Safe Ambients: Control Flow Analysis and Security.

Pierpaolo Degano, Francesca Levi, Chiara Bodei

Dipartimento di Informatica, Università di Pisa
Corso Italia 40, I-56100 Pisa, Italy
{degano,levifran,chiara}@di.unipi.it

Abstract. We present a Control Flow Analysis (CFA) for the Safe Ambients, a variant of the calculus of Mobile Ambients. The analysis refines [12] and computes an approximation of the run-time topology of processes. We use the result of the analysis to establish a secrecy property.

1 Introduction

Mobile Ambients (MA [6]) has recently emerged as a core programming language for the Web and, at the same time, as a model for reasoning about properties of mobile processes. Differently from other process algebras based on name communication like the π-calculus [11], MA is based on the notion of *ambient*. An ambient is a bounded place, where multi-threaded computation happens; in a sense, it generalizes both the idea of agent and the idea of location. Each ambient has a *name*, a collection of *local processes* and a collection of *subambients*. Ambients are organized in a hierarchy that can be dynamically modified, according to three basic capabilities, associated with ambient names and used for access control. They are the following: $\mathtt{in}\, n$ allows an ambient to enter into ambient (named) n: $(m[\,\mathtt{in}\, n.P_1|P_2\,]|n[\,Q\,] \longrightarrow n[\,m[\,P_1|P_2\,]|Q\,])$; $\mathtt{out}\, n$ allows an ambient to exit from ambient n: $(n[\,m[\,\mathtt{out}\, n.P_1|P_2\,]|Q\,] \longrightarrow m[\,P_1|P_2\,]|n[\,Q\,])$; $\mathtt{open}\, n$ allows to destroy the boundary of ambient n: $(\mathtt{open}\, n.P|n[\,Q\,] \longrightarrow P|Q)$. As shown by the rules, an ambient m moves as a whole with all its subambients. The movements of m only depend on the capabilities exercised by the local processes contained at its top level (for instance $\mathtt{in}\, n.P_1$ in the first rule); instead the local processes of subambients of m cannot directly influence the movements of their parent m. Also, the ambient n, that is affected by the rules, has no control on whether or not the action takes place.

Safe Ambients (SA [10]) is a modification of MA, where a movement or an ambient dissolution can take place only when the affected ambient agrees, offering the corresponding *coaction*: $\overline{\mathtt{in}}\, n, \overline{\mathtt{out}}\, n, \overline{\mathtt{open}}\, n$. This variation does not change the expressiveness of MA, yet makes it easier both to write programs and to formally prove their correctness, especially by using behavioural equivalences.

Several techniques, both dynamic and static, have been devised to study and establish various security properties of mobile calculi, based on notions of classifications and information flow. Encouraging results have been obtained by the use of static approaches, such as Type Systems [10, 4, 3, 5] and Control Flow Analysis (CFA) [12–14]. These techniques predict safe and computable approximations to the set of values or behaviours arising dynamically. For example, in calculi for concurrency, the approximations concern the values that variables may assume at run-time or the values that may flow on channels. In the ambient calculi, particularly relevant for security issues is information about the dynamic evolution of the ambient hierarchy and about which and where capabilities may be exercised.

The CFA given in [12] computes an approximation of the (run-time) topological structure of MA processes and follows the lines of previous work on the π-calculus [1]. More in detail, the analysis predicts for each ambient n, which ambients and capabilities (say α) may be contained at top level inside n. The analysis has been applied to prove a security property of the firewall protocol from [6, 7].

In this paper, we refine the analysis of [12], adapted to SA, by introducing a sort of contextual information, extending the proposal made in [9]. The analysis predicts for each ambient n:

1. which ambient surrounds ambient n whenever α (capability or ambient) is contained inside at top level;
2. which ambient surrounds n whenever α (capability or ambient), besides being contained inside (at top level), is also ready to interact.

Such a contextual information allows to considerably restrict the space of possible movements. In this way we gain in precision, and thus we may consider more properties and prove statically more programs safe.

As a simple example, we apply the result of the analysis for proving a *secrecy* property. We classify ambients into *trustworthy* and *untrustworthy*. Secrecy of data is preserved if an untrustworthy ambient can never open a trustworthy one. We show that if a program passes a simple static test, then its secret information is dynamically protected.

In proving a security property it is essential to be able to guarantee that properties are preserved when programs work in unknown, possibly hostile contexts. In general, not all contexts preserve our property, but we can say which of them do. As expected, the needed restrictions concern the usage of names. Actually, some of them cannot occur as ambient names in the context, but can occur inside its capabilities (cf. the assumption made in [12]). We define a tester process E that represents the most hostile context matching these requirements. If a process P in parallel with E passes the static test, then the secrecy property holds dynamically in any context represented by E.

Due to lack of space, we omit all the proofs.

2 Mobile Safe Ambients

In this section we briefly recall the *Mobile Safe Ambients* ([10]) calculus, without communication primitives. The only difference with the syntax of MA is given by coactions $\overline{\texttt{in}}\, n, \overline{\texttt{out}}\, n, \overline{\texttt{open}}\, n$.

Because of a peculiar treatment of bound names (see below), we partition names as follows. Given $\widehat{\mathcal{N}} = \{n, h, k, \ldots\}$ the infinite set of *names*, let $\mathcal{N}' = \biguplus_{n \in \widehat{\mathcal{N}}} \mathcal{N}_n$, where $\mathcal{N}_n = \{n_0, n_1, \ldots\}$. The name n stands for a generic element of $\mathcal{N}_n$.

Definition 1 (syntax). Processes *(denoted by $P, Q, R, \ldots \in \mathcal{P}$) and* capabilities *(denoted by $M, N, \ldots \in \mathcal{C}$) Processes are built according to the following syntax*

$P ::=$	*processes*	$M ::=$	*capabilities*
$\mathbf{0}$	*nil*	$\mathtt{in}\, n$	*enter n*
$M.P$	*prefix*	$\overline{\mathtt{in}}\ n$	*allow enter n*
$n[P]$	*ambient*	$\mathtt{out}\, n$	*exit n*
$P \vert P$	*parallel composition*	$\overline{\mathtt{out}}\ n$	*allow exit n*
$(\nu n)P$	*restriction*	$\mathtt{open}\, n$	*open n*
$!P$	*replication*	$\overline{\mathtt{open}}\ n$	*allow open n*

In the following capabilities are ranged over by $M, N \ldots$, *actions* by μ and *coactions* by $\bar{\mu}$. Furthermore, capabilities and names are ranged over by α. Standard syntactical conventions are used: the trailing $\mathbf{0}$ in processes $M.\mathbf{0}$ is omitted, and parallel composition has the least syntactic precedence.

