A Parametric Model for the Analysis of Mobile Ambients

Dino Distefano

Department of Computer Science,
Queen Mary, University of London
{ddino@dcs.qmul.ac.uk}

Abstract. In this paper we propose a new *parametric abstract finite* model of Mobile Ambients able to express several properties on processes. The model can be used for the analysis of these properties by means of model checking techniques. The precision of the model can be increased by modifying certain numeric parameters increasingly avoiding thereby the occurrences of false counterexamples in the analysis.

1 Introduction

The calculus of Mobile Ambients (MA) is meant to model *wide area* computations. Introduced in [2], MA has as main characteristic to allow active processes to move between different sites.

A wide range of work has been recently carried out on the analysis of mobile ambients [1,8,13,14,18], mostly based on static-analysis techniques and abstract interpretation [7].

In this paper we propose a *parametric finite abstract* model to analyse properties of mobile ambients processes by model checking — as an alternative to static analysis. Our model is based on techniques introduced first in [11]. Such techniques provide a general framework for modelling and verifying systems whose computation involves manipulation of pointer structures. The model we define here is suitable for verifying a wide range of safety properties of systems among which security properties such as *secrecy.* It has the following features: *(i)* It provides a *safe approximation* of the concrete transition system of processes. *(ii)* It models *finitely* (by means of abstraction) processes that are in principle infinite due to replication (i.e., $!P$). *(iii)* The model depends on two (numeric) *parameters* that can be increased to tune its precision in case false counterexamples are returned by the model checking algorithm.

The analysis we propose is based on the following strategy. Our models, called HABA, are special Büchi automata with some typical characteristic of history-dependent automata [17]. HABA are used to represent the behaviour of an ambient process P. Properties of interest are expressed in the temporal logic *NTL* (introduced in [11]) which is interpreted over HABA runs. Then, the model checking algorithm defined in [9,12] can be used to verify the validity of the properties against the model.

The first contribution of our approach w.r.t. existing analyses of MA lays on its ability to deal finitely with replication. The model distinguishes between P and $!P$ at several levels of precision (due to parametricity). Existing techniques can cope with replication only to a limited extent. They are designed only for abstraction $\{0, 1, \omega\}$ (i.e., none, one, many). Our abstraction goes beyond this range by considering a range $\{0, 1, \ldots, M, \omega\}$ with $M > 1$ parameter of the model. Therefore it is able to detect properties of the kind *"a certain number of copies of ambient n is inside ambient m at the same time"*. The second contribution of our approach is that the model introduced here provides a general and completely automated framework for the verification of properties of MA. This means that the model is *not* limited to some specific safety properties (like static analysis techniques). Many temporal properties expressible by NTL-formulae can be automatically checked on the abstract model giving us the possibility to infer safe answer on ambient processes.

Related work. Our model takes inspiration from the following works. The paper [18] proposes an algorithm detecting process firewalls that are not protective. The technique is based on a control flow analysis and does not distinguish between a process P and $!P$. This technique is enhanced in [13] where the precision of the analysis is improved by the use of information about the multiplicity of the number of ambients occurring within another ambient. The distinction is within the range $\{0, 1, \omega\}$. Another refinement of the analysis proposed in [18], for the special case of Safe Ambients [15], is introduced in [8]. However, the analysis proposed — as the one in [18] — *does not* distinguish between different copies of the same ambient. An abstract interpretation framework for MA is proposed in [14]. Based on [13] and [8] the analysis given in this paper considers some information about multiplicity of the ambients and contextual information. Again, based on [13], the paper [1] defines a more accurate analysis for capturing boundary crossing. Also in this work no information on multiplicities is provided.

A parallel stream of work considers model checking for mobile ambients using spatial logics [5] and in particular ambient logic[3]. In [6] the authors identify a fragment of mobile ambients (where replication is replaced by recursion) verifiable by model-checking. For this fragment, a model-checking algorithm for the ambient logic is proposed. The paper [4] introduces a spatial logic for synchronous π-calculus and investigate its power. A model-checking algorithm is then presented for a class of bounded processes. Our contribution stands somehow between these two independent streams of work in that it applies model checking in a static analysis oriented fashion.

Organisation of the paper. This paper is organised as follows: Section 2 reviews some background on the ambient calculus. Section 3 gives an overview of *NTL* and HABA. Section 4 defines an operational semantics for MA using HABA. Section 5 provides some concluding remarks.

Due to space limitation this paper presents the main ideas and results. More details and proofs are reported in the full version of this paper [10].

2 An Overview of Mobile Ambients

We consider the pure *Mobile Ambients* calculus [2] without communication primitives. Let $\mathcal{N}$ be a denumerable set of *names* (ranged over by a, b, n, m). The set of processes over $\mathcal{N}$ is defined according to the following grammar:

$$N ::= \mathsf{in}\, n \;\Big|\; \mathsf{out}\, n \;\Big|\; \mathsf{open}\, n \qquad\qquad \text{(capabilities)}$$

$$P, Q ::= \mathbf{0} \;\Big|\; (\nu n)P \;\Big|\; P|Q \;\Big|\; !P \;\Big|\; n[P] \;\Big|\; N.P \qquad \text{(processes)}$$

For a process P we write $n(P)$ for its set of names. $\mathbf{0}$ does not perform any action. The restriction $(\nu n)P$ creates a new name called n that is private in the scope of P. $P \mid Q$ is the standard parallel composition of processes P and Q. Replication $!P$ represents an arbitrary number of copies of P and it is used to introduce recursion as well as iteration. $n[P]$ represents an ambient with name n enclosing a running process P. Ambients can be arbitrarily nested. Capabilities provide ambients with the possibility to *interact* with other ambients. In particular, $\mathsf{in}\, n$ has the effect to move the ambient that performs it into a sibling ambient called n (if there exists one). Symmetrically, by $\mathsf{out}\, n$, an ambient nested inside n moves outside; $\mathsf{open}\, n$ dissolves an ambient n nested inside the one performing this capability.

The standard semantics of Mobile Ambients is given in [2] on the basis of a structural congruence between processes, denoted by $\equiv$ (see [2]), and a reduction relation $\rightarrow$. Processes are identified up to α-conversion. Moreover, note that: $n[P]|n[Q] \not\equiv n[P|Q]$ that is, multiple copies of an ambient n have distinct identities; and $!(\nu n)P \not\equiv (\nu n)!P$ that is, the replication operator combined with restriction creates an infinite number of new names. The reduction relation $\rightarrow$ is defined by the rules listed in Table 1.

Table 1. Reduction rules for Mobile ambients

$n[\mathsf{in}\, m.P\|Q]\|m[R] \rightarrow m[n[P\|Q]\|R]$	$\dfrac{P \rightarrow Q}{n[P] \rightarrow n[Q]}$	$\dfrac{P \rightarrow Q}{P\|R \rightarrow Q\|R}$	$\dfrac{P \rightarrow Q}{(\nu n)P \rightarrow (\nu n)Q}$
$\mathsf{open}\, n.P\|n[Q] \rightarrow P\|Q$	$m[n[\mathsf{out}\, m.P\|Q]\|R] \rightarrow n[P\|Q]\|m[R]$	$\dfrac{P' \equiv P \quad P \rightarrow Q \quad Q \equiv Q'}{P' \rightarrow Q'}$	

3 An Overview on *NTL* and HABA

In this section we summarise the framework for modelling and model checking systems with pointers introduced in [11].

