Safe Ambients: Abstract machine and distributed implementation

Paola Giannini\textsuperscript{a}, Davide Sangiorgi\textsuperscript{b,∗}, Andrea Valente\textsuperscript{c}

\textsuperscript{a}Dipartimento di Informatica, Università del Piemonte Orientale, Via Bellini 25/B, 15100 Alessandria, Italy
\textsuperscript{b}Dipartimento di Scienze dell’Informazione, Università di Bologna, Mura Anteo Zamboni 7, 40126 Bologna, Italy
\textsuperscript{c}Department of Software and Media Technology, Aalborg University Esbjerg, Niels Bohrs Vej 8, 6700 Esbjerg, Denmark

Received 1 April 2004; received in revised form 20 April 2005; accepted 2 May 2005
Available online 17 June 2005

Abstract

The abstract machine PAN for a distributed implementation of an ambient calculus is presented. PAN is different from, and simpler than, previous implementations of ambient-like calculi, mainly because: the underlying calculus is typed Safe Ambients (SA) rather than the untyped Ambient calculus and therefore does not present certain forms of interferences among processes (the grave interferences). In PAN the logical structure of an ambient system and its physical distribution are separated. A translation from SA terms to PAN terms is defined. The correctness of such a translation, which asserts that an SA term and its translation exhibit the same observational behavior, is proved. Moreover, a description of a distributed implementation of the abstract machine in Java is given.

Keywords: Safe ambients; Abstract machine; Proof of correctness; Distributed implementation

∗ Corresponding author.

\textit{E-mail addresses:} giannini@mfn.unipmn.it (P. Giannini), Davide.Sangiorgi@cs.unibo.it (D. Sangiorgi), av@cs.aue.auc.dk (A. Valente).

© 2005 Published by Elsevier B.V.

doi:10.1016/j.scico.2005.05.002
1. Introduction

The Ambient calculus [5] is a model for mobile distributed computing. An ambient is the unit of movement. Processes within the same ambient may exchange messages; ambients may be nested, so to form a hierarchical structure. The three primitives for movement allow: an ambient to enter another ambient, \( n[\text{in} m. P | Q | m[ R] \rightarrow m[ n[ P | Q ] | R ] \); an ambient to exit another ambient, \( m[ n[\text{out} m. P | Q ] | R ] \rightarrow n[ P | Q | m[ R ] \); a process to dissolve an ambient boundary thus obtaining access to its content, \( \text{open} n. P | n[ Q ] \rightarrow P | Q \).

Several studies of the basic theory of the Ambient calculus have recently appeared, concerning for instance behavioral equivalences, types, logics, static analysis techniques [6,7,1,8,17]. In comparison, little attention has been given to implementations. The only implementations of Ambients we are aware of are Cardelli’s [3,4], and Fournet, Lévy and Schmitt’s [10]. The latter, formalized as a translation of Ambients into the distributed Join Calculus, is the only distributed implementation. Although ingenious, the algorithms that these implementations use for simulating the ambient reductions are fairly complex.

One of the difficulties of a distributed implementation of an ambient-like language is that each movement operation involves ambients on different hierarchical levels. For instance, the ambients affected by an \text{out} operation are the moving ambient, and its initial and its final parent; at the beginning they reside on three different levels. In [3,4] locks are used to achieve a synchronization among all ambients affected by a movement. In a distributed setting, however, this lock-based policy can be expensive. For instance, the serializations introduced diminish the parallelism of the whole system. In [10] the synchronization is simulated by means of protocols of asynchronous messages. The problems of implementation have been a restraint to the development of programming languages based on Ambients and to experimentation of Ambients on concrete examples.

In our opinion, implementation is one of the aspects of Ambients that most need investigations.

In this paper we study an abstract machine for a distributed implementation of an ambient-like calculus, called PAN, Pervasive Ambient Network. The calculus we implement is the typed Safe Ambients [14] (SA) rather than untyped Ambients. SA is a variant of the original Ambient calculus that eliminates certain forms of interference in ambients, the grave interferences. They are produced when an ambient tries to perform two different movement operations at the same time, as for instance \( n[\text{in} h. P | \text{out} n. Q | R ] \).

The control of mobility is obtained in SA by a modification of the syntax and a type system. The type system allows two kinds of ambients: single-threaded and the immobile ambients, and a limited interaction between the two. In [14] the absence of grave interferences is used to develop an algebraic theory and prove the correctness of some examples. One of the contributions of this paper is to show that the absence of grave interferences also brings benefits in the implementations.

The algorithms of our abstract machine are quite different from, and simpler than, those of [3,4,10], mainly for two reasons. First, and most important, the calculus that we take is typed SA rather than untyped Ambients. The absence of grave interferences allows us to obtain important simplifications in the structure of the abstract machine and the protocols needed to realize the movement operations.
The second reason is that in our abstract machine there is a separation between the logical structure of an ambient system and its physical distribution. Exploiting this, the interpretation of the movement associated with the various operations is reversed: the movement of the open capability is physical, that is, the location of some processes changes, whereas that of in and out is only logical, that is, some hierarchical dependencies among ambients may change, but not their physical location. Intuitively, in and out are acquisition of access rights, and open is exercise of them. We may say that the movement given by in and out is first-order (some names move, and some pointers are updated), whereas the movement given by open is higher order (some terms move).

Our algorithms are formulated as an abstract machine. This leads to a relatively simple proof of correctness with respect to the one of the Join Calculus implementation. Moreover, the machine is independent of a specific implementation language, and can thus be used as a basis for implementations on different languages. In the paper we present one such implementation, written in Java.

As a final consideration we would point out that with PAN it becomes evident that Ambient-based calculi really need two kinds of operations, namely physical and logical ones. Creation of new subambients and open belong to the first category, being related to actual creation and destruction of unique elements in the tree of ambients; logical operations are the ones that only reshape the topology of the tree, i.e., in and out.

The paper is organized as follows. In Section 2 we introduce the syntax, semantics and relevant properties of the SA calculus. In Section 3 we introduce informally how the movement operations are realized in the abstract machine whose formal syntax and reduction relation are defined in Section 4. The proof of correctness for the fragment of SA including only single-threaded ambients is given in Section 5, and in Section 6 the proof is extended to the whole SA, which includes also immobile ambients. In Section 7 we describe the Java implementation of the abstract machine. Comparisons with the Join implementation are considered in Section 8, and in Section 9 we outline improvements and optimizations to the abstract machine and its implementation.

This paper is an expanded and improved version of a paper presented at ICALP 01, see [21].

2. Safe Ambients: Syntax and semantics

We briefly describe typed Safe Ambient (SA), from [14]. In the reduction rules of the original Ambient calculus, mentioned in Section 1, an ambient may enter, exit, or open another ambient. The second ambient undergoes the action; it has no control on when the action takes place. In SA this is rectified: coaction \( \overline{\text{in}} \, n \), \( \overline{\text{out}} \, n \), \( \text{open} \, n \) are introduced with which any movement takes place only if both participants agree. The syntax of SA is presented in Fig. 1. Expressions that are not variables or names are the capabilities.

These are divided into actions \( \text{actions} \ (\text{in} \, n, \text{out} \, n, \text{open} \, n) \) and coaction \( \text{coaction} \ (\overline{\text{in}} \, n, \overline{\text{out}} \, n, \overline{\text{open}} \, n) \). We often omit the trailing \( \textbf{0} \) in processes \( M.\textbf{0} \). Parallel composition has the least syntactic precedence, thus \( m[ M.\textbf{P} | Q ] \) reads \( m\{( M.\textbf{P}) | Q \} \). An ambient, or a parallel composition, or variable, is unguarded if it is not underneath a capability or an abstraction.
In a recursion $\text{rec } X. P$, the recursion variable $X$ should be guarded in $P$. For simplicity of presentation we omit path expressions in the syntax.

Below are the reduction axioms: those for movement, that for exiting, that for opening, and the communication rule (communication is asynchronous, takes place inside ambients, and is anonymous—it does not use channel or process names):

$$
\begin{align*}
&n[\text{in } m. P_1 | P_2] | n[\text{in } m. Q_1 | Q_2] \rightarrow m[n[P_1 | P_2] | Q_1 | Q_2] & \text{[R-IN]} \\
&m[n[\text{out } m. P_1 | P_2] | \text{out } m. Q_1 | Q_2] \rightarrow n[P_1 | P_2] | m[Q_1 | Q_2] & \text{[R-OUT]} \\
&\text{open } n. P | n[\text{open } n. Q_1 | Q_2] \rightarrow P[Q_1 | Q_2] & \text{[R-OPEN]} \\
&(M) | (x) P \rightarrow P[M/x] & \text{[R-MSG]}
\end{align*}
$$

Structural congruence ($\equiv$) is used to bring the participants of a potential interaction into contiguous positions; its definition is given below.

**Definition 2.1 (Structural Congruence).** Structural congruence is the smallest congruence $\equiv$ such that:

1. parallel composition, $\mid$, and $0$ form an abelian monoid;
2. $(wn) (P_1) | P_2 \equiv (wp) (P_1 | P_2)$ if $n$ not free in $P_2$;
3. $(wp) 0 \equiv 0$, and $(wp) (wq) P \equiv (vq) (wp) P$;
4. $\text{rec } X. P \equiv P[\text{rec } X. P/X]$.

The inference rules below allow a reduction to occur underneath a restriction, a parallel composition, and inside an ambient. Moreover, structural congruence relation can be applied before and after a reduction step.
\[
\begin{align*}
\frac{P \rightarrow P'}{(\nu n) P \rightarrow (\nu n) P'} & \quad \text{[R-Res]} \\
\frac{P \rightarrow P'}{P \mid Q \rightarrow P' \mid Q} & \quad \text{[R-Par]} \\
\frac{P \rightarrow P'}{n[P] \rightarrow n[P']} & \quad \text{[R-Amb]} \\
& \quad \text{[R-Struct]}
\end{align*}
\]

We write \(\Longrightarrow\) for the reflexive and transitive closure of \(\rightarrow\). The use of coaction, in the syntax and operational rules, is the only difference between (untyped) SA and the original Ambient calculus.

Up to structural congruence, every ambient in a term can be rewritten as
\[
(\nu n)[P_1 \mid \ldots \mid P_s \mid m_1[Q_1] \mid \ldots \mid m_r[Q_r]]
\]
where the \(P_i\) (\(i = 1 \ldots s\)) are neither ambients nor parallel compositions. In this case, \(P_1, \ldots, P_s\) are the local processes, and \(m_1[Q_1], \ldots, m_r[Q_r]\) are the subambients of the ambient \(n\).

SA has two main kinds of types: single-threaded and immobile. We consider them separately. We begin with the single-threaded types, which we informally describe below. We consider immobility types in Section 6.

The capabilities of the local processes of an ambient control the activities of that ambient. In an untyped (or immobile) ambient such control is distributed over the local processes: any of them may exercise a capability. In a single-threaded (ST) ambient, by contrast, at any moment at most one local process has the control thread, and may therefore use a capability. An ST ambient \(n\) is willing to engage in at most one interaction at a time with external or internal ambients. Inside \(n\), however, several activities may take place concurrently: for instance, a subambient may reduce, or two subambients may interact with each other. Thus, if an ambient \(n\) is ST, the following situation, where at least two local processes are ready to execute a capability, cannot occur: \(n[\text{in} m. P \mid \text{out} h. Q \mid R]\). The control thread may move between processes local to an ST ambient by means of an open action. Consider, for instance, a reduction
\[
n[\text{open} m. P \mid m[\text{open} m. Q]] \rightarrow n[P \mid Q]
\]
where \(n\) and \(m\) are ST ambients. Initially \(\text{open} m. P\) has the control thread over \(n\), and \(\text{open} m. Q\) over \(m\). At the end, \(m\) has disappeared; the control thread over \(n\) may or may not have moved from \(P\) to \(Q\), depending on the type of \(m\). If the movement occurs, \(Q\) can immediately exercise a capability, whereas \(P\) cannot; to use further capabilities within \(n\), \(P\) will have to get the thread back.

For simplicity, we assume here a strong notion of ST, whereby a value message \(\langle M \rangle\) never carries the thread. In [14] a weaker notion is used, where also messages may carry the thread. In this case, the control thread over an ambient may move, other than by an open operation, as a result of the consumption of a value by an abstraction. The results in our paper can be adapted to this weaker notion.
In the remainder, all processes are assumed to be well-typed, and closed, i.e., without free variables.

Let $P \equiv P_1 \mid \ldots \mid P_s$, where the $P_i$'s ($i = 1..s$) are not parallel compositions, be an SA term. The unguarded prefixes/capabilities of $P$ are the $M \in \{\text{in } n, \text{in } n, \text{out } n, \text{open } n, \text{open } n\}$ such that for some $i$, $P_i = (\nu \tilde{n}) M. P'_i$. A term $P'$ is an unboxed subcomponent of $P$ if $P'$ is a subterm $P$ such that $P'$ is not subterm of an ambient $n[Q]$. The main property that ST ambients have is expressed from the following statement (a simple variant of a theorem in [14]).

**Proposition 2.2.** If $P$ is a well-typed SA term and the subterm $n[Q]$ is a single-threaded ambient, then no unboxed subcomponent of $Q$ has more than one unguarded capability.

As proved in [14], well-typedness is preserved under reduction (Subject Reduction), so the previous property is true under reduction.

The expressiveness of SA is studied in [14,15]. By considering the main examples in the Ambient literature, it is argued that the modifications that lead from the original Ambient Calculus to SA do not reduce its expressiveness. In some cases, such as the encoding of channels, SA programs are actually simpler, since the use of cocapabilities and the absence of grave interferences allows a tighter control on the ambient movements.

3. The abstract machine, informally

We describe the data structures and the algorithms of the abstract machine, called PAN. PAN separates between the logical and the physical distribution of the ambients. The logical distribution is given by the tree structure of the ambient syntax (an ambient can contain other ambients). The physical distribution is given by the association of a location with each ambient.

In PAN, an ambient named $n$ is represented as a located ambient $h:n[P]_k$, where $h$ is the location, or site, at which the ambient runs, $k$ is the location of the parent of the ambient, and $P$ collects the processes local to the ambient. While the same name may be assigned to several ambients, a location univocally identifies an ambient; it can be thought of as its physical address.

A tree of ambients is rendered, in PAN, by the parallel composition of the (unguarded) ambients in the tree. In this sense, the physical and the logical topology are separated: the space of physical locations is flat, and each location hosts at most one ambient, each ambient resides at a distinct physical location (this gives us the physical distribution), but each ambient knows the location at which its parent resides (this gives us the logical topology). For instance, an SA term $n[P_1 \mid P_2 \mid m_1[Q_1] \mid m_2[Q_2]]$, where $P_1$ and $P_2$ are the local processes of $n$, and $Q_i$ $(i = 1, 2)$ is a local process of $m_i$ (i.e., $m_i$ has no subambients), becomes in PAN:

$$h:n[P_1 \mid P_2]_{\text{root}} \parallel k_1:m_1[Q_1]_{h} \parallel k_2:m_2[Q_2]_{h}$$

where $h$, $k_1$, $k_2$ are different location names, $\text{root}$ is a special name indicating the outermost location, and $\parallel$ is the parallel composition of located ambients. (The above configuration is actually obtained after two creation steps, in which the root ambient
spawns off the two ambients located at \( k_1 \) and \( k_2 \). Since ambients may run at different physical sites, they communicate with each other by means of asynchronous messages.