We refer to the usual notions of names, free names, and bound names of a process P, denoted by $\mathrm{n}(P)$, $\mathrm{fn}(P)$, $\mathrm{bn}(P)$, respectively. Hereafter, it will be convenient to assume an external ambient, $\top \notin \mathcal{N}'$, so we let $\mathcal{N} = \mathcal{N}' \cup \{\top\}$.

The *structural congruence* $\equiv$ on processes is defined as the least congruence satisfying the following clauses:

- $(\nu\, n_i)P \equiv (\nu\, n_j)P\{n_j/n_i\}$, if $n_j \notin \mathrm{fn}(P)$;
- $(\mathcal{P}/_{\equiv}, |, \mathbf{0})$ is a commutative monoid;
- $M.P \equiv M.Q$ if $P \equiv Q$;
- $n[P] \equiv n[Q]$ if $P \equiv Q$;
- $(\nu\, n)P \equiv (\nu\, n)Q$ if $P \equiv Q$;
- $(\nu\, n)\mathbf{0} \equiv \mathbf{0}$, $(\nu\, n)(\nu\, n')P \equiv (\nu\, n')(\nu\, n)P$, $(\nu\, n)P \equiv P$ if $n \notin \mathrm{fn}(P)$,
  $(\nu\, n)(P \,|\, Q) \equiv (\nu\, n)P \,|\, Q$ if $n \notin \mathrm{fn}(Q)$,
  $m[\,(\nu n)P\,] \equiv (\nu n)m[\,P\,]$ if $n \neq m$;
- $!P \equiv P|!P$.

The structural congruence is the standard one, apart from the treatment of α-conversion. A name $n_i \in \widehat{\mathcal{N}}_n$ the name n_i can only be replaced by a name $n_j \in \widehat{\mathcal{N}}_n$ (not occurring free in the process). Moreover, we assume that the names occurring bound inside restrictions are all distinct from each other and from free variables. These assumptions are only used to keep the definition of our analysis more compact, and can be easily removed (see, e.g. [1]).

The reduction relation $\longrightarrow$ of SA in Table 1 is defined in the usual way. The only differences with the reduction semantics of MA are the basic movement rules. We write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$.

We introduce some notions which are necessary to formalize the soundness of the analysis and also the security property.

$\texttt{In}: \ n[\,\texttt{in}\, m.P_1 \vert P_2\,] \vert m[\,\overline{\texttt{in}}\ m.Q_1 \vert Q_2\,] \longrightarrow m[\,n[\,P_1 \vert P_2\,] \vert Q_1 \vert Q_2\,]$	
$\texttt{Out}: \ m[\,n[\,\texttt{out}\, m.P_1 \vert P_2\,] \vert\ \overline{\texttt{out}}\ m.Q_1 \vert Q_2\,] \longrightarrow n[\,P_1 \vert P_2\,] \vert m[\,Q_1 \vert Q_2\,]$	
$\texttt{Open}: \ m[\,\overline{\texttt{open}}\, m.P_1 \vert P_2\,] \vert \texttt{open}\ m.Q_1 \vert Q_2 \longrightarrow P_1 \vert P_2 \vert Q_1 \vert Q_2$	
$\texttt{Res}: \ \dfrac{P \longrightarrow P'}{(\nu n)P \longrightarrow (\nu n)P'}$	$\texttt{Par}: \ \dfrac{P \longrightarrow P'}{P \vert Q \longrightarrow P' \vert Q}$
$\texttt{Amb}: \ \dfrac{P \longrightarrow P'}{n[\,P\,] \longrightarrow n[\,P'\,]}$	$\texttt{Struct}: \ \dfrac{P \equiv P' \quad P' \longrightarrow P'' \quad P'' \equiv P'''}{P \longrightarrow P'''}$

Table 1. Reduction Rules for SA.

Definition 2. *Let P be a process and $\alpha \in \mathcal{N} \cup \mathcal{C}$ a name or a capability. We say that α is*

1. top level *in P iff α does not occur inside any ambient of P;*
2. enabled *in P iff α is top level in P and it does not occur underneath a prefix.*

For instance, in $P = \texttt{in}\, k.\overline{\texttt{out}}\, n \vert m[\,\texttt{out}\, n.\overline{\texttt{open}}\, m.\texttt{in}\, a\,]$, both capabilities $\texttt{in}\, k$ and $\overline{\texttt{out}}\, n$ and ambient m are top level. Capability $\texttt{in}\, k$ and ambient m are also enabled, while $\overline{\texttt{out}}\, n$ is only top level. Intuitively, if α is enabled, then it can be exercised (if it is a capability) and can interact with other processes at the same level (if it is an ambient). Hence, whenever P is placed inside an ambient with name n, capability $\texttt{in}\, k$ may be exercised and cause a movement of n and ambient m may interact inside n. By contrast, capability $\overline{\texttt{out}}\, n$ is not ready to be exercised, because it is guarded by $\texttt{in}\, k$: it is top level but not enabled.

We say that a *context* $C[-]$ (a process with a hole) is flat iff the hole is enabled in $C[-]$. Also, we call Q *configuration* of P if $P \equiv C[Q]$, and we write $\top[P]$ when C is empty.

3 Control Flow Analysis for *SA*

Our analysis predicts for each ambient n which capabilities and ambients *may* be contained inside n at run-time, when it occurs within a selected ambient. The solution to the analysis of a process P is a function ϕ, such that a typical element of $\phi(n)$ is $\alpha^{(\ell_1,\ell_2)}$, where α is either a name or a capability and ℓ_1, ℓ_2 are sets of names. Intuitively, when $\alpha^{(\ell_1,\ell_2)} \in \phi(n)$, the analysis says that the derivatives of P may include the following configurations:

1. for each $k \in \ell_1$, $k[\,\boldsymbol{\nu}\tilde{p}\,(R \vert n[\,Q\,])\,]$ where α is *top level* in Q;

2. for each $k \in \ell_2$, $k[\, \boldsymbol{\nu}\tilde{p}\,(R|n[\,P\,])\,]$ where α is *enabled* in P.

Before introducing the formal definitions we illustrate the above through a simple example. We omit brackets in singletons.

