An Abstract Interpretation Framework for Analysing Mobile Ambients

Francesca Levi and Sergio Maffeis

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

Abstract We introduce an abstract interpretation framework for Mobile Ambients, based on a new fixed-point semantics. Then, we derive within this setting two analyses computing a safe approximation of a property about the run-time topological structure of processes which is relevant to security.

1 Introduction

Mobile Ambients (MA) [5] 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. MA is based on the notion of *ambient*. An ambient is a bounded place, where multi-threaded computation takes place; roughly speaking, 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 tree, which can be dynamically modified, according to three basic capabilities: $\mathtt{in}\, n$ allows an ambient to enter into an ambient n ($m[\mathtt{in}\, n.\, P_1 \mid P_2] \mid n[Q] \mapsto n[m[P_1 \mid P_2] \mid Q]$); $\mathtt{out}\, n$ allows an ambient to exit from an ambient n ($n[m[\mathtt{out}\, n.\, P_1 \mid P_2] \mid Q] \mapsto m[P_1 \mid P_2] \mid n[Q]$); $\mathtt{open}\, n$ allows to destroy the boundary of an ambient n ($\mathtt{open}\, n.\, P \mid n[Q] \mapsto P \mid Q$).

Several static techniques, such as Type Systems [10, 3, 2, 4, 8, 1] and Control Flow Analysis (CFA) [12, 13, 14, 9], have been devised to study and establish various security properties of MA, based on notions of classification and information flow. Although these approaches are strictly related and compute safe approximations of similar properties concerning the run-time topological structure of processes, their formulation as "ad hoc" syntax-directed systems makes it difficult to formally compare them.

We follow here the approach to program analysis of *abstract interpretation* [7, 6]. The main idea of abstract interpretation is that program analyses effectively compute an approximation of the program semantics so that the specification of program analyses should be formally derivable from the specification of the semantics. The typical approach suggested by abstract interpretation consists of: replacing the concrete domain of computation with an abstract domain modelling the property we are interested in; establishing a relation between the concrete and the abstract domain which formalizes safeness and precision of approximations; deriving, in a systematic way, an approximate safe semantics over the abstract domain. We refer the reader to Appendix A for more details on the abstract interpretation theory. One of the most important and critical steps for applying abstract interpretation consists in the choice of the concrete semantics one should start from. The standard reduction semantics of MA [5] is not adequate to abstraction, because it heavily relies on the syntax and uses structural rules and structural congruence to bring the participants of a potential reaction into contiguous positions. In order to overcome the above problems, we propose a new semantics for MA (equivalent to the standard reduction semantics) based on the simple observation that an MA process is essentially a tree of ambients, each one containing a set of active processes controlling its movements. Then we obtain, by stepwise approximation of the semantics, two analyses computing a safe approximation of the following (run-time) information: for any ambient which ambients and capabilities may be contained inside. This information has been fruitfully used to establish security properties, such as secrecy [9], and to validate a firewall protocol [12].

Because of space limitations we omit the presentation of the standard reduction semantics of MA [5] and the comparison with our normal semantics which can be found in the extended version of the paper [11]). The normal semantics is presented in Section 3, and the two derived abstractions in Sections 4 and 5.

2 Mobile Ambients

We enhance the standard syntax for the *Mobile Ambients* calculus without communication, adding labels to capabilities, restrictions and ambients, and partitioning the set of names.

Let $\mathcal{L}$ be a set of labels (ranged over by $\ell, \ell', \ldots$) and let $\mathcal{L}_I = \{\ell_i \mid \ell \in \mathcal{L}, i \in I\}$ be the corresponding set of indexed labels (ranged over by $\lambda, \mu, \gamma, \ldots$). Let $\mathcal{N}$ (ranged over by $n, m, h, k, \ldots$) and let $\widehat{\mathcal{N}}$ (ranged over by $\widehat{n}, \widehat{m}, \widehat{h}, \widehat{k}, \ldots$) be sets of *names*, such that $\mathcal{N} \cap \widehat{\mathcal{N}} = \emptyset$. Moreover, let $\widehat{\mathcal{N}}_I = \{\hat{n}_i \mid \hat{n} \in \widehat{\mathcal{N}}, i \in I\}$ be the set of indexed names derived from $\widehat{\mathcal{N}}$. To have a more compact notation we may use $n, m, h, k, \ldots$ also for generic elements of $\widehat{\mathcal{N}}_I$, when the meaning is clear from the context.

Definition 2.1 (Processes). Processes over indexed labels $\mathcal{L}_I$ and names $\mathcal{N} \cup \widehat{\mathcal{N}}_I$ are built according to the following syntax:

M,N::=		(capabilities)	P,Q::=		(processes)
	$\mathtt{in}\, n$	enter n		0	inactivity
	$\mathtt{out}\, n$	exit n		$(\nu n_\lambda)\ P$	restriction
	$\mathtt{open}\, n$	open n		$P \mid Q$	parallel composition
				$!P$	replication
				$n_\lambda[P]$	ambient
				$M_\lambda.\, P$	prefix

Standard syntactical conventions are used: trailing zeros are 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 as $n(P)$, $fn(P)$, $bn(P)$.

