta \quad \text{implies that } Q \downarrow \beta \\
(2) & \quad P \mathcal{R} Q \text{ and } P \rightarrow P' \quad \text{implies that there is } Q', Q \rightarrow Q' \text{ and } P' \mathcal{R} Q'
\end{align*}
\]
A relation $R$ on processes is a strong barbed bisimulation if $R$ and $R^{-1}$ are barbed simulations. Strong barbed equivalence $\sim$ is the greatest barbed bisimulation. (Weak) barbed bisimulation is obtained by replacing all barbs by weak barbs in the above definition. Barbed equivalence $\approx$ is the greatest barbed bisimulation.

### 5.3 Barbed Congruence

Barbed equivalence is not a congruence, it is easy to construct processes that while barbed equivalent will be differentiated when placed in a particular context. Barbed congruence, written $\equiv$, is obtained from barbed equivalence by quantifying over all contexts:

$$ P \equiv Q \overset{def}{=} \forall C[] \in \text{Context} : C[P] \approx C[Q] $$

Where contexts are defined by

\[
C[] := [] | C[] | P | P | C[] | (\nu x)C[] | \alpha .C[] | !C[] | x[C[]]
\]

and $C[P]$ denotes the process obtained from the substitution of the process $P$ for the single occurrence of the hole $[]$ in the context $C[]$.

Strong barbed congruence, written $\sim$, is obtained similarly from strong barbed equivalence.

Two useful lemmas characterize barbed equivalences.

**Lemma 6** If $P \Downarrow \beta$ is provable and $P \equiv Q$ then $Q \Downarrow \beta$ is provable by a demonstration of the same depth.

**Lemma 7** For every process $P$ and context $C[]$ and barb $\beta$, if $C[P] \Downarrow \beta$, then either $P \Downarrow \gamma$ for some barb $\gamma$, or for all process $Q$, $C[Q] \Downarrow \beta$.

Let $\equiv^+$ denote the transitive closure of $\equiv$, that is the relation obtained by adding a transitive rule to the three axioms of $\equiv$.

We use $\pi$ to denote a permutation (that is, a bijective function from an initial segment of natural numbers to itself). We also use $P_1 | P_2 | \ldots | P_n$ to denote a term obtained by some valid parenthesing of $P_1 | P_2 | \ldots | P_n$. Then we have the following lemma

**Lemma 8** If $P \equiv^+ Q$ then:

1. $P = (P_1 | \ldots | P_n) | (P_{n+1} | \ldots | P_{n+k})$ with $n, k \geq 1$.

2. Two cases for $Q$:

   a. $Q = (P_{\pi(1)} | \ldots | P_{\pi(i)}) | (P_{\pi(i+1)} | \ldots | P_{\pi(n+k)})$ with $1 \leq i < n + k$

   b. $Q = (\nu x)(P_{\pi(1)} | \ldots | P_{\pi(n+k-1)}) | P'$ with $P_{\pi(n+k)} = (\nu x)P'$ and $x \notin fn(P_{\pi(i)})$, $1 \leq i < n + k$.  


Proof Easy induction on the depth of the derivation. □

This lemma is useful since it describes the derivations ending by (Comm ≡). Indeed consider a derivation of \[ P \xrightarrow{\ell} \omega P' \] ending by (Comm ≡). The last rule is preceded by a (possibly empty) suite of applications of (Comm ≡) topped by a different rule deriving \[ Q \xrightarrow{\ell} \omega P' \], and \[ P \equiv \top Q \]. The lemma above describes all the possible combinations of \( P \) and \( Q \).

6 Confinement

The property of confinement is one of the first security properties that must be proved in order to reason about security of mobile computations. Intuitively, a process is confined if it is completely isolated from the environment it executes in, that is, it cannot interact in any way with this environment. More precisely, we say that a process is confined if and only if in whatever system we use this process, if the system perform an action, then this action is either completely due to the process, or completely due to the environment that contains the process.

In presence of replication it is very difficult to track down the actions of a single process. Thus, we limit our treatment of confinement to the linear version of the Seal calculus, that is, where replication is not allowed. Therefore, all the results in this section hold only for terms in which no subterm has the form \( ! P \) and where all the receive actions have a single parameter (they are of the form \( x^y \)). At the end of this section we informally discuss generalization to non-linear case.

Formally, we define confinement in two steps, first by introducing immediate confinement, that characterizes processes which cannot interact with their environment in a single step of reduction; then we define general confinement that takes into account an arbitrary number of reduction steps.

Definition 3 A process \( P \) is immediately confined if and only if for all context \( C[] \) if \( C[P] \rightarrow Q \) then one of the following properties holds:

1. \( Q \equiv C[P'] \) and \( P \rightarrow P' \)
2. \( Q \equiv C'[P\sigma] \) and for all process \( R \), \( C[R] \rightarrow C'[R\sigma] \)

Definition 4 (Confinement) A process \( P \) is confined, if it is immediately confined and for all substitution \( \sigma \) if \( P\sigma \rightarrow Q \), then \( Q \) is immediately confined.

The rest of this section is devoted to the proof of the two following results:

1. \( P \approx 0 \) if and only if \( P \) is confined.
2. \( (\nu x) x[P] \approx 0 \)

The combination of these two results gives us an effective way to confine every seal. This is an important property from a security viewpoint as it means it is always possible for the owner of a system to curtail all communication abilities of programs running on his or her machine.

We also contrast our calculus with Ambient Calculus [3] for which barbed congruence with \( 0 \) does not imply confinement, and discuss the security implications of this fact.
Lemma 9 If $P$ is confined then for every substitution $\sigma$, if $P\sigma \rightarrow Q$, then $Q$ is confined.

Proof Suppose by contradiction that there exists $Q$ and $\sigma$ such that $P\sigma \rightarrow Q$ and $Q$ not confined. By definition of confinement $Q$ is immediately confined. But since it is not confined, then there must exist $Q'$ and $\sigma'$ such that $Q'$ is not immediately confined and $Q\sigma' \rightarrow Q'$. But then by Corollary 5 we have $(P\sigma)\sigma' \rightarrow Q'$ contradicting the fact that $P$ is confined.