Example 1. Consider the process

$$P = n[\,\mathtt{in}\,k.\overline{\mathtt{out}}\,n|m[\,\mathtt{out}\,n.\overline{\mathtt{open}}\,m.\mathtt{in}\,a\,]\,]|k[\,\overline{\mathtt{in}}\,k.\mathtt{open}\,m\,].$$

The process P evolves (only) as follows. The ambient n can go inside k

$$P \longrightarrow k[\,\mathtt{open}\,m|n[\,\overline{\mathtt{out}}\,n|m[\,\mathtt{out}\,n.\overline{\mathtt{open}}\,m.\mathtt{in}\,a\,]\,]\,] = Q$$

When n is inside k, the ambient m can exit from n

$$Q \longrightarrow k[\,\mathtt{open}\,m|n[\,\mathbf{0}\,]|m[\,\overline{\mathtt{open}}\,m.\mathtt{in}\,a\,]\,] = R$$

When m is inside k, it can be opened and it liberates the capability $\mathtt{in}\,a$ inside k

$$R \longrightarrow k[\,n[\,\mathbf{0}\,]|\mathtt{in}\,a\,] = S.$$

A solution for the analysis of P is as follows

$\phi(\top)$	$\{n^{(\emptyset,\emptyset)}, k^{(\emptyset,\emptyset)}\}$
$\phi(n)$	$\{m^{(\{\top,k\},\{\top,k\})}, \mathtt{in}\,k^{(\top,\top)}, \overline{\mathtt{out}}\,n^{(\{\top,k\},k)}\}$
$\phi(m)$	$\{\mathtt{out}\,n^{(n,n)}, \overline{\mathtt{open}}\,m^{(\{n,k\},k)}, \mathtt{in}\,a^{(\{n,k\},k)}\}$
$\phi(k)$	$\{\overline{\mathtt{in}}\,k^{(\top,\top)}, \mathtt{open}\,m^{(\top,\top)}, \mathtt{in}\,a^{(\{n,k,\top\},\{k,\top\})}, n^{(\top,\top)}, m^{(\top,\top)}\}$

– $\phi(\top)$ contains ambients n and k with label $(\emptyset,\emptyset)$, to indicate that they occur at the outermost level.
– $\phi(n)$ contains: $m^{(\{\top,k\},\{\top,k\})}$, to predict that m is enabled when n is inside $\top$ or k (see processes P and Q); $\mathtt{in}\,k^{(\top,\top)}$, to show that $\mathtt{in}\,k$ is enabled, when n is inside $\top$ (see P) and will be consumed when n moves elsewhere (see Q and R); $\overline{\mathtt{out}}\,n^{(\{\top,k\},k)}$ recording that $\overline{\mathtt{out}}\,n$ is top level when n is inside $\top$ or k (see P and Q), while it is only enabled when $\mathtt{in}\,k$ has been consumed, and thus n has moved inside k (see Q).

– $\phi(m)$ contains: $\texttt{out}\, n^{(n,n)}$ predicting that $\texttt{out}\, n$ is enabled when m is inside n (see P and Q); $\overline{\texttt{open}}\, m^{(\{n,k\},k)}$ showing that $\overline{\texttt{open}}\, m$ is top level when m is inside n or k (see P and Q), while it is only enabled after the execution of $\texttt{out}\, n$, that is in k (see R). Similarly for $\texttt{in}\, a^{(\{n,k\},k)}$.
– $\phi(k)$ contains: n and m with label $(\top, \top)$ to indicate that both ambients may enter in k (n by exercing $\texttt{in}\, k$ and m by exercing $\texttt{out}\, n$); $\overline{\texttt{in}}\, k$ and $\texttt{open}\, m$ with label $(\top, \top)$, showing that m does not move from $\top$. Moreover, it contains also $\texttt{in}\, a^{(\{n,k,\top\},\{k,\top\})}$, to predict that $\texttt{in}\, a$ is top level and enabled when k is inside $\top$ (see S).

Although, when $\alpha^L \in \phi(n)$, $\ell_2 \subseteq \ell_1$ (α enabled is also top level), both sets are useful for an accurate prediction of process behaviour:

1. ℓ_1 approximates the set of ambients that surround n when α is top level and that therefore may acquire α by opening n. For instance, in Example 1 the ambient k acquires capability $\texttt{in}\, a$ from $\phi(m)$ through $\texttt{open}\, m$, because k belongs to (the first component of) its label. Capabilities $\texttt{out}\, n$ and $\overline{\texttt{open}}\, m$ are not acquired, because they are not top level when m is inside k.
2. ℓ_2 approximates the set of ambients that surround n when α is enabled, namely when α is ready to interact. Hence, ℓ_2 can be used to say if a capability is executable and to predict its effect. For instance, in Example 1 the (second component of the) labels of $\overline{\texttt{out}}\, n$ in $\phi(n)$ and of $\texttt{out}\, n$ in $\phi(m)$ are used to predict the effect of $\texttt{out}\, n$ on the ambient m. Since $\texttt{out}\, n$ is enabled when m is inside n and $\overline{\texttt{out}}\, n$ is enabled when n is inside k, m may go out of n and may end up inside k, the current parent ambient of n.

NOTATION. In the following we shall make use of some notation and abbreviations. The set of *labels* is $\mathcal{L} = \{(\ell_1, \ell_2) \mid \ell_1, \ell_2 \in \wp(\mathcal{N}) \wedge \ell_2 \subseteq \ell_1\}$. Hereafter, L will stand for (ℓ_1, ℓ_2) and L_i will stand for $(\ell_{1,i}, \ell_{2,i})$. Inclusion and union of labels are defined componentwise. Moreover, with an abuse of notation, for $N \in \wp(\mathcal{N})$ we write $L \subseteq N$ in place of $L \subseteq (N, N)$ and $L \cup N$ in place of $L \cup (N, N)$.
The set of *located capabilities* and *names* is denoted $\mathcal{LCA}$, ranged over by α^L, where α is either a capability or an ambient name and L is a label. An *element* I is a subset of $\mathcal{LCA}$. Over elements we define

1. $\alpha^{L_1} \in I$ if $\alpha^{L_2} \in I$ and $L_1 \subseteq L_2$;
2. $I_1 \sqsubseteq I_2$ iff $\alpha^L \in I_1$ implies $\alpha^L \in I_2$.

For example $\{\texttt{in}\, n^{(k,k)}\} \sqsubseteq \{\texttt{in}\, n^{(\{h,k\},k)}\}$.

THE ANALYSIS. We now formalize the intuition given above. Our CFA does not distinguish among different elements of the same equivalence class $\mathcal{N}_n$. This amounts to saying that $\phi(n)$ stands for the analysis of every n_i and, similarly, each n occurring in a located capability stands for every n_i, as well. In this way, we statically maintain the identity of names that may be lost by applying the standard α-conversion: for a different approach see the naming environment and the stable names in [12].