All the actions (in, out, and open) can modify the logical distribution. Only open, however, can modify the physical distribution. The algorithms that PAN adopts to model reduction in SA are based on three steps: first, a request message is sent upward, from a child ambient that wants to move (logically or physically) to its parent; second, a match is detected by the parent itself; third, a completion message is sent back to the child, for its relocation. The only exception is the algorithm for open, where a further message is needed to migrate the child’s local processes to the parent. These steps are sketched in Figs. 2–4, where \( a, b, c \) represent three ambients, a straight line represents a pointer from an ambient to its parent, and a curved line represents the sending of a message. Thus in Fig. 2, at the beginning \( a \) and \( b \) are sibling ambients and \( c \) is their parent. This figure illustrates an R-IN reduction in which \( a \) becomes a child of \( b \). In the first phase, \( a \) demands to enter \( b \) (precisely, if \( n \) is the name of \( b \), then \( a \) demands of entering an ambient with name \( n \)), and \( b \) accepts an ambient in. For this, \( a \) and \( b \) send requests in and in to their parent \( c \) (the actual messages may also contain the name and location of the sender; these are not shown in the figures). In the second phase, \( c \) sees that two matching requests have been sent and authorizes the movement. Finally, in the third phase, \( c \) sends completion messages to \( a \) and \( b \). The message sent to \( a \) also contains the location of \( b \), which \( a \) will use to update its parent field. An ambient that has sent a request to its parent but has not yet received an acknowledgement back, goes into a wait state, in which it will not send further requests. In the figures, this situation is represented by a circle that encloses the ambient. An ambient in a wait state, however, can still receive and answer requests from its children and can perform local communications.

Fig. 3 sketches an R-OUT reduction. In the first phase, ambient \( a \) demands its parent \( b \) to exit. When \( b \) authorizes the movement (phase 2), it sends \( a \) an acknowledgement
containing the location of the parent of \( b \), namely \( c \), and upon receiving this message (phase 3) \( a \) updates its parent field. Note that the grandparent ambient \( c \) is not affected by the dialog between \( a \) and \( b \). Fig. 4 sketches an \( R\)-OPEN reduction. Ambient \( a \) accepts to be opened, and thus notifies its parent \( c \). If a matching capability exists, that is, one of the processes local to \( c \) demands to open \( a \), then \( c \) authorizes \( a \) to migrate its local processes into \( c \). Ambient \( a \) then becomes a forwarder (\( a \triangleright c \) in the figure) whose job is just to forward any messages sent to \( a \) on to \( c \). Such a forwarder is necessary, in general, because \( a \) may have subambients, which would run at different locations and which would send their requests of movement to \( a \). (With appropriate optimizations, forwarders can be further simplified, or even altogether removed; see the discussion in Section 9.)

The other reduction rule of SA, the \( R\)-MSG rule, is an interaction between two processes local to the same ambient. In PAN, this is simulated by essentially the same rule: no messages need to be exchanged between locations.

Using \( R\)-OPEN, rather than \( R\)-IN or \( R\)-OUT, for the physical movements may appear counterintuitive. One should however bear in mind that, in an ambient-like formalism, entering and exiting ambients is not very useful without opening some ambients. The processes local to an ambient \( a \) can interact with the processes local to ambient \( b \) only if one of the ambients moves inside the other and is opened. For instance, suppose we want to model a traveler that starts on a base site, goes to a distant server, interacts with its services, and then comes back reporting a result. For simplicity, we suppose that interaction just consists in loading a value \( v \) that is also the final result. The distant server is defined thus:

\[
SERVER \triangleq s[ \text{in } s. \text{open go. out } s. P | (v) ]
\]

The base site and the traveler \( T \) are:

\[
BASE \triangleq n[ T | \text{out } n. \text{in } n. \text{open return. }(x) R ]
\]

\[
T \triangleq \text{go[ out } n. \text{in } s. \text{open go. }(x) \text{return[ out } s. \text{in } n. \text{open return. }(x) ]]]
\]

We have:

\[
SERVER | BASE \implies s[ \text{out } s. P | (v) | (x) \text{return[ out } s. \text{in } n. \text{open return. }(x) ]] | n[ \text{in } n. \text{open return. }(x) R ] \]

\[
\implies s[ P ] | n[ (x) R | (v) ] \]

\[
\implies s[ P ] | n[ R | v ]]
\]

In the PAN execution, the initial configuration is (omitting some local processes of \( s \) and \( n \))

\[
h; s[ \ldots ]_{\text{root}} \parallel k; n[ T | \ldots ]_{\text{root}}
\]
and the first step is to spawn off a new location hosting the traveler, obtaining:

\[ h : s[\ldots]_\text{root} \parallel k : n[\ldots]_\text{root} \parallel \ell : \text{go}[\ldots]_k \]

In the reductions in (3), two R-OPEN are executed. When this is modeled in PAN, in the first R-OPEN the (derivative of the) traveler physically moves into the distant server so to load \( v \); in the second, the (derivative of the) traveler comes back to deliver its result to \( R \).

4. The abstract machine, formally

In this section we introduce the syntax and reduction semantics of the abstract machine. Syntax. The syntax of PAN is shown in Fig. 5. A term of PAN, a net, is the parallel composition of agents and messages, with some names possibly restricted. An agent can be a located ambient or a forwarder. Located ambients are the basic unit of PAN, and represent ambients of SA with their local processes. A located ambient becomes a forwarder when opened. Messages are of two forms: requests and completions. The former are the messages that an ambient sends to its parent to request a movement operation; the latter are the messages that the parent sends back to the children to complete the operation. An open needs however a third completion message (a register message) in which the processes local to a child are migrated to its parent. The syntax of the processes inside located ambients is similar to that of processes in SA. The only additions are: the prefix wait.\( P \), which appears in an ambient when this has sent a request to its parent but has not received an answer yet; and the requests, which represent messages received from the children and not yet served. We use \( A \) to range over nets.

Semantics. The reduction relation of PAN, \( \rightarrow \), from nets to nets, is defined by the rules of Figs. 6 and 7. The axioms are divided into five groups, for, respectively: reductions local to an ambient; creation of new ambients and new restrictions; forwarding; consumption of request messages; emission of request messages; consumption of completion messages. There is finally a group of inference rules. The rules for local reductions, and the associated inference rule PAR-PROC, have a special format. We write

\[ P \xrightarrow{k}{h,n} Q \gg \tilde{\text{Msg}} \]

to mean a process \( P \), local to an ambient \( n \) that is located at \( h \), and whose parent is located at \( k \), becomes \( Q \) and, as a side effect, the messages in \( \tilde{\text{Msg}} \) are generated. We use \( \tilde{\text{Msg}} \) to indicate a possibly empty parallel composition of messages. When \( n \) or \( h \) or \( k \) are unimportant, we replace them with \(-\), as in

\[ P \xrightarrow{k}{-} Q \gg \tilde{\text{Msg}}. \]

When all the three of them are unimportant, we simply write

\[ P \xrightarrow{-} P' \gg \tilde{\text{Msg}}. \]

The local reductions correspond to the reduction rule of the SA calculus. The rule LOCAL-COM is simply local message exchange, no completion is generated. In the rule LOCAL-IN both (action and coaction) request messages are consumed and we generate two completions: one for the ambient requesting the action that has to change its parents, with the indication of the location of the new parent (\( k \)), and the other for the ambient requesting the coaction saying that the process can consume the coaction. In rule LOCAL-OUT, since
\[ m, n, \ldots \in \text{Names} \quad h, k, \ldots \in \text{Locations} \quad p, q, \ldots \in \text{Names} \cup \text{Locations} \]

**Nets**

\[
\begin{align*}
A & := \emptyset \quad \text{(empty)} \\
& | \text{Agent} \quad \text{(agent)} \\
& | h[M_{\text{MsgBody}}] \quad \text{(message)} \\
& | A_1 \parallel A_2 \quad \text{(composition)} \\
& | (\nu p) A \quad \text{(restriction)} \\
\end{align*}
\]

**Agents**

\[
\begin{align*}
\text{Agent} & := h \triangle k \quad \text{(forwarder)} \\
& | h[n] P \quad \text{(located ambient)} \\
\end{align*}
\]

**Message body**

\[
\begin{align*}
\text{MsgBody} & := \text{Request} \quad \text{(request)} \\
& | \text{Completion} \quad \text{(completion)} \\
\end{align*}
\]

\[
\begin{align*}
\text{Request} & := \text{in } n, h \quad \text{(the agent at } h \text{ wants to enter } n) \\
& | \text{out } n, h \quad \text{(the agent at } h \text{ named } n \text{ accepts someone in)} \\
& | \text{open } n, h \quad \text{(the agent at } h \text{ wants to go out of } n) \\
\end{align*}
\]

\[
\begin{align*}
\text{Completion} & := \text{go } h \quad \text{(change the parent to be } h) \\
& | \text{OK} \quad \text{(request } \text{in accepted)} \\
& | \text{migrate} \quad \text{(request } \text{open accepted)} \\
& | \text{register } P \quad \text{(add } P \text{ to the local processes)} \\
\end{align*}
\]

**Process-related syntax**

\[
\begin{align*}
P & := 0 \\
& | P_1 \mid P_2 \\
& | (\nu n) P \\
& | M.P \\
& | M[P] \\
& | \langle M \rangle \\
& | (x) P \\
& | X \\
& | \text{rec } X.P \\
& | \text{wait } P \\
& | \langle \text{Request} \rangle \\
\end{align*}
\]

no request message is generated for the coaction, only one completion message is generated for the ambient requesting to be moved (again the completion contains the location of the new parent). The reduction, however, must occur in the ambient containing the coaction, and the coaction is consumed allowing the process \( P \) to go on. For the LOCAL-OPEN again the matching happen in the ambient containing the coaction. However, the process \( P \), is stopped, and has to wait for the register completion to continue. The register completion will contain the local processes of the ambient that is opened.

Fig. 5. The syntax of PAN.
Local reductions

\[\langle M \rangle \upharpoonright (x) \cdot P \xrightarrow{\cdot x} P[M/x] \gg 0\]  \[\text{[LOCAL-COM]}\]

\[\{in \cdot n \cdot h \upharpoonright \{in \cdot n \cdot k \} \xrightarrow{\cdot k} \theta \gg h(k) \parallel k[\text{OK} \cdot n]\} \]  \[\text{[LOCAL-IN]}\]

\[\{out \cdot n \cdot h \upharpoonright \{out \cdot n \cdot P \xrightarrow{k} P \gg h(k) \} \]  \[\text{[LOCAL-OUT]}\]

\[\text{open} \cdot n \cdot P \upharpoonright \{\text{open} \cdot n \cdot h \} \xrightarrow{k} \text{wait} \cdot P \gg h[\text{migrate}]\]  \[\text{[LOCAL-OPEN]}\]

Creation

\[\text{h} \cdot n \cdot m \cdot P \cdot Q \cdot h' \rightarrow \text{h} \cdot n \cdot Q \cdot h' \parallel (vk) (k \cdot m \cdot P \cdot h)\]  \[\text{[NEW-LOCAMB]}\]

\[\text{h} \cdot n \cdot (vm) \cdot P \cdot k \rightarrow (vm) (h \cdot n \cdot P \cdot k) \quad n \neq m\]  \[\text{[NEW-RES]}\]

Forwarder

\[\text{h} \triangleright k \parallel h[\text{MsgBody}] \rightarrow h \triangleright k \parallel k[\text{MsgBody}]\]  \[\text{[FW-MSG]}\]

Consumption of request messages

\[\text{h} \cdot n \cdot P \cdot [\text{Request}] \cdot h' \rightarrow \text{h} \cdot n \cdot P \cdot [\text{Request}] \cdot h'\]  \[\text{[CONSUME-REQ]}\]

Emission of request messages (should be h \neq \text{root})

\[\text{in} \cdot m \cdot P \xrightarrow{k} \text{wait} \cdot P \gg k[\text{in} \cdot m \cdot h]\]  \[\text{[REQ-IN]}\]

\[\text{in} \cdot n \cdot P \xrightarrow{k} \text{wait} \cdot P \gg k[\text{in} \cdot n \cdot h]\]  \[\text{[REQ-COIN]}\]

\[\text{out} \cdot m \cdot P \xrightarrow{k} \text{wait} \cdot P \gg k[\text{out} \cdot m \cdot h]\]  \[\text{[REQ-OUT]}\]

\[\text{open} \cdot n \cdot P \xrightarrow{k} \text{wait} \cdot P \gg k[\text{open} \cdot n \cdot h]\]  \[\text{[REQ-COOPEN]}\]

Consumption of completion messages

\[\text{h} \cdot n \cdot [\text{wait} \cdot Q \cdot k \parallel h[\text{go} \cdot h'] \rightarrow \text{h} \cdot n \cdot [P \cdot Q] \cdot h'\]  \[\text{[COMPL-PARENT]}\]

\[\text{h} \cdot n \cdot [P \cdot wait \cdot Q \cdot k \parallel h[\text{OK} \cdot n] \rightarrow h \cdot n \cdot [P \cdot Q] \cdot h\]  \[\text{[COMPL-COIN]}\]

\[\text{h} \cdot n \cdot [P \cdot wait \cdot Q \cdot k \parallel h[\text{migrate}] \rightarrow h \triangleright k \parallel k[\text{register} \cdot P \cdot Q]\]  \[\text{[COMPL-MIGR]}\]

\[\text{h} \cdot n \cdot [P \cdot wait \cdot Q \cdot k \parallel h[\text{register} \cdot R] \rightarrow h \cdot n \cdot [P \cdot Q \cdot R] \cdot k\]  \[\text{[COMPL-REG]}\]

Fig. 6. Reduction Rules of PAN.
The creation rule, \textsc{new-locamb} assigns to the ambient \(m\) subambient of \(n\) a new location \(k\). The logical structure is realized by assigning the location of \(n\) (\(h\)) as parent of the location \(k\). The rule \textsc{new-res} enlarges the scope of a restriction. The rule \textsc{fw-msg}, if there is a forwarder from \(h\) to \(k\), redirects the messages sent to \(h\) to \(k\), so that in the rule \textsc{consume-req} such messages may enter the right ambient (and be processed by the local reduction).

The rules for emission of requests, generate the request messages corresponding to all the actions and coaction except for \textsc{open} and \textsc{out}. The request message is sent to the location of the parent of the ambient and contains the location of the ambient requesting the action. This allows the parent which does not know the location of its children, to do the matching and send the completion to the right location. Notice that the emission of requests for the coaction \(\text{\textsc{in}}\) is done only if the capability is an unguarded prefix of the enclosing ambient \(n\). For the \textsc{out} coaction, for which no request message is generated, the same property is insured by the fact that the local reduction \textsc{local-out} must happen in the ambient \(n\).