Navigation Temporal Logic. Let LVAR be a countable set of logical variables ranged over by x, y, z, and *Ent* be a countable set of entities ranged over by e, e', e_1 etc. $\perp \notin \mathit{Ent}$ is used to represent "undefined"; we denote $E^{\perp} = E \cup \{\perp\}$ for arbitrary $E \subseteq \mathit{Ent}$. Navigation Temporal Logic (*NTL*) is a linear temporal logic where quantification ranges over logical variables that can denote entities, or may be undefined. The syntax is defined by the grammar:

$$\alpha \ ::= \ nil \ \Big| \ x \ \Big| \ \alpha\uparrow \qquad\qquad \text{(navigation expressions)}$$

$$\Phi \ ::= \ \alpha = \alpha \ \Big| \ \alpha \ \mathsf{new} \ \Big| \ \alpha \rightsquigarrow \alpha \ \Big| \ \Phi \wedge \Phi \ \Big| \ \neg\Phi \ \Big| \ \exists\, x.\, \Phi \ \Big| \ \mathsf{X}\,\Phi \ \Big| \ \Phi\, \mathsf{U}\, \Phi \quad \text{(formulae)}$$

nil denotes the null reference, x denotes the entity that is the value of x (if any), and $\alpha\uparrow$ denotes the entity referred to by (the entity denoted by) α (if any). Let $x\uparrow^0 = x$ and $x\uparrow^{n+1} = (x\uparrow^n)\uparrow$ for natural n. The basic proposition α new states that the entity (referred to by) α is fresh, $\alpha = \beta$ states that α and β are aliases, and $\alpha \rightsquigarrow \beta$ expresses that (the entity denoted by) β is reachable from (the entity denoted by) α. The boolean connectives, quantification, and the linear temporal connectives X (next) and U (until) have the usual temporal interpretation. We denote $\alpha \neq \beta$ for $\neg(\alpha = \beta)$, $\alpha \not\rightsquigarrow \beta$ for $\neg(\alpha \rightsquigarrow \beta)$ and $\forall x.\, \Phi$ for $\neg(\exists\, x.\, \neg\Phi)$. The other boolean connectives and temporal operators $\Diamond$ (eventually) and $\Box$ (always) are standard [19]. For example, $\Diamond(\exists x.\, x \neq v \wedge x \rightsquigarrow v \wedge v \rightsquigarrow x)$ expresses that eventually v will point to a non-empty cycle.

Formulae are interpreted over infinite sequences of triples, called *allocation sequences*, $(E_0, \mu_0, \mathcal{C}_0)(E_1, \mu_1, \mathcal{C}_1)(E_2, \mu_2, \mathcal{C}_2)\ldots$ where for all $i \geqslant 0$, $E_i \subseteq \mathit{Ent}$ and $\mu_i : E_i^{\perp} \to E_i^{\perp}$ such that $\mu_i(\perp) = \perp$; μ_i encodes the pointer structure of E_i. $\mathcal{C}_i$ is a function on E_i such that $\mathcal{C}_i(e) \in \mathbb{M} = \{1, \ldots, M\} \cup \{*\}$ for some fixed constant $M > 0$. The number $\mathcal{C}_i(e)$ is called the *cardinality* of e. Entity e for which $\mathcal{C}_i(e) = m \leqslant M$ represents a chain of m "concrete" entities; if $\mathcal{C}_i(e) = *$, e represents a chain that is longer than M. In the latter case, the entity is called *unbounded*. (Such entities are similar to summary nodes [20], with the specific property that they always abstract from chains.) The cardinality of a set is defined as $\mathcal{C}(\{e_1, \ldots, e_n\}) = \mathcal{C}(e_1) \oplus \ldots \oplus \mathcal{C}(e_n)$ where $n \oplus m = n+m$ if $n+m \leqslant M$ and $*$ otherwise.

Automata-based models. States in our automata are triples $(E, \mu, \mathcal{C})$, called *configurations*. Let CONF denote the set of all configurations ranged over by γ and γ'. Configurations that represent the same pointer structure at different abstraction levels are related by *morphisms*. For $\gamma, \gamma' \in$ CONF, a morphism is surjective function $h : E_\gamma \to E_{\gamma'}$ which maintains the abstract shape of the pointer dependencies represented by the two related configurations. Moreover, a (pure) chain may be abstracted to a single entity while keeping the cardinality invariant. That is, the cardinality of an entity $e \in \gamma'$ is equal to the sum of the cardinalities of the entities in $h^{-1}(e)$. Collapsing chains to single entities —provided correspondence of the cardinality— is the mechanism used by morphisms to associate to a configuration another more abstract configuration.

Although morphisms provide us with a tool for abstraction of pointer structures, they do not model the dynamic evolution of such structures. To reflect the execution of pointer-manipulating operations as well as the creation or deletion of entities we use *reallocations*. For $\gamma, \gamma' \in$ CONF, a *reallocation* is a multi-set $\lambda : (E^{\perp} \times E'^{\perp}) \to \mathbb{M}$ which redistributes (but preserves) cardinalities on E to

E'. More precisely, the total cardinality $\bigoplus_{e' \in E'} \lambda(e, e')$ allocated by λ to $e \in E$ equals $\mathcal{C}(e)$; and the total cardinality $\bigoplus_{e \in E} \lambda(e, e')$ assigned to $e' \in E'$ equals $\mathcal{C}'(e')$. As in the case of morphisms, one entity can be related by a reallocation to more than one entity only if these form a chain. Note that the identity function id is a reallocation. We write $\gamma \overset{\lambda}{\rightsquigarrow} \gamma'$ if there is a reallocation (named λ) from γ to γ'. Reallocations are a generalisation of the idea of identity change as present in history-dependent automata [17]: besides the possible change of identity of entities, it allows for the evolution of pointer structures[1].

To model the dynamic evolution of a system manipulating (abstract) linked lists, we use a generalisation of Büchi automata where each state is a configuration and transitions exist between states only if these states can be related by means of a reallocation reflecting the possible change in the pointer structure.

Definition 3.1. *A* high-level allocation Büchi automaton *(HABA)* $\mathcal{H}$ *is a tuple* $\langle X, C, \rightarrow, I, \mathcal{F} \rangle$ *with:* (i) $X \subseteq$ LVAR*, a finite set of logical variables;* (ii) $C \subseteq$ CONF*, a set of configurations (also called* states*);* (iii) $\rightarrow \subseteq C \times (Ent \times Ent \times \mathbb{M}) \times C$*, a* transition relation*, s.t.* $c \rightarrow_\lambda c' \Rightarrow c \overset{\lambda}{\rightsquigarrow} c'$*;* (iv) $I : C \rightharpoonup 2^{Ent} \times (X \rightharpoonup Ent)$*, an* initialisation *function such that for all* c *with* $I(c) = (N, \theta)$ *we have* $N \subseteq E$ *and* $\theta : X \rightharpoonup E$*.* (v) $\mathcal{F} \subseteq 2^C$ *a set of sets of* accept *states.*

HABA can be used to model the behaviour of systems at different levels of abstraction. In particular, when all entities in any state are concrete (i.e., $\mathcal{C}(e) = 1$ for all e), a concrete model is obtained that is very close to the actual system behaviour.

Model Checking NTL. In [9,12] a model checking algorithm which establishes whether a formula Φ is valid on a given (finite) HABA $\mathcal{H}$ was developed. The model checking algorithm is based on the construction of a tableau graph $G_{\mathcal{H}}(\Phi)$ out of $\mathcal{H}$ and Φ as in [16]. We give here a short summary of this construction.