Definition 3 (Solutions). *Given a process P, let* $\mathrm{n}(P) \cup \{\top\} \subseteq N_P$. *A* solution *for P is a total function $\phi : N_P \to \wp(\mathcal{LCA})$.*
Furthermore, we order solutions by letting $\phi_1 \sqsubseteq \phi_2$ iff $\forall n \in N_P.\ \phi_1(n) \sqsubseteq \phi_2(n)$.

A solution ϕ for a process Q is *validated* if and only if $\phi \models^{\top,(\emptyset,\emptyset)}_{\phi(\top)} Q$ (shortly $\phi \models Q$) according to the set of clauses in Table 2. Clauses operate on judgments of the form

$$\phi \models^{k,L}_{I} P,$$

where k is a name of an ambient or $\top$, L is a label, and I is an element. The rules in Table 2 use some auxiliary notions that follow.

AUXILIARY NOTIONS. Let ϕ be a solution for P and $n, k \in N_P$. Some of the functions defined below depend on ϕ, that we shall often omit when clear from the context.

We introduce a function $\mathtt{f}$ that collects for each ambient n a superset of its possible fathers or parent ambients, i.e. the ambients that may surround n. In the same definition, we introduce the function ENAB which, given an ambient n and a capability M, collects those fathers of n in which M may be enabled. Moreover, we define a sort of transitive closure w.r.t. the open operation.

Definition 4 (Fathers, ENAB, $k^{\uparrow}$).

- $\mathtt{f}^{\phi}(n) = \{k \mid n^L \in \phi(k)\}$;
- $\textit{ENAB}^{\phi}(M, n) = \{\ell_2 \mid M^{(\ell_1,\ell_2)} \in \phi(n)\}$;
- $k^{\uparrow}_{\phi}$ *is the least set of names such that* $k \in k^{\uparrow}_{\phi}$ *and* $\forall h \in k^{\uparrow}_{\phi}$, *if* $n \in \textit{ENAB}^{\phi}(\overline{\mathtt{open}}\, h, h)$ *and* $\mathtt{open}\, h^L \in \phi(n)$ *then* $n \in k^{\uparrow}_{\phi}$.

For instance in Example 1 $\text{ENAB}(\overline{\mathtt{out}}\, n, n) = k$ and $\text{ENAB}(\mathtt{in}\, k, n) = \top$ which show that $\overline{\mathtt{out}}\, n$ may be exercised when n is inside k only, and that $\mathtt{in}\, k$ may be exercised when n is inside $\top$ only.

Based on the above, the function $\mathtt{t}$ predicts the target ambients in which an ambient ends up after a movement action ($\mathtt{in}$ or $\mathtt{out}$).

Definition 5 (Target). *The* target *of α in k, $\mathtt{t}^{\phi}(\alpha, k) \subseteq N_P$, is given by*

1. $\mathtt{t}^{\phi}(\mathtt{in}\, n, k) = \begin{cases} n & \text{if } \bigcup_{h \in k^{\uparrow}_{\phi}} \textit{ENAB}^{\phi}(\mathtt{in}\, n, h) \ \cap \ \textit{ENAB}^{\phi}(\overline{\mathtt{in}}\, n, n) \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$
2. $\mathtt{t}^{\phi}(\mathtt{out}\, n, k) = \begin{cases} \textit{ENAB}^{\phi}(\overline{\mathtt{out}}\, n, n) & \text{if } n \in \bigcup_{h \in k^{\uparrow}_{\phi}} \textit{ENAB}^{\phi}(\mathtt{out}\, n, h) \\ \emptyset & \text{otherwise} \end{cases}$
3. $\mathtt{t}^{\phi}(\alpha, k) = \emptyset$ *if* $\alpha \in \{\mathtt{open}\, n, \overline{\mathtt{in}}\, n, \overline{\mathtt{out}}\, n, \overline{\mathtt{open}}\, n\}$.

The target of a movement capability is empty whenever the capability is deadlocked. In details: $\mathtt{in}\, n$ may be executed only if there exists an ambient h, such that $\mathtt{in}\, n$ is enabled when k in inside h and $\overline{\mathtt{in}}\, n$ is enabled when n is also inside h; $\mathtt{out}\, n$ is executable only if it is enabled when k is also inside n. When a capability $\mathtt{in}\, n$ is executable, its effect is that of moving the ambient k inside

n; when a capability $\mathtt{out}\, n$ is executable, its effect is that of moving the ambient k inside some of the fathers of n, the ones where $\overline{\mathtt{out}}\, n$ is enabled.

For instance in Example 1, $\mathtt{t}(\mathtt{in}\, k, n) = k$, since $\mathtt{in}\, k$ is enabled when n is inside $\top$, while $\overline{\mathtt{in}}\, n$ is enabled when n is inside $\top$. Moreover, $\mathtt{t}(\mathtt{out}\, n, m) = k$ since $\mathtt{out}\, n$ is enabled when m is inside n and $\overline{\mathtt{out}}\, n$ is enabled when n is inside k.

We introduce also some operators acting on elements, i.e. on subsets of $\mathcal{LCA}$. They are used in the clauses (**open**) and (**par**): we shall give some intuition later on.

Definition 6. *We define*

- $I \lhd\ k = \{\alpha^L \mid \alpha^L \in I \ \wedge\ k \in \ell_1\}$;
- $\gamma^\phi(k, I, n) = \begin{cases} I \lhd\ k \ \textit{if}\ k \in \textit{ENAB}^\phi(\overline{\mathtt{open}}\, n, n) \\ \emptyset \qquad \textit{otherwise} \end{cases}$;
- $I \oplus\ L = \{\alpha^{L' \cup L} \mid \alpha^{L'} \in I\}$;
- $I_1 \otimes^\phi_n I_2 = \Big(I_1 \oplus \{\mathtt{t}^\phi(M, n) \mid M^L \in I_2\}\Big) \cup \Big(I_2 \oplus \{\mathtt{t}^\phi(M, n) \mid M^L \in I_1\}\Big)$.

By abuse of notation, we write $I_1 \oplus\ I_2$ for $I_1 \oplus\ (I_2, I_2)$.