The following lemma describes what happens when a context containing a process $P$ emits some label. In particular, it shows that there are three possible cases: either the label has been emitted by the process $P$ and it surfaced at the top-level without interacting with the context, or the context has emitted the label, without interacting with the process, or the process has emitted a barb that possibly interacted with the context.

Lemma 10 (Main Lemma) If $C[P] \xrightarrow{\ell} \omega Q$, then at least one of the following cases holds:

1. $Q \equiv C[P'] \land P \xrightarrow{\ell} \omega P'$
2. $Q \equiv C'[P\sigma] \land \forall R, C[R] \xrightarrow{\ell} \omega C'[R\sigma]$
3. $P \downarrow \beta$ for some barb $\beta$.

Proof The result follows by a straightforward (but long) induction on the depth of the derivation of $C[P] \xrightarrow{\ell} \omega Q$, by performing a case analysis on the last applied rule.

Lemma 11 If $C[0] \downarrow \beta$ then for every process $P$, $C[P] \downarrow \beta$

Proof By induction on the depth of the derivation of $C[0] \downarrow \beta$:

(depth=1) Then $C[0] \downarrow \beta$. The result follows by Lemma 7 and the observation that there does not exist any barb $\gamma$ such that $0 \downarrow \gamma$.

(depth>1) Then $C[0] \rightarrow P$ and $P \downarrow \beta$. By Theorem 3 the reduction above implies that $C[0] \rightarrow \epsilon Q$ and $Q \equiv P$. We can thus use Lemma 10; in particular since only point (2) of Lemma 10 applies we deduce that $C'[0] \equiv Q \equiv P$ and

$$\forall R, C[R] \rightarrow \epsilon C'[R\sigma]$$

Since $P \downarrow \beta$ then by transitivity of $\equiv$ and Lemma 6 $C'[0] \downarrow \beta$ and it is possible to apply the induction hypothesis, deducing that for all $R, C'[R] \downarrow \beta$. The result follows by combining this last result with (3).

Lemma 12 If $P \approx 0$ then $P$ is immediately confined

Proof Let $C[\ ]$ be any context. By Theorem 3 if $C[P] \rightarrow Q$ then $C[P] \rightarrow \epsilon Q'$ with $Q' \equiv Q$. We can thus use Lemma 10; in particular since $P \approx 0$ then the case 3 of this lemma cannot apply, and therefore by Theorem 3 $P$ satisfies the definition of immediate confinement.
Lemma 13 If $P \approx 0$ and $P \sigma \rightarrow Q$ then $Q \approx 0$

Proof We have to prove that for all context $C[\_], C[Q] \downarrow \beta \iff C[0] \downarrow \beta$.

The implication ($\iff$) follows immediately from Lemma 11. The converse is proved by induction on the depth of the derivation of $C[Q] \downarrow \beta$:

\textbf{(depth=1) } Then $C[Q] \downarrow \beta$ By Lemma 7 there are two possible sub-cases:

- $Q \downarrow \beta$. This case is impossible since this would imply that $P \sigma \downarrow \beta$. But if you consider the context $D[\_] = e(\lambda \vec{x}).[\_] | \tau(\sigma \vec{x})$ with $\vec{x} = dom(\sigma)$ and $c \notin fn(\beta)$ then we would have $D[P] \downarrow \beta$ and $D[0] \not\not \downarrow \beta$ that contradicts the hypothesis $P \approx 0$

- $\forall R, C[R] \downarrow \beta$, thus in particular $C[0] \downarrow \beta$.

\textbf{(depth> 1) } Then $C[Q] \rightarrow Q'$ and $Q' \downarrow \beta$. By Theorem 3, transitivity of $\equiv$ and Lemma 10 there are three possible cases:

1. $Q' \equiv C[Q'']$ and $Q \rightarrow Q''$. Then $P \sigma \rightarrow Q''$. Since $Q' \downarrow \beta$, then by Lemma 6 we have $C[Q''] \downarrow \beta$; furthermore it is possible to apply the induction hypothesis yielding $C[0] \downarrow \beta$.

2. $Q' \equiv C'[Q\sigma']$ and

$$\forall R, C[R] \rightarrow C'[R\sigma']$$ (4)

By Corollary 5, $(P\sigma)\sigma' \rightarrow Q\sigma'$. Since $Q' \downarrow \beta$, then by Lemma 6, $C'[Q\sigma'] \downarrow \beta$. By induction hypothesis we deduce then that $C'[0] \downarrow \beta$. Therefore by (4) we obtain $C[0] \downarrow \beta$.

3. $Q \downarrow \gamma$. We already showed in the base case of the induction that this case is impossible.

$\square$

Theorem 14 $P \approx 0$ if and only if $P$ is confined

\textbf{(only if) } By Lemma 12, $P \approx 0$ implies that $P$ is immediately confined. Let $\sigma$ be any substitution and $P'$ be a process such that $P\sigma \rightarrow P'$. By Lemma 13, $P' \approx 0$. By applying once more Lemma 12 we deduce that also $P'$ is immediately confined. In summary we have that $P$ is immediately confined and for all $\sigma$, if $P\sigma \rightarrow P'$ then $P'$ is immediately confined. Then, by definition, $P$ is confined.

\textbf{(if) } We suppose that $P$ is confined and prove that for all context $C[\_], C[P] \downarrow \beta \iff C[0] \downarrow \beta$

- ($\leftarrow$) Immediate from Lemma 11

- ($\Rightarrow$) By induction on the depth of the derivation of $C[P] \downarrow \beta$.

Proposition 15 $(\nu x)x[P] \approx 0$
Proof. It is easy to prove by Lemma 10 that \((\nu x)x[P]\) is confined. Indeed \((\nu x)x[P]\) cannot exhibit any barb since all barbs that can be exhibited by \(x[P]\) contain \(x\) as free name. Thus only the first two cases of Lemma 10 apply and, by Theorem 3 \((\nu x)x[P]\) is confined. □

This proposition provides us with an effective way to confine a seal, whatever it may contain or do: just restrict its name. This operation is always possible in the Seal calculus since the environment always knows the name of a seal. Indeed every seal is either native (and therefore it is known by the environment) or it arrived by a move (but then it has the name the environment assigned it.)