States of $G_{\mathcal{H}}(\Phi)$ are pairs (q, D) where q is a state of $\mathcal{H}$ and D is the collections of sub-formulae of Φ, and their negations, that possibly hold in q. A transition from (q, D) to (q', D') exists in $G_{\mathcal{H}}(\Phi)$ if $q \rightarrow_\lambda q'$ in $\mathcal{H}$ and, moreover, for each sub-formula $\mathsf{X}\Psi$ in D there exists a "corresponding" Ψ in D'. Here, the correspondence is defined modulo the reallocation λ. A *fulfilling path* in $G_{\mathcal{H}}(\Phi)$ is then an infinite sequence of transitions — starting from an initial state — that also satisfies all the "until" sub-formulae $\Psi_1 \mathsf{U} \Psi_2$. That is, if $\Psi_1 \mathsf{U} \Psi_2$ is in a given state in the sequence, then a corresponding Ψ_2 (modulo a sequence of reallocations) occurs in a later state. Fulfilling path are related with the validity of Φ. More precisely, Φ is valid in $\mathcal{H}$ (written $\mathcal{H} \models \Phi$) iff there does not exist a fulfilling path in $G_{\mathcal{H}}(\neg\Phi)$. The existence of a *self-fulfilling strongly connected sub-component* (SCS) in $G_{\mathcal{H}}(\neg\Phi)$ provides us with a necessary criterion for the existence of a fulfilling path. The tableau graph of a finite HABA is always finite and its number of SCSs is finite as well. Moreover, since the property of

[1] A complete treatment of morphisms and reallocations of pointer structures can be found in [9,11,12].

self-fulfilment is decidable, this gives rise to a mechanical procedure for verifying the validity of formulae.

In [9,12] we also showed that if a formula is valid in an abstract HABA $\mathcal{H}$, then it is valid in all concrete (infinite-state) automata $\mathcal{H}_c$ represented by $\mathcal{H}$. Therefore it is enough to verify the validity on the finite-state abstract automata to infer the validity of the property in all its concretizations. As usual in model checking of infinite-state systems in the presence of abstraction the algorithm is sound but not complete in the sense that it might return *false negatives.* This means that if the algorithm fails to show that Φ is valid in $\mathcal{H}$ then it cannot be concluded that Φ is *not* satisfiable (by some run of $\mathcal{H}$). However, since such a failure is always accompanied by a "prospective" counterexample of Φ, further analysis or testing may be used to come to a more precise conclusion.

4 An Abstract Operational Model for Mobile Ambients

Before defining our model we give two motivating examples.

Example 1. In [8] the following system is considered. Ambient m wants to send a message to ambient b. Messages are delivered enclosed in a wrapper ambient that moves inside the receiver which acquires the information by opening it. For secret messages we want to be sure that they can be opened *only* by the receiver b: $SYS_1 = m[mail[\mathsf{out}\, m.\mathsf{in}\, b.msg[\mathsf{out}\, mail.D]]] \mid b[\mathsf{open}\, msg] \mid \mathsf{open}\, msg.$

Data D is secret, *mail* is the pilot ambient that goes out of m to reach b. The outer-most ambient attempts to access the secret by $\mathsf{open}\, msg$. Once inside b, the wrapper *mail* is opened and b reads the secret D. For the process SYS_1 we want to guarantee that the property (UA): *"no untrusted ambients can access D"* holds.

The previous example illustrates the relevance of *secrecy* in wide-area computations. However, there are other important properties which are relevant for the safety of systems. An instance is given in the following.

Example 2. Let us assume that a distributed network of an organisation (e.g., a bank) has a server used by a certain number of clients to execute critical operations (e.g., buying/selling stocks). A rather trivial implementation could be the following system:

$$\begin{aligned} SYS_2 &= Serv[PORT|PORT|PORT|Exec] \mid Cl[REQ] \mid Cl[REQ] \\ PORT &= P[\mathsf{in}\, Req.\mathsf{in}\, Exec] \\ REQ &= !Req[\mathsf{out}\, Cl.\mathsf{in}\, Serv.\mathsf{open}\, P.DATA] \end{aligned}$$

A client Cl asks the server to execute an order (buying/selling) by sending a request. Details of orders are contained in $DATA$. The ambient Req, implementing a request, leaves the client and goes into the server. Once there, Req uses one of the available $PORT$s which are the data structures used by the server to execute the requests. The port P moves the request to some process $Exec$

which executes the order in *DATA* (and then it gives back P to *Serv*). As any other real-life server, the bank's server can accept a limited amount of requests at the same time. The risk is that its finite number of internal data structures (ports Ps) are consumed in pending requests not yet completed. This can result in an overflow and the server must be rebooted while — in the meanwhile — the bank may be loosing millions. Predicting the number of requests may be difficult. This is mostly because the clients place their orders following some mathematical models which depend from several random variables.

Safety for such kind of systems involves the *number* of requests that the server has to deal with at each time. It is essential that the property no-overflow (NO) holds, i.e. *at any point in time the server has to deal with a number of requests smaller or equal to the size of its data structures (in this case 3 ports).*

Now, suppose that the system is expanded and new clients are added to the bank's network. Therefore the designer of the system decides to implement some strategy meant to avoid overflows. The system is upgraded with a buffer using the following strategy. If the server gets shorter in ports it sends a broadcast message (*BCAST*) to its clients and informs them to address their requests to the buffer (instead that the server). From that moment the server accepts only requests from the buffer which forwards client's orders when the server ask for one (by the ambient *ASK_BUF*). When the server has executed enough requests and its number of free ports get back to normal, the server broadcasts another message (*ADDR_to_Ser*) to the clients to inform them that from that moment on they can again address their requests directly to the server. The designer implements this idea in the following new system:

$$\begin{aligned}
SYS_2 &= SER \mid Cl[REQ|to\_Ser] \mid Cl[REQ|to\_Ser] \\
&\quad \mid Cl[REQ|to\_Ser]| \; Cl[REQ|to\_Ser] \\
REQ &= !Req[\mathsf{in}\ to\_Ser.\mathsf{out}\ to\_Ser.\mathsf{out}\ Cl.\mathsf{in}\ Serv.\mathsf{open}\ P.DATA|inB.DATA] \\
BUF &= Buf[!Req_B[\mathsf{open}\ Ask\_Buf.\mathsf{open}\ B.\mathsf{out}\ Buf.\mathsf{in}\ Serv.\mathsf{open}\ P]] \\
ASK\_BUF &= !Ask\_Buf[\mathsf{open}\ Ready\_for\_req.\mathsf{out}\ Serv.\mathsf{in}\ Buf.\mathsf{in}\ Req_B] \\
BCAST &= BCast[\mathsf{out}\ Serv.\mathsf{in}\ Cl.\mathsf{open}\ to\_Ser.B[\mathsf{open}\ Req.\mathsf{out}\ Cl.\mathsf{in}\ Buf.\mathsf{in}\ Req_B]] \\
ADDR\_to\_Ser &= \mathsf{open}\ Norm\_St.to\_Ser[\mathsf{out}\ Serv.\mathsf{in}\ Cl] \\
SER &= Serv[ASK\_BUF|PORT|PORT|PORT|BCAST|BCAST|BCAST \\
&\quad |BCAST|ADDR\_to\_Ser|ADDR\_to\_Ser|ADDR\_to\_Ser|ADDR\_to\_Ser]
\end{aligned}$$

Now, it should be formally verified that this patch properly avoids any overflows, i.e., in this new system the property no-overflow (NO) holds. Note that (NO) cannot be accurately verified by analyses dealing only with multiplicities $\{0, 1, \omega\}$ as those found in the literature. Other example properties that this system should have and we might wish to verify are: (REQ): *any request eventually reaches the server*; and (REQB): *an ambient Req_B leaves the buffer only after the server has asked for a new request by sending the message Ask_Buf.*

In this paper we are concerned with the verification of the kind of properties described in these two examples.