Intuition on the CFA clauses The intuitive meaning of the elements of judgement $\phi \models^{k,L}_I P$, with $L = (\ell_1, \ell_2)$, is the following: the process P to be examined is (statically) contained in the current ambient k; the element I, subset of $\phi(k)$, contains enough information to validate process P assuming the following configurations are reachable:

1. for any $m \in \ell_1$, $m[\,\boldsymbol{\nu}\tilde{p}\,(R|k[\,Q\,]]$, where P is a subprocess of Q (possibly equal to Q itself), and
2. for any $m \in \ell_2$, $m[\,\boldsymbol{\nu}\tilde{p}\,(R|k[\,P\,]]$.

Using subsets of $\phi(k)$ increases the accuracy of the analysis, especially in the case of parallel processes (see the explanation of clause (**par**) below). The clauses operate on the structure of process P, updating k, L and I. The current ambient k is updated, whenever we pass inside a new ambient. The current label L is updated in the same situation and whenever we pass a movement capability (input and output). Each time an ambient or a capability is under consideration, the analysis checks that the current element I contains the corresponding located component. The check skips to another element I' to analyse the continuation of the process in almost every clause, except for **pref** and $\overline{\textbf{pref}}$. The new element I' or a suitable modification of it must be part of the old subset I (and finally part of the analysis of the current ambient). We now illustrate the more relevant clauses.

(**amb**) Ambient n is (statically) contained inside the ambient k. Label L says that n may be enabled, when k is inside an ambient of ℓ_2, while it may be top level but not necessarily enabled, when k is inside an ambient of ℓ_1.

Hence, the located ambient n^L has to occur in I, i.e. in the solution for k. The process P has to be validated in the ambient n with label (k, k), because the configuration $k[\,\boldsymbol{\nu}\tilde{p}\,(R|n[\,P\,])\,]$ may be reachable. To analyse P it suffices then to find out an element I' such that $I' \sqsubseteq \phi(n)$.

nil: $\phi \models_I^{k,L} \mathbf{0}$	iff true
amb: $\phi \models_I^{k,L} n[\,P\,]$	iff $n^L \in I \;\wedge \exists I' : \phi \models_{I'}^{n,(k,k)} P \;\wedge\; I' \sqsubseteq \phi(n)$
pref: $\phi \models_I^{k,L} \mu.P \;\wedge\; \mu \neq \mathtt{open}\, n$	iff $\begin{cases} \mu^L \in I \;\wedge\; \phi \models_I^{k,(\ell_1 \cup \mathtt{t}(\mu,k),\mathtt{t}(\mu,k))} P \;\wedge \\ \forall h \in \mathtt{t}(\mu,k) : k^{(\mathtt{f}(h),\mathtt{f}(h))} \in \phi(h) \end{cases}$
$\overline{\textbf{pref}}$ $\phi \models_I^{k,L} \overline{\mu}.P$	iff $\overline{\mu}^L \in I \wedge\; \phi \models_I^{k,L} P$
open: $\phi \models_I^{k,L} \mathtt{open}\, n.P$	iff $\begin{cases} (\mathtt{open}\, n^L \in I) \;\wedge\; \exists I_1 :\; \phi \models_{I_1}^{k,L} P \;\wedge \\ I_1 \otimes_k (\gamma(k,\phi(n),n) \oplus L) \sqsubseteq I \;\wedge \\ \forall k \in \textsc{enab}(\overline{\mathtt{open}}\, n, n),\; h \in \mathcal{N}, \\ \alpha^L \in \phi(h) :\; n \in \ell_i \Rightarrow k \in \ell_i,\; \forall i = 1,2 \end{cases}$
par: $\phi \models_I^{k,L} P \vert Q$	iff $\begin{cases} \exists I_1, I_2 : \phi \models_{I_1}^{k,L} P \;\wedge\; x\phi \models_{I_2}^{k,L} Q \;\wedge \\ I_1 \otimes_k I_2 \sqsubseteq I \end{cases}$
repl: $\phi \models_I^{k,L} !P$	iff $\exists I' : \phi \models_{I'}^{k,L} P \;\wedge\; I' \otimes_k I' \sqsubseteq I$

Table 2. Control Flow Analysis for SA

(**pref**) As in the previous rule the located capability μ^L must belong to the element I. The residual process P has to be validated still inside the ambient k considering that ambient k, as effect of μ, may have moved. Indeed, function $\mathtt{t}(\mu, k)$ gives the set of ambients in which k may enter because of the execution of μ. Hence, the following configurations may be reachable: $m[\, \boldsymbol{\nu}\tilde{p}\,(R|k[\,P\,])\,]$, for $m \in \mathtt{t}(\mu,k)$, and $m[\,\boldsymbol{\nu}\tilde{p}\,(R|k[\,Q\,])\,]$, where P is a subprocess of Q, for $m \in \ell_1 \cup \mathtt{t}(\mu,k)$. Label $(\ell_1 \cup \mathtt{t}(\mu,k), \mathtt{t}(\mu,k))$ is thus used to validate P.

For instance in Example 1 we have $\overline{\mathtt{out}}\, n^{(\top,k\},k)} \in \phi(n)$, because $\mathtt{t}(\mathtt{in}\, k, n) = k$ and

$$\phi \models_I^{n,(\top,\top)} \mathtt{in}\, k.\overline{\mathtt{out}}\, n \text{ requires } \phi \models_I^{n,(\{\top,k\},k)} \overline{\mathtt{out}}\, n$$

while

$$\phi \models_I^{n,(\{k,\top\},k)} \overline{\mathtt{out}}\, n \text{ requires } \overline{\mathtt{out}}\, n^{(\{k,\top\},k)} \in I \sqsubseteq \phi(n).$$

The remaining condition guarantees that $\mathtt{t}(\mu,k)$ have been recorded as possible fathers of k.