We present some basic concepts, which are necessary to define the semantics and the abstractions. A *substitution* of names and indexed names is a function $\eta : \mathcal{N} \cup \widehat{\mathcal{N}}_I \to \mathcal{N} \cup \widehat{\mathcal{N}}_I$. Standard notation $P[m/n]$ and $P\eta$ is used for the application of a substitution to a process. As far as it concerns indexed labels, we denote by $\Lambda(P)$ the set of labels occurring in P. A *renaming* of indexed labels is a function $\rho : \mathcal{L}_I \to \mathcal{L}_I$. Standard notation $P[\lambda/\mu]$ and $P\rho$ is used for the application of a renaming to a process.

Indexed names $\widehat{\mathcal{N}}_I$ are used to replace bound names with fresh names. To that purpose we assume two injective functions $H_{\mathcal{L}} : \mathcal{L} \to \widehat{\mathcal{N}}$ and $H_{\mathcal{L}_I} : \mathcal{L}_I \to \widehat{\mathcal{N}}_I$ such that $H_{\mathcal{L}_I}(\ell_i) = \hat{n}_i$ if $H_{\mathcal{L}}(\ell) = \hat{n}$. We also adopt a notion of equivalence over processes. We define the renaming and substitution of indexes, induced by an injective function $\iota : I \to I$, as $\rho_\iota : \mathcal{L}_I \to \mathcal{L}_I$ and $\eta_\iota : \widehat{\mathcal{N}}_I \to \widehat{\mathcal{N}}_I$, such that $\rho_\iota(\ell_i) = \ell_{\iota(i)}$ and $\eta_\iota(\hat{n}_i) = \hat{n}_{\iota(i)}$. We say that P and Q are *equivalent up to renaming and substitution of indexes* ($P \sim Q$) if $P\rho_\iota\eta_\iota = Q$, for a renaming ρ_ι and a substitution η_ι induced by some ι.

Moreover, we say that a process P is *active* if $P = M_\lambda.\,Q$ or $P = !Q$. In the following we use $\mathcal{P}$ and $\mathcal{AP}$ respectively for the set of processes and for the subset of active processes. Furthermore, we use $\mathcal{A}$ (ranged over by a,b,c,...) for the set of labelled names n_λ, such that $n \in \mathcal{N} \cup \widehat{\mathcal{N}}_I$ and $\lambda \in \mathcal{L}_I$, augmented with a distinct symbol @.

Remark 2.2. It is worth mentioning that the idea of annotating processes is typical of Flow Logic [15], where labels are introduced to keep distinct multiple occurrences of the same object and to handle α-conversion. In our approach indexed labels and names are similarly used both in the normal semantics and in the abstract interpretation. In particular, indexes are meaningless as far as it concerns the normal semantics and the second abstraction. Instead, they are necessary to define the first abstraction (see Example 4.9 and Example 4.10 of Section 4).

3 The Normal Semantics

The normal semantics is based on the intuitive interpretation of a process as a tree of ambients, each one containing a set of active processes. We use a set, called a *topology*, to represent the tree of ambients, and a set, called a *configuration*, to represent the active processes contained in each ambient.

For instance a process $a[\mathtt{in}\,k_\mu.\,\mathtt{in}\,m_\gamma \mid \mathtt{out}\,m_\nu.\,d[0] \mid b[0]] \mid c[!\mathtt{in}\,m_\lambda]$ (where $a, b, c, d \in \mathcal{A}$ are labelled ambients) is represented by

$$(\{\ {}^{@}_{a},\ {}^{@}_{c},\ {}^{a}_{b}\}, \{\ {}^{a}\mathtt{in}\,k_\mu.\,\mathtt{in}\,m_\gamma,\ {}^{a}\mathtt{out}\,m_\nu.\,d[0],\ {}^{c}!\mathtt{in}\,m_\lambda\}).$$

The topology contains ${}^{a}_{b}$ because b is a son of a, and contains ${}^{@}_{a}$ and ${}^{@}_{c}$ because a and c are sons the outermost ambient @. The configuration contains

${}^{c}!\mathtt{in}\, m_\lambda$ because the active process $!\mathtt{in}\, m_\lambda$ is executable inside c. Moreover, it contains ${}^{a}\mathtt{in}\, k_\mu.\, \mathtt{in}\, m_\gamma$ and ${}^{a}\mathtt{out}\, m_\nu.\, d[0]$ because both the active processes $\mathtt{in}\, k_\mu.\, \mathtt{in}\, m_\gamma$ and $\mathtt{out}\, m_\nu.\, d[0]$ are executable inside a.