The rules for consumption of completions realize the actual movement operations, consuming the completion message. The \textsc{compl-parent} rule modifies the parent of the ambient located at \(h\) to be \(h'\). This completion was sent as the result of an \textsc{in} or \textsc{out} request. The process \(Q\), which was the one prefixed by the corresponding capability, is allowed to continue. For the \textsc{in} completion we just allow the process \(Q\), which was the one prefixed by the \textsc{in} capability, to go on. The process was stopped to avoid the possibility that it may flood the parent with requests, before satisfying the current one. The \textsc{compl-migr} rule dissolves the ambient \(n\) located at \(h\) leaving a forwarder to the location of its parent, to whom it sends (through the \textsc{register} completion) the local processes of \(n\). The processes will be added to the local processes of the parent ambient via the \textsc{compl-reg} rule. In the implementation, the completion messages are first read, and then consumed, whereas here these two phases are merged. This abstraction is completely harmless—changing the machine to reflect it quite straightforward.
The inference rules allow us to execute local reductions inside located ambients, restrictions, and parallel compositions. First the side condition of rule PAR-PROC insures that local reductions and emissions of requests may be done only in located ambients not containing unguarded subambients. So subambients of an ambient are always activated as soon as possible, before any local reduction takes place (here we exploit the fact that recursions are guarded, otherwise there could be an infinite number of ambients to create). The rule NEW-LOCAMB, therefore, has the precedence on all the other rules. Rule PROC-AGENT transforms a message (request or completion) into an agent. For instance, if \( P \xrightarrow{\text{process}} A \xrightarrow{\text{wait}} A \), then, using PROC-AGENT and PAR-AGENT, we have, for any net \( A \):

\[
A \parallel h: n[ P ]_k \longrightarrow A \parallel h: n[ Q ]_k \parallel \tilde{\text{Msg}}
\]

The inference rule STRUCT-CONG make use of the structural congruence relation \( \equiv \), whose definition is similar to that for SA.

**Definition 4.1.** Structural congruence is the smallest congruence \( \equiv \) extending structural congruence of Definition 2.1, and such that

- parallel composition, \( \parallel \), and \( 0 \) form an abelian monoid on agents, and
- \((v p)(A_1) \parallel A_2 \equiv (v p)(A_1 \parallel A_2)\).

Reductions are characterized in proper and administrative reductions. Proper reductions mirror the reductions of the SA calculus, whereas administrative reductions do the bookkeeping needed to generate locations, requests and consume completions.

**Definition 4.2.** A reduction \( A \longrightarrow A' \) is an administrative reduction, written \( A \xrightarrow{\text{adm}} A' \), if there is a derivation of \( A \longrightarrow A' \) that does not use local reductions (axioms LOCAL-COM, LOCAL-IN, LOCAL-OUT, or LOCAL-OPEN).

A reduction \( A \longrightarrow A' \) is a proper reduction, written \( A \xrightarrow{\text{pr}} A' \), if there is a derivation of \( A \longrightarrow A' \) that uses one of the local reductions.

As usual, \( \longrightarrow_{\text{adm}} \) and \( \longrightarrow_{\text{pr}} \) are the reflexive and transitive closures of \( \longrightarrow \) and \( \longrightarrow \), respectively.

We can give the following characterization of administrative and proper reductions.

- \( A \xrightarrow{\text{adm}} A' \) iff \( A \equiv (v p)(A_1 \parallel A_2) \), and \( A' \equiv (v p)(A'_1 \parallel A_2) \) and either:
  (1) \( A_1 \) is the antecedent of one of the rules: \([\text{NEW-...}]\), \([\text{FW-MSG}]\), \([\text{CONSUME-REQ}]\), \([\text{COMPL-...}]\), and \( A'_1 \) is the consequent, or
  (2) \( A_1 = h: n[M.Q | Q']_k \), where \( M \in \{\text{in } m, \text{out } m, \text{in } n, \text{open } n\} \), \( Q' \) does not have unguarded ambients, and \( A'_1 = h: n[\text{wait. } Q | Q']_k \parallel k[M, h] \).

- \( A \xrightarrow{\text{pr}} A' \) iff \( A \equiv (v p)(A_1 \parallel A_2) \), and \( A' \equiv (v p)(A'_1 \parallel A_2) \) and either:
  (1) \( A_1 = h: n[M] \parallel (x).Q | Q'\}_k \), and \( A'_1 = h: n[Q[M/x] | Q']_k \), or
  (2) \( A_1 = h: n\{[\text{in } m, h'] | \text{in } m, k'\} | Q'\}_k \), and \( A'_1 = h: n[Q']_k \parallel h'[\text{go } k'] | k'[\text{out } m] \), or
  (3) \( A_1 = h: n\{[\text{out } n, h'] | \text{out } n, Q | Q'\}_k \), and \( A'_1 = h: n[Q | Q']_k \parallel h'[\text{go } k] \), or
  (4) \( A_1 = h: n\{[\text{open } m, h'] | \text{open } m, Q | Q'\}_k \), and \( A'_1 = h: n[\text{wait. } Q | Q']_k \parallel h'[\text{migrate}] \)

where \( Q' \) does not have unguarded ambients.
5. Correctness of the abstract machine for single-threaded ambients

In this section we present the proof of adequacy of the translation of SA terms into PAN when the only ambients in a term are single threaded. In particular, we prove that if $P$ is a well-typed single-threaded SA terms, $P$ exhibits the same observable behavior as the PAN term which is its translation.

We first introduce some definitions. The name of a located ambient $h: n [ P ]_k$ is $n$, the home location is $h$, the parent location (of $h$) is $k$. The home location of a forwarder $h \triangleright k$ is $h$, and the parent location is $k$. The target location of a message $h[\text{MsgBody}]$ is $h$. The source location of a request $\{M, k\}$ (that can be part of a message or of a local process) is $k$. There is no source location in a completion message. A special location is root, indicating the outermost (logical) location. By convention, the ambient located at root has name rootname and parent rootparent. We can extract a relation among locations from a net $A$, whereby two locations are related if they are the home and the parent location of the same agent. We call this the dependency relation on the locations of $A$.

As for SA terms (Eq. (1) of Section 2) also a located ambient, up to structural congruence, can be written as

$$(\tilde{w} n) (h: n [ P_1 | \ldots | P_i | m_1[Q_1] | \ldots | m_r[Q_r] ])_k$$

Note that the subambients of a located ambient are not located. A located ambient is in wait state if one of the local processes of the ambient, $P_i$, has an unguarded wait prefix.

Since PAN separates between the logical and physical distribution of ambients, we need to make sure that the two are consistent. For instance, the graph of the dependencies among locations in the physical distribution, which represents the logical structure, should be a tree. We also need conditions that ensure that the wait prefix is used as described informally in previous sections.

Therefore, we introduce the notion of well-formedness, which is a property that holds for all the nets that are translation of SA terms, and that is preserved by administrative reductions. We will show that proper reductions preserve a weaker version of well-formedness.

In the definition below, the dependency relation on the locations of $A$ is required to be a tree. Hence we can define a partial order $\leq$ on locations, where $h \leq k$ indicates that either $k = h$, or that $k$ is between $h$ and the root of the tree in the ordering. The target location of a message $h[\text{MsgBody}]$ is the least location $k$ with $h \leq k$ such that the agent at $k$ is a located ambient (not the home location of a forwarder). Moreover, we define the relations parent ambient and child ambient: the parent ambient of an ambient located at $h$ is $k$ if $k$ is a located ambient, $h < k$, and for no located ambient $h'$, such that $h' \neq h$ and $h' \neq k$, $h < h'$ and $h' < k$. Similarly, $h$ is a child ambient of an agent $k$ (not necessarily an ambient), if $h$ is a located ambient, $h < k$, and for no located ambient $h'$, such that $h' \neq h$ and $h' \neq k$, it holds that $h < h'$ and $h' < k$.

**Definition 5.1 (Well-formedness).** A net $A$ is well-formed if root and rootparent are the only free locations of $A$, and the following conditions are true for $A$ (by alpha-conversion, we assume that all restricted locations and names are different from each other and from free locations and names):
(1) The dependency relation on the locations of $A$ forms a tree.
(2) If there is a message $h\{\text{go } k\}$, then $k \not\leq h$.
(3) There is a special located ambient, called the root, located at $\text{root}$, with name $\text{rootname}$, and with parent $\text{rootparent}$. The name $\text{rootname}$ and the location $\text{rootparent}$ cannot appear anywhere else. The location $\text{root}$ cannot be 
(a) the source of a request message, or 
(b) the target of a completion message that is not a register.
(4) For every location $h$, there is at most one agent located at $h$.
(5) For every forwarder $h \triangleright k$, the home location $h$ does not appear 
(a) as source location of a request message or a request process, or 
(b) as target of a migrate message.
(6) The prefix $\text{wait}$ can only appear as unguarded prefix of a local process of a located ambient.
(7) A located ambient with home location $h$ exists and is in wait state if and only if one of the following holds:
(a) $h$ is the source of at most a request (in a message or in a local process);
(b) $h$ is the target of at most a completion message;
(c) $h$ is the parent ambient of the target of at most a migrate message.
(8) A located ambient with home location $h$ such that $h$ is the target of a completion does not contain unguarded subambients.
(9) No ambient (located or not) has an unboxed subterm with more than one unguarded prefix.
(10) If there is a message $h\{\text{register } R\}$ and its ambient target (up to $\equiv$) is $h:n[P_1 | \text{wait}. P_2 | k]$, then $P_1 | P_2 | R$ has at most one unguarded prefix, and this prefix is not $\text{wait}$.
(11) If there is a message $h\{\text{migrate}\}$ and the ambient at $h$ (up to $\equiv$) is $h:n[Q_1 | \text{wait}. Q_2]_k$, and the ambient parent of $h$ (up to $\equiv$) is $k:m[P_1 | \text{wait}. P_2]_{k'}$ then $Q_1 | Q_2 | P_1 | P_2$ has at most one unguarded prefix.
(12) (a) In every request message $k[M, h]$, location $h$ is a child of $k$, and 
(b) if inside an ambient located at $k$ there is a request $\{M, h\}$ then $h$ is a child of $k$.

Clause 1 asserts that the dependency relation mirrors the nested structure of ambients, and clause 3 says that the root ambient represents a top level ambient enclosing the whole term. Note that, since $\text{root}$ is the root of the dependency relation, for all located ambients different from the ambient located at $\text{root}$, the parent ambient exists and it is unique, whereas there can be any number of children ambients (also zero).

Clause 2 is needed to insure that execution of the COMPL-PARENT reduction preserves the tree structure of the dependency relation. Looking at the reductions we can see that a completion $h\{\text{go } k\}$ is generated by two rules. The first is by applying rule LOCAL-IN where $h$ is the location of the agent containing the $\text{in}$ capability and $k$ is the location of the agent containing the $\text{in}$ capability. In this case the two agents represent siblings ambients so $k \not\leq h$. The second is by applying rule LOCAL-OUT, in this case $h$ is the location of the agent containing the $\text{out}$ capability, and $k$ is the location of the parent ambient of the ambient containing the $\text{out}$ capability. So also in this case $k \not\leq h$.

Clause 4 implies that if there is a forwarder $h \triangleright k$, then there is no ambient $h:n[\cdots]_k$ (located at $h$). This is consistent with the fact that forwarders represent ambients that
have been dissolved by an open capability. (Forwarders are generated by a COMPL-MIGR reduction.)

Clause 5 reflects the fact that a forwarder, h △ k, is generated by an ambient (located at h) containing an open capability, and the ambient is dissolved when the forwarder is generated. Since the ambient was single threaded, it did not have any other capability at the outermost level (except for the open that was consumed). Therefore, no other request could have h as its source location, and consequently h could not be the target of a completion. However, h could be the target of a register since h is the location of the agent enclosing the ambients containing the open and open capabilities, which may have active capabilities.

Since wait prefixes replace capabilities when generating a request message, and ambients are single threaded, there can be only one wait unguarded prefix in a located ambient (clause 6).

Clause 7 says how wait prefixes are matched with request or completion messages in a well-formed net. Note that the rules REQ-... generate a wait prefix, and a matching request (clause 7a). Moreover, the rules LOCAL-... replace a request (which had a corresponding wait) with a completion. This is now the completion matching the wait prefix (clause 7b), and the rules COMPL-PAR and COMPL-COIN, remove a wait prefix and its matching completion. A wait prefix generated by a LOCAL-OPEN rule, is generated without a matching request. However, this wait is contained in the parent ambient of the ambient containing the open prefix (clause 7c). Rule COMPL-MIGR substitutes a wait prefix, and its matching completion with a register completion, which now matches the wait prefix introduced by the LOCAL-OPEN rule. Rule COMPL-REG removes this matching pair.

Clause 8 reflects the fact that a completion message is sent to an agent that was the source of a request. By rule PAR-PROC a request message can be generated by a process in a located ambient only if this ambient does not contain unguarded subambients.

Clause 9 holds since all the ambients are single threaded (so they can contain only single-threaded ambients).

Clauses 10, and 11 are needed to asserts that after consuming a migrate or register completion we obtain a well-formed net, which satisfies clauses 6 and 7. This clause is satisfied by a net representing an SA term since all the ambients are single threaded, and this property is preserved by reduction.

Finally, clause 12 asserts that messages travel from child to parent, and get correctly in the location of the parent.

**Lemma 5.2.** If A is well-formed and A ≡ A' then also A' is well-formed.

**Proof.** Straightforward. □

The following lemma proves that administrative reductions on well-formed nets are weakly confluent.

**Lemma 5.3.** Suppose A is well-formed. If A → adm A' and A → adm A" then either A' ≡ A", or there is A‴ such that A' → adm A‴ and A" → adm A‴.

**Proof.** If the reductions of A → adm A' and A → adm A" involve non-overlapping redexes, the result is obvious. Therefore let us consider the possible reductions involving overlapping redexes.
Let \( A \equiv (\nu \exists P) (A_1 \parallel A_2) \).

1. \( A_1 = h: n[M, Q \parallel Q']_k \), where \( M \in \{in, out, \text{IN}, \text{OUT}, m\} \), and \( A \mapsto_{\text{adm}} A' \) applying a rule \([\text{REQ} \ldots]\). So \( A' \equiv (\nu \exists P) (A'_1 \parallel A_2) \) where

\[
A'_1 = h: n[\text{wait}, Q | Q']_k \parallel k[M, h]
\]

Since clause 9 of Definition 5.1 holds for \( A \), \( Q' \) cannot have unguarded prefix so neither a rule \([\text{COMPL} \ldots]\), nor another rule \([\text{REQ} \ldots]\) involving \( A_1 \) can be applied. Moreover rule \([\text{NEW-LOCAMB}]\) cannot be applied since \( Q' \) does not have unguarded ambient.

The only conflict may arise when \( A_2 = h[\text{Request}] \parallel A_3 \), and rule \([\text{CONSUME-REQ}]\) is applied to \( A_1 \parallel h[\text{Request}] \). In this case we would have \( A'' = (\nu \exists P) (A''_1 \parallel A_3) \) where

\[
A''_1 = h: n[M, Q | Q' | \{\text{Request}\}]_k \parallel k[M, h] \parallel A_3.
\]

We get that \( A' \mapsto_{\text{adm}} A'' \) via \([\text{CONSUME-REQ}]\), and \( A'' \mapsto_{\text{adm}} A''' \) via the corresponding \([\text{REQ} \ldots]\).