In the Ambient calculus it is possible to prove a property similar to the one of Proposition 15, and that Cardelli and Gordon christen the perfect firewall equation:

\[
x \not\in fn(P) \Rightarrow (\nu x)x[P] \approx 0
\]

The extra requirement on free names is necessary because of the different philosophy underlying the Ambient calculus. In Ambient calculus an ambient is not moved by its environment but it autonomously initiates its own moves. In particular, if it knows its own name (i.e., \(x \not\in fn(P)\)) it can rename itself and, in case it were bound by a restriction, escape it.

Contrary to Seal calculus, in Ambient calculus barbed congruence with 0 does not imply confinement, since the ambient is still able to interact with all the ambients whose name it is aware of. In particular it can use the action \(\text{out}\) to escape the current ambient since

\[
m[P \mid (\nu n)n[\text{out} m.Q]] \rightarrow (\nu n)(m[P] \mid n[Q])
\]

is a valid ambient reduction even if \(n \not\in fn(Q)\). Similarly by the action \(\text{in}\) it can enter a sibling ambient and use it as a Trojan horse (for escaping rather than entering):

\[
m[P \mid (\nu n)n[\text{in} m.Q] \rightarrow (\nu n)m[P \mid n[Q]]
\]

and such a technique can be used to circumvent the technique (efficacious in the first case) of using use of different names (i.e., passwords) for entering and exiting an ambient.

It is true that as long as \((\nu n)n[P]\) evolves in a trusted environment it is not allowed to communicate any information in its possession, say, a stolen secret. But since it can (nearly) freely move it is much more difficult to ensure that it will not enter a distrusted ambient where this information could be communicated. For ensuring that, it would be necessary to perform a textual examination of the ambient and check that the only free names it knows are those of trusted ambients.

In any case, the “perfect firewall equation” of the Ambient calculus already requires textual examination of the ambient at issue in order to verify that the it does not know its own name. This is due to the fact that in the Ambient calculus ambient names are used like passwords rather than denotations, and therefore they cannot be freely set by the environment. We believe that the search of free names of an ambient would be awkward in an Internet-wide distributed system. The need of such a search contrasts with the simple approach of the seal calculus where a modest restriction provides confinement, independently from the contents and the behavior of the seal.
In this section we discussed the case of linear Seal calculus. We believe that these results naturally generalize to the full case. This generalization is at the moment of writing, under work. The difficulty is that in presence of replication the hole of a context may be duplicated, so it becomes hard to trail the contents of the hole. In particular Lemma 10 becomes false. We are actually exploring the following alternatives.

1. Define **linear contexts** (that is, contexts in which the hole is not in the scope of a replication and all receive actions have a single parameter) and use them instead of generic contexts $C[ ]$ in Definition 3.
2. Use multi-hole contexts to generalize Lemma 10.
3. Redefine confinement in terms of residual of a process, in the spirit of the residuals of $\lambda$-calculus.

### 7 Related work

The difficulties of modeling some key aspects of distributed computing, in particular failures, within the $\pi$-calculus have driven a number of researchers to specifying distributed extensions [2, 1, 10]. The Linda model was extended with explicit localities and the ability to evaluate (dynamically scoped) processes at a given locality [5]. The distributed join-calculus of Fournet and Gonthier is a calculus specially designed for a distributed implementation on trusted networks [6] in which every channel is rooted at a given location. All of these calculi adopt a higher level view than the Seal calculus, allowing direct communication with remote processes. In programming terms this means that a mobile entity may always communicate with its creator and thus leak any information that it gleaned along the way. So policies such as the strong sandbox of Java 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 much more sensitive to failures. Our goal with mobility is to emphasize local interaction. We have already mentioned the Ambient calculus [3], but several other calculi have been developed recently with related goals [11, 7].

Several researchers have proposed to rely on types for resource access control [7, 4, 11]. The work of Hennessy and Riely is innovative as it deals with open networks where a subset of hosts may be malicious. This raises very interesting problems: for instance, handing out an ill-typed value to a mobile application can not be detected right away if the value is a non-local channel name but may break the application later on. Their type system detects these kinds of error before the value is used, but this remains a genuine attack (if the goal is to prevent the mobile agent from carrying out its task). In the Seal calculus we did not choose types for controlling access to resources as we feel that resource allocation in real systems is very dynamic, typically the set of resources (memory, communication channels, cpu time, etc.) available to an entity will evolve over time. Types appear too rigid to model this aspect well. For our part, we plan to study the use of type systems for constraining other characteristics of the behavior of seals.
8 Conclusions

The Seal calculus is a calculus of mobile computations which is well suited to the task of modeling Internet applications since some of the key elements of such applications, namely, parallelism, locations, mobility and protection boundaries, are represented directly in the calculus. In this paper we have given a commitment semantics to the Seal calculus and proved that this semantics corresponds to the standard reduction semantics of [14]. The difficulty of the proof and, to some extent, the complexity of the semantics stem from the higher order nature of the calculus and in particular the presence of seal copying capabilities (which demands the introduction of the heating relation).

In this paper, we also formulated an important security property called confinement that we hope to generalize to generic process algebras since it provides a characterization of processes that are unable to interact with their environment. Finally, we have shown that in (linear) Seal calculus \( (\nu x) x[P] \) is confined.

In the future we plan to continue our investigation of notions of equivalence and look at the use of types to ease the proof of security properties of agents. In parallel, we are implementing a Java–based system which incorporates many of the ideas developed in these pages [13].

Acknowledgments

Our work benefited greatly from multiple discussion with Martín Abadi, Luca Cardelli, Andrew Gordon and Peter Sewell, we gratefully acknowledge their suggestions and comment which have improved our work in many ways. Furthermore, we thank Cedric Fournet for spotting a mistake in the statement of confinement. Jan Vitek is funded by the Swiss PP project ASAP: Agent System Architectures and Platforms No 5003-45335.

References


Appendix A

We define free names of processes and agent prefixes and bound names of agent prefixes.