(**par**) The main idea is that of validating separately the process P and the process Q and to combine the results through the operator $\otimes$. Consider the process $R = \mathtt{in}\, n|\mathtt{out}\, m$. We obtain for instance

$$\phi \models_{I_1}^{k,L} \mathtt{in}\, n \;\text{ and }\; \phi \models_{I_2}^{k,L} \mathtt{out}\, m$$

for $I_1 = \{\mathtt{in}\, n^L\}$, $I_2 = \{\mathtt{out}\, m^L\}$. Neither I_1 nor I_2 contains enough information to validate R, because capabilities $\mathtt{in}\, n$ and $\mathtt{out}\, m$ may be executed in every order. Hence, the solution is sound only if it predicts that $\mathtt{out}\, m$ may be exercised in ℓ_2 and *also* after $\mathtt{in}\, n$, i.e. when k is inside n. Symmetrically it predicts that $\mathtt{in}\, n$ may be exercised in ℓ_2 and *also* after $\mathtt{out}\, m$, namely when k has moved out of m. Since label L does not contain this information, the labels of capabilities and ambients of the solutions I_1 and I_2 have to be updated according to the movements of the parallel processes. The operation $\otimes$ derives from elements I_1 and I_2 a correct element for R. Assuming that $\mathtt{t}(\mathtt{in}\, n, k) = n$ and $\mathtt{t}(\mathtt{out}\, m, k) = \mathtt{f}(m)$, we have

$$I_1 \otimes_k I_2 = \{\mathtt{in}\, n^{(\ell_1 \cup \{\mathtt{f}(m)\}, \ell_2 \cup \{\mathtt{f}(m)\})}, \mathtt{out}\, m^{(\ell_1 \cup \{n\}, \ell_2 \cup \{n\})}\}.$$

(**open**) The effect of an $\mathtt{open}\, n$ exercised inside k is that of liberating the process (say Q), which is currently contained in n, and to place it in parallel with the continuation of $\mathtt{open}\, n$ (say P). Considering a simplified case where $P = \mathbf{0}$ we have that $k[\,\mathtt{open}\, n|n[\,\overline{\mathtt{open}}\, n.Q\,]\,]$ reduces to $k[\,Q\,]$.

The idea is that the element I is valid iff also $\phi \models_I^{k,L} Q$, i.e. if I contains the ambients and capabilities which may be liberated inside k by opening n. If $\alpha^L \in \phi(n)$ and $k \in \ell_1$, then the the following configurations may be reachable $k[\,\boldsymbol{\nu}\tilde{p}\,(R|n[\,S\,])\,]$, where α is top level of S and may be acquired by k by opening n. Hence, the subset of $\phi(n)$, $\phi(n) \triangleleft\; k$ (that is the set of α^L such that $k \in \ell_1$), gives an approximation of the ambients and capabilities that k may acquire.

To be more precise we require that $\phi(n) \triangleleft\; k$ is a subset of the element I only when $\mathtt{open}\, n$ is executable, i.e. when $\overline{\mathtt{open}}\, n$ is enabled when n is inside k (see Definition 6 of γ).

For instance in Example 1 we have

$$\phi(m) = \{\mathtt{out}\, n^{(n,n)}, \overline{\mathtt{open}}\, m^{(\{n,k\},k)}, \mathtt{in}\, a^{(\{n,k\},k)}\}.$$

Therefore, $\gamma(k, \phi(m), m) = \{\mathtt{in}\, a^{(\{n,k\},k)}\}$: only capability $\mathtt{in}\, a$ is acquired by k through $\mathtt{open}\, m$.

If $\mathtt{open}\, n$ has a continuation P then $k[\,\mathtt{open}\, n.P | n[\,\overline{\mathtt{open}}\, n.Q\,]\,]$ reduces to $k[\,P|Q\,]$. Hence, the element $\gamma(k, \phi(n), n)$ has to be combined in parallel with the element that validates P. The remaining condition in the clause ensures that the effect of $\mathtt{open}\, n$ is properly recorded in all the labels. Since n may be dissolved inside k, every capability or ambient which was enabled (or top level) in n is also enabled (or top level) in k.

Remark 1. Now, we can better comment the role of labels ℓ_1 and ℓ_2: label ℓ_1 is used to handle more accurately the effect of $\mathtt{open}\, n$ (see rule (**open**)); label ℓ_2 restricts the cases where capabilities are executable and also allows one to predict more precisely where an ambient may end up after its execution (see Definition 5 of $\mathtt{t}$).

If capabilities and ambients were not labelled in our manner the prediction of the analysis would be less precise. For instance in Example 1, the analysis in [12] would imprecisely predict that k may acquire by opening m also $\mathtt{out}\, n$ and $\overline{\mathtt{open}}\, m$, i.e. the whole set $\phi(m)$. Moreover, the same analysis would imprecisely predict that the ambient m may end up, by executing $\mathtt{out}\, n$, in any father of n, thus in k and also in $\top$.

4 Properties of the analysis

We state some standard results of our analysis. First there always exists a least solution which is valid according to the clauses of Table 2. It is enough to show that the set of valid solutions is a complete lattice w.r.t. $\sqsubseteq$ (here defined componentwise), and that it forms a Moore Family [1] [8, 15].

Theorem 1. *The set* $\mathcal{J} = \{\phi \mid \phi \ \models_I^{k,L} P\}$ *is a* Moore family.

The analysis satisfies a standard Subject Reduction theorem: validity of solutions is preserved under reduction.

Theorem 2. *Let* $\phi \models_I^{k,L} P$, $I \sqsubseteq \phi(k)$ *and* $L \subseteq \mathtt{f}(k)$. *If* $P \Longrightarrow Q$ *then there exists* I' *such that* $\phi \models_{I'}^{k,L} Q$ *and* $I \sqsubseteq I' \sqsubseteq \phi(k)$.

Corollary 1. *Let* $\phi \models_{\phi(\top)}^{\top,(\emptyset,\emptyset)} P$. *If* $P \Longrightarrow Q$ *then* $\phi \models_{\phi(\top)}^{\top,(\emptyset,\emptyset)} Q$.

The following theorem states the relation between the dynamic and static behaviour. It says that a valid solution for a process P contains all the configurations of any derivative of P. In details, for every configuration $k[\,\boldsymbol{\nu}\tilde{p}\,(R_2|n[\,R_1\,])\,]$ of any derivative of P, and for every α (capability or name) top level (resp. enabled) in R_1, the solution for n contains α^L where $k \in \ell_2$ (resp. $k \in \ell_1$).

[1] $(\mathcal{I}, \sqsubseteq)$ is a Moore family whenever any of its subsets admits a g.l.b. in $\mathcal{I}$

Theorem 3 (Correctness). *Let $\alpha \in \mathcal{N} \cup \mathcal{C}$ and $n, k \in \mathcal{N}$. Whenever $\phi \models P$, $P \Longrightarrow Q$, and $k[\boldsymbol{\nu}\tilde{p}\,(n[\,R_1\,]|R_2)]$ is a configuration of Q:*

1. *if α is enabled in R_1, then $\alpha^{(\ell_1, \ell_2 \cup \{k\})} \in \phi(n)$;*
2. *if α is top level in R_1, then $\alpha^{(\ell_1 \cup \{k\}, \ell_2)} \in \phi(n)$.*

5 A Security Property

An ambient m may acquire the information contained in another ambient by opening it. We wish to maintain some information confidential to a group of ambients, considered trustworthy. To do so, names $\mathcal{N}$ (including $\top$) are partitioned into *trustworthy* $\mathcal{T}$ and *untrustworthy* $\mathcal{U}$. Secrecy of data is preserved if an untrustworthy ambient can never open a trustworthy one. Based on this partition, we first formalise our dynamic notion of secrecy: *protection*; then we give a static notion that implies the dynamic one.