4.1 HABA Modelling Approach

Due to replication, the concrete transition systems of processes are infinite. Since we want to use model checking as analysis technique for processes it is essential that their representation in the model is finite. A naive encoding of the process topology would be hopeless. Therefore, we focus only on essential information which allow us to infer the properties we need. Along the lines of [8,13,14,18], the information we retrieve from a mobile ambient process P is: *which ambients may end up in which other ambient.* To model P we introduce a classification among the entities in use. For any ambient a occurring in P we have:

– A special entity a^{ho} (called a's *host*) is used to record, at any point in the computation, the ambients (hosted) directly inside *any* copy of ambient a. a^{ho} is fixed, i.e., during the computation its position within the topology of the process does not change.
– A special entity a^{is} (called the *inactive site* of a). It is the repository where the copies of a are placed when this ambient is inactive. Informally speaking, inactive means that a cannot yet execute any action (see Section 4.2). As a^{ho}, also a^{is} does not move during the computation.
– All other entities —distinct from a^{ho} and a^{is}— represent instances of the ambient a. A concrete entity can move according to the capabilities of the particular copy of a it represents. Several instances of a may be represented by a single multiple or unbounded entity.

Example 3. State q_{in} in Figure 4 depicts how process SYS_1 of Example 1 is represented in our model. Outgoing references define the child/parent relation μ. Notation $e{:}n$ says that e denotes an ambient with name n. The host of an ambient, say a, keeps track of the ambients directly contained in *any* copy of a. Thus, ambients m and b are inside the outer-most ambient @, whereas *mail* is inside m. Ambients b and *mail* are empty. Hosts entities are depicted as squares and inactive sites as patterned squares. *msg* is *inactive* since in the beginning it cannot execute any action. Only when both $\mathsf{out}\, m$ and $\mathsf{in}\, b$ have been consumed, *msg* becomes *active*. Inactive ambients are modelled by having the copies pointing to their inactive site. Figure 2 (left) shows the use of the unbounded entity e_2 (depicted as patterned circle) to model more than M copies of the ambient n.

Preliminary notation. We assume the existence of a global function $\mathsf{A} : \mathit{Ent} \to n(P)$ that associates to every entity e a name of the ambient in P represented by e. For $e \in \mathit{Ent}$, we write $e{:}n$ as a shorthand for $\mathsf{A}(e) = n$ and $e{:}n \in E$ as a shorthand for $e \in E \wedge \mathsf{A}(e) = n$.

We consider two fixed disjoint sets of entities: the set of inactive sites $E^{\mathsf{is}}_P = \{n^{\mathsf{is}} \in \mathit{Ent} \mid n \in n(P)\}$ and the set of hosts $E^{\mathsf{ho}}_P = \{n^{\mathsf{ho}} \in \mathit{Ent} \mid n \in n(P)\}$. For every ambient n we assume $\mathsf{A}(n^{\mathsf{is}}) = \mathsf{A}(n^{\mathsf{ho}}) = n$ and in every state of the model n^{is} points to n^{ho}. A HABA state modelling mobile ambients is of the form:

$$q = \langle \gamma, \mathsf{P} \rangle \in \textsc{States}$$

where $\textsc{States} = \textsc{Conf} \times (\mathit{Ent} \rightharpoonup 2^{\mathbf{Proc}})$. The first component $\gamma = (E, \mu, \mathcal{C}) \in \textsc{Conf}$ is a standard HABA configuration as defined in Definition 3.1. Given an entity e, the second component $\mathsf{P} : \mathit{Ent} \rightharpoonup 2^{\mathbf{Proc}}$ associates to e the set of processes e must execute. In figures the component $\mathsf{P}(e)$ is depicted close to e. It is not written if it is the empty process.

Pre-initial state and Initial state. The *pre-initial state* is an artificial state added to the model in order to identify by *NTL*-formulae which ambient an entity represents. The pre-initial state of a process P is constructed in such a way that every entity representing a copy of the ambient n points to the inactive site n^{is}. The structure of the pre-initial state does not reflect the initial topology described by P. *NTL*-formulae exploit the fact that an entity e in the pre-initial state leads to n^{is} to express that e stands for a copy of the ambient n. State q_{pre} in Figure 4 illustrates the pre-initial state of the process SYS_1 of Example 1. Although $\mathsf{A}(e_1) = m$, this information cannot be exploited in *NTL*. However, *NTL*-formulae can refer to the set X of logical variables in the model (see Def. 3.1). By having a variables x_m for any $m \in n(P)$, and by interpreting x_m into m^{is} (see ϑ in Def. 4.2) *NTL*-formulae can refer to m^{is} and therefore to all the other entities. Hence, by the special shape of the pre-initial state, we can use the formula $\exists x : x \rightsquigarrow x_m$ to express that x as a copy of the ambient m.

The *initial-state* models the child/parent relation (i.e. the topology) described by the process in terms of entities and pointers. For example, in Figure 4, q_{in} is the initial state of the process SYS_1 of Example 1. Note that the ambient @ does not have a real instance (it is modelled only by $@^{\mathsf{is}}$ and $@^{\mathsf{ho}}$), therefore we use $@^{\mathsf{is}}$ for the execution of capabilities.

Example 4. The security property (UA) of Example 1 is violated if and only if the following *NTL* formula is satisfied

$$\Phi_{\mathsf{UA}} \equiv \exists x : x \rightsquigarrow x_{msg} \wedge \Diamond(x \not\rightsquigarrow x_{msg} \;\wedge\; x\!\uparrow \neq x_{mail}\!\uparrow \;\wedge\; x\!\uparrow \neq x_b\!\uparrow).$$

Φ_{UA} states that *msg* eventually will be included inside an ambient different from *mail* and b (which are the only trustworthy ones). Note the use of x_{msg}, x_{mail}, x_b to refer to ambient names.

The property no-overflow (NO) (see Example 2) is violated if there are at least four distinct requests inside the server at the same time:

$$\begin{aligned}\Psi_{\mathsf{NO}} \equiv \exists x, y, z, w : x \rightsquigarrow x_{Req} \wedge y \rightsquigarrow x_{Req} \wedge z \rightsquigarrow x_{Req} \wedge w \rightsquigarrow x_{Req} \wedge \\ (x \neq y \wedge x \neq z \wedge x \neq w \wedge y \neq z \wedge y \neq w \wedge z \neq w) \wedge \\ \Diamond(x\!\uparrow = x_{Server} \wedge y\!\uparrow = x_{Server} \wedge z\!\uparrow = x_{Server} \wedge w\!\uparrow = x_{Server})\end{aligned}$$

REQ and REQB (see Example 2) are satisfied if the following formulae hold:

$$\begin{aligned}\Psi_{\mathsf{REQ}} &\equiv \forall x : x \rightsquigarrow x_{Req} \Rightarrow \Diamond(x\!\uparrow = x_{Serv}) \\ \Psi_{\mathsf{REQB}} &\equiv \Box(\forall x{:}x \rightsquigarrow x_{Req_B} \wedge x\!\uparrow = x_{Buf} \Rightarrow (x\!\uparrow \mathsf{U}\, \exists y{:}y \rightsquigarrow x_{Ask\_Buf} \wedge y\!\uparrow{=}x_{Buf}))\end{aligned}$$

Hence, if $\mathcal{H}_{SYS_1}$ and $\mathcal{H}_{SYS_2}$ are the HABA modelling SYS_1 and SYS_2, the properties are guaranteed to hold if we verify $\mathcal{H}_{SYS_1} \models \neg\Phi_{\mathsf{UA}}$, $\mathcal{H}_{SYS_2} \models \neg\Psi_{NO}$,

$\mathcal{H}_{SYS_2} \models \Psi_{REQ}$ and $\mathcal{H}_{SYS_2} \models \Psi_{REQB}$. That can be automatically checked using the model checking algorithm defined in [9,12].