<table>
<thead>
<tr>
<th>Table 13: Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$fn(0) = \emptyset$</td>
</tr>
<tr>
<td>$fn((x) P) = fn(P) \setminus {x}$</td>
</tr>
<tr>
<td>$fn(P</td>
</tr>
<tr>
<td>$fn(!P) = fn(P)$</td>
</tr>
<tr>
<td>$fn(\lambda y.P) = {x, \eta} \cup {\bar{y}} \cup fn(P)$</td>
</tr>
<tr>
<td>$fn(\bar{y} x.P) = {x, \eta} \cup \bar{y} \cup fn(P)$</td>
</tr>
<tr>
<td>$fn(\bar{y} x.P) = {x, \eta} \cup \bar{y} \cup fn(P)$</td>
</tr>
<tr>
<td>$fn(\bar{y} x.P) = {x, \eta} \cup \bar{y} \cup fn(P)$</td>
</tr>
<tr>
<td>$fn(\bar{y} x.P) = {x, \eta} \cup \bar{y} \cup fn(P)$</td>
</tr>
<tr>
<td>$fn(\bar{y} x.P) = {x, \eta} \cup \bar{y} \cup fn(P)$</td>
</tr>
</tbody>
</table>

In the rules above $\eta$ is in $fn$ only if it is a name (not $\uparrow$ or $\ast$).
Table 14: Agent Prefixes

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Expression</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>fn</td>
<td>((\nu x)(\check{y}))</td>
<td>(\check{y}\backslash x)</td>
</tr>
<tr>
<td>bn</td>
<td>((\nu x)(\check{y}))</td>
<td>(x)</td>
</tr>
<tr>
<td>fn</td>
<td>((\nu x)(P))</td>
<td>(fn(P)\backslash x)</td>
</tr>
<tr>
<td>bn</td>
<td>((\nu x)(P))</td>
<td>(x)</td>
</tr>
<tr>
<td>fn</td>
<td>((\lambda y))</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>bn</td>
<td>((\lambda y))</td>
<td>(\check{y})</td>
</tr>
<tr>
<td>fn</td>
<td>((\lambda X))</td>
<td>(\emptyset)</td>
</tr>
<tr>
<td>bn</td>
<td>((\lambda X))</td>
<td>(\emptyset)</td>
</tr>
</tbody>
</table>

Note that in the case of concretions of process values, the rule for bound names disregards the contents of \(P\) as bound variables can only escape from a seal body during communication.

Appendix B

We define substitutions of names over processes and locations. Let \(\sigma = \{z'/z\}\).

Table 15: Substitutions

\[
\begin{align*}
\uparrow \sigma &= \uparrow & \star \sigma &= \star & x \sigma &= x \text{ if } x \neq z & z \sigma &= z'
\end{align*}
\]

<table>
<thead>
<tr>
<th>Expression</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0\sigma)</td>
<td>(0)</td>
</tr>
<tr>
<td>((\nu x)P\sigma)</td>
<td>((\nu x)P\sigma\text{ if } x \neq z, x \neq z')</td>
</tr>
<tr>
<td>((P \mid Q)\sigma)</td>
<td>(P\sigma \mid Q\sigma)</td>
</tr>
<tr>
<td>((!P)\sigma)</td>
<td>(!P\sigma)</td>
</tr>
<tr>
<td>(x^n(\lambda y).P\sigma)</td>
<td>(x^n\sigma^{y\sigma}(\lambda y).P\sigma\text{ if } z \notin y, z' \notin \check{y})</td>
</tr>
<tr>
<td>(\bar{x}^n(\check{y}).P\sigma)</td>
<td>(\bar{x}^n\sigma^{y\sigma}(y\sigma).P\sigma)</td>
</tr>
<tr>
<td>(x^n(\check{y}).P\sigma)</td>
<td>(x^n\sigma^{y\sigma}(\check{y}\sigma).P\sigma)</td>
</tr>
<tr>
<td>(\bar{x}^n(y).P\sigma)</td>
<td>(\bar{x}^n\sigma^{y\sigma}(y\sigma).P\sigma)</td>
</tr>
<tr>
<td>(\text{open}_n x.P\sigma)</td>
<td>(\text{open}_n x\sigma^{x\sigma}(P\sigma))</td>
</tr>
<tr>
<td>(x[P]\sigma)</td>
<td>(x\sigma[P\sigma])</td>
</tr>
<tr>
<td>(x[X]\sigma)</td>
<td>(x\sigma[X])</td>
</tr>
</tbody>
</table>

Appendix C

We define structural congruence rules over agent prefixes.
<table>
<thead>
<tr>
<th>Table 16: Structural congruence of agent prefixes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega \equiv \omega$</td>
</tr>
<tr>
<td>$\omega \equiv \omega' \Rightarrow \omega' \equiv \omega$</td>
</tr>
<tr>
<td>$\omega \equiv \omega', \omega' \equiv \omega'' \Rightarrow \omega \equiv \omega''$</td>
</tr>
<tr>
<td>$P \equiv Q \Rightarrow (\nu \bar{x})(\nu \bar{x})P \equiv (\nu \bar{x})(\nu \bar{x})Q$</td>
</tr>
<tr>
<td>$\omega \equiv \omega' \Rightarrow (\nu x)\omega \equiv (\nu x)\omega'$</td>
</tr>
<tr>
<td>$\omega \equiv \omega' \Rightarrow (\nu y)(\nu x)\omega \equiv (\nu x)(\nu y)\omega'$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 17: Structural congruence of agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega P \equiv \omega P$</td>
</tr>
<tr>
<td>$\omega P \equiv \omega' Q \Rightarrow \omega' Q \equiv \omega P$</td>
</tr>
<tr>
<td>$\omega P \equiv \omega' Q, \omega' Q \equiv \omega'' R \Rightarrow \omega P \equiv \omega'' R$</td>
</tr>
<tr>
<td>$\omega \equiv \omega', P \equiv Q \Rightarrow \omega P \equiv \omega' Q$</td>
</tr>
</tbody>
</table>

The following properties will prove useful in our later work. The first states that structurally congruent terms have the same free variables.

**Lemma 16**

1. If $P \equiv Q$ then $fn(P) = fn(Q)$.
2. If $\omega_1 \equiv \omega_2$ then $fn(\omega_1) = fn(\omega_2)$.

**Proof** By induction on the depths of the derivations of $P \equiv Q$ and $\omega_1 \equiv \omega_2$, respectively. Note that only reflexivity holds for $x[X]$.

The following is a substitution lemma for structurally congruent terms.