2. \( A_1 = h: n[mP \parallel Q']_k \), and \( A \mapsto_{\text{adm}} A' \) applying a rule \([\text{NEW-LOCAMB}]\). In this case no rule \([\text{COMPL} \ldots]\) involving \( A_1 \) can be applied, since clause 8 of Definition 5.1 holds for \( A \). As in the previous case, the only conflict may arise with rule \([\text{CONSUME-REQ}]\), and \( A''' \) can be defined in a similar way.

3. \( A_1 = h \triangleright k \parallel h[\text{Request}] \), and \( A \mapsto_{\text{adm}} A' \) applying a rule \([\text{FW-MSG}]\). From clause 4 of Definition 5.1, no ambient can be located at \( h \) so no other rule involving \( h[\text{Request}] \) can be applied.

4. \( A_1 = h: n[P \mid \text{wait}, Q]_k \parallel h[\text{Completion}] \), and \( A \mapsto_{\text{adm}} A' \) applying any rule \([\text{COMPL} \ldots]\). First notice that, since clause 7 of Definition 5.1 holds for \( A \), no other rule \([\text{COMPL} \ldots]\) can be applied. However, if \( A_2 = h[\text{Request}] \parallel A_3 \) also rule \([\text{CONSUME-REQ}]\) can be applied. When \( \text{Completion} \neq \text{migrate} \), applying first \([\text{CONSUME-REQ}] \) and then the appropriate \([\text{COMPL} \ldots] \) produces the same result as applying first \([\text{COMPL} \ldots] \) and then \([\text{CONSUME-REQ}] \).

For \( \text{Completion} = \text{migrate} \), the situation is more complex. We have

\[
A_1 = h: n[P \mid \text{wait}, Q]_k \parallel h[\text{migrate}].
\]

\( A \mapsto_{\text{adm}} A' \) applying rule \([\text{COMPL-MIGR}]\) where

\[
A' = (\nu \exists P) (h \triangleright k \parallel k[\text{register}, P | Q] \parallel h[\text{Request}] \parallel A_3)
\]

\( A \mapsto_{\text{adm}} A'' \) applying rule \([\text{CONSUME-REQ}]\), where

\[
A'' = (\nu \exists P) (h: n[P \mid \text{wait}, Q | \{\text{Request}\}]_k \parallel h[\text{migrate}] \parallel A_3).
\]

From clause 7c of Definition 5.1, we have that \( A_3 = k: m[P | \text{wait}, Q']_L \parallel A_4 \). Therefore

\[
A' \mapsto_{\text{adm}} (\nu \exists P) (h \triangleright k \parallel k[\text{register}, P | Q] \parallel k[\text{Request}] \parallel A_3) \quad \text{with rule [FW-MSG]}
\]

\[
\mapsto_{\text{adm}} (\nu \exists P) (h \triangleright k \parallel k[\text{Request}] \parallel k: m[P | Q']_L \parallel A_4) \quad \text{with rule [COMPL-REG]}
\]

\[
\mapsto_{\text{adm}} A''' \quad \text{with rule [CONSUME-REQ]}
\]
where $A'' = (v p) (h \triangleright k \text{~} k'[P' \mid Q' \mid P \mid Q \mid \text{[Request]}]) \triangleright k' \text{~} A_4$, and

$$A'' \xrightarrow{adm}(v p) (h \triangleright k \text{~} k'[\text{register} \text{~} P \mid Q \mid \text{[Request]}]) \triangleright k' \text{~} A_4 \text{~} A_3 \text{~} \text{[COMPL-MIGR]}$$

$$A'' \xrightarrow{adm} A_4 \text{~} \text{[COMPL-REG]}$$

So $A' \xrightarrow{adm} A''$ and $A'' \xrightarrow{adm} A'''$.

This concludes the proof. □

We now prove that well-formedness is preserved by administrative reductions.

**Lemma 5.4.** If $A$ is well-formed, and $A \xrightarrow{adm} A'$, then also $A'$ is well-formed.

**Proof.** By case analysis on the reduction $A \xrightarrow{adm} A'$. For each case one has to consider all the clauses of well-formedness (Definition 5.1).

Let $A \equiv (v p) (A_1 \parallel A_2)$, and $A' \equiv (v p') (A'_1 \parallel A_2)$. For the rule [FW-MSG] the clauses of well-formedness hold for $A'$ since they hold for $A$.

For the rule [CONSUME-REQ] notice only that clause 12b holds for $A'$ since clause 12a holds for $A$, and for rule [NEW-...] since $k$ is new the dependency relation in $A'$ still forms a tree. All the other clauses hold since they hold for $A$.

For the [REQ-...] rules:

- $A_1 = h:n[M, Q \mid Q']$\_k, where $M \in \{\text{in} ~ m, \text{out} ~ m, \text{open} ~ m\}$, and $Q'$ does not contain an unguarded ambient, and
- $A'_1 = h:n[\text{wait} \mid Q \mid Q']\_k \parallel k(M, h)$.

Note that in $A$, $h$ can be neither root nor the source of a forwarder, therefore in $A'$ clauses 3 and 5 hold. Since in $A$ the process $Q'$ cannot have any unguarded prefix, then clauses 8 and 9 hold in $A'$. Clause 7 holds for $A'$ since it holds for $A$ and $A'_1$. Finally since $h$ is a child of $k$, clause 12a holds for $A'$. The other clauses hold since they hold for $A$.

For the rule [COMPL-PARENT], since clause 2 holds for $A$, we have that the dependency relation in $A'$ still forms a tree. Since clause 9 holds for $A$, $P$ cannot have unguarded prefixes, and so clause 9 holds also for $A'$. Moreover, from clause 6 for $A$, such a prefix cannot be wait. So also the resulting ambient is not in wait state, and clauses 7, 9, and 12 hold for $A'$. The other clauses hold since they hold for $A$. Similarly for rule [COMPL-COIN].

For the rule [COMPL-MIGR]:

- $A_1 = h:n[P \mid \text{wait \~} Q]_k \parallel h[\text{migrate}]$, and
- $A'_1 = h \triangleright k \text{~} k[\text{register} \text{~} P \mid Q]$.

All the clauses except for 5, 7, and 10 hold for $A'$ because they hold for $A$. Consider clause 5. Since clause 7 holds for $A$, the ambient $h$ cannot be either the source of a request, or the target of a completion, so clause 5 holds for $A'$. Since clauses 7b and 7c hold for $A$, we know that in $A_2$ there should be an agent in wait state located at $k$. So clause 7 is verified for $A'$. Moreover, clause 10 for $A'$ is insured from the fact that clauses 11 and 6 hold for $A$.

For the rule [COMPL-REG] we only have to check clauses 7 and 9. In $A'$ the ambient located at $h$ is not in wait state because 10 holds for $A'$, so clause 7 holds for $A'$. Clause 9 is derived from the fact that clause 10 holds for $A$. □
Administrative reductions are strongly normalizing.

Lemma 5.5. There is a function $\mu$ from well-formed terms to ordinals such that $\mu(A) > 0$ and for $A, A'$, if $A \overset{\text{adm}}{\rightarrow} A'$, then $\mu(A) > \mu(A')$.

Proof. To define $\mu(A)$ we define also $\mu(P)$ for a process $P$, by structural induction on $A$ and $P$.

- $\mu(0) = \mu(h \triangleright k) = 0$,
- $\mu(h : n[P]_k) = \mu(P)$,
- $\mu(h[\text{MsgBody}]) = d + 1$ if $\text{MsgBody} \neq \text{register}P$ where $d$ is the distance of $h$ from root,
- $\mu(h[\text{register}P]) = d + 1 + \mu(P)$ where $d$ is the distance of $h$ from root,
- $\mu(A_1 \parallel A_2) = \mu(A_1) + \mu(A_2)$,
- $\mu((v \cdot P) A) = \mu(A)$,
- $\mu(P_1 \parallel P_2) = \mu(P_1) + \mu(P_2)$,
- $\mu((\nu n) P) = \mu(P) + 1$,
- $\mu(M.P) = \mu(P) + \omega$, where $M \in \{\text{in} : v, \text{open} : v, \text{in} : v, \text{out} : v\}$, and $v = x$ or $v = n$,
- $\mu(M[P]) = \mu(\text{rec}X.P) = \mu((x) P) = \mu(\text{wait}.P) = \mu(P) + 1$,
- $\mu((M)) = \mu(X) = \mu(\text{Request}) = 1$.

First note that for all $A$, $\mu(A) \geq 0$. We have to show that $A \overset{\text{adm}}{\rightarrow} A'$ implies $\mu(A) > \mu(A')$.

Let us just consider the [REQ-...] rules. In this case $A \overset{\text{adm}}{\rightarrow} A'$ iff $A \equiv (\nu \overline{\nu}A) A_1 \parallel A_2$, $A' \equiv (\nu \overline{\nu}A') A_1' \parallel A_2$ and

- $A_1 = h : n[M, Q | Q' ]_k$,
- $A_1' = h : n[\text{wait} \cdot Q | Q' ]_k \parallel k[M, h]$.

Therefore $\mu(A) = \mu(A_2) + \mu(Q') + \mu(Q) + \omega$, and $\mu(A') = \mu(A_2) + \mu(Q') + \mu(Q) + 1 + d + 1$ where $d$ is the distance of $k$ from root. So $\mu(A) > \mu(A')$. The other rules are similar. □

We can now conclude that administrative reductions are Church-Rosser.

Theorem 5.6. Suppose $A$ is well-formed. If $A \Rightarrow_{\text{adm}} A_1$ and $A \Rightarrow_{\text{adm}} A_2$ then either $A_1 \equiv A_2$, or there is $A_3$ such that $A_1 \overset{\text{adm}}{\Rightarrow} A_3$ and $A_2 \overset{\text{adm}}{\Rightarrow} A_3$.

Proof. A consequence of Lemma 5.3, using standard term-rewriting techniques. (Note that since we have just weak confluence, we need also termination.) □

Definition 5.7. A net $A$ is in normal form if it cannot perform an administrative reduction.

Corollary 5.8. If $A$ is well-formed, then there is a normal form $A^*$ such that $A \Rightarrow_{\text{adm}} A^*$. Moreover, the normal form $A^*$ is unique up to $\equiv$, in the sense that if there is another process $A'$ which satisfies the above clause, then $A' \equiv A^*$.

Proof. From Theorem 5.6 and Lemma 5.5. □
Normal forms are characterized by the following well-formed nets.

**Lemma 5.9.** Let $A$ be a well-formed net in normal form. Then $A \equiv (\nu \tilde{P}) (A_1 \parallel A_2 \parallel \cdots \parallel A_r)$ where for all $i$, $1 \leq i \leq r$,

- $A_i$ is an agent (either a forwarder or a located ambient)
- if $A_i$ is a located ambient, then $A_i \equiv h: n[P_1 | \cdots | P_s]k$, where all $j$, $1 \leq j \leq s$,
  - $P_j = \langle M \rangle$, or $P_j = (x) P$, or
  - $P_j = X$, or $P_j = \text{rec } X \cdot P$, or
  - $P_j = \text{wait } X \cdot P$, or $P_j = \{\text{Request}\}$

To prove the correctness of the abstract machine we define the translation of a term of $SA$, $P$, into a term of $PAN$, $\[[P]]$, and the reconstruction of an $SA$ term, $\{A\}$, from a well-formed term of $PAN$, $A$. We prove that the observable behavior of $P$ coincides with the observable behavior of $\[[P]]$, by a bisimulation technique. That is, we show that $P \rightarrow P'$ implies $\[[P]] \rightarrow A'$ with $\{A'\} = P'$, i.e., a reduction in $SA$ corresponds to a sequence of reductions in $PAN$. Vice versa, $A \rightarrow A'$ implies $\{A\} \rightarrow \{A'\}$. In particular, a reduction in $PAN$ corresponds to zero or one reduction in $SA$.

The translation of a term of $SA$ into a term of $PAN$ is defined by putting the term in the ambient $\text{root}\text{name}$ located at $\text{root}$.

**Definition 5.10.** Let $\[[.]]$ be the translation of terms of $SA$ into terms of $PAN$, so defined:

$$\[[P]] \overset{\text{def}}{=} \text{root: } \text{root}\text{name}[P]_{\text{rootparent}}$$

**Lemma 5.11.** Suppose that $P$ is well-typed. Then $\[[P]]$ is well-formed.

**Proof.** All the clauses are trivially verified, since there is only one located ambient, and there are neither requests nor ambients in wait state. We use Proposition 2.2 to insure clause 9 of the definition of well-formedness. □

To reconstruct a term of $SA$ from a well-formed $PAN$ term we define a reduction, $\sim_{\text{adm}}$, that removes messages (generated by administrative reductions) either by reintroducing the corresponding capabilities, or by executing the corresponding completion. Moreover, the reduction removes locations reconstructing the nesting of ambients.

The reduction relation of $PAN$, $\sim_{\text{adm}}$, is defined by the reduction and inference rules of Fig. 8. With $\sim_{\text{adm}}^*$ we denote the reflexive and transitive closure of $\sim_{\text{adm}}$. Notice that we do not have the inference rule [PAR-AGENT], such a rule being replaced by the explicit presence of the context $A$ in the axioms. For the rules [REM-FW], and [REM-LOC] the context $A$ should not contain location $h$.

**Definition 5.12.** A net $A$ is in $\sim_{\text{adm}}$-normal form if it cannot perform any $\sim_{\text{adm}}$ reduction. We write $\mathcal{W}(A)$ for a normal form of $A$ w.r.t. $\sim_{\text{adm}}$.

The following lemma shows the main properties of the reduction $\sim_{\text{adm}}$ which are confluence and termination. From these properties we derive that the normal form (w.r.t. $\sim_{\text{adm}}$) of a well-formed net is a term containing a single ambient located at $\text{root}$ containing a $SA$-process $P$. 
Lemma 5.13. (1) If A is a well-formed net and \( A \xrightarrow{\sim_{\text{adm}}} A' \), then \( A' \) is also well-formed. 
(2) Reductions \( \sim_{\text{adm}} \) on well-formed nets are confluent. 
(3) Reductions \( \sim_{\text{adm}} \) on well-formed nets are terminating. 
(4) \( \mathcal{W}(A) \) is unique and \( \mathcal{W}(A) = \text{root:rootname}[P]_{\text{rootparent}} \) for some \( P \in \mathcal{S}A \).

Proof. (1) If \( A \xrightarrow{\sim_{\text{adm}}} A' \) by applying one of the rules consuming completions, then the proof is as for the proof of the corresponding clause of Lemma 5.4.

If \( A \xrightarrow{\sim_{\text{adm}}} A' \) by applying the rules removing requests, then we remove a message and a \text{wait} prefix, and therefore the fact that clause 6 holds for \( A \) implies that it also holds for \( A' \).

The clauses of Definition 5.1 that are relevant to the rule removing locations are: 1, 3, and 4. Rules \([\text{REM-FW}]\) and \([\text{REM-LOC}]\) remove a leaf from the dependency relation between locations, and rule \([\text{REM-INDIR}]\) moves the subtree rooted at \( h \) from being a subtree of \( k \) to be a subtree of \( k' \), preserving the tree structure of the dependency relation. So since clause 1 of Definition 5.1 holds for \( A \), then clause 1 holds also for \( A' \). Clause 3 of Definition 5.1 is preserved by the application of the rules removing
locations, since the fact that clause 3 holds for \( A \) implies that rule \([\text{REM-LOC}]\) and \([\text{REM-FW}]\) cannot remove the location \( \text{root} \). Finally no agent is moved from a location, therefore clause 4 of Definition 5.1 holds for \( A' \).