Canonical form for configurations. As models we use HABA whose configurations are in a special form called canonical. The main advantage of canonical configurations is that the resulting HABA is proved to be finite-state [9]. Informally, given a $L > 0$, a configuration γ is *L-canonical* (or *in L canonical form*) if (a) only concrete entities are closer than $L + 1$ pointer dereferences from a host; and, (b) there are no pure chains longer than $L + 1$. For every configuration γ its canonical form exists and it is unique (denoted by $\mathsf{cf}(\gamma)$). $\mathsf{cf}(\gamma)$ is determined by the unique morphism $h_{\mathsf{cf}} : \gamma \to \mathsf{cf}(\gamma)$.

The Parameters M and L. The precision of automaton $\mathcal{H}$ is ruled by two parameters: L controlling the canonical form; and M defining the minimum number of copies of an ambient represented by a single unbounded entity. Due to canonical form, non-concrete entities are not direct children of hosts: there are L concrete entities in between although all of them represent different instances of the *same* ambient. In other words, a chain of entities $e{:}b, e'{:}b \ldots$ pointing to a host, say a^{ho} represents a set of instances of b inside a. By L and M we are able to distinguish that inside a there are no instances of the ambient b; or there are *precisely* i instances of b with $1 \leq i \leq L + M$; or there are more than $L + M$ instances of b. Since L and M are parameters of the model they can be properly tuned to accomplish a more precise model. For example, assume $M = 1$ and $L = 3$. In q_3 of Figure 3, we know that inside a there are *exactly* two instances of n and *any* number of b's copies strictly greater than 4.[2]

4.2 Coding Processes into HABA Configurations

In this section we define a function $\mathcal{D}$ that codes a given process P into a HABA state. $\mathcal{D}$ returns: *(i)* a configuration γ that models P's topology; and *(ii)* a function P that associates to every entity the set of capabilities. We first define all the auxiliary elements necessary to $\mathcal{D}$'s definition.

Union configuration. For a configuration γ, let $E^{\mathsf{c}}_{\gamma} = E_{\gamma} \backslash (E^{\mathsf{ho}}_{P} \cup E^{\mathsf{is}}_{P})$ be its set of non-fixed entities. For configurations γ, γ' such that $E^{\mathsf{c}}_{\gamma} \cap E^{\mathsf{c}}_{\gamma'} = \varnothing$, the *union configuration* is $\gamma \uplus \gamma' = (E_{\gamma} \cup E_{\gamma'}, \mu, \mathcal{C})$ where: $\mathcal{C}(e) = \mathcal{C}_{\gamma'}(e)$ if $e \in E_{\gamma'} \backslash E_{\gamma}$ and $\mathcal{C}(e) = \mathcal{C}_{\gamma}(e)$ otherwise; and

$$\mu(e) = \begin{cases} \mu_{\gamma}(e) & \text{if } e \in E_{\gamma} \\ \mu_{\gamma'}(e) & \text{if } e \in E_{\gamma'} \backslash E^{\mathsf{c}}_{\gamma'} \\ \mathit{first}(\{e'{:}a \in E^{\mathsf{c}}_{\gamma} \mid b^{\mathsf{ho}} \in \mu^*(e')\} \cup \{b^{\mathsf{ho}}\}) & \text{if } e{:}a \in E^{\mathsf{c}}_{\gamma'} \text{ and } \mu_{\gamma'}(e) = b^{\mathsf{ho}} \\ \mu_{\gamma'}(e) & \text{otherwise} \end{cases}$$

[2] Between copies of the same ambient, we draw dashed horizontal arrows to stress that, at the conceptual level, these arrows do not describe a child/parent relation as the solid vertical ones.

For $e{:}a \in E_{\gamma'}$, $\mu(e)$ assigns the first entity in the queue of copies of a. If both configurations have copies of an ambient, say b, inside the same ambient, say a, the union appends the copies of the second configuration to those of the first one. The union for P is defined point-wise: let $q = \langle \gamma, \mathsf{P} \rangle$ and $q' = \langle \gamma', \mathsf{P}' \rangle$ and $e \in E_\gamma \cup E_{\gamma'}$, then

$$(\mathsf{P} \uplus \mathsf{P}')(e) = \begin{cases} \mathsf{P}(e) \cup \mathsf{P}'(e) & \text{if } e \in E_\gamma \cap E_{\gamma'} \\ \mathsf{P}(e) & \text{if } e \in E_\gamma \backslash E_{\gamma'} \\ \mathsf{P}'(e) & \text{if } e \in E_{\gamma'} \backslash E_\gamma \end{cases}$$

Finally the union of states is $\langle \gamma, \mathsf{P} \rangle \uplus \langle \gamma', \mathsf{P}' \rangle = \langle \gamma \uplus \gamma', \mathsf{P} \uplus \mathsf{P}' \rangle$.

Sub-processes executed by ambients. Given a process P the function $\rho : \mathbf{Proc} \to 2^{\mathbf{Proc}}$ returns the set of sub-processes that the ambient containing P can execute:

$$\begin{array}{lll} \rho(\mathbf{0}) = \varnothing & \rho(M.Q) = \{M.Q\} & \rho(Q \mid Q') = \rho(Q) \cup \rho(Q') \\ \rho(m[Q]) = \varnothing & \rho(!Q) = \{!Q\} & \rho((\nu n)Q) = \rho(Q) \end{array}$$

Processes belonging to nested ambients are not returned. Note that because we do not distinguish between $\nu!P$ and $!\nu P$ we can delete restriction[3].

Enabled and active ambients. An *enabled* ambient is an ambient which is ready to perform some action. Syntactically enabled ambients are those not guarded by a capability. The corresponding semantic notion is being active. In state q, the ambient n *is active* if $\nexists e \in E_{\gamma_q} : \mu_{\gamma_q}(e) = n^{\mathsf{is}}$. If n is not active it is called *inactive*. In the operational model only entities related to active ambients can execute capabilities.

Constructing the state. We use the following abbreviation for a state composed only by two entities.

$$\begin{aligned}(e_1, k_1, P_1) \twoheadrightarrow (e_2, k_2, P_2) = \langle \{e_1, e_2\}, \{e_1 \mapsto e_2\}, \{e_1 \mapsto k_1,\ e_2 \mapsto k_2\}, \\ \{e_1 \mapsto P_1,\ e_2 \mapsto P_2\} \rangle\end{aligned}$$

The next function $\Omega(a, P, k, \mathsf{act})$ returns a HABA state representing the process P contained inside the ambient a. The parameter k deals with cardinalities. The parameter act is a boolean that instructs Ω to construct the configuration with the active or with the inactive representation of its ambients. Formally,

[3] However, we assume that names occurring bound inside restriction are all distinct from each other and from the free names.

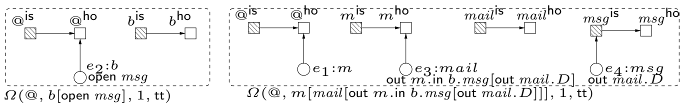

Fig. 1. HABA states returned by $\Omega(@, m[mail[\mathsf{out}\ m.\mathsf{in}\ b.msg[\mathsf{out}\ mail.D]]], 1, \mathsf{tt})$ and $\Omega(@, b[\mathsf{open}\ msg], 1, \mathsf{tt})$

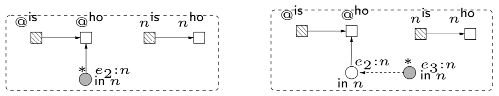

Fig. 2. Left: HABA state returned by $\Omega(@, !n[\mathsf{in}\ n], 1, \mathsf{tt})$. Right: Its 1-canonical form.