In this representation restriction is handled in a particular way. For instance a process $(\nu n_\lambda)\ (n_\gamma[\mathtt{in}\, m_\mu])$ is represented by $(\{\ {}_{(\hat{n}_i)_\gamma}{}^{@}\}, \{\ {}^{(\hat{n}_i)_\gamma}\mathtt{in}\, m_\mu\})$, where $\hat{n}_i \in \widehat{\mathcal{N}}_I$ is the fresh name provided by the indexed label attached to restriction, namely $H_{\mathcal{L}_I}(\lambda) = \hat{n}_i$.

Notice that, since in this representation the constructs of restriction, parallel composition and ambient are implicitly represented, the standard structural rules and structural congruence (including α-conversion) of the reduction semantics are no longer necessary.

States. Formally, a state is a pair which consists of a topology and a configuration: the topology is a standard representation of a tree via a set of pairs (son, father) and the configuration is simply a set of pairs associating each active process to its enclosing ambient.

Definition 3.1 (States). A *state* S is a pair (T, C) where

1. $T \in \wp(\mathcal{A} \times \mathcal{A})$ such that, if $(a, b), (a, c) \in T$ then $b = c$ (*topology*);
2. $C \in \wp(\mathcal{A} \times \mathcal{AP})$ (*configuration*).

We extend to states in the obvious way the notions of labels, renaming, substitution, equivalence up to renaming and substitution of indexes $\sim$. Moreover, we use $\mathcal{S}$ for the set of states.

Normalization function. In order to derive the initial state from a process and to handle the processes, which became executable after a step, we introduce a normalization function $\delta : (\mathcal{A} \times \mathcal{P}) \to \mathcal{S}$. Intuitively, $\delta(a, P)$ gives the state representing process P assuming that P is contained in ambient a. Thus, the initial state corresponding to a process P is $\delta(@, P)$.

DRes	$\delta\ {}^{a}(\nu n_\lambda)\ P$	$=$	$\delta\ {}^{a}P[H_{\mathcal{L}_I}(\lambda)/n]$
DAmb	$\delta\ {}^{a}b[P]$	$=$	$\delta\ {}^{b}P\ \cup\ (\{\ {}_{b}{}^{a}\}, \emptyset)$
DZero	$\delta\ {}^{a}0$	$=$	$(\emptyset,\ \emptyset)$
DPar	$\delta\ {}^{a}P \mid Q$	$=$	$\delta\ {}^{a}P\ \cup\ \delta\ {}^{a}Q$
DBang	$\delta\ {}^{a}!P$	$=$	$(\emptyset, \{{}^{a}!P\})$
DPref	$\delta\ {}^{a}N_\lambda.\, P$	$=$	$(\emptyset, \{{}^{a}N_\lambda.\, P\})$

Table 1. The normalization function δ

The normalization function δ is shown in Tab. 1 (where $\delta\ {}^{a}P$ stands for $\delta(a, P)$). Rule **DRes** eliminates the restriction by replacing the bound name

n with the fresh name $H_{\mathcal{L}_I}(\lambda)$ provided by the indexed label λ. Rule **DAmb** adds ambient b to the topology as son of the enclosing ambient a and normalises the process contained in b. Rule **DPar** gathers the processes and the topologies built in each of its two branches. The last rules simply add the active process to the configuration.

Transitions. Transitions are obtained by the rules of Tab. 2 which realize the unfolding of recursion, the movement in and out of ambients, and the opening of ambients. The rules use the normalization function to handle the continuations. Moreover, they use a function $new : \mathcal{P} \to \mathcal{P}$, such that $new(P) = Q$, where $Q = P\rho_\iota$ for some fresh renaming of indexes ρ_ι. In the following, $T[\ {}_b{}^a/\ {}_d{}^c]$ denotes the replacement in the topology of every pair ${}_d{}^c$ with ${}_b{}^a$. Analogously for configurations.

$$\textbf{Bang}\ \frac{{}^a!P \in C}{(T,C) \mapsto \delta\ {}^a new(P)\ \cup\ (T,C)}$$

$$\textbf{In}\ \frac{c =\ {}^a\mathtt{in}\, m_\gamma.\, P \in C \qquad {}_a{}^b,\ {}_{m_\mu}{}^b \in T \qquad a \neq m_\mu}{(T,C) \mapsto \delta\ {}^a P\ \cup\ ((T \setminus \{\ {}_a{}^b\}) \cup \{\ {}_a{}^{m_\mu}\}, C \setminus \{c\})}$$

$$\textbf{Out}\ \frac{c =\ {}^a\mathtt{out}\, m_\gamma.\, P \in C \qquad {}_a{}^{m_\mu},\ {}_{m_\mu}{}^b \in T \qquad a \neq m_\mu}{(T,C) \mapsto \delta\ {}^a P\ \cup\ ((T \setminus \{\ {}_a{}^{m_\mu})\} \cup \{\ {}_a{}^b\}, C \setminus \{c\})}$$

$$\textbf{Open}\ \frac{c =\ {}^a\mathtt{open}\, m_\gamma.\, P \in C \qquad {}_{m_\mu}{}^a \in T \qquad a \neq m_\mu}{(T,C) \mapsto \delta\ {}^a P\ \cup\ ((T \setminus \{\ {}_{m_\mu}{}^a\})[\ {}_b{}^a/\ {}_b{}^{m_\mu}], (C \setminus \{c\})[{}^a Q_\beta/{}^{m_\mu} Q_\beta])}$$