(2) Assume that \( A \xrightarrow{adm} A' \) and \( A \xrightarrow{adm} A'' \), we want to show that \( A' \xrightarrow{adm} A''' \) and \( A'' \xrightarrow{adm} A''' \) for some \( A''' \). If the reductions do not involve overlapping redexes the result is obvious.

The fact that \( A \) is well-formed (clauses 1, and 4 of Definition 5.1 hold), and the side condition of the rule \([\text{REM-LOC}]\), imply that the only possible overlapping redexes are when

- \( A \equiv (\nu \tilde{p})(h:n[P]\parallel k;m[Q]\parallel h'\colon n'[P']\parallel B) \),
- \( A' \equiv (\nu \tilde{p})(k;m[Q\parallel n[P]\parallel h'\colon n'[P']\parallel B] \), and
- \( A'' \equiv (\nu \tilde{p})(h:n[P]\parallel k;m[Q\parallel n'[P']\parallel B] \),
- \( A \xrightarrow{adm} A' \) by applying \([\text{REM-LOC}]\), and
- \( A \xrightarrow{adm} A'' \) by applying \([\text{REM-LOC}]\).

Note that \( m \) is the parent ambient of both \( n \) and \( n' \). It is easy to see that \( A''' = (\nu \tilde{p}) k:m[Q\parallel n'[P']\parallel n[P]\parallel B] \) is such that

- \( A' \xrightarrow{adm} A''' \) applying rule \([\text{REM-LOC}]\), and
- \( A'' \xrightarrow{adm} A''' \) applying rule \([\text{REM-LOC}]\).

(3) Notice that each reduction step, either

- removes a \textit{wait} prefix, or
- removes a location, or
- shortens the length of the path from a location to \textit{root}.

Define \( \mu(A) = n_w + \sum_{l\in A} d_l + 1 \) where \( n_w \) is the number of \textit{wait} prefixes, and \( d_l \) is the distance of the location \( l \) from \textit{root}. From clause 1 and 3 of Definition 5.1 for all location \( l \neq \text{root} \) we have \( d_l > 0 \). If \( A \xrightarrow{adm} A' \) then \( \mu(A) > \mu(A') \), and therefore \( \xrightarrow{adm} \) is terminating.

(4) Confluence and termination implies that \( W(A) \) exists and it is unique. Clause 7 of Definition 5.1 implies that: if \( A \) does not contain a \textit{wait} prefix, then it does not contain any message and vice-versa. So when no rule for removing requests or consuming completions is applicable, then in the net there can be neither messages nor \textit{wait} prefixes. Therefore in a net in normal form all the processes \( P \) inside ambients must be such that \( P \in SA \). Moreover, clause 1 of Definition 5.1 insures that if there is more than one agent, then one of the rules for removing locations is applicable, and clause 3 insures that location \textit{root} is the only location that is not removed. So in a net in normal form there will be only the ambient \textit{rootname} located at \textit{root} and, from the previous considerations, containing an SA process. \( \square \)

The reconstruction of an SA term from a PAN net is defined as follows.

**Definition 5.14.** Given a well-formed net \( A \), \( \llbracket A \rrbracket = P \) if

\[ W(A) \equiv (\nu \tilde{p}) \text{root}:\text{rootname}[P] \text{rootparent} \]

We now show the relation between reductions in PAN and reductions in SA. In particular, administrative reductions do not change the SA term encoded, whereas proper reductions may be simulated by reductions in SA.
Observe that, from Lemma 5.4, we know that administrative reductions preserve well-formedness. However, it is not true that $A \xrightarrow{pr} A'$ and $A$ well-formed implies $A'$ well-formed.

On one side, applying rule [LOCAL-OPEN] to a well-formed net may produce a net in which clause 11 of Definition 5.1 does not hold. For example, let $A$ be

$$h : n[\text{open } m, h'] | \text{open } m. M. Q | Q' | h': m[\text{wait}. M'. P | P']_h$$

If $P'$ and $Q'$ do not have unguarded prefixes, then $A$ is well-formed. However,

$$A \xrightarrow{pr} h : n[\text{wait}. M. Q | Q']_k | h'[\text{migrate}] | h': m[\text{wait}. M'. P | P']_h \quad (5)$$

and the net obtained violates clause 11 of Definition 5.1.

On the other side, applying [LOCAL-COM] to a well-formed net may produce a net in which clause 9 of Definition 5.1 does not hold. For example,

$$h : n[\langle M \rangle | (x). M'. P | M. Q]_k \xrightarrow{pr} h : n[M'. P[M/x] | M. Q]_k \quad (6)$$

produces a net that violates clause 9. (A similar example can be given for clauses 11 and 10.) Requiring that the term encoded by $A$ be well-typed eliminates the problems of equations (5) and (6), since a single-threaded ambient cannot have (or reduce to a term with) more than one unguarded prefix.

However, let $A = h : n[\langle M \rangle | (x). M'. P | \text{wait}. Q]_k \parallel h\{\text{Completion}\}$, even though $A$ is well-formed and encodes a well-typed SA term, the reduction

$$A \xrightarrow{pr} h : n[M'. P[M/x]] | \text{wait}. Q]_k \parallel h\{\text{Completion}\} \quad (7)$$

produces a net that violates clause 8.

Moreover, let $A' = h : n[\langle M \rangle | (x). M'. P | \text{wait}. Q]_k \parallel h\{\text{Completion}\}$, where $Q$ does not have unboxed prefixes. Again $A'$ is well-formed and encodes a well-typed SA term, but

$$A' \xrightarrow{pr} h : n[M'. P[M/x]] | \text{wait}. Q]_k \parallel h\{\text{Completion}\} \quad (8)$$

produces a net that violates clause 9. The problem of Eqs. (7) and (8) is caused by the fact that wait prefixes corresponding to completions do not correspond to capabilities in the encoded term. To cope with these problems, we define a weaker notion of well-formedness, quasi-well-formedness, and show that this property is preserved by proper reductions.

**Definition 5.15.** A net $A$ is quasi-well-formed if it verifies clauses 1 $\div$ 7 and 10 $\div$ 12 of Definition 5.1, and the following:

9. (a) If an ambient (located or not) is not the target of a completion, then it cannot have an unboxed subterm with more than one unguarded prefix.

(b) If there is a completion $h\{\text{Completion}\}$, and the ambient at $h$ (up to $\equiv$) is $h : n[Q_1 | \text{wait}. Q_2]_h$, then $Q_1 | Q_2$ has at most one unguarded prefix.

So, in quasi-well-formedness, clause 8 of Definition 5.1 is dropped and clause 9 is replaced by its weaker version above. Note that for migrate and register completions clauses 11 and 10 of Definition 5.1 imply 9.b above. Therefore, a quasi-well-formed net without completion messages is well-formed.
Let \( A \) be a well-formed net, and \( \| A \| = P \) where \( P \) is a single-threaded SA term.

(1) \( A \xrightarrow{\text{adm}} A' \) then \( \| A' \| = P \).

(2) \( A \xrightarrow{\text{pr}} A' \) then \( A' \) is quasi-well-formed, \( \| A' \| = P \) and \( P \xrightarrow{} P' \).

**Proof.** (1) Let \( A \xrightarrow{\text{adm}} A' \). Then, \( A \equiv (v \bar{p}) A_1 \parallel A_2 \), and \( A' \equiv (v \bar{p}) A'_1 \parallel A_2 \).

If \( A_1 = h: n(m[P]) \parallel Q \), \( A'_1 \) is the antecedent of the rule \([\text{NEW-LOCAMB}]\), then
\[
A'_1 = h: n(Q)_{h'} \parallel (wk) k: m[P]_h
\]
where \( k \) can be chosen such that it does not occur in \( h: n(Q)_{h'} \parallel A_2 \). Therefore \( A'_1 \parallel A_2 \equiv (wk) A'' \) where,
\[
A'' = h: n(Q)_{h'} \parallel k: m[P]_h \parallel A_2.
\]

Since \( A'' \xrightarrow{\text{adm}} A_1 \parallel A_2 \) using rule \([\text{REM-LOC}]\), we also have that, applying rule \([\text{STRUCT-CONG}]\), \( A'_1 \parallel A_2 \xrightarrow{\text{adm}} A_1 \parallel A_2 \). Therefore \( A \) and \( A' \) have the same \( \xrightarrow{\text{adm}} \) normal form, from which we derive that \( \| A' \| = \| A \| \).

If \( A_1 \) is the antecedent of the rule \([\text{FW-MSG}]\), that is, \( A_1 = h \triangleright k \parallel h[M_{\text{MsgBody}}] \), then \( A'_1 = h \triangleright k \parallel k[M_{\text{MsgBody}}] \). From \( A \) well-formed, we derive that \( M_{\text{MsgBody}} \) is not a completion. So all the rules of \( \xrightarrow{\text{adm}} \) applicable to \( A \) are also applicable to \( A' \) (the rules \([\text{REM-REQ-...}]\) do not depend on the target of the message) and so, since the normal forms \( \text{w.t.} \ x \rightarrow \) \( \xrightarrow{\text{adm}} \) do not contain messages and forwarders, \( \forall \nu(A'') \equiv \nu(A) \).

Therefore \( \| A' \| = \| A \| \).

If \( A_1 \equiv h: n[M, Q \mid Q']_k \), where \( M \in \{ \text{in} m, \text{out} m, \text{in} m, \text{out} m \} \), and \( Q' \) does not have unguarded ambients (\( A \) is the antecedent of a \([\text{REQ-...}]\) rule), then
\[
A'_1 = h: n[\text{wait}. Q \mid Q']_k \parallel k[M, h], \quad A'_1 \xrightarrow{\text{adm}} A_1 \text{ using } [\text{REM-REQ-PROCESS}], \text{ and so } \| A' \| = \| A \| .
\]

If \( A_1 \equiv h: n[P]_k \parallel h[M, h'] \) (the antecedent of the rule \([\text{CONSUME-REQ}]\)), then
\[
A'_1 = h: n[P \parallel \{M, h' \}]_k . \quad \text{Since } A \text{ is well-formed, then } A_2 \equiv A_3 \parallel A'_2 \text{ where }
A_3 = h': m[Q \mid \text{wait}. Q']_k . \quad \text{Note that, } A_1 \parallel A_3 \xrightarrow{\text{adm}} h: n[P]_k \parallel h': m[Q \mid M. Q']_k \text{ using } [\text{REM-REQ-PROCESS}], \quad A'_1 \parallel A_3 \xrightarrow{\text{adm}} h': m[Q \mid M. Q']_k \text{ using } [\text{REM-REQ-MESS}], \text{ and therefore } \| A' \| = \| A \| .
\]

If \( A_1 \) is the antecedent of a rule \([\text{COMPL-...}]\), the result is obvious since the \([\text{COMPL-...}]\) rules are also rules of \( \xrightarrow{\text{adm}} \).

(2) Let \( A \xrightarrow{\text{pr}} A' \). Then \( A \equiv (vp) (A_1 \parallel A_2) \), \( A' \equiv (vp) (A'_1 \parallel A_2) \), and \( A_1 \xrightarrow{\text{pr}} A'_1 \). Consider the four possible derivations corresponding to the application of the \([\text{LOCAL-...}]\) rules.

\([\text{LOCAL-COM}]\): Let \( A_1 = h: n[M \mid (\chi). Q \mid Q']_k \), and \( A'_1 = h: n[M_{/\chi} \mid Q']_k \). To show that \( A' \) is quasi-well-formed note that: all the clauses of the definition of quasi-well-formed (Definition 5.15) except 9, 10, and 11 are easily verified for \( A' \).
To verify clause 9 note that \( Q' \equiv Q_1 \| \cdots \| Q_r \) where none of the \( Q_i \) is an unguarded ambient. If \( Q^{(M/k)} \) or all the \( Q_i \)'s do not have an unguarded prefix, then clause 9 holds for \( A' \), since it holds for \( A \).

Assume that \( Q^{(M/k)} = M' \). \( Q'' \) and for some \( i, Q_i \) has an unguarded prefix (since clause 9 holds for \( A \) there is at most one of such \( Q_i \)). Assume that \( Q_i = \text{wait} \cdot Q'_i \).

From clause 7, there is either a corresponding request (agent or process), or a corresponding completion.

If the \text{wait} prefix corresponds to a request, then from the definition of \( \sim^{adm}_{\text{adm}} \) in \( \llbracket A \rrbracket \) the ambient \( n \) is \( n[(M) \mid (x), Q \mid M''. Q_i \mid \cdots] \) for some \( M'' \in (\overline{\tau} m, \overline{\text{open}} m, \text{in} m, \text{out} m) \). So in \( P' \) the ambient \( n \) would be \( n[M'. Q'' \mid M''. Q_1 \mid \cdots] \). However, this is a contradiction, since \( P \) single threaded and Proposition 2.2 imply that \( P' \) is single threaded. (The same proof holds when \( Q_i = M''. Q'_i \) where \( M'' \) is a capability.)

If the \text{wait} prefix corresponds to a completion, then from the definition of \( \sim^{adm}_{\text{adm}} \) in \( \llbracket A \rrbracket \) (given the completion rules of \( \sim^{adm}_{\text{adm}} \)) there is an ambient \( m \) (that is not necessarily \( n \)) such that \( m[(M) \mid (x), Q \mid Q_i \mid \cdots] \). Therefore, as before, the fact that \( P \) is well-typed, and types are preserved by reduction, implies that \( Q_i \) does not have an unboxed prefix, and so clause 9.b of Definition 5.15 holds for \( A' \).

Note that all the reductions \( \sim^{adm}_{\text{adm}} \) applicable to \( A \) are also applicable to \( A' \). So, since \( \llbracket A' \rrbracket = P \) we have that \( \llbracket A' \rrbracket = P' \), where \( P' \) is obtained from \( P \) by replacing \( (M) \mid (x), Q \) in ambient \( n \) with \( Q^{(M/k)} \), and \( P \longrightarrow P' \) via [R-COM].

**LOCAL-IN.** Let \( A_1 = h'n[(\text{in} m, h') \mid (\overline{\text{in}} m, k') \mid Q''_k], \) and \( A'_1 = h':n[Q' \mid k'| \overline{\text{in}} m] \). The only nontrivial clauses to verify are clauses 2, 3, and 7. To verify 2, note that from clause 12 for \( A \) we derive that \( k \) is the parent location of both \( h' \) and \( k' \), and so \( k' \leq h' \) (also \( h' \leq k' \)). Hence, (assuming \( A' \) is well-formed (clause 7) the ambients located at \( h' \) and \( k' \) are in wait state, and \( h' \neq k' \). Therefore clause 7 holds also for \( A' \) with the completions generated by the reduction.