$\Omega : \mathcal{N} \times \mathbf{Proc} \times \mathbb{M}^* \times \mathbb{B} \to$ STATES is given by:

$$\begin{aligned}
\Omega(a, \mathbf{0}, k, \mathsf{act}) &= (a^{\mathsf{is}}, 1, \mathbf{0}) \twoheadrightarrow (a^{\mathsf{ho}}, 1, \mathbf{0}) \\
\Omega(a, m[Q], k, \mathsf{act}) &= \Omega(a, \mathbf{0}, k, \mathsf{act}) \uplus \Omega(m, Q, k, \mathsf{act}) \\
&\quad \uplus \begin{cases} (e, k, \rho(Q)) \twoheadrightarrow (a^{\mathsf{ho}}, 1, \mathbf{0}), \text{ if } \mathsf{act} \\ (e, k, \rho(Q)) \twoheadrightarrow (m^{\mathsf{is}}, 1, \mathbf{0}) \text{ otherwise} \end{cases} \\
&\qquad\qquad \text{where } e{:}m \text{ is fresh} \\
\Omega(a, Q_1|Q_2, k, \mathsf{act}) &= \Omega(a, Q_1, k, \mathsf{act}) \uplus \Omega(a, Q_2, k, \mathsf{act}) \\
\Omega(a, (\nu n)Q, k, \mathsf{act}) &= \Omega(a, Q, k, \mathsf{act}) \\
\Omega(a, !Q, k, \mathsf{act}) &= \Omega(a, Q, *, \mathsf{act}) \\
\Omega(a, N.Q, k, \mathsf{act}) &= \Omega(a, \mathbf{0}, k, \mathsf{act}) \uplus \Omega(a, Q, k, \mathsf{ff})
\end{aligned}$$

The representation of $m[Q]$ in a comprehends $a^{\mathsf{is}}, a^{\mathsf{ho}}$, the sub-state of Q inside m and a configuration with a non fixed entity e standing for the copy of m in a. Depending on the parameter act, this representation can be either the active or the inactive one. $\Omega(a, !Q, k)$ changes the cardinality from k to $*$. Finally, the representation of $N.Q$ inside a has the *inactive* representation for the process Q.

Example 5. Figure 1 shows $\Omega(@, m[mail[\mathsf{out}\ m.\mathsf{in}\ b.msg[\mathsf{out}\ mail.D]]], 1, \mathsf{tt})$ and $\Omega(@, b[\mathsf{open}\ msg], 1.\mathsf{tt})$. In the former, note the different representation between active ambients (@, m, $mail$) and inactive (msg). The left part of Figure 2 shows a state involving replication. We have $\Omega(@, !n[\mathsf{in}\ n], 1, \mathsf{tt}) = \Omega(@, n[\mathsf{in}\ n], *, \mathsf{tt})$ therefore, the entity e_2 modelling the copies of n, becomes unbounded.

Definition 4.1 (Process encoding). *The* process encoding *function* $\mathcal{D}$:$\mathbf{Proc}\!\to$ STATES *is given by* $\mathcal{D}(P) = \langle \mathsf{cf}(\gamma), \mathsf{P}[@^{\mathsf{is}} \mapsto \rho(P)] \rangle$ *where* $\Omega(@, P, 1, tt) = \langle \gamma, \mathsf{P} \rangle$.

The existence of a unique canonical form is proved in [9,12]. The state in the left part of Figure 2 is not L-canonical for any $L > 0$. The canonical form for $L = 1$ is shown on the right side. Note that $\mathcal{D}$ assigns to $@^{\mathsf{is}}$ the set $\rho(P)$ with the capabilities to be executed by @. For any process P, its pre-initial state is given by $q_{pre} = \Omega(@, P, 1, \mathsf{ff})$ and the initial state by $q_{in} = \mathcal{D}(P)$. Figure 4 shows q_{pre} and q_{in} of $SYS1$.

Table 2. Functions for moving ambients used in the operational rules

$act : \mathbf{Proc} \times \mathcal{N} \times \textsc{Conf} \to \textsc{Conf}$ defined by
$act_{Q,a}(\gamma) = (\gamma \setminus \gamma_{\Omega(a,Q,1,\mathsf{ff})}) \uplus \gamma_{\Omega(a,Q,1,\mathsf{tt})}$

$move : \textsc{Conf} \times Ent \times Ent \to \textsc{Conf}$ defined by
$move(\gamma, e, \hat{e}) = (E_\gamma, \mu_\gamma[e \mapsto \hat{e},\ \mu_\gamma^{-1}(e) \mapsto \mu_\gamma(e),\ e' \mapsto e], \mathcal{C}_\gamma)$
where $\mu_\gamma(e') = \hat{e}$, $\mathsf{A}(e') = \mathsf{A}(e)$

$\text{IOUp} : (\textsc{States} \times \mathbf{Proc} \times Ent \times Ent) \to \textsc{States}$ defined by
$\text{IOUp}(q, N.Q, e, \hat{e}) = \langle act_{Q,\mathsf{A}(e)} \circ move(\gamma_q, e, \hat{e}),\quad \mathsf{P}[e \mapsto \mathsf{P}(e) \setminus \{N.Q\} \cup \rho(Q)] \rangle$

$diss : (\textsc{Conf} \times Ent \times Ent) \to \textsc{Conf}$ defined by
$diss(\gamma, a^{\mathsf{ho}}, e{:}b) = (E_\gamma \setminus \{e\},\ \mu_\gamma[\mu_\gamma^{-1}(e) \mapsto a^{\mathsf{ho}},\ \mu_\gamma^{-1}(b^{\mathsf{ho}}) \mapsto a^{\mathsf{ho}}],\ \mathcal{C}_\gamma \restriction E_\gamma \setminus \{e\})$

$\text{OpenUp} : (\textsc{States} \times Ent \times Ent \times \mathbf{Proc}) \to \textsc{States}$ defined by
$\text{OpenUp}(q, N.Q, e'{:}a, e) = \langle act_{Q,a} \circ diss(\gamma_q, a^{\mathsf{ho}}, e), \mathsf{P}[e' \mapsto \mathsf{P}(e') \setminus \{N.Q\} \cup \rho(Q) \cup \mathsf{P}(e)] \rangle$

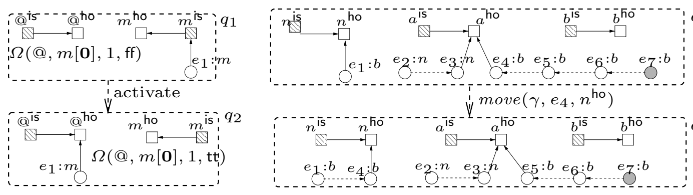
q_1
$@^{\mathsf{is}}$ $@^{\mathsf{ho}}$ m^{ho} m^{is}
$\Omega(@, m[\mathbf{0}], 1, \mathsf{ff})$
$e_1{:}m$
activate
q_2
$\Omega(@, m[\mathbf{0}], 1, \mathsf{tt})$
q_3
n^{is} n^{ho} a^{is} a^{ho} b^{is} b^{ho}
$e_1{:}b$ $e_2{:}n$ $e_3{:}n$ $e_4{:}b$ $e_5{:}b$ $e_6{:}b$ $e_7{:}b$
$move(\gamma, e_4, n^{\mathsf{ho}})$
q_4

Fig. 3. Left: Rearrangements of pointers performed by $act_{m[\mathbf{0}],@}(\gamma)$. Right: Rearrangements of pointers carried out by $move(\gamma, e_4, n^{\mathsf{ho}})$.

4.3 Configuration Link Manipulations

In our operational model, the computation of a process P corresponds to specific pointer manipulations mimicking the movements of P's ambients (see Figures 4). We will now introduce the functions implementing these pointer manipulations. They will be used in the rules of operational semantics given in Table 3.