**Lemma 17**

1. If $P \equiv Q$, then $P\{y/x\} \equiv Q\{y/x\}$.
2. If $P \equiv Q$ and $R \equiv S$, then $P\{R/x\} \equiv Q\{R/x\}$.
3. If $\omega P \equiv \omega' Q$, then $\omega P\{y/x\} \equiv \omega Q\{y/x\}$.
4. If $\omega P \equiv \omega' Q$ and $R \equiv S$, then $\omega P\{R/x\} \equiv \omega Q\{R/x\}$.

**Proof** By induction on the structure of the derivation of $P \equiv Q$ and $\omega P \equiv \omega' Q$, respectively.

The last lemma simply states that reduction and heating can only decrease the set of free names of a term.

**Lemma 18** If $P \rightarrow Q$ or $P \rightarrow x^n.\omega Q$, then $fn(Q) \subseteq fn(P)$.
For point 3 we have (Comm In) ones: (Comm PreSend), (Comm Recv), (Comm Port), and (Seal Send)— followed by the inductive analysis on the last applied rule, starting with the base cases — i.e., (Comm In), (Comm Out), (Comm PreSend), (Comm Recv), (Comm Port), and (Seal Send)— followed by (Heat In) we get P, by setting x = x_1 ∪ x_2, x_1 = x ∩ fin(P_1), and x ∉ x.

Appendix D — Equivalence between reduction and commitment

Lemma 19 If P ⊩ Q then P ⊩ Q.

Proof By inspection of the rules of ⊩; every rule is matched by an equivalent rule in ⊩.

Lemma 20 (Genericity Lemma)

1. If P x → ω Q then ω = ϵ.

2. If P ⊢ x₀ → ω Q then ω = (ν x)(P_1), P ⊢ (ν x)(x[P_1] | P_2) and Q ⊢ (ν x)P_2 where x = x_1 ∪ x_2, x_1 = x ∩ fin(P_1), and x ∉ x.

3. If P x₀ → ω Q then ω = (λ x)(λ x_1)(P_1) or ω = (λ x)(λ x_1)(x[P_1]), and there exist R, S such that R ⊢ P, S ⊢ Q and R ≤ x₀.ωS.

4. If P x₀ → ω Q then ω = (ν x)(P_1) or ω = (ν x)(x[P_1]), and there exist R, S such that R ⊢ P, S ⊢ Q, and R ≤ x₀.ωS.

5. If P x₀(x) → ω Q then ω = ϵ, P ⊢ (ν x)(x)(P_1), Q ⊢ (ν x)(x)(P_1 | P_2), and x ∩ {x, y} = φ.

6. If P x₀ → ω Q then ω = ϵ, P ⊢ (ν x)(open x)(P_1 | P_2), Q ⊢ (ν x)(x)(P_1 | P_2), and x ∩ {x, y} = φ.

Proof Simultaneously by induction on the depth of the commitment derivation. We proceed by analysis on the last applied rule, starting with the base cases — i.e., (Comm In), (Comm Out), (Comm PreSend), (Comm Recv), (Comm Port), and (Seal Send)— followed by the inductive ones:

(Comm In) For point 3 we have P = x₀(λ y).Q and P x_0 → (λ y)Q. Let R = P and S = Q, by (Heat In) we get P ⊢ x₀. (λ y)Q, that is the result. The remaining points do not apply and, thus, are trivially satisfied.

(Comm Out) For point 4 we have P = x_0(y).Q and P x_0 → (y)Q. Let R = P and S = Q, by (Heat Out) we get P ⊢ x₀. (y)Q, that is the result. The remaining points do not apply and, thus, are trivially satisfied.

(Comm PreSend) For point 5 we have P = x_0(y).Q and P x_0 → εQ. The result is obtained by setting P_1 = Q, P_2 = 0, and x the empty vector. The remaining points do not apply.

(Comm Recv) Only point 3 applies. Let y = y_1 . . . y_n, we have P = x_0(y).Q, and P x_0 → (λ y)(Q | y_1[X] | . . . | y_n[X]). Let R = P, by (Heat Recv) we get P ⊢ x_0, (λ y)(Q | y_1[X] | . . . | y_n[X]). The result follows by setting S = Q = Q’ | y_1[X] | . . . | y_n[X].
(Comm Port) Only point 6 applies. We have \( P = \text{open}_\eta x \cdot Q \) and \( P \xrightarrow{\text{open}_\eta x} \epsilon Q \). The result follows by setting \( P_1 = Q \), \( P_2 = 0 \), and \( \vec{x} \) the empty vector.

(Comm Seal Send) Only point 2 applies and we have \( P = x[P_1], \omega = \langle P_1 \rangle \), and \( Q = 0 \). The result follows by setting \( P_2 = 0 \) and \( \vec{x} \) the empty vector.

(Comm Seal Up) We have \( P = y[P'], Q = y[Q'], \eta = y[] \), \( y \not\in bn(\omega) \) and \( P' \xrightarrow{x} \omega Q' \). Both 3 and 4 apply:

3. Then \( x = x \). By induction hypothesis, \( \omega = \langle \lambda X \rangle \) or \( \omega = \langle \lambda \vec{x} \rangle \) and there exist \( R', S' \) such that \( R' \equiv P', S' \equiv Q' \), and \( R' \prec x^\uparrow \cdot \omega S' \). Since \( y \not\in bn(\omega) \) we can apply (Heat Seal-1) obtaining \( y[R'] \prec x^\uparrow \cdot \omega S' \). The result follows by setting \( R = y[R'] \), \( S = y[S'] \) and observing that by (Struct Seal) \( P = y[P'] \equiv y[R'] = R \) and \( Q = y[Q'] \equiv y[S'] = S \).

4. As the point above. Only prefixes change.

(Comm Seal Down) As the previous cases. Just substitute \( \overset{\uparrow}{\uparrow} \) for \( \uparrow \) and \( \overline{y} \) for \( y \) and use (Heat Seal-2) instead of (Heat Seal-1)

(Comm Seal \( \tau \)) Only point 1 applies and it is straightforwardly satisfied.