Now we want to show that \( \llbracket A' \rrbracket = P' \) where \( P' \) is such that \( P \longrightarrow P' \). First consider the case that \( \llbracket A' \rrbracket = P' \). From clause 7 we have that \( A_2 \equiv A'_2 \equiv A''_2 \) where \( A'_2 = h':m'[\text{wait} \cdot P_1 \mid P_2] \|
\|
 k':m[\text{wait} \cdot Q_1 \mid Q_2] \text{k''} \text{. Applying rule } \llbracket \text{REM-REQ-MESS} \rrbracket \text{ twice we have that} \)

\[
A \sim^{adm}_{\text{adm}} h:n[Q' \mid k \parallel h';m'[\text{in} m \cdot P_1 \mid P_2] \text{k''} \parallel k':m[\overline{\text{in}} m \cdot Q_1 \mid Q_2] \text{k''} \parallel A''_2
\]

From clause 12.b, both \( h' \) and \( k' \) are children of \( h \) (there could be forwards from \( h'' \) and/or \( k'' \) to \( h \)), therefore from the definition of \( \llbracket [] \rrbracket \),

\[
\llbracket A \rrbracket = (v\tilde{n})(m_1 \mid m_2 \cdots m_s[ n[m'[\text{in} m \cdot P_1 \mid P_2] \mid m[\overline{\text{in}} m \cdot Q_1 \mid Q_2] \mid Q'' \mid Q_3] \mid \cdots] \mid Q_1] \mid Q_0)
\]

Applying the rules [COMPL-PARENT] and [COMPL-COIN] to \( A' \) we get

\[
A' \sim^{adm}_{\text{adm}} h:n[Q' \mid k \parallel h';m' \mid P_1 \mid P_2] \parallel k':m[Q_1 \mid Q_2] \parallel A''_2
\]

and therefore

\[
\llbracket A' \rrbracket = (v\tilde{n})(m_1 \mid m_2 \cdots m_s[ n[m[Q_1 \mid Q_2 \mid m'[P_1 \mid P_2]]] \mid Q'' \mid Q_3] \mid \cdots] \mid Q_1] \mid Q_0)
\]
Let $P' = (\mathcal{A}'), P \rightarrow P'$ applying rule [R-IN].

[LOCAL-OUT]. Let $A_1 = h : n[\text{out}(n, h'), \text{out}(n, Q'), k], A_1' = h : n[Q | Q_1]'k || h'[\text{go} k]$. Again, the only non-trivial clauses to verify are: 2, 7, and 9.

As before, from the fact that clause 12 holds for $A$, we derive that clause 2 holds for $A'$, and since 3a holds for $A$, then 3b holds for $A'$.

From $A$ well-formed (clause 7) we derive that the ambient located at $h'$ is in wait state. Therefore clause 7 holds also for $A'$ with the completion generated by the reduction.

Finally, since clause 9 holds for $A$, $Q'$ cannot have an unguarded prefix (in $A_1$ there cannot be more than one unguarded prefix). Therefore in $A'$ there can be at most one unguarded prefix (exactly one if $Q$ has an unguarded prefix). So clause 9 holds for $A'$.

The proof that $\mathcal{A}' = P'$ for $P \rightarrow P'$ is as for the case of [LOCAL-IN].

[LOCAL-OPEN]. Let $A_1 = h : n[\text{open}(m, h') | \text{open}(m, Q | Q')_k], A_1' = h : n[\text{wait}(Q | Q_1')_k || h'[\text{migrate}]). We verify only clauses 7 and 11; all the others are either trivial or have a proof similar to the previous cases. Since $A$ is well-formed, $A_2 = h' : m[\text{wait}(P | P')_k]. || A_2'$ and $h$ is the parent location of $h'$. Consider clause 7. The ambient in wait state located at $h'$ was in $A$ the source of a request (that does not occur in $A'$) and is in $A'$ the target of a migrate completion. The new ambient in wait state (located at $h$) is the ambient parent of the target of a migrate message, so clause 7 holds for $A'$. Finally we want to verify proposition 11 for $A'$. Note that, from the well-formedness of $A$, we derive that $Q'$ cannot contain a subcomponent with a wait prefix. From the definition of $\mathcal{A}$ in $P$ we have that the ambient $n$ is $n[m[\text{open}(m, R | R')_k | \text{open}(m, Q | Q')_k], Q, Q' \rightarrow P' \rightarrow P'$ where $P'$ contains $[Q | Q' | R | R' | \ldots]$ and $P'$ (from subject reduction) is well-typed (with $m$ single threaded). Therefore from Proposition 2.2, clause 11 holds for $A'$.

Again the proof that $\mathcal{A}' = P'$ for $P \rightarrow P'$ is as for the case of [LOCAL-IN].

We now prove the properties relating reductions in SA with reductions in PAN.

**Lemma 5.17.** Let $P$ be a single-threaded SA term, and $A$ be a well-formed net such that $\mathcal{A} = P$. If $P \rightarrow P'$, then there is $A'$ well-formed such that $\mathcal{A}' = P'$ and $A \Rightarrow A'$.

**Proof.** If $P \rightarrow P'$ there are $m_1, \ldots, m_s, Q_0, \ldots, Q_s, Q$, and $Q'$ such that

- $P \equiv (\text{magn}) (m_1[m_2[\ldots m_s[Q | Q_1] \ldots] | Q_1] | Q_0),$
- $P' \equiv (\text{magn}) (m_1[m_2[\ldots m_s[Q' | Q_1] \ldots] | Q_1] | Q_0),$

$Q$ is the left-hand-side of one of the rules [R-MSG], [R-IN], [R-OUT], or [R-OPEN], and $Q'$ is the right-hand-side of the corresponding rule.

Let $A$ be such that $\mathcal{A} = P$. Let $B$ be the administrative normal form of $A$, $A \Rightarrow A$. From Lemma 5.16.1 $\mathcal{B} = \mathcal{B}$. From Lemma 5.9, for some $Q'_i, h_j (1 \leq j \leq s - 1)$ and $Q''$

$B \equiv (\text{magn}) (\text{root:rootname}[R]_{\text{rootparent}} || h_1 : m[Q'_1]_{\text{rootname}} || h_2 : m[Q'_2]_{\text{rootparent}} || \ldots || h_s : m[Q''_{k_{s-1}}]_{\text{rootname}})$

where there are no messages in $Q'_0$, and $k_i \leq h_i, 1 \leq i \leq s - 1$, that is in $Q'_0$ there are agents:
From Lemma 5.16.2 $A'$ is quasi-well-formed. Moreover, from definition of $A'$ we have that $|\{A\}| = P'$. Since $B$ is in administrative normal form, $B$ does not contain any completion and no completion is generated by the [LOCAL-COM] rule. So also $A'$ does not contain any completion, and therefore $A'$ is well-formed.

If the reduction axiom applied is [R-OPEN], then $Q = \text{open} n. R | n[\text{open} n. R_1 | R_2 ]$ and $Q' = R | R_1 | R_2$. So

- $P = (v\overline{n}) (m_1 [m_2 [\cdots m_s [\text{open} n. R | n[\text{open} n. R_1 | R_2 ] | Q_1 ] | \cdots ] | Q_1 ] | Q_0)$.
- $P' = (v\overline{n}) (m_1 [m_2 [\cdots m_s [R | R_1 | R_2 | Q_1 ] | \cdots ] | Q_1 ] | Q_0)$.

Since $B$ is in normal form,

$$B = (v\overline{n}) (\text{root} : \text{rootname}[ R ] \text{rootparent} | h_1 : m_1 [ Q_1 ] \text{rootname} | h_2 : m_2 [ Q_2 ] \text{rootparent} | \cdots | h_i : m_i [ Q_i ] \text{rootname} | \text{open} n. R | \{\text{open} n. h\} | Q_{i-1} | h : n[\text{wait}. R | R_1 | R_2 ] \text{rootname} | Q')$$

where as before $k_i \leq h_i, 1 \leq i \leq s$. Applying rule [LOCAL-OPEN], we obtain

$$B' = (v\overline{n}) (\text{root} : \text{rootname}[ R ] \text{rootparent} | h_1 : m_1 [ Q_1 ] \text{rootname} | h_2 : m_2 [ Q_2 ] \text{rootparent} | \cdots | h_i : m_i [ Q_i ] \text{rootname} | \text{wait}. R | Q_{i-1} | h[\text{migrate}] | h : n[\text{wait}. R | R_1 | R_2 ] \text{rootname} | Q')$$

and then applying [COMPL-MIGR] followed by [COMPL-REG] we get

$$A' = (v\overline{n}) (\text{root} : \text{rootname}[ R ] \text{rootparent} | h_1 : m_1 [ Q_1 ] \text{rootname} | h_2 : m_2 [ Q_2 ] \text{rootparent} | \cdots | h_i : m_i [ Q_i ] \text{rootname} | R | R_1 | R_2 ] \text{rootparent} | A')$$

From Lemma 5.16.2 $B'$ is quasi-well-formed. Since administrative reductions preserve quasi-well-formedness and $A'$ does not contain completions, $A'$ is well-formed. Moreover, from definition of $A'$, we get that $|\{A\}| = |\{B\}| = P'$.

If the reduction axioms applied are [R-IN] or [R-OUT] the proof is similar to the previous one.

We now define a notion of observability both for SA and PAN term. This notion corresponds to the standard idea that an external observer may interact with the system by opening or entering a given ambient.

**Definition 5.18 (Observability).** We write $A \downarrow_n$ if

$$A = (v\overline{n}) (\text{root} : \text{rootname}[ M, h ] | P ] \text{rootparent} | A')$$
Theorem 5.19

where $M \in \overline{\{\text{in} n, \text{open} n\}}$ and $n \not\in \tilde{p}$. We write $A \downarrow_n$ if $A \xrightarrow{\text{adm}} A'$ for some $A'$ such that $A' \downarrow_n$.

Observability in SA is defined similarly as for PAN: $P \downarrow_n$ if

$$P \equiv (\tilde{w}n) (n[M, Q_1 | Q_2 ] | Q_3)$$

where $M \in \overline{\{\text{in} n, \text{open} n\}}$ and $n \not\in \tilde{n}$. We write $P \downarrow_n$ if $P \xrightarrow{\text{adm}} P'$, for some $P'$ such that $P' \downarrow_n$

**Theorem 5.19 (Adequacy).** Let $P \in SA$ be such that $P$ contains only single-threaded ambients. It holds that, for all $n$, $P \downarrow_n$ if $P \downarrow_n$.

**Proof.** (Only if) By induction on the length of the reduction from $P$ to $P'$ such that $P' \downarrow_n$.

We will prove the following stronger statement (needed for the inductive step):

$$P \downarrow_n \text{ implies that for all } A \text{ well-formed such that } \llbracket A \rrbracket = P \text{ we have that } A \downarrow_n.$$  

Since $\llbracket P \rrbracket = P$, this statement implies the result.

- (Base case) If $P \downarrow_n$, then $P \equiv (\tilde{w}n) (n[M, Q_1 | Q_2 ] | Q_3)$ where $M \in \overline{\{\text{in} n, \text{open} n\}}$.

Let $A$ be such that $\llbracket A \rrbracket = P$, then $\forall V(A) = A'$ where

$$A' = (\tilde{w}n) (\text{root:rootname}[n[M, Q_1 | Q_2 ] | Q_3]_{\text{rootparent}})$$

We will prove that $A \downarrow_n$, that is $A \xrightarrow{\text{adm}} A''$ where $A''$ is congruent to

$$(\tilde{w}n) (\text{root:rootname}[M, h] | P']_{\text{rootparent}} \parallel A'')$$

(9)

where $M \in \overline{\{\text{in} n, \text{open} n\}}$ for some $P'$ and $A''$. By induction on the length of the $\xrightarrow{\text{adm}}$ reduction from $A$ to its $\xrightarrow{\text{adm}}$ normal form $A'$.

- (Base case) Let $A' = A$.

$A \xrightarrow{\text{adm}} A_1$ where $A_1 = B \parallel A'_1$ and $B = (\tilde{w}n) \ (\text{root:rootname}[Q_3]_{\text{rootparent}}$ and $A'_1 = (vk)k:n[M, Q_1 | Q_2 ]_{\text{root}}$ (using rule [NEW-LOCAMB])

$$A_1 \xrightarrow{\text{adm}} A_2$$ where $A_2 = B \parallel A'_2$ where $A'_2 = (vk)k:n[M, Q'_1 | Q']_{\text{root}}$ (does not have unguarded ambients) and $A'_2 = 1_{\leq i \leq r} (vk_i)k:n_i[Q''_i]_{\text{root}}$ (using a number of application of [NEW-LOCAMB])

$$A_2 \xrightarrow{\text{adm}} A_3$$ where $A_3 = B \parallel A''_{1'} \parallel \text{root}(M, k) \parallel A'_3$ where $A'_3 = (vk)k:n[\text{req}(Q'_{1'}, Q'_{1'})_{\text{root}}$ (using [REQ-...])

$$A_3 \xrightarrow{\text{adm}} A_4$$ where $A_4 = (\tilde{w}n) (\text{root:rootname}[M, h] | Q_3]_{\text{rootparent}} \parallel A''_{1'} \parallel A''_4$

Since $A_4 \downarrow_n$ we have that $A \downarrow_n$.

- (Inductive case) Let $A = A_1 \xrightarrow{\text{adm}} \cdots \xrightarrow{\text{adm}} A_r = A'$ where $r > 1$. We want to show that $A \downarrow_n$ using the inductive hypothesis that $A_2 \downarrow_n$, that is $A_2 \xrightarrow{\text{adm}} B$ where $B$ is congruent to (9).

By case analysis on the rule used for the reduction $A_1 \xrightarrow{\text{adm}} A_2$. Let $A_1 \equiv (\tilde{w}n) A'_1$ and $A_2 \equiv (\tilde{w}n) A'_2$.

If the rule applied is one of the [COMPL-...], then $A_1 \xrightarrow{\text{adm}} A_2$ by applying the same [COMPL-...], $A'_1 \equiv h \parallel k\parallel A''_1$, and $A''_2 = A''_1$. Since $h$ does not
(If) By induction on the length of the reduction from \( A = \llbracket P \rrbracket \) to \( B \) such that \( B \downarrow_n \).

- (Base case) \( A \downarrow_n \) implies
  \[
  A \equiv (\nu \gamma) \left( \text{root: rootname}(\{M, h\} | P') \right) \text{rootparent} \llbracket A' \rrbracket
  \]
where $M \in \{\text{in}, \text{open}\}$ and $n \not\in \bar{p}$. Since $A$ is well-formed clause 7 of Definition 5.1 implies that $A' \equiv h_n[\text{wait}.Q | Q']_k || A''$. Therefore applying rule \textsc{rem-req-mess} $A \rightsquigarrow_{\text{adm}} B'$ where

$$B' = \text{root}:\text{rootname}[P']_{\text{rootparent}} || h_n[M.Q | Q']_k || A''.$$  

Moreover, from clause 12 of Definition 5.1 we have that $\text{root}$ is the ambient parent of $h$. So we also have that $W(B') = B''$ where

$$B'' \equiv \text{root}:\text{rootname}[P'' | n[M.Q | Q']_{\text{rootparent}}$$

for some $P''$. Therefore $\ll B'' \gg_n$.

- (Inductive case) Let $A = A_1 \rightsquigarrow \cdots \rightsquigarrow A_r = A' \rightsquigarrow A''$ with $r > 1$, and $A' \downarrow_n$. If $A_1 \rightsquigarrow_{\text{adm}} A_2$ then from Lemma 5.16.1 we have that $\ll A_1 \gg = \ll A_2 \gg$. Since by inductive hypothesis $\ll A_2 \gg \downarrow_n$, then also $\ll A_1 \gg \downarrow_n$.