Table 2. Transitions $\mapsto$

Rule **Bang** creates a fresh copy (equivalent up to renaming of indexes) of the process under replication. The last three rules correspond to the usual reduction rules of movements and opening (shown in the Introduction). Rule **In** verifies that the capability $\mathtt{in}\, m$ is enabled by checking if there exists a parallel ambient named m. Then, it modifies both the topology and the configuration accordingly: (i) it updates the father of a, (ii) it removes the executed capability and adds its (normalized) continuation. **Out** acts in an analogous way. Rule **Open** also extends the state by adding the set of processes and subambients acquired by a from the destroyed ambient m.

The collecting semantics. We can now define the core of the abstract interpretation framework, the *collecting semantics*. In order to have a correct semantics we have to consider only processes and states which satisfy some restrictions on indexed names and labels ensuring that multiple occurrences of the same process or of ambients with the same name are keep distinct.

$\leftarrow$ *Ho spostato l'esempio alla fine per uniformita'*

Definition 3.2 (well-labelled). A process P is *well-labelled* if: (i) for each $\hat{n}_i \in n(P)$, such that $\hat{n}_i = H_{\mathcal{L}_I}(\ell_i)$, $\ell_i \notin \Lambda(P)$; (ii) the (indexed) labels used in capabilities, ambients and restrictions are distinct one from each other. A state $S = (T, C)$ is *well-labelled* if condition (i) holds (for P replaced by S) and for each ${}^{a}P \in C$, P is well-labelled.

In the following we assume that both $\mathcal{P}$ and $\mathcal{S}$ range only over well-labelled processes and states.

The domain is the power-set of well-labelled states up to renaming and substitution of indexes. We use $[S]$ for equivalences classes of states with respect to $\sim$. We assume that $\subseteq$ and $\cup$ over states are defined component-wise.

Definition 3.3. The *concrete domain* is $\langle \wp(\mathcal{S}_{/\sim}), \subseteq \rangle$.

The concrete semantics is defined in a standard way as the least fixed-point of a continuous operator.

Definition 3.4 (Collecting Semantics). Let $X \in \wp(\mathcal{S}_{/\sim})$ and let $P \in \mathcal{P}$ a process. We define $\mathfrak{S}_{Coll}[\![P]\!] =$ lfp $\Psi([\delta\ {}^{@}P])$ where $\Psi(X) = \bigcup_{[S] \in X} \{[S'] \mid S \rightarrow S'\}$.

Examples. The following example better explain the reason to consider well-labelled processes only.

Example 3.5. Labels have to be distinct to guarantee that the translation function δ does not lose different instances of the same process or of ambients with the same name. For instance, for $P = \mathtt{in}\, m_\lambda \mid \mathtt{in}\, m_\lambda$ we would have $\delta(\ {}^{@}P) = (\emptyset,\ {}^{@}\mathtt{in}\, m_\lambda)$. The remaining condition concerning the free names of $\widehat{\mathcal{N}}_I$ is necessary for a correct treatment of restrictions. For instance, for $Q = (\nu n_\lambda)\ (n_\gamma[0]) \mid (\hat{n}_i)_\mu[0]$, where $H_{\mathcal{L}_I}(\lambda) = \hat{n}_i$, we would have $\delta(\ {}^{@}Q) = (\{\ {}_{(\hat{n}_i)_\mu}{}^{@},\ {}_{(\hat{n}_i)_\gamma}{}^{@}\}, \emptyset)$. Even if the two ambients are keep distinct by labels μ and γ, this is not correct because they share the same name $\hat{n}_i$.

The following example stresses an important aspect concerning indexed labels and names: replication produces processes equivalent up to renaming and substitution of indexes.

Example 3.6. Consider the following process $Q = n_{\ell_1}[\mathtt{in}\, n_{\ell'_1}]$.
We have $\mathfrak{S}_{Coll}[\![!Q]\!] = X$, such that X is the minimal set of states (up to renaming and substitution of indexes) where

1. $(\emptyset,\ {}^{@}!Q) \in X$, and for each $(T, C) \in X$, ${}^{@}!Q \in C$;

2. for each $k \in N$ there exists $S = (T, C) \in X$ such that $\Lambda(S) \subseteq \bigcup_{j \in \{1,k\}} \{\ell_j, \ell'_j\}$ and $fn(S) = \{n\}$, and for each $j \in \{1, k\}$ either ${}_{n_{\ell_j}}^{@} \in T$ and ${}^{n_{\ell_j}}\mathtt{in}\, n_{\ell'_j} \in C$ or ${}_{n_{\ell_j}}^{n_{\ell_h}} \in T$, with $h \neq j$, and ${}^{n_{\ell_j}}\mathtt{in}\, n_{\ell'_j} \notin C$.