State update for in/out. The function $\text{IOUp}(q, N.Q, e, \hat{e})$ in Table 2 performs the overall update of the state q when e moves inside $\hat{e}$ because of the execution of $N \in \{\mathsf{in}, \mathsf{out}\}$ and continue with Q. There are three kinds of updates to carry out during the execution of N: *(i)* First the pointer rearrangements moving e from its current location to the target location. These updates are performed by $move(\gamma, e, \hat{e})$. *(ii)* Then by applying act, IOUp carries out those rearrangements needed for the activation of the ambients becoming enabled in Q because of the execution of N. *(iii)* Finally, the set of capabilities $\mathsf{P}(e)$ is updated to record that e has executed N and that it must continue with Q. Figure 3 (right part) shows how the configuration changes when e_4 moves inside n^{ho}. In Figure 4, state q_1, $\mathsf{in}\, b$ moves e_3 inside b by making it point to b^{ho}; moreover msg becomes active and it points to $mail^{\mathsf{ho}}$ instead of msg^{is}. Figure 3 (left part) depicts the activation of m by the outer-most ambient @. It corresponds to $act_{m[\mathbf{0}],@}(\gamma)$.

State update for open. The function $\mathrm{OpenUp}(q, N.Q, e'{:}a, e)$ updates the state when $e'{:}a$ executes open of the ambient represented by e. $\mathrm{OpenUp}(q, N.Q, e'{:}a, e)$ performs the following operations: *(i)* It dissolves e using *diss*; *(ii)* It activates the ambients that become enabled; *(iii)* It updates the set of sub-processes that remain to be done by the entity executing open. Note that e' takes the processes $\mathsf{P}(e)$ which were supposed to be executed by e. See the transition between q_3 and q_4 in Figure 4.

4.4 The HABA Semantics of Processes

We can now define HABA $\mathcal{H}_P$ defining the abstract model for process P.

Definition 4.2. *The abstract semantics of a process P is the HABA $\mathcal{H}_P = \langle X_P, S, \rightarrow, I, \mathcal{F}\rangle$ where*

- $X_P = \{x_n \mid n \in n(P)\} \cup \{x_@\}$;
- $S \subseteq$ STATES *such that* $q_{pre}, q_{in} \in S$;
- *let* $\mathcal{R} \subseteq S \times (Ent \times Ent \rightarrow \mathbb{M}) \times S$ *be the smallest relation satisfying the rules in Table 3. Then* $\rightarrow = \mathcal{R} \cup \{(q_{pre}, \lambda_{pre}, q_{in})\} \cup \{(q, id, q) \mid \neg\exists q', \lambda : (q, \lambda, q') \in \mathcal{R}\}$;
- $\mathrm{dom}(I) = \{q_{pre}\}$ *and* $I(q_{pre}) = \langle \varnothing, \vartheta\rangle$ *where* $\vartheta(x_n) = n^{\mathsf{is}}$ $(n \in n(P))$.
- $\mathcal{F} = \{\{q \in S \mid (\exists q', \lambda : q \rightarrow_\lambda q') \Rightarrow q = q'\}\}$.

X_P contains a logical variable for each ambient name in P and $x_@$ for the outermost ambient. The transition relation $\rightarrow$ includes a transition from the pre-initial state to the initial state and an "artificial" self-loop for each deadlocked state in $\mathcal{R}$. $\mathcal{F}$ is defined as the set of states whose only outgoing transition is a self-loop. The set I contains only the pre-initial state. The interpretation ϑ allows us to refer to ambient names in *NTL*-formulae (see discussion at page 409).

Operational rules. The execution of a capability $N.Q$, in a given state q, applies the following pattern: γ_q is first modified with the needed link rearrangements into γ'. This is performed by IOUp (for in and out) or OpenUp (for open). Because of the rearrangements of the links, γ' may be not canonical. Therefore, we consider its *safe expansion* $\mathsf{SExp}(\gamma')$[4] and for each of its element γ'' we take the canonical form $\mathsf{cf}(\gamma'')$. The reallocation is defined as: $\lambda = h_{\mathsf{cf}} \circ h^{-1}(\gamma')$ where the morphism h is determined by the safe expansion of γ' and h_{cf} is the morphism giving the canonical form of $h^{-1}(\gamma')$. [12] shows that this is a good definition of reallocation. In q only concrete non-fixed entities modelling an active ambient and directly pointing to a host can move, i.e., $E^{\mathsf{m}} = \{e \in E^{\mathsf{c}}_q \mid \mathsf{A}(e) \text{ is active}, \ \mu_q(e) \in E^{\mathsf{ho}}_P\}$. In the rules $E^{\mathsf{m}}_@ = E^{\mathsf{m}} \cup \{@^{\mathsf{is}}\}$. Moreover, *siblings*$(e)$ is the set of ambients having an instance with the same parent of e. *child*(a) returns the entities that are children of the ambient a. *parents*(b) is the set of parents of ambients b. In the **In rule**, if e has $\mathsf{in}\ b.Q$

[4] The safe expansion of a (possibly unsafe) configuration γ' is a finite set of L-safe configurations γ'' which are related to γ' by a morphism (i.e., they represent the same topological structure). Formally: $\mathsf{SExp}(\gamma') = \{\gamma'' \mid \gamma'' \text{ is } L\text{-safe and } h : \gamma'' \rightarrow \gamma'\}$. See [12] for an exhaustive treatment.

Table 3. Operational rules for Mobile ambients

In	$\dfrac{e \in E^{\mathsf{m}}, \quad \mathsf{in}\, b.Q \in \mathsf{P}_q(e), \quad b \in \mathit{siblings}(e)}{q \rightarrow_{\lambda} \mathsf{cf}(\gamma''), \mathsf{P}'}$
	where $\langle \gamma', \mathsf{P}' \rangle = \mathrm{IOUp}(q, \mathsf{in}\, b.Q, e, b^{\mathsf{ho}})$ and $\gamma'' \in \mathsf{SExp}(\gamma')$
Out	$\dfrac{e \in E^{\mathsf{m}}, \quad \mathsf{out}\, b.Q \in \mathsf{P}(e), \quad \mu(e) = b^{\mathsf{ho}} \quad a \in \mathit{parents}(b)}{q \rightarrow_{\lambda} \mathsf{cf}(\gamma''), \mathsf{P}'}$
	where $\langle \gamma', \mathsf{P}' \rangle = \mathrm{IOUp}(q, \mathsf{out}\, b.Q, e, a^{\mathsf{ho}})$ and $\gamma'' \in \mathsf{SExp}(\gamma')$
Open	$\dfrac{e \in E^{\mathsf{m}}_{@}, \quad \mathsf{open}\, b.Q \in \mathsf{P}(e), \quad e'{:}b \in \mathit{child}(\mathsf{A}(e))}{q \rightarrow_{\lambda} \mathsf{cf}(\gamma''), \mathsf{P}'}$
	where $\langle \gamma', \mathsf{P}' \rangle = \mathrm{OpenUp}(q, \mathsf{open}\, b.Q, e, e')$ and $\gamma'' \in \mathsf{SExp}(\gamma')$
Bang	$\dfrac{e \in E^{\mathsf{m}}_{@}, \quad !Q \in \mathsf{P}(e)}{q \rightarrow_{\lambda} \mathsf{cf}(\gamma'), \mathsf{P}'}$ where $\mathsf{P}' = \mathsf{P}_q[e \mapsto \mathsf{P}_q(e) \cup \rho(Q)]$ and $\gamma' \in \mathsf{SExp}(\mathit{act}_{Q,\mathsf{A}(e)}(\gamma_q))$

and there exists a sibling ambient b then e moves inside b. In the **Out rule**, if e executes $\mathsf{out}\, b.Q$ and its father is ambient b, i.e. $\mu(e) = b^{\mathsf{ho}}$ then e must move in every ambient containing a copy of b. In the **Open rule**, e can execute $\mathsf{open}\, b$, if there exists a $\mathit{child}(\mathsf{A}(e))$ e' modelling a copy of b. Entity e' is dissolved and the component $\mathsf{P}(e)$ acquires the processes contained in $\mathsf{P}(e')$. In the **Bang rule**, if a process $!Q$ is contained in the set of processes that e must execute, then $!Q$ is expanded using the equivalence $!Q \equiv Q|!Q$. Note that we do not need structural rules for parallel composition, restriction, ambients since those constructs are implicitly represented in the configuration of a state.