- If $A_1 \rightsquigarrow_{\text{pr}} A_2$, then from Lemma 5.16.2 we have that $\ll A_1 \gg \rightarrow \ll A_2 \gg$. Again from the inductive hypothesis $\ll A_1 \gg \downarrow_n$. ☐

6. Correctness of the abstract machine for immobile ambients

A well-typed SA program may contain both single-threaded and immobile ambients. In this section we show how to extend the proof of correctness of the abstract machine to include also the immobile ambients. These are ambients that:

1. cannot jump into or out of other ambients;
2. cannot be opened.

Thus the only capabilities that an immobile ambient can exercise are $\text{in}$, $\text{out}$, and $\text{open}$. The main property that immobile ambients have is expressed from the following statement (a simple variant of a theorem in [14]).

**Proposition 6.1.** If $P$ is well-typed SA term and the subterm $n[Q]$ is an immobile ambient, then no unboxed subcomponent of $Q$ has the unguarded capabilities $\text{in}$, $\text{out}$, $\text{open}$.

Moreover, well-typedness is preserved under reduction (Subject Reduction), so the previous property is true under reduction.

Since immobile ambients can exercise several capabilities at the same time, in the abstract machine, an immobile ambient can contain several $\text{wait}$ prefixes. To be able to distinguish among them, and make sure that a completion message wakes up the right process, we add an index $i$ (where $i$ is a natural number) to $\text{wait}$ prefixes and messages. Thus in the syntax for nets we add the production

$$h[M\text{gBody}]_i$$

and in the process-related syntax we add the productions

$$\text{wait}_i.P \mid \{\text{Request}\}_i$$

The additions to the reduction rules are simple. We report them in Fig. 9. We assume that the set of names is partitioned into two (infinite) subsets, one for the names of the single-threaded ambients, the other for the names of the immobile ambients.
Local reductions

\[
\{ \text{in } n, k \} \mid \{ \text{in } n, k \} \xrightarrow{\text{IMM-LOCAL-IN}} h \{ \text{go } k \} \mid \{ \text{in } n, k \}
\]

open. \text{n. } P \mid \{ \text{open } n, h \} \xrightarrow{\text{IMM-LOCAL-OPEN}} \text{wait}_i. P \gg h \{ \text{migrate}\}_i \text{ i new and } m \text{ immobile}

Forwarder

\[
h \triangleright k \parallel h \{ \text{MsgBody}\}_i \xrightarrow{\text{FW-MSG-IMM}} h \triangleright k \parallel h \{ \text{MsgBody}\}_i
\]

Consumption of request messages

\[
h : n \{ P \} | \text{wait}_i, Q | h \{ \text{Request}\}_i \xrightarrow{\text{IMM-CONSUME-REQ}} h : n \{ P \} | \{ \text{Request}\}_i | h'
\]

Emission of request messages (should be \( h \neq \text{root} \), and \( n \) immobile)

\[
\{ \text{in } n \}, P \xrightarrow{\text{IMM-REQ-COIN}} \text{wait}_i. P \gg k \{ \text{in } n, h \}
\]

Consumption of completion messages

\[
h : n \{ P \} | \text{wait}_i, Q | h \{ \text{OK}\}_i \xrightarrow{\text{IMM-COMPL-COIN}} h : n \{ P \} | Q | h
\]

\[
h : n \{ P \} | \text{wait}_i, Q | h \{ \text{migrate}\}_i \xrightarrow{\text{IMM-COMPL-MIGR}} h : n \{ P \} | \{ \text{register } P \} | Q | h
\]

\[
h : n \{ P \} | \text{wait}_i, Q | h \{ \text{register } R\}_i \xrightarrow{\text{IMM-COMPL-REG}} h : n \{ P \} | Q \parallel R | Q | h
\]

Inference rules

\[
P \xrightarrow{\text{IMM-PAR-PROC}} Q
\]

\[
P \xrightarrow{\text{IMM-PAR-PROC}} Q
\]

The local rules are: the local rules for single threaded ambients of Fig. 6 and the rules IMM-LOCAL-IN and IMM-LOCAL-OPEN of Fig. 9.

Rule IMM-LOCAL-IN reflects the fact that when a \( \{ \text{in } n \} \) request from an immobile ambient matches an \( \text{in} \) request from a ST ambient an indexed completion for the immobile ambient (which corresponds to an indexed \text{wait} \) and a regular completion for the moving ST ambient are generated.

Rule IMM-LOCAL-OPEN reflects the fact that, in a well-typed SA term, a single-threaded ambient may be opened by an immobile ambient. (However, it must be the case that the local processes of the immobile ambient do not have any unguarded prefix.) Hence, if the ambient sending the \text{open} request is immobile, then an indexed request is generated. When combining the rules for immobile and ST ambients, to avoid ambiguity, we need to add to the rule LOCAL-OPEN of Fig. 6 the side condition that the ambient is single threaded, and the rule becomes

\[
\text{open. } n. P \mid \{ \text{open } n, h \} \xrightarrow{\text{IMM-LOCAL-OPEN}} \text{wait}_i. P \gg h \{ \text{migrate}\}
\]

where \( m \) is single threaded.
The rules FW-MSG-IMM, IMM-CONSUME-REQ, and the completion rules are obvious. 
Note that, since the local processes of immobile ambients can only have the unguarded prefixes \(\text{in} \), \(\text{out} \), and \(\text{open} \), the only requests generated by immobile ambients are \(\text{in} \) requests (rule IMM-REQ-COIN). Again, when combining the rules for immobile and single-threaded ambients, to avoid ambiguity, we need to add to the rule REQ-COIN of Fig. 6 the side condition that \(n\) is single threaded.

Finally, in rule IMM-PAR-PROC, we write \(\text{wait}(R)\), where \(R\) is a process, for the set
\[
\{i : \text{wait}_i \text{ appears in } R\}
\]

The side condition in the rules ensures that all \(\text{wait}\) prefixes inside an ambient have different indices.

We report below the modifications to the definition of well-formedness (Definition 5.1).

**Definition 6.2 (Well-formedness, with Immobile Ambients).** Clauses 1–5 are as in Definition 5.1. The other clauses are modified as follows.

6. The prefixes \(\text{wait}\), and \(\text{wait}_i\) (for some \(i\)) can only appear as the unguarded prefixes of the local process of a located ambient.

7. (a) A located ambient with home location \(h\) exists and is in \(\text{wait}\) state iff one of the following holds:
   (i) \(h\) is the source of at most a request (in a message or in a local process);
   (ii) \(h\) is the target of at most a completion message;
   (iii) \(h\) is the parent ambient of the target of at most a migrate message (which may be indexed).

   (b) A located ambient with home location \(h\) exists and is in \(\text{wait}_i\) state iff one of the following holds:
   (i) \(h\) is the source of a \(\text{in}\) request (in a message or in a local process) with index \(i\);
   (ii) \(h\) is the target of a register or a \(\text{OK}\text{in}\) completion with index \(i\);
   (iii) \(h\) is the parent ambient of the target of a migrate message with index \(i\).

8. A located ambient with home location \(h\) such that \(h\) is the target of a completion (indexed or not) does not contain subambients.

9. (a) No single-threaded ambient (located or not) has an unboxed subterm with more than one unguarded prefix. No local process of a (located) single-threaded ambient has an unguarded prefix \(\text{wait}_i\) for some \(i\).

   (b) No immobile ambient (located or not) has an unboxed subterm with the unguarded prefixes \(\text{in}, \text{out}, \text{open}\) or \(\text{wait}\). For all \(i\), no local process of a (located) immobile ambient has more than one unguarded prefix \(\text{wait}_i\) (that is, all the indices of \(\text{wait}\) prefixes must be distinct).

10. (a) If there is a message \(h[\text{register } R]\) and its ambient target (up to \(\equiv\)) is \(h:n[P_1 | \text{wait}.P_2],\) then \(P_1 | P_2 | R\) has at most one unguarded prefix, and it is not a \(\text{wait}\).

     (b) If there is a message \(h[\text{register } R]_i\), then \(R\) does not have the unguarded prefixes \(\text{in}, \text{out}, \text{open}\), and \(\text{wait}\).
(11) (a) If there is a message \( h \{ \text{migrate} \} \) and the ambient at \( h \) (up to \( \equiv \)) is \( h: n[Q_1 \mid \text{wait}.P_1 \mid P_2]\) then \( Q_1 \mid Q_2 \mid P_1 \mid P_2 \) has at most one unguarded prefix.

(b) If there is a message \( h \{ \text{migrate} \} \), and the ambient at \( h \) (up to \( \equiv \)) is \( h: n[Q_1 \mid \text{wait}.Q_2]_h \), then \( Q_1 \mid Q_2 \) does not have the unguarded prefixes \( \text{in}, \text{out}, \text{open}, \) and \( \text{wait} \).

(12) (a) In every request message \( k\{M, h\} \), or \( k\{M, h\} \) the location \( h \) is a child of \( k \), and

(b) if inside an ambient located at \( k \) there is a request \( \{M, h\} \) or \( \{M, h\} \) then \( h \) is a child of \( k \).

In the previous definition the clauses of well-formedness for the combination of immobile and single-threaded ambients coincide with the one of Definition 5.1 for non-indexed \( \text{wait} \), requests, and completions (which come from single-threaded ambients).

Consider clause 7b. These conditions are motivated by the fact that a \( \text{wait}_i \) may either come from applying \( \text{IMM-REQ-COIN} \) in which case there is a matching request, or from applying \( \text{IMM-LOCAL-OPEN} \). In this case the \( \text{wait}_i \) replaces an \( \text{open} \) prefix which is contained in the parent ambient of the ambient containing the migrate completion indexed by \( i \), and the \( \text{open} \) request is substituted by a migrate completion indexed by \( i \). This is why in clause 7a.iii a \( \text{wait} \) prefix may correspond to an indexed migrate. So we have a \( \text{wait} \) matching an indexed migrate completion that is substituted with a register completion by rule \( \text{IMM-COMPL-MIGR} \). The register completion and the matching \( \text{wait}_i \) prefix generated by \( \text{IMM-LOCAL-OPEN} \) will be consumed by an application of the \( \text{IMM-COMPL-REG} \) rule.

Clauses 10 and 11 assert that after consuming a migrate or register completion we obtain a well-formed net. This clause is satisfied by a net representing an SA term, since ambients are either immobile or single threaded and this property is preserved by reduction. So, if an ambient is opened inside a single-threaded (immobile) ambient the resulting ambient is still single threaded (immobile).

Administrative reductions are Church-Rosser also when we have immobile ambients. The proof of the theorem requires only minor additions and modifications to the proof for the case of single-threaded ambients.

**Theorem 6.3.** Suppose \( A \) is well-formed. If \( A \models \text{adm} A_1 \) and \( A \models \text{adm} A_2 \) then either \( A_1 \equiv A_2 \), or there is \( A_3 \) such that \( A_1 \models \text{adm} A_3 \) and \( A_2 \models \text{adm} A_3 \).

For the translation \( [P] \) of Definition 5.10 to be a well-formed net, we need the properties of well-typed ambients expressed by Propositions 2.2 and 6.1.

To extend the mapping \( [\cdot] \) to include also immobile ambients we add to \( \sim_{\text{adm}} \) the rules in Fig. 10. Lemma 5.13 holds also for this reduction relation. The only difference in the proof is that, for clause 2, there are more critical pairs, since an ambient may have more than one \( \text{wait}_i \) prefix. However, from well-formedness, the indexes of such prefixes are distinct, and the result follows easily.

The definition of quasi-well-formedness is extended as follows.

**Definition 6.4.** A net \( A \) is quasi-well-formed if it verifies clauses 1 \( \div \) 7 and 10 \( \div \) 12 of Definition 6.2, and the following clause 9:

(a1) No single-threaded ambient (located or not), that is not the target of a completion, has an unboxed subterm with more than one unguarded prefix.
Removing indexed requests

\[ h: n[P | \text{wait}_i, Q|k] \parallel k': \text{m} \{[M, h] | P\}'|k' \parallel A \nLeftarrow \text{adm} h: n[P | M, Q|k] \parallel k': \text{m} | P\}'|k' \parallel A \]  
\text{[REM-IND-REQ]}

\[ h: n[P | \text{wait}_i, Q|k] \parallel k'[M, h] \parallel A \nLeftarrow \text{adm} h: n[P | M, Q|k] \parallel A \]  
\text{[REM-IND-REQ-PROCESS]}

Consuming indexed completions

\[ h: n[P | \text{wait}_i, Q|k] \parallel h[\text{migrate}] | h | \text{register} P | Q_i | k \parallel A \nLeftarrow \text{adm} h: n[P | M, Q|k] \parallel A \]  
\text{[COMPL-IND-MIGR]}

\[ h: n[P | \text{wait}_i, Q|k] \parallel h[\text{register} R_i] | A \nLeftarrow \text{adm} h: n[P | Q | R|k] \parallel A \]  
\text{[COMPL-IND-REG]}

Fig. 10. Rules of \( \nLeftarrow \text{adm} \) for immobile ambients.

(a2) No local process of a (located) single-threaded ambient has an unguarded prefix \( \text{wait}_i \) for some \( i \).

(a3) If there is completion \( h[\text{Completion}] \), and at \( h \) there is a single-threaded environment which (up to \( \equiv \)) is \( h: n[Q_1 | \text{wait}_i, Q_2|k] \), then \( Q_1 \parallel Q_2 \) has at most one unguarded prefix.

(b1) No immobile ambient (located or not) has an unboxed subterm with the unguarded prefixes \( \text{in}, \text{out}, \text{open} \) or \( \text{wait} \).

(b2) For all \( i \), no local process of a (located) immobile ambient has more than one unguarded prefix \( \text{wait}_i \) (that is, all the indices of \( \text{wait} \) prefixes must be distinct).

Lemmas 5.16 and 5.17 can be proved also for nets containing immobile ambients.

The notion of observability for SA terms is as for single-threaded ambients. For well-formed PAN nets we have to add the clause corresponding to the fact that the ambient willing to engage in an interaction with the environment be an immobile ambient.

**Definition 6.5 (Observability).** We write \( A \Downarrow_n \) if

- \( A \equiv (\nu n) (\text{root} : \text{rootname} \{[M, h] | P\}'|\text{rootparent} \parallel A') \), or
- \( A \equiv (\nu \tilde{p}) (\text{root} : \text{rootname} \{[M, h] | P\}'|\text{rootparent} \parallel A') \) (for some \( i \)),

where \( M \in \{\text{in} n, \text{open} n\} \) and \( n \not\in \tilde{p} \). We write \( A \Downarrow_n \) if \( A \nRightarrow \Downarrow_n \).

**Theorem 6.6 (Adequacy).** Let \( P \) be a well-typed SA term. It holds that \( P \Downarrow_n \) iff \( P \Downarrow_n \), for all \( n \).