The idea is that every unfolding of recursion produces a new copy of ambient n labelled by ℓ_j for a fresh index j. Any ambient n_{ℓ_j} may enter inside any other n_{ℓ_h} provided that $h \neq j$.

4 A First Abstraction

We devise a first abstraction aimed at capturing the following property: for each ambient, which ambients and capabilities can be contained inside. We obtain a safe and computable approximation of the property above by combining two basic abstractions.

We begin by considering abstract states where the topology is a hierarchy of ambients, possibly not a tree, and where the configuration contains also the father of the enclosing ambient of any active process. For example, a state $S = (\{\ {}_{a}^{@},\ {}_{b}^{@}\},\ {}^{a}\mathtt{in}\, k_\mu.\, \mathtt{in}\, m_\gamma)$ is abstracted to $S^\diamond = (\{\ {}_{a}^{@},\ {}_{b}^{@}\},\ {}^{{}_{a}^{@}}\mathtt{in}\, k_\mu.\, \mathtt{in}\, m_\gamma)$. The abstract state is equivalent to the concrete one with a minor modification: every active process is annotated by a *partial topology*, that is by the father of the enclosing ambient. For instance, we have ${}^{{}_{a}^{@}}\mathtt{in}\, k_\mu.\, \mathtt{in}\, m_\gamma$, because $\mathtt{in}\, k_\mu.\, \mathtt{in}\, m_\gamma$ is an active process of ambient a, whenever a is contained in @.

To understand the relevance of the partial topology it is necessary to look at the abstraction of a set of states. Consider a state $S' = (\{\ {}_{a}^{b},\ {}_{b}^{@}\},\ {}^{a}\mathtt{in}\, m_\gamma)$ which is abstracted to $S'^\diamond = (\{\ {}_{a}^{b},\ {}_{b}^{@}\},\ {}^{{}_{a}^{b}}\mathtt{in}\, m_\gamma)$. The set $\{S, S'\}$ is abstracted to the unique abstract state which is the union:

$$(\{\ {}_{a}^{b},\ {}_{b}^{@},\ {}_{a}^{@}\}, \{\ {}^{{}_{a}^{b}}\mathtt{in}\, m_\gamma,\ \ {}^{{}_{a}^{@}}\mathtt{in}\, k_\mu.\, \mathtt{in}\, m_\gamma\}).$$

Obviously the obtained abstract topology is not a tree. For instance we have both ${}_{a}^{b}$ and ${}_{a}^{@}$. In this case the partial topology allows us to distinguish between the possibly multiple fathers of an ambient. For instance, it says that process $\mathtt{in}\, k_\mu.\, \mathtt{in}\, m_\gamma$ is executable when a is inside @, while $\mathtt{in}\, m_\gamma$ is executable when a is inside b.

This abstraction is of course not enough to achieve a computable semantics, in that we may have infinite processes equivalent up to renaming of indexes (see Example 3.6). Hence, we abstract also indexes by keeping only the following information concerning labels: whether there is *at most* one occurrence or *any number* of occurrences.

For example, consider the states

$$S = (\ {}_{n_\lambda}^{@},\ {}^{n_\lambda}\mathtt{open}\, m_\mu) \text{ and } S' = (\ {}_{n_\lambda}^{@}, \{\ {}^{n_\lambda}\mathtt{open}\, m_\mu,\ {}^{n_\lambda}\mathtt{open}\, m_\gamma\})$$

where $\lambda = \ell'_1$, $\mu = \ell_1$ and $\gamma = \ell_2$. The abstraction gives the following $S^\diamond$ and $S'^\diamond$

$$S^\diamond = (\, {}_{n_\lambda}{}^{@},\ \ {}_{n_\lambda}{}^{@}\,\mathtt{open}\, m_{\ell_1}) \text{ and } S'^\diamond = (\, {}_{n_\lambda}{}^{@},\ \ {}_{n_\lambda}{}^{@}\,\mathtt{open}\, m_{\ell_\omega}).$$

Capability $\mathtt{open}\, m$ in state S is abstracted to $\mathtt{open}\, m_{\ell_1}$, and the two copies of $\mathtt{open}\, m$ in state S' are abstracted $\mathtt{open}\, m_{\ell_\omega}$. The label ℓ_ω (with multiplicity ω) represents any number of occurrences of the corresponding object equivalent up to renaming of indexes, while a label ℓ_1 (with multiplicity one) represents at most one occurrence of the corresponding object.