Example 6. The HABA modelling SYS_1 of Example 1 is depicted in Figure 4. For SYS_1 we want to check the secrecy property (UA) *"no untrusted ambients can access D"* expressed by the *NTL*-formula $\varPhi_{\mathsf{UA}}$ in Example 4. No runs of the HABA satisfies $\varPhi_{\mathsf{UA}}$ therefore in SYS_1 only b can access the secret data D.

Theorem 1. *If $P \rightarrow Q$ then there exists Q' and a finite sequence of $\lambda_1, \lambda_2, \ldots, \lambda_k$ such that $\mathcal{D}(P) \rightarrow_{\lambda_1} \cdots \rightarrow_{\lambda_k} \mathcal{D}(Q')$ and $Q' \equiv Q$.*

This theorem ensures that the HABA semantics of a process P provides a safe approximation of all P behaviours. Although for many processes it provides rather precise information, some limitations occur on processes which combine name restriction and replication. Like other analyses based on static analysis [8,13,14,18], our semantics does not distinguish between processes $!(\nu n)P$ and $(\nu n)!P$. However, our model is able to capture precise information on the number of copies of the same ambients that may be inside another ambient. Therefore it is able to distinguish between P and $!P$. The precision can easily be increased by increasing the parameters L and M.

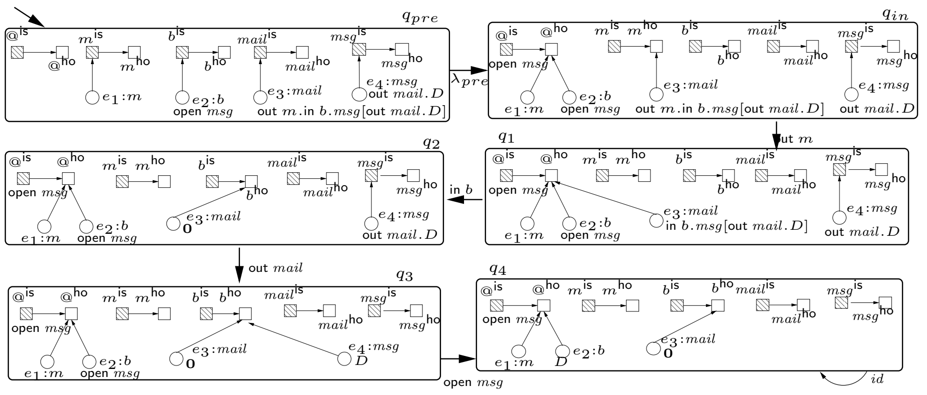

Fig. 4. HABA modelling the system described in Example 1 by the process $SYS_1 = m[mail[\mathsf{out}\ m.\mathsf{in}\ b.msg[\mathsf{out}\ mail.D]]]|b[\mathsf{open}\ msg]|\mathsf{open}\ msg$

5 Conclusions

The analysis we have developed in this paper represents an alternative approach which goes beyond the analyses for MA found in the literature. A strong point of our technique seems to rely on its power of counting occurrences of ambients, as well as its flexibility on tuning the precision. Another advantage of our approach w.r.t. static analysis is that the model can be used to investigate properties of the evolution of the computation (via *NTL*). Beside the information “which ambient end up in in which other ambient” our model is able to answer other involved questions which cannot be answered by existing analyses. For example, properties like “it is always the case that whenever a and b are inside n, a exit n before b”. Or, ”a copy of a does not leave b until another copy of a enters b”.

Acknowledgments. I would like to thank Arend Rensink, Joost-Pieter Katoen, Ivana Mijajlović, Josh Berdine, Marco Carbone, Giuseppe Milicia, and Sergio Maffeis for fruitful discussions and helpful suggestions on different stages of this paper. Thanks also to the anonymous referees for detailed comments.

References

1. C. Braghin, A. Cortesi, and R. Focardi. Control flow analysis of mobile ambients with security boundaries. In B. Jacobs and A. Rensink, editors, *FMOODS 2002*, pp. 197–212. Kluwer, 2002.
2. L. Cardelli and A.D. Gordon. Mobile ambients. In *FOSSACS ’98*, *LNCS* 1378 pp. 140–155. Springer, 1998.
3. L. Cardelli and A.D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *POLP 2000*, pp. 365–377. ACM Press, 2000.
4. L. Caires. Behavioral and spatial observations in a logic for π-calculus In I. Walukiewicz, editor *FOSSACS 2004*, *LNCS* 2987, pp. 72–89, Springer 2004.
5. L. Caires and L. Cardelli. A spatial logic for concurrency (part I). *Inf. and Comp.*, 186(2):194–235, 2003.

6. W. Charatonik, A.D. Gordon, and J.-M. Talbot. Finite-control mobile ambients. In D. Le Métayer, editor, *ESOP 2002*, *LNCS* 2305, pp. 295–313. Springer, 2002.
7. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixed points. In *POPL77*, pp. 238–252. ACM, 1977.
8. P. Degano, F. Levi, and C. Bodei. Safe ambients: Control flow analysis and security. In *Asian Computing Science Conference*, *LNCS* 1961, pp. 199–214. Springer, 2000.
9. D. Distefano. On model checking the dynamics of object-based software: a foundational approach. PhD thesis, University of Twente, Nov 2003.
10. D. Distefano. A parametric model for the analysis of mobile ambients. Full version with proofs.
11. D. Distefano, J.-P. Katoen, and A. Rensink. Who is pointing when to whom? on the automated verification of linked list structures. In K. Lodaya, M. Mahajan editors: *FSTTCS 2004*, *LNCS* 3328, pp. 250–262, Springer 2004.
12. D. Distefano, A. Rensink, and J.-P. Katoen. Who is pointing when to whom: on model-checking pointer structures. Tech. Report TR-CTIT-03-12, University of Twente, 2003.
13. R.R. Hansen, J.G. Jensen, F. Nielson, and H.R. Nielson. Abstract interpretation of mobile ambients. In A.Cortesi and G. Filé, editors, *SAS '99*, *LNCS* 1694, pp. 135–148. Springer, 1999.
14. F. Levi and S. Maffeis. An abstract interpretation framework for analysing mobile ambients. In P. Cousot, editor, *SAS 2001*, *LNCS* 2126, pp. 395–411. Springer, 2001.
15. F. Levi and D. Sangiorgi. Controlling interference in ambients. In *POPL 2000*, pp. 352–364. ACM Press, 2000.
16. O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *POPL'85*, pp. 97–107. ACM, 1985.
17. U. Montanari and M. Pistore. An introduction to history-dependent automata. In A. Gordon, A. Pitts, and C. Talcott, editors, *HOOTS II*, vol.$\tilde{1}$0 of *ENTCS*. Elsevier Science Publishers, 1997.
18. F. Nielson, H.R. Nielson, R.R. Hansen, and J.G. Jensen. Validating farewalls in mobile ambients. In J.C.M. Baeten and S. Mauw, editors, *CONCUR '99*, *LNCS* 1664, pp. 463–477. Springer, 1999.
19. A. Pnueli. The temporal logic of programs. In *Proceedings FOCS-77*, pp. 46–57. IEEE Computer Society Press, 1977.
20. M. Sagiv, T. Reps, and R. Wilhelm. Solving shape-analysis problems in languages with destructive updating. *TOPLAS*, 20(1):1–50, 1998.