### 7. Implementation architecture

Our implementation, written in Java, follows the definition of the abstract machine. As usual in implementation of process calculi, rules for arbitrarily changing the order of parallel components can be taken into account with some randomization mechanism; in our implementation we do not adopt this, which may reduce non-determinism. The main difference is that the implementation allows clustering of agents on the same IP node, i.e.,
a physical machine. The implementation is made of three layers: agents, nodes and the network (Figs. 11 and 12). The address $k$ of an agent is composed of the IP-name of the node on which it resides, plus a suffix, which is different for each agent in that node. This ensures that each agent has a unique location name. Each agent is executed by an independent Java thread; the processes local to an ambient are scheduled using a round-robin policy. Each agent knows its name, its address, its parent’s address, and keeps a link to its node.

From a physical point of view, the messages exchanged between agents are of two kinds: local, when both agents reside on the same node, and remote, when two distinct nodes are involved. In each node a special Java RMI object, with its own thread of execution, takes care of inter-node communications. For this, nodes act alternatively as clients (requiring that a message is sent to another computer) and as servers (receiving a message and pushing it into a local mail-box). A node can also spawn a new local agent in response to a remote creation request. The node layer is implemented using Java RMI and serialization, and the network layer simply provides IP-name registry for RMI communications to take place (using Java RMIregistry).

7.1. Agents from creation to destruction.

An agent acts as an interpreter for the ambient expressions that constitute its local processes. When the agent wants to create a subambient, it sends a special message to its node, which will spawn a new agent hosting the subambient code. We also allow remote creation of new agents: an agent may send a message to a node different from its own, to demand the creation of subambients. This corresponds to the addition of a primitive create $n[P]$ at $h$, where $h$ is the IP-name of a node, to the abstract machine. When the execution of an ambient expression begins on a given node, the first action is the local creation of a root agent. An agent resides on the same node until it is opened; then, its processes are serialized and sent via RMI to the parent agent.
A forwarder is a special kind of agent with the form \( h \leadsto k \), and its intended meaning is that every message arriving at \( h \) is sent to \( k \); agents of this kind do not contain running processes, nor subambients, so they absorb very little memory and computational power, but there is no way to remove them from the system once they are created.

As already noted in and out operations simply change the logical topology between agents, and do not alter the physical distribution of processes.

8. Comparisons and remarks

Cardelli [3,4] has produced the first implementation, called Ambit, of an ambient-like language; it is a single-machine implementation of the untyped Ambient calculus, written in Java. The algorithms are based on locks: all the ambients involved in a movement (three ambients for an in or out movement, two for an open) have to be locked for the movement to take place.

More recently, Fournet, Lèvy and Schmitt [10] have presented a distributed implementation of the untyped Ambient calculus, as a translation of the calculus into Jicama [11] (a programming language based on the distributed Join Calculus [9]). Our abstract machine is quite different from the above mentioned implementations mainly because:

(i) We are implementing a variant of the Ambient calculus (the Safe Ambients) that has coaction and types for single-threadedness and immobility.

(ii) We have a different interpretation of the logical and physical distribution of an ambient system.
The combination of (i) and (ii) allows us considerable simplifications, both in the abstract machine and in its correctness proof. We are not aware of correctness proofs for Ambit. The correctness proof for the Join implementation is very ingenious and makes use of sophisticated techniques, such as coupled simulation and decreasing diagram techniques. Below, we focus on the differences with the Join distributed implementation, to which we will refer as AtJ (Ambients to Join); comparison is focused on the algorithms underlying the two implementations. It is worth mentioning that one of the goals of AtJ was to provide as much parallelism as possible, even with moving ambients, which is irrelevant for PAN since moving ambients are single threaded.

Although the design of AtJ is very clever, the differences between Ambients and Join inevitably give some complications.

- In AtJ open is by far the most complex operation, because the underlying Jicama language does not have primitives with a similar effect. In AtJ, every ambient has a manager that collects the requests of operations from the subambients and from the local processes. If the ambient is opened, its manager becomes a forwarder of messages towards the parent ambient. The processes local to the opened ambient are not moved. As a consequence, in AtJ the processes local to an ambient can be distributed on several locations (precisely: to sublocations of the location of the given ambient). Therefore, also the implementation of the communication rule R-MSG may require exchange of messages among sites, which does not occur in PAN, where forwarders are always empty.

Moreover it is important to notice that PAN is an abstract machine, therefore it is independent of a specific target language; so many additional implementation improvements can eventually be achieved by carefully adopting different target language (similar considerations hold also for coordination languages such as Linda).

- In PAN, the presence of coaction dispenses us from having backward pointers from an ambient to its children. In the example of Fig. 1, without \( \Sigma \), the ambient \( c \) would not know the location of \( b \) and therefore could not communicate this location to \( a \). Backward pointers, present in AtJ, make bookkeeping and correctness proof more complex. In PAN, the absence of backward pointers, and the presence of coaction make the implementation of forms of dynamic linking straightforward: new machines hosting ambients can be connected to existing machines running an ambient system; it suffices that the new machines know the location (the IP number) of one of the running ambients; no modifications or notifications is needed to the running ambients themselves.

Note that, strictly speaking, the translation \( \lbrack P \rbrack \) is not compositional, w.r.t. the parallel operator. In fact, the parallel composition of two well-formed nets is not well-formed, due to the fact that it is a forest rather than a tree, and there are name clashes. However, the translation could be easily modified to achieve this property.

- In PAN, since any moving ambient (an ambient that tries to enter or exit another ambient, or that can be opened) is single threaded, each moving ambient requests at most one operation at a time to its parent. By contrast, in AtJ an ambient can send an unbounded number of requests to the parent (an example is \( n[\text{!in } m_1 \mid \text{!out } m_2] \)).

Due to this property, in PAN no ambient needs a log of pending requests received from a given child or sent to the parent. Without the property, both forms of log are
needed, as happens in AtJ. To see why, consider two ambients $a$ and $b$, where $b$ is the parent of $a$. If moving ambients can request several operations concurrently, $b$ must of course keep a log of the pending requests from $a$. A copy of the same log must however be kept by $a$, because messages exchanged among ambients are asynchronous and therefore the following situation could arise. Suppose $a$ requests two operations, say $\text{in} \ n$ and $\text{in} \ m$. The request for $\text{in} \ n$ could reach $b$ first. The request for $\text{in} \ m$ could reach $b$ only when the movement for $\text{in} \ n$ has been completed (indeed, $a$ might have completed other movements). The request $\text{in} \ m$ must now be resent to the new parent of $a$, but $b$ does not possess this information. This task must therefore be accomplished by $a$, which, for this, must have stored $\text{in} \ m$ in its log of pending requests to the parent.

The example also shows that, aside from message retransmission in forwarders, some requests may have to be retransmitted several times, to different parents (in the example, $\text{in} \ m$); in PAN every request is sent at most once.

If we consider the extension to immobile ambients (with the bang) situations such as $n[!a \text{in} \ n \mid P]$ could occur, but in PAN the immobile ambient will not move, therefore it will be there to receive any answer for its parent, and again no logs are needed and messages are sent just once.

- In PAN, any movement for a given ambient is requested to the parent, which (assuming this is not a forwarder) makes decisions and gives authorizations; the grandparent is never contacted. This homogeneity property breaks in the presence of backward pointers from an ambient to its children. For instance, the simulation of the $\text{out}$ reduction of Fig. 2 would then need also the involvement of the grandparent $c$.

- In AtJ, the domain of physical distribution is a tree. The $\text{in}$ and $\text{out}$ operations produce physical movements in which an ambient, and all its tree of subambients, must move. To achieve this, the tree of ambients is first “frozen” so that all the activities in the ambients of the tree stop while the movement takes place. In PAN, where the domain of physical distribution is flat, $\text{in}$ and $\text{out}$ only give logical movement; no freezing of ambients is required. On the other hand in PAN $\text{open}$ gives physical movement, which is not the case in AtJ.

By the time the revision of the present paper was completed, a few more abstract machines for Ambient-like calculi had appeared.

In [18], a distributed abstract machine, for a variant of the boxed ambient calculus with channels is presented. The machine, CAM, uses a list syntax for terms listing the top-level processes and ambients (which coincides with our Eq. (1)), and blocked processes (similar to our $\text{wait}$ prefixed process) to identify possible interactions. In their calculus there are $\text{in}$, $\text{out}$ action/coaction, but no $\text{open}$, which is replaced by processes asking explicitly to be moved (up or down). CAM is proved to be correct by showing that reductions in the original calculus are simulated by sets of reductions in the abstract machine, and vice versa. The proof is similar to ours and requires us to classify the reductions of the abstract machine into administrative (housekeeping and blocking in CAM terminology) and proper (interaction). However, the CAM machine described in [18] does not make an explicit separation between physical and logical distribution of ambients. In particular, the absence of the $\text{open}$ capability in the boxed calculus makes the use of logical forwarders to represent mobility (one of the main issues of PAN) not applicable to CAM.
Optimizations of the machine presented in the present paper are studied in \cite{13}; in particular, some well-known algorithms for distributed systems (e.g., algorithms based on reference counting and union-find) are exploited to improve the management of the forwarders created upon code migration. The correctness of such optimizations is establishing by proving a weak bisimilarity between the new machine and \textsc{Pan} and then appealing to the correctness of \textsc{Pan}.

9. Further developments

In this section we present some improvements and extensions that may be done, on one side to the abstract machine, and on the other to its implementation.

9.1. Immobile ambients

To handle immobile ambients, we have extended the syntax of \textsc{Pan}, and the set of reduction rules. We believe that these extensions are not necessary: the same syntax and rules presented for single-threaded ambients work also in the presence of immobile ambients. However, the correctness proof is harder.

We describe an improvement to the solution of adopting the rules of Section 4. Consider the process

\[
P \overset{\text{def}}{=} n[\text{rec } X. (\overline{\text{in}} n | (\nu m)(\text{open } m.X | m[\text{open } m]))]
\]

(Using replication, the behavior of \(P\) can be expressed as \(n[\overline{\text{in }} n]\).) With the rules of Section 4, ambient \(n\) could flood its parent with \(\overline{\text{in}}\) requests. We can avoid the problem by modifying \textsc{Par-Proc} thus:

\[
P \overset{\text{h}}{\rightsquigarrow} P' \parallel \overline{\text{Msg}}
\]

\[
\text{Q does not have unguarded ambients}
\]

\[
\text{Q or } P' \text{ do not contain any wait}
\]

\[
\text{[IMM-PAR-PROC]}
\]

We then have to modify also \textsc{Local-Open} and \textsc{Par-Proc}, so that an immobile ambient does not go into a \texttt{wait} state while opening a child ambient:

\[
\text{open } m. P | \{\text{open } m, h\} \overset{\text{h}}{\rightsquigarrow} P \parallel \text{h[migrate]}
\]

\[
\text{n is an immobile ambient}
\]

\[
\text{[IMM-LOCAL-OPEN]}
\]

\[
\text{h:n[ P ]k} \parallel h[\text{register } R] \overset{\text{h}}{\rightsquigarrow} h:n[ P | R]k
\]

\[
\text{n is an immobile ambient}
\]

\[
\text{[IMM-COMPL-REG]}
\]

The original rules \textsc{Local-Open}, \textsc{Par-Proc}, and \textsc{Compl-Reg} are now used only for ST ambients, therefore the corresponding side conditions are added.

Rule \textsc{Imm-Local-Open} could actually be used also for ST ambients. This however makes the correctness proof harder and under some more refined notion of behavioral equivalence (example: a real-time model) would not be correct.

The effect of adopting rule \textsc{Imm-Par-Proc} in place of \textsc{Par-Proc} is that an immobile ambient sends to its parent only one \(\overline{\text{in}}\) request at a time. This property already holds
for ST ambient with the rules of Section 4. An immobile ambient can exercise several capabilities at the same time. Sending one request at a time to the parent is correct because the only capability that may produce a request from an immobile ambient named \( n \) to its parent is \( \mathsf{in} \) (the protocol for \( \mathsf{in} \) can however be executed in parallel with several protocols for \( \mathsf{out} \) and \( \mathsf{open} \) operations).

9.2. Forwarders

In the abstract machine presented, a message may have to go through a chain of forwarders before getting to destination. A (partial) solution to this problem is a modification of the rules that guarantees the following property: every agent sends a message to a given forwarder at most once. The modification consists in adding the source field to the completion messages \( h\{\mathsf{OK} n, k\} \), which thus becomes \( h\{\mathsf{OK} n, k, h'\} \), where \( k \) is the ambient that is authorizing the move. Thus the rules \( \mathsf{LOCAL-IN} \) and \( \mathsf{COMPL-COIN} \) become

\[
\begin{align*}
&\{\mathsf{in} n, h\} \mid \{\mathsf{in} n, k\} \xrightarrow{h'} 0 \Rightarrow h\{\mathsf{go} k\} \parallel k\{\mathsf{OK} n, h'\} \quad \text{[\( \mathsf{LOCAL-IN2} \)]} \\
&h:n[P \mid \mathsf{wait}.Q]k \parallel h\{\mathsf{OK} n, h'\} \mapsto h:n[P \mid Q]h' \quad \text{[\( \mathsf{COMPL-COIN2} \)]}
\end{align*}
\]

The reason why these rules may be useful is that the parent of an ambient that has sent a \( \mathsf{in} \) request may have become a forwarder; thus the real parent is another ambient further up in the hierarchy. With the new rules, the parent of the ambient that has sent the \( \mathsf{in} \) request is updated and hence this ambient will not go through the forwarder afterwards. With the other capabilities that may originate a request from an ambient to his parent (\( \mathsf{open} \), \( \mathsf{out} \), \( \mathsf{in} \)), the issue does not arise, because either the requesting ambient is dissolved (\( \mathsf{open} \)), or its parent is modified (\( \mathsf{out} \), \( \mathsf{in} \)).

Even with the rules above, however, the forwarder introduced by an \( \mathsf{open} \) operation is permanent. We plan to study the problem of the garbage-collection of forwarders. We also plan to experiment with the addition of backwards pointers, from an ambient to its children; this should avoid the introduction of forwarders in an \( \mathsf{open} \), however, it is likely to complicate other parts of the abstract machine.

9.3. Other issues

At the moment, we have an implementation of the abstract machine, a compiler from SA to Java, and a simple user interface that allows the execution of SA terms. We hope that this will be helpful for the design of implementations of programming languages based on ambients. To be useful, in assessing the practical impact of the SA calculus, however, we have to embed this core language into a real language. We intend to pursue this direction, taking advantage also of the experience of \( \pi \)-calculus-based programming languages such as Pict and Join. An orthogonal direction would be to make the ambient constructs into a framework that could be used in conjunction with, e.g., Java.

9.4. Extensions to other calculi

It would be interesting to adapt PAN to a recent extension, SAP, of the SA calculus, see [16]. In this calculus in order to interact with an ambient \( n \), an ambient \( m \) must exercise
a capability indicating both \( n \) and a password to access \( n \). We think that our abstract machine could be modified to deal with SAP, but we have not yet explored this possibility.

Acknowledgements

We would like to thank Jean-Jacques Lèvy and Alan Schmitt for their useful comments, and the anonymous referees for their observations and suggestions.

Giannini’s work was partially supported by the IST-2001-33477 DART, and MIUR Cofin’01 NAPOLI projects. Sangiorgi’s work was partially supported by the IST-2001-33100 PROFUNDIS project. Valente’s work was partially supported by the IST-2001-33477 DART, MIUR Cofin’01 NAPOLI, and MIUR Cofin’02 McTati projects. The founding bodies are not responsible for any use that might be made of the results presented here.

References