(Comm Open) We have \( P = P_1 | P_2, Q = Q_1 | Q_2, P_1 \xrightarrow{x^*} \omega Q_1, P_2 \xrightarrow{\text{open}_\eta x} \epsilon Q_2 \), and \( bn(\omega) \cap fn(Q_2) = \emptyset \). Both 3 and 4 apply:

3. Then \( x = x \) and we have to prove that there exist \( R \equiv P_1 | P_2 \) and \( S \equiv Q_1 | Q_2 \) such that \( R \prec x^\tau \cdot \omega S \). By induction hypothesis we immediately obtain that \( \omega = \langle \lambda X \rangle \) or \( \omega = \langle \lambda \vec{x} \rangle \). Furthermore, again for induction hypothesis, there exist \( R_1, R_2, R_3, R_4 \) such that

\[
R_1 \prec x^* \cdot \omega R_2
\]  
\[
P_1 \equiv R_1
\]  
\[
Q_1 \equiv R_2
\]  
\[
P_2 \equiv (\nu \vec{x})(R_3 | \text{open}_\eta x.R_4)
\]  
\[
Q_2 \equiv (\nu \vec{x})(R_3 | R_4)
\]

We can consider (by performing silent alpha conversions) that \( \vec{x} \not\in fn(P_1) \cup bn(\omega) \). Whence we deduce (by Lemmas 16 and 18) that \( \vec{x} \not\in fn(R_1) \cup fn(R_2) \).

Set \( R = (\nu \vec{x})(R_1 | R_3 | \text{open}_\eta x.R_4) \) and \( S = (\nu \vec{x})(R_2 | R_3 | R_4) \). Since \( \vec{x} \not\in fn(R_1) \cup fn(R_2) \) then by (Struct Res Par) \( P \equiv R \) and \( Q \equiv S \). To obtain the result we just need to prove that \( R \prec x^\tau \cdot \omega S \).

From \( bn(\omega) \cap fn(Q_2) = \emptyset \) and Lemma 16 we deduce that

\[
bn(\omega) \cap fn(R_3) = bn(\omega) \cap fn(R_4) = \emptyset
\]

We can thus apply (Heat Par) to (5) and obtain

\[
(R_1 | R_3) \prec x^* \cdot \omega (R_2 | R_3)
\]
From (10) and (11) we deduce
\[
(R_1 | R_3) \mid \text{open}_\eta x.R_4 \prec x^\eta \cdot \omega(R_2 | R_3 | R_4) \tag{12}
\]
Finally by \(\bar{x} \notin bn(\omega)\), (12), and (Heat Res-1), we obtain the result:
\[
(\nu \bar{x})(R_1 | R_3 | \text{open}_\eta x.R_4) \prec x^\eta \cdot \omega(\nu \bar{x})(R_2 | R_3 | R_4)
\]

4. As the previous point. Only prefixes change.

(Comm Local), (Comm Remote) Only point 1 applies and it is trivial to verify that \(\omega = \epsilon\).

(Comm Send) This is the most difficult case. Only point 4 applies. The rule is
\[
\frac{P_1 \xrightarrow{\pi(y)} \epsilon Q_1 \quad P_2 \xrightarrow{y} Q_2}{P_1 | P_2 \xrightarrow{\pi'} \omega(Q_1 | Q_2) \quad bn(\omega) \cap fn(Q_1) = \emptyset} \quad \text{(Comm Send)}
\]
where \(P = P_1 | P_2\) and \(Q = Q_1 | Q_2\). We have to prove that there exist \(R \equiv P\), \(S \equiv Q\), \(\bar{z}\), and \(R'\) such that \(R \prec \pi^n \cdot \omega S\) and \(\omega = (\nu \bar{z})(R')\).

By induction hypothesis we have:
\[
P_1 \equiv (\nu \bar{y})(\pi^n(y) . R_1 | R_2)
Q_1 \equiv (\nu \bar{y})(R_1 | R_2)
\omega = (\nu \bar{x}_1)(R_4)
P_2 \equiv (\nu \bar{x})(R_3 | y[R_4])
Q_2 \equiv (\nu \bar{x}_2)R_3
\]
where \(\bar{x} = \bar{x}_1 \cup \bar{x}_2, \bar{x}_1 = \bar{x} \cap fn(R_4)\), and \(y \notin \bar{x}\).

We can perform silent alpha conversions and thus consider without loss of generality that
\[
\bar{y} \notin fn(R_3) \cup fn(R_4) \cup \{y\} \land \bar{x} \notin fn(R_1) \cup fn(R_2) \cup \{x, \eta\} \tag{13}
\]

Now, set:
\[
R = (\nu \bar{y})(\nu \bar{x})(\pi^n(y) . R_1 | y[R_4] | R_2 | R_3)
S = (\nu \bar{y})(\nu \bar{x})(R_1 | R_2 | R_3)
\bar{z} = \bar{x}_2
R' = R_4
\]

From (13) we obtain
\[
R \equiv (\nu \bar{y})(\pi^n(y) . R_1 | R_2) \mid (\nu \bar{x})(R_3 | y[R_4]) = P
\]
and
\[
S \equiv (\nu \bar{y})(R_1 | R_2) \mid (\nu \bar{x}_2)R_3 = Q
\]
Thus it remains to prove that \(R \prec \pi^n \cdot \omega S\)

By (Heat Send) we have
\[
\pi^n(y) . R_1 | y[R_4] \prec \pi^n \cdot (R_4)_1 R_1
\]
From this and the fact that $bn(\langle R_1 \rangle) = \emptyset$ we deduce by (Heat Par)

$$\bar{x}^\eta(y).R_1 \mid y[R_1] \mid R_2 \mid R_3 \prec \bar{x}^\eta(\langle R_1 \rangle(\langle R_1 \rangle(\langle R_1 \rangle(\langle R_1 \rangle)))$$

Since $\bar{x}_1 \in fn(\langle R_1 \rangle)$ and $\bar{x}_2 \not\in fn(\langle R_1 \rangle)$, it is possible to apply (Heat Res-1) and (Heat Res-2) obtaining

$$(\nu \bar{x})(\bar{x}^\eta(y).R_1 \mid y[R_1] \mid R_2 \mid R_3) \prec \bar{x}^\eta. (\nu \bar{x}_1)(\langle R_1 \rangle(\nu \bar{x}_2)(\langle R_1 \rangle(\langle R_1 \rangle)))$$

By (13) $\bar{y} \not\in fn(\langle \bar{x}^\eta, \bar{x}_1 \rangle(\langle R_1 \rangle))$, thus we can apply (Heat Res-1) yielding the result:

$$(\nu \bar{y})(\nu \bar{x})(\bar{x}^\eta(y).R_1 \mid y[R_1] \mid R_2 \mid R_3) \prec \bar{x}^\eta. (\nu \bar{x}_1)(\langle R_1 \rangle(\nu \bar{y})(\nu \bar{x}_2)(\langle R_1 \rangle(\langle R_1 \rangle)))$$

**Comm $\equiv$** All the points apply and they all follow by a straightforward application of the induction hypothesis and of Lemma 19.