Abstract domain. Abstract labels are $\mathcal{L}^\diamond = \{\ell_1, \ell_\omega \mid \ell \in \mathcal{L}\}$ (ranged over by $\lambda^\diamond, \mu^\diamond, \gamma^\diamond, \ldots$). Abstract names are $\mathcal{N} \cup \widehat{\mathcal{N}}$ (ranged over by $n^\diamond, m^\diamond, k^\diamond, h^\diamond, \ldots$). Analogously, we use $\mathcal{A}^\diamond$ (ranged over by $a^\diamond, b^\diamond, c^\diamond, \ldots$) for the set of abstract labelled names $n^\diamond{}_{\lambda^\diamond}$, augmented with the distinct symbol @. The relation between names and labels has to be modified accordingly. We define $H_{\mathcal{L}^\diamond} : \mathcal{L}^\diamond \to \widehat{\mathcal{N}}$ such that $H_{\mathcal{L}^\diamond}(\ell_1) = H_{\mathcal{L}^\diamond}(\ell_\omega) = H_{\mathcal{L}}(\ell)$.

Abstract processes are built according to the syntax of Def. 2.1 over names $\mathcal{N} \cup \widehat{\mathcal{N}}$ and labels $\mathcal{L}^\diamond$. In the following we use $\mathcal{P}^\diamond$ and $\mathcal{AP}^\diamond$ for the set of abstract processes and active abstract processes.

Definition 4.1 (Abstract States). An *abstract state* $S^\diamond$ is a pair $(T^\diamond, C^\diamond)$ where

1. $T^\diamond \in \wp(\mathcal{A}^\diamond \times \mathcal{A}^\diamond)$ (*abstract topology*);
2. $C^\diamond \in \wp((\mathcal{A}^\diamond \times \mathcal{A}^\diamond) \times \mathcal{AP}^\diamond)$ (*abstract configuration*).

All the previously defined notions on states and processes are adapted to abstract states and processes in the expected way. Only the notion of well-labelling requires some care.

Definition 4.2 (well-labelled). An abstract process $P^\diamond$ is *well-labelled* if : (i) for each $\hat{n} \in n(P^\diamond)$, such that $\hat{n} = H_{\mathcal{L}^\diamond}(\lambda^\diamond)$, $\lambda^\diamond \notin \Lambda(P^\diamond)$; (ii) $\ell_1 \in \Lambda(P^\diamond)$ implies $\ell_\omega \notin \Lambda(P^\diamond)$ (and vice-versa); (iii) there is at most one occurrence of any label ℓ_1. An abstract state $S^\diamond = (C^\diamond, T^\diamond)$ is *well-labelled* if conditions (i) and (ii) hold (with $P^\diamond$ replaced by $S^\diamond$), and for each ${}_{a}{}^{b} P^\diamond \in C^\diamond$, $P^\diamond$ is well-labelled.

In the following, we consider only well-labelled abstract states and processes denoted by $\mathcal{S}^\diamond$ and $\mathcal{P}^\diamond$, respectively.

We consider an ordering over abstract states which reflects the intuition that ℓ_1 is more precise than ℓ_ω. Therefore, let $\subseteq^\diamond$ be the minimal ordering over $\mathcal{S}^\diamond$, such that $S^\diamond \subseteq S'^\diamond$ implies $S^\diamond \subseteq^\diamond S'^\diamond$, and such that $S^\diamond \subseteq^\diamond S^\diamond[\ell_\omega/\ell_1]$. As usual we use $\cup^\diamond$ as the least upper bound w.r.t. $\subseteq^\diamond$. Notice that for $\lambda = \ell_1$ and $\gamma = \ell_\omega$

$$(\, {}_{n_\lambda}{}^{b^\diamond},\ \ {}_{n_\lambda}{}^{b^\diamond} P^\diamond) \cup^\diamond (\, {}_{n_\gamma}{}^{b^\diamond},\ \ {}_{n_\gamma}{}^{b^\diamond} P^\diamond) = (\, {}_{n_\gamma}{}^{b^\diamond},\ \ {}_{n_\gamma}{}^{b^\diamond} P^\diamond).$$

Definition 4.3. The *abstract domain* is $\langle \mathcal{S}^\diamond, \subseteq^\diamond \rangle$.

To simplify the notation in the following we may omit the over-script $-^\diamond$ for any syntactic category, when the meaning is clear from the context.

The Galois connection. We present now the relation between the concrete and the abstract domain establishing a standard Galois connection (see Appendix A). A state is abstracted, as explained at the beginning of the section, by introducing the partial topology in processes, by replacing indexed labels $\mathcal{L}_I$ by labels with multiplicity $\mathcal{L}^\diamond$ and by modifying names $\widehat{\mathcal{N}}_I$ accordingly. A set of states is abstracted to the abstract state containing the abstraction of each element.

Let $S \in \mathcal{S}$ be a state. We define $\rho^\diamond_S : \mathcal{L}_I \to \mathcal{L}^\diamond$ such that $\rho^\diamond_S(\ell_i) = \ell_\omega$, if $\ell_i, \ell_j \in \Lambda(S)$ with $i \neq j$, and $\rho^\diamond_S(\ell_i) = \ell_1$ otherwise. Moreover, we define $\eta^\diamond_S : \widehat{\mathcal{N}}_I \to \widehat{\mathcal{N}}$ such that $\eta^\diamond_S(\hat{n}_i) = \hat{n}$.