We say that n is *opened* by k in P iff $k[\,\boldsymbol{\nu}\tilde{p}\,(n[\,R_1\,]|R_2)\,]$ is a configuration of P and $\overline{\texttt{open}}\,n$ and $\texttt{open}\,n$ are enabled in R_1 and in R_2, respectively.

Definition 7 (Dynamic Property). *A process P is* protected *iff $\forall k \in \mathcal{N}$, $n \in \mathcal{T}$, whenever $P \Longrightarrow Q$ and n is opened by k in Q then $k \in \mathcal{T}$.*

Definition 8 (Static Property). *A process P is* defended *if there exists a solution ϕ such that $\phi \models P$ and, $\forall k \in \mathcal{N}$, $n \in \mathcal{T}$, whenever $\texttt{open}\,n^{L_1} \in \phi(k)$, $\overline{\texttt{open}}\,n^{L_2} \in \phi(n)$ and $k \in \ell_{2,2}$, then $k \in \mathcal{T}$.*

Our static property is a correct approximation of the dynamic one. In fact, Theorem 3 suffices to prove the following.

Lemma 1. *If P is* defended *then it is* protected.

The property above is not enough, because it does not guarantee that a *defended* process P will still be such when plugged in a hostile context $C[-]$, unless the whole $C[P]$ is analysed again and proved to be *defended*. However, we can characterize those contexts that do not break the secrecy of P and for which there is no need to analyse $C[P]$. Technically, we first define a tester process, called enemy, which represents the most hostile context in which P can be put. The enemy satisfies some mild conditions on the names of its ambients: they should not clash with a subset N of those occurring in P. Note that the enemy may know the names of P, even if trustworthy: they need not to be kept secret (e.g. by restricting them). Also, the check on the names of the enemy can be done at run-time by P: entering, leaving or opening an ambient n of the enemy should be forbidden if $n \in N$. This can be easily done by a dynamic check.

Then, we show that if the system where P and its enemy run in parallel is *defended*, then P is *protected* in any context represented by the enemy.

Definition 9 (Enemy). *The* enemy *of a process P w.r.t. $N \subseteq \mathrm{fn}(P)$ is*

$$E(P, N) = (|_{m \in \mathrm{fn}(P) \setminus N}\, m[\,Q\,]) \mid Q$$

where $Q = |_{n \in \mathrm{fn}(P)}(!\texttt{in}\,n|!\texttt{out}\,n|!\texttt{open}\,n|!\overline{\texttt{in}}\,n|!\overline{\texttt{out}}\,n|!\overline{\texttt{open}}\,n)$.

Theorem 4. *If* $P|E(P,N)$ *is* defended *for* $N \subseteq \mathrm{fn}(P)$ *then* $C[P]$ *is* protected *for every* flat *context* $C[-]$ *with no configuration* $n[\,Q\,]$, $n \in N$.

Note that it is not necessary to actually analyse $E(P,N)$: its solutions have all the same shape and can be easily built. Furthermore, given the standard solution for $E(P,N)$ and a solution for P, it is possible to build a valid solution for $P|E(P,N)$.

6 An Example

Consider an ambient a willing to send a message to an ambient b. Since the ambient calculus has no primitives for remote communication, a message is delivered by enclosing it in an ambient that moves inside the receiver. This acquires the message by opening it. When the message is secret, it is essential to guarantee that no ambient can open the ambient carrying the message, except for the designated receiver b. In other words, we would like our system to be *protected* (but we do not consider whether b receives the message or not).

Assuming that the path from a to b is known, an abstract specification could be the following, where *mail* carries the message D [2]:

$$SYS \;= a[\,\overline{\mathtt{out}}\,a.\overline{\mathtt{in}}\,a|\mathtt{open}\,mail \mid MAIL\,]|b[\,\overline{\mathtt{in}}\,b.\mathtt{open}\,msg\,]$$

$$\begin{aligned} MAIL = mail[\,&\mathtt{out}\,a.\mathtt{in}\,b.\overline{\mathtt{out}}\,mail.\mathtt{out}\,b.\mathtt{in}\,a.\overline{\mathtt{open}}\,mail| \\ &msg[\,\mathtt{out}\,mail.\overline{\mathtt{open}}\,msg.D\,]\,] \end{aligned}$$

The ambient *mail* goes out of a and carries within b the message, included in *msg*. When there, *msg* exits from *mail* and b reads D through $\mathtt{open}\,msg$. Then, *mail* goes back to a, willing to deliver the ack of b through $\overline{\mathtt{open}}\,mail$.

When the message is secret, it is essential to guarantee that no ambient can open *msg* and therefore read the message, except for the receiver b. Assume that $\mathcal{T} = \{b, msg\}$ is the set of trustworthy ambients and that all the other ambients (including) $\top$, are trustworthy and form the complementary set $\mathcal{U}$. We wish to prove that SYS is protected (even when placed into a context which knows names *msg*, *mail* and b.)

Actually, we show that $P = SYS|E(SYS, \{b, msg, mail\})$ is defended. Then, Theorem 4 suffices to prove that $C[SYS]$ is protected in any context where there are no ambients with name b, *msg* and *mail*.

The solution in Table 3 is valid for P and shows it protected. Indeed ϕ satisfies the requirements of Definition 8:

1. *msg* can be opened in b only because $\overline{\mathtt{open}}\,msg^{(\mathcal{U}\cup\{b\},b)} \in \phi(msg)$;
2. ambient b is locked (it is not openable), because $\overline{\mathtt{open}}\,b$ does not occur in $\phi(b)$.

[2] For the sake of simplicity, we assume here that D is a passive datum, that may only occur in the same position of $\mathbf{0}$ in processes, and that requires no analysis.