**Comm Par, Comm Res-1** Straightforward application of the induction hypothesis.

**Comm Res-2** Note that the point 1, 3, 5, 6 do not apply because by induction $fn(\omega) = \emptyset$.

The cases 2 and 4 are proved by a straightforward application of the induction hypothesis.

**Comm Repl** Use the induction hypothesis. For the last four points simply note that $P \mid \mid P \equiv \mid P$.

\[
\begin{proof}
\end{proof}

\[\text{Lemma 21} \text{ If } P \overset{x}{\rightarrow} \omega Q \text{ then there exist } Q' \text{ such that } P \rightarrow Q' \text{ and } Q' \equiv Q\]

\[\text{Proof} \quad \text{First note that by genericity (Lemma 20) we have } \omega \equiv \epsilon. \text{ We prove this lemma by induction on the derivation of } P \overset{x}{\rightarrow} \epsilon Q \text{ performing a case analysis on the last applied rule.} \]

**Comm Local** We have $P = P_1 \mid P_2, P_1 \overset{x}{\rightarrow} \omega_1 P'_1, P_2 \overset{x}{\rightarrow} \omega_2 P'_2$, and $Q = (\omega_1 P'_1) \bullet (\omega_2 P'_2)$.

By genericity we have two possible subcases:

1. $\omega_1 = \langle \lambda \bar{y} \rangle, \omega_2 = (\nu \bar{x})(\bar{z}), Q = (\nu \bar{z})(P'_1 \{\bar{z}/y\} \mid P'_2)$ and

$$R_1 \prec x^* \cdot \omega_1 R'_1 \quad R_2 \prec x^* \cdot \omega_2 R'_2$$

with $P_i \equiv R_i$ and $P'_i \equiv R'_i$ for $i = 1, 2$. From (14) and (Red Local) it follows

$$R_1 \mid R_2 \rightarrow (\nu \bar{z})(R'_1 \{\bar{z}/y\} \mid R'_2)$$

By Lemma 17 $P'_1 \{\bar{z}/y\} \equiv R'_1 \{\bar{z}/y\}$. Thus since $\equiv$ is a congruence we have $P \equiv R_1 \mid R_2$ and $Q \equiv (\nu \bar{z})(R'_1 \{\bar{z}/y\} \mid R'_2$. The result follows by (Red $\equiv$)

2. $\omega_1 = \langle \lambda X \rangle, \omega_2 = (\nu \bar{x})(\bar{R}), Q = (\nu \bar{z})(P'_1 \{R/X\} \mid P'_2)$. As the case above.

**Comm Remote** As the previous case.
Straightforward application of the induction hypothesis (note that \(\omega = \epsilon\) and therefore \(bn(\omega) = \emptyset\)).

All remaining cases do not apply. \(\Box\)

**Lemma 22** If \(P \prec x^n \cdot \omega Q\) then there exists \(Q'\) such that \(Q \equiv Q'\) and \(P \xrightarrow{x^n} \omega Q'\).

**Proof** We prove this lemma by induction on the structure of a derivation \(P \prec x^n \cdot \omega Q\), by a case analysis on the last applied rule:

(\textbf{Heat Send}) By (Comm \textit{Pre Send}), (Comm \textit{Seal Send}), (Comm \textit{Send}) and (Struct \textit{Dead Par}).

(\textbf{Heat Open}) By induction hypothesis, (Comm \textit{Port}), and (Comm \textit{Open}).

(\textbf{Heat In}), (\textbf{Heat Out}), (\textbf{Heat Recv}) Immediate from the homonymous Comm rules.


\(\Box\)

**Lemma 23** If \(P \equiv Q\) then \((\omega P) \cdot (\omega' R) \equiv (\omega Q) \cdot (\omega' R)\) and \((\omega P) \equiv (\omega' R) \bullet (\omega Q)\)

**Lemma 24** If \(P \equiv P'\) and \(P \xrightarrow{l} \omega Q\), then there exists \(\omega' Q'\) such that \(P' \xrightarrow{l} \omega' Q'\) and \(\omega Q \equiv \omega' Q'\).

**Proof** By a fairly long induction on the depth of the derivation of \(P \xrightarrow{l} \omega Q\) by a case analysis of last rule of the derivation of \(P \equiv Q\):

(\textbf{Equivalence}) If the last rule is for reflexivity or symmetry then the result easily follows. If the last rule is for transitivity then the result follows from (Weak Trans) and (Comm \(\equiv\)).

(\textbf{Struct Par Comm}) By (Weak Par Comm) and (Comm \(\equiv\)).

(\textbf{Struct Par Assoc}) By (Weak Par Assoc) and (Comm \(\equiv\)).

(\textbf{Struct Repl Par}) By (Weak Repl) and (Comm \(\equiv\)).

(\textbf{Struct Res Par}) By (Weak Res Par) and (Comm \(\equiv\)).

(\textbf{Struct Dead Res}) We have \(P = \nu x 0\) and \(P' = 0\). Therefore the result follows vacuously since neither \(P\) does not have any commitment.

(\textbf{Struct Dead Par}) We have \(P = P' | 0\). In part (1), the commitment \(P \xrightarrow{l} \omega Q\) can be only obtained either by (Comm \(\equiv\)) or, since 0 has no commitments, by (Comm Par). In the former case the result follows from a straightforward application of the induction hypothesis and of Lemma 19. In the latter there is a \(\omega''\) such that \(P' \xrightarrow{l} \omega' Q''\) and \(\omega Q = \omega'(Q'' | 0)\). By induction hypothesis then we have that \(P' \xrightarrow{l} \omega' Q'\) with \(\omega' Q' \equiv \omega Q''\). The result follows by (Struct Dead Par).
This is the most difficult (or, rather, long) case. We have $P = C[\hat{P}]$, $P' = C[\hat{P}]$ and $P \equiv P'$ is deduced from $\hat{P} \equiv \hat{P}'$. We check all the possible cases of $C[]$

$(C[] = [])$ Immediate.