Definition 4.4. Let $X \in \wp(\mathcal{S}_{/\sim})$, $(T, C) \in \mathcal{S}$ and $S^\diamond \in \mathcal{S}^\diamond$. We define $\alpha : \wp(\mathcal{S}_{/\sim}) \to S^\diamond$ and $\gamma : S^\diamond \to \wp(\mathcal{S}_{/\sim})$ as follows

1. $\alpha((T, C)) = (T, C^\diamond)\rho^\diamond_S \eta^\diamond_S$, where $C^\diamond = \{ \ {}_a{}^b P \mid \ {}_a^b \in T,\ {}^a P \in C\}$.
2. $\alpha(X) = \bigcup^\diamond_{[S] \in X} \alpha([S])$, where $\alpha([S]) = \bigcup^\diamond_{S' \in [S]} \alpha(S')$;
3. $\gamma(S^\diamond) = \bigcup\{[S] \ \mid \ \alpha([S]) \subseteq^\diamond S^\diamond\}$.

Lemma 4.5 (Galois insertion). *The pair of functions* (α, γ) *forms a Galois insertion between* $\langle \wp(\mathcal{S}_{/\sim}), \subseteq \rangle$ *and* $\langle \mathcal{S}^\diamond, \subseteq^\diamond \rangle$.

Abstract semantics. The abstract normalization function $\delta^\diamond : (\mathcal{A}^\diamond \times \mathcal{A}^\diamond) \times \mathcal{P}^\diamond \to \mathcal{S}^\diamond$, shown in Tab 3, is the obvious adaptation of δ to the abstract domain, where $\cup^\diamond$ is used instead of $\cup$ to properly handle labels with multiplicity.

$$
\begin{array}{lcl}
\delta^\diamond \ \ {}_a{}^b(\nu n_\lambda)\, P & = & \delta^\diamond \ \ {}_a{}^b P[H_{\mathcal{L}^\diamond}(\lambda)/n] \\
\delta^\diamond \ \ {}_a{}^b c[P] & = & \delta^\diamond \ \ {}_c{}^a P \cup^\diamond (\{ \ {}_c^a\}, \emptyset) \\
\delta^\diamond \ \ {}_a{}^b 0 & = & (\emptyset,\ \emptyset) \\
\delta^\diamond \ \ {}_a{}^b P \mid Q & = & \delta^\diamond \ \ {}_a{}^b P \cup^\diamond \delta^\diamond \ \ {}_a{}^b Q \\
\delta^\diamond \ \ {}_a{}^b !P & = & (\emptyset, \{ \ \ {}_a{}^b !P\}) \\
\delta^\diamond \ \ {}_a{}^b M_\lambda.\, P & = & (\emptyset, \{ \ \ {}_a{}^b M_\lambda.\, P\})
\end{array}
$$

Table 3. The normalization function $\delta^\diamond$

The abstract transitions are obtained by the rules of Tab. 4. The rules use a function $new_\omega : \mathcal{P}^\diamond \to \mathcal{P}^\diamond$, such that $new_\omega(P) = P\rho$ for the renaming ρ, where $\rho(\ell_1) = \ell_\omega$ for any $\ell_1 \in \Lambda(P)$. We also use

$$
C \setminus^{\diamond} \{ \ {}_{a}^{b} M_{\lambda^{\diamond}}.P \} = \begin{cases} C \text{ if } \lambda^{\diamond} = \ell_{\omega} \\ \\ (C \setminus \ {}_{a}^{b} M_{\lambda^{\diamond}}.P) \text{ if } \lambda^{\diamond} = \ell_{1} \end{cases}
$$

The rules are similar to the ones of Tab. 2. We explain the relevant differences only. The rule for the unfolding of replication creates a copy of the replicated process where every label has multiplicity ω, instead of creating a fresh copy (equivalent up to renaming of indexes). Consider the rule corresponding to the execution of a capability $\mathtt{in}\, m$. First, it verifies that the capability is executable by looking if there exist an ambient named m, which is contained in the father b of a. Then, it extends both the topology and the configuration accordingly: (i) it adds the new father $m_{\mu^{\diamond}}$ to a, (ii) it adds both the normalized continuation and the parallel processes to the set of processes executable when a is contained in $m_{\mu^{\diamond}}$. The other rules are adapted in an analogous way. Notice that in the rule for opening also the processes executable in any ambient c, when this is contained in the destroyed ambient, have to be updated.