$\phi(mail)$	$\{\mathtt{out}\, a^{(\mathcal{U},\mathcal{U})}, \mathtt{in}\, b^{(\mathcal{U},\mathcal{U})}, \overline{\mathtt{out}}\, mail^{(\mathcal{U}\cup\{b\},b)}, \mathtt{out}\, b^{(\mathcal{U}\cup\{b\},\mathcal{U}\cup\{b\})},$ $\mathtt{in}\, a^{(\mathcal{U}\cup\{b\},\mathcal{U})}, \overline{\mathtt{open}}\, msg^{(\mathcal{U}\cup\{b\},a)}\} \cup \{msg^{(\mathcal{U}\cup\{b\},\mathcal{U}\cup\{b\})}\}$
$\phi(b)$	$\{\bigcup_{n\in\mathcal{U}} n^{\mathcal{U},\mathcal{U}}, msg^{(\mathcal{U},\mathcal{U})}\}\cup$ $\{\overline{\mathtt{in}}\, b^{(\mathcal{U},\mathcal{U})}, D^{(\mathcal{U}\cup\{b\},\mathcal{U}\cup b\})}, \mathtt{open}\, msg^{(\mathcal{U},\mathcal{U})}\}$
$\phi(msg)$	$\{\mathtt{out}\, mail^{(\mathcal{U},\mathcal{U})}, \overline{\mathtt{open}}\, msg^{(\mathcal{U}\cup\{b\},b)}, D^{(\mathcal{U}\cup\{b\},b)}\}$
$\phi(n)$	$\{\bigcup_{m\in\mathcal{U}\cup\mathcal{T}} m^{(\mathcal{U},\mathcal{U})}\} \cup \{\bigcup_{m\in\mathcal{U}\cup\mathcal{T}} \mu^{(\mathcal{U},\mathcal{U})}, \bar{\mu}^{(\mathcal{U},\mathcal{U})}\},\ n \in \{a, \top\}$

Table 3. A Solution for SYS

Now we can better explain why the restrictions on the use of names in the context are necessary. The constraint on the use of trustworthy names b and msg are necessary for both the dynamic and the static property to hold. Dynamically, a context may contain a trustworthy ambient, say msg, that accept to be opened inside any other ambients. ¿From a static point of view, ϕ is a function and so ambients with the same name are mapped to the same element. So, there is no way to distinguish the located capabilities and ambients of the "good" and trustworthy ambient *mail* and the ones of the "bad " and trustworthy ambient *mail* occurring in the context.

Note that labels are essential to prove (statically) that SYS satisfies the security property (see the discussion on the precision of the analysis contained in Remark 1).

7 Conclusions

The idea of using static techniques, such as those based on CFA [12–14] and type systems [10, 4, 3, 5], to prove properties of mobile ambients is not new. The analysis of [12] is similar to ours but less accurate (if adapted to SA), since it does not use any contextual information (see Remark 1). The analyses contained in [13, 14] show how to perform a rational reconstruction of the analysis [12] in the framework of Abstract Interpretation as well as how to specify stronger analyses using powerful counting analyses. Moreover, in [2] it is proposed an approach similar to our, even though based on type systems.

Many research lines are still open, besides the use of shape analysis [14] for improving the precision of our analysis in the case of recursive processes. Among them, there is the problem of constructing solutions efficiently and of considering also communication. Furthermore, the generality of the analysis (which is not ad hoc for a specific property) suggests to define other static tests on the solutions for establishing other security properties, like for instance mobility and locking of [3, 5] or integrity.

It is worth mentioning that, even though SA was not introduced for security reasons, it seems to be more adequate than MA for better writing secure programs. Coaction $\overline{\mathtt{open}}\, n$ for instance offers a control on the access to the private resources of the ambient n. In the example of Section 6 a proper use of coactions guarantees that trustworthy ambients may be contained inside untrustworthy ambients without being opened. The protocol should be substantially complicated to achieve the same security property if MA is used.

Acknowledgments. We wish to thank Flemming and Hanne Riis Nielson for many useful discussions. This work has been partially supported by MURST Project TOSCA.

References

1. C. Bodei, P. Degano, F. Nielson and H. Riis Nielson. *Static Analysis for the π-calculus with their application to security.* To appear in *Information and Computation.* Also available at http://www.di.unipi.it/~chiara/publ-40/BDNNi00.ps.
2. M. Bugliesi and G. Castagna *Secure Safe Ambients and JVM Security.* Proceedings of the Workshop on Issues in the Theory of Security 2000 (co-located with ICALP'00), pages 18-23, 2000.
3. L. Cardelli, G. Ghelli and A. Gordon *Mobility types for mobile ambients.* Proceedings of ICALP' 99, LNCS 1644, pages 230-239. Springer-Verlag, 1999.
4. L. Cardelli and A. Gordon *Types for mobile ambients.* Proceedings of 26th ACM Principles of Programming Languages (POPL' 99), pages 79-92. ACM Press, 1999.
5. L. Cardelli and G. Ghelli *Ambient Groups and Mobility Types.* Proceedings of IFIP TCS 2000. To appear.
6. L. Cardelli and A. Gordon. *Mobile Ambients* Proceedings of FoSSaCS' 98, LNCS 1378, pages 140-155. Springer-Verlag, 1998.
7. L. Cardelli and A. Gordon. *Equational Properties of Mobile Ambients* Proceedings of FoSSaCS' 99, LNCS 1578, pages 212-226. Springer-Verlag, 1999.
8. P. Cousot and R. Cousot, *Systematic Design of Program Analysis Frameworks*, Proceedings of POPL '79, pages 269–282, ACM Press, 1979.
9. F. Levi and C. Bodei *Security Analysis of Mobile Ambients.* Proceedings of the Workshop on Issues in the Theory of Security 2000 (co-located with ICALP'00), pages 18-23, 2000.
10. F. Levi and D. Sangiorgi *Controlling Interference in Ambients.* Proceedings of the 27th ACM Principles of Programming Languages (POPL' 00), pages 352-364. ACM Press, 2000.
11. R. Milner and J. Parrow and D. Walker. *A calculus of mobile processes, (Part I and II)* Information and Computation, number 100, pages 1-77, 1992.

12. F. Nielson, H.R. Nielson, R.R. Hansen and J.G. Jensen *Validating firewalls in mobile ambients.* Proceedings of CONCUR' 99, LNCS 1664, pages 463-477. Springer-Verlag, 1999.
13. R. R. Hansen and J. G. Jensen and F. Nielson and H. R.Nielson *Abstract Interpretation of Mobile Ambients* Proceedings of SAS'99, LNCS 1694, pages 135-148, Springer-Verlag, 1999.
14. H. R. Nielson and F. Nielson *Shape Analysis for Mobile Ambients* Proceedings of POPL' 00, pages 135-148, ACM Press, 2000.
15. F. Nielson and H. R. Nielson and C. Hankin, *Principles of Program Analysis*, Springer,1999.