$(C[] = D[] | R)$ We perform a case analysis on the last rule of the deduction of $D[\hat{P}] | R \xrightarrow{\ell} Q$:

$(Comm \equiv)$ $D[\hat{P}] | R \equiv P''$ and $P'' \xrightarrow{\ell} \omega Q$. By Lemma 19, transitivity and reflexivity of $\equiv$ we obtain $D[\hat{P}] | R \equiv P''$. We can thus apply the induction hypothesis obtaining that there exists $\omega'Q' \equiv \omega Q$ such that $D[\hat{P}] \xrightarrow{\ell} \omega'Q'$.

$(Comm \ Open), (Comm \ Send)$ We have $\ell = x^T, D[\hat{P}] \xrightarrow{\omega'x} \omega R_1, R \xrightarrow{\omega'x} \epsilon R_2$, and $Q = \omega(R_1 | R_2)$. Since $\hat{P} \equiv \hat{P}'$ then $D[\hat{P}] \equiv D[\hat{P}]$, and we can apply the induction hypothesis obtaining $D[\hat{P}] \xrightarrow{\omega'x} \omega' R_1$ where $\omega R_1 \equiv \omega' R_1$. By silent alpha-conversion of $\omega' R_1$ we can have that $bn(\omega') \cap fn(R) = \emptyset$. We can thus apply (Comm Open) obtaining that $P'' \xrightarrow{\omega'x} \omega'(R_1 | R_2)$. The result follows from the definition of $\equiv$.

$(Comm \ Send)$ As the previous case.

$(Comm \ Local), (Comm \ Remote)$ These cases follow by induction hypothesis and Lemma 23.

The remaining cases do not apply.

$(C[] = R | D[])$ As the case before.

$(C[] = (\nu x)D[])$ There are only three possible rules that apply. If the last rule of the deduction of $P \xrightarrow{\ell} \omega Q$ is (Comm $\equiv$), then proceed as the corresponding case of $C[] = D[] | R$. If the last rule is (Comm Res-1) or (Comm Res-2) then use the induction hypothesis and Lemma 16 extended to agents.

$(C[] = \alpha . D[])$ We distinguish two subcases according to the last rule of the deduction of $P \xrightarrow{\ell} \omega Q$:

1. The last rule is one of the following: (Comm Port), (Comm Out), (Comm In), (Comm Pre Send), (Comm Recv). Then the result follows immediately from the fact that $D[\hat{P}] \equiv D[\hat{P}]$.
2. The last rule is (Comm $\equiv$): proceed as in the corresponding subcase of the previous cases.

$(C[] = !D[])$ The only possible case is that the commitment deduction ends by (Comm $\equiv$). Then the result follows as in the previous cases.

$(C[] = x[D[]])$ Again we proceed by a case analysis of the last rule of the derivation of $P \xrightarrow{\ell} \omega Q$. We can distinguish three subcases:

1. (Comm $\equiv$): as in the previous cases.
2. (Comm Seal Up), (Comm Seal Down), and (Comm Seal $\tau$): a simple application of the induction hypothesis, with an eventual silent alpha-conversion for the first two rules.
3. (Comm Seal Send): then we have \( x[D[\hat{P}] ] \xrightarrow{\approx} \langle D[\hat{P}] \rangle 0 \). By the same rule we obtain \( x[D[\hat{P}] ] \xrightarrow{\approx} \langle D[\hat{P}] \rangle 0 \), and by definition \( \langle D[\hat{P}] \rangle 0 \equiv \langle D[\hat{P}] \rangle 0 \). Just note that it is because of this case that we require \( \equiv \) to be defined on agents rather than just on processes.

The remaining rules do not apply.

(Struct Res Res) Proceed exactly as in case \( C[] = (\nu x)D[] \) of (Congruence).

\[ \Box \]

**Theorem 1** \( P \rightarrow Q \) if and only if there is \( R \equiv Q \) with \( P \xrightarrow{\tau} eR \).

**Proof** The backward direction follows from Lemma 21. For the forward direction, we show by induction on the depth of the derivation of \( P \rightarrow Q \) that there exists \( R \equiv Q \) such that \( P \xrightarrow{\tau} eR \). We start with the base cases.

(\textbf{Red Local}) We have \( P = P_1 | P_2, P_1 \prec x^\omega_1Q_1 \) and \( P_2 \prec x^\omega_2Q_2 \).

The result follows by a straightforward application of the induction hypothesis, of (Red Local), and of Lemma 23.

(\textbf{Red Remote}) The result is obtained as in the previous case.

For the inductive case.

(\textbf{Red Res}) We have \( (\nu x)P \rightarrow (\nu x)Q \) and must show that there is \( R \equiv Q \) such that \( (\nu x)P \xrightarrow{\tau} e(\nu x)R \). We also have that \( P \rightarrow Q \). By inductive hypothesis we may assume that \( P \xrightarrow{\tau} eR \) and \( R \equiv Q \). Since \( fn(e) = \emptyset \), only (Comm Res-1) may apply and we get \( (\nu x)P \xrightarrow{\tau} e(\nu x)R \).

(\textbf{Red Par}), (\textbf{Red Seal}) By inductive hypothesis and (Comm Par) and (Comm Seal) respectively. Note that for (Comm Par) the side condition is trivially true since \( bn(\omega) = \emptyset \).

(\textbf{Red \( \equiv \)}) We have \( P \equiv P_1, P_1 \rightarrow Q_1 \) and \( Q_1 \equiv Q \) implies \( P \rightarrow Q \). We must prove that there is an \( R \) such that \( P \xrightarrow{\tau} eR \) and \( R \equiv Q \). By induction hypothesis, we have \( P_1 \xrightarrow{\tau} eR \) and \( R \equiv Q_1 \equiv Q \). By Lemma 24, the fact that \( P_1 \equiv P \), and point (1) of Lemma 20 we obtain \( P \xrightarrow{\tau} eR' \) and \( R' \equiv R \equiv Q \).

\[ \Box \]