It is important to stress that both the partial topology and the multiplicity of labels are fruitfully exploited. Consider for instance the rule corresponding to the execution of a capability $\mathtt{in}\, m$. The father b of a, recorded in the partial topology, is used both to establish the enabling of $\mathtt{in}\, m$ and to find out which processes are executable inside a in parallel with $\mathtt{in}\, m.P$. The movement of an ambient named m inside itself is permitted only if the multiplicity of m is ω. Moreover, a capability $\mathtt{in}\, m$ with multiplicity one can be exercised only once, and so is not executable in the new father $m_{\mu^{\diamond}}$ of a. By contrast, a capability $\mathtt{in}\, m$ with multiplicity ω is still executable in the new father $m_{\mu^{\diamond}}$ of a, as the other parallel processes.

Definition 4.6 (The abstract collecting semantics). Let $S^{\diamond} \in \mathcal{S}^{\diamond}$ and let $P \in \mathcal{P}$. We define $\mathfrak{S}_{Coll^{\diamond}}[\![P]\!] =$ lfp $\Psi^{\diamond}(\{\alpha([\delta \ {}^{@}P])\ \})$ where $\Psi^{\diamond}(S^{\diamond}) = \bigcup^{\diamond} S'^{\diamond}$ such that $S^{\diamond} \mapsto^{\diamond} S'^{\diamond}$.

The abstract collecting semantics is a safe approximation of the concrete one and so is an upper approximation of the property we are interested in. Safeness is stated in classical abstract interpretation style (see Appendix A).

Theorem 4.7 (Safeness). *Let $S \in \mathcal{S}$ and $P \in \mathcal{P}$. We have $\alpha(\Psi(S)) \subseteq^{\diamond} \Psi^{\diamond}(\alpha(S))$ and $\alpha(\mathfrak{S}_{Coll}[\![P]\!]) \subseteq^{\diamond} \mathfrak{S}_{Coll^{\diamond}}[\![P]\!]$.*

Examples. We present some examples to summarize the more interesting aspects of the abstraction. The following example shows the advantages obtained by combining together multiplicity and partial topology.

$$\frac{{}_{a}{}^{b}!P \in C}{(T,C) \mapsto^{\diamond} \delta^{\diamond}\ {}_{a}{}^{b} new_{\omega}(P) \cup^{\diamond} (T,C)}$$

$$\frac{c = {}_{a}{}^{b}\mathtt{in}\, m_{\lambda^{\diamond}}.P \in C \qquad {}_{a}{}^{b},\ {}_{m_{\mu^{\diamond}}}{}^{b} \in T \qquad (a = m_{\ell'_1} \Rightarrow \ell'_1 \neq \mu^{\diamond})}{(T,C) \mapsto^{\diamond} \delta^{\diamond}\ {}_{a}{}^{m_{\mu^{\diamond}}} P \cup^{\diamond} (T \cup^{\diamond} \{ {}_{a}{}^{m_{\mu^{\diamond}}} \}, C \cup^{\diamond} (C \setminus^{\diamond} \{c\})[{}_{a}{}^{m_{\mu^{\diamond}}} Q / {}_{a}{}^{b} Q])}$$

$$\frac{c = {}_{a}{}^{m_{\mu^{\diamond}}}\mathtt{out}\, m_{\lambda^{\diamond}}.P \in C \qquad {}_{a}{}^{m_{\mu^{\diamond}}} \in T \qquad (a = m_{\ell'_1} \Rightarrow \ell'_1 \neq \mu^{\diamond})}{(T,C) \mapsto^{\diamond} \bigcup^{\diamond}_{\{b \mid {}_{m_{\mu^{\diamond}}}{}^{b} \in T\}} \left(\delta^{\diamond}\ {}_{a}{}^{b} P \cup^{\diamond} (T \cup^{\diamond} \{ {}_{a}{}^{b} \}, C \cup^{\diamond} (C \setminus^{\diamond} \{c\})[{}_{a}{}^{b} Q / {}_{a}{}^{m_{\mu^{\diamond}}} Q])\right)}$$

$$\frac{{}_{a}{}^{b}\mathtt{open}\, m_{\lambda^{\diamond}}.P \in C \qquad {}_{m_{\mu^{\diamond}}}{}^{a} \in T \qquad (a = m_{\ell'_1} \Rightarrow \ell'_1 \neq \mu^{\diamond})}{(T,C) \mapsto^{\diamond} \delta^{\diamond}\ {}_{a}{}^{b} P \cup^{\diamond} (T \cup^{\diamond} T[c^{a}/c^{m_{\mu}}], C \cup^{\diamond} C[{}_{a}{}^{b} Q / {}_{m_{\mu}}{}^{a} Q][{}_{c}{}^{a} Q' / {}_{c}{}^{m_{\mu}} Q'])}$$

Table 4. Abstract transitions $\mapsto^{\diamond}$

Example 4.8. Consider the process $P = n_{\lambda}[\mathtt{in}\, k_{\epsilon}.\, \mathtt{in}\, m_{\nu} \mid m_{\gamma}[0]] \mid k_{\mu}[\mathtt{open}\, n_{\beta}]$, where labels $\{\epsilon, \mu, \nu, \lambda, \beta, \gamma\}$ are distinct also up to renaming of indexes. We have, $\mathfrak{S}_{Coll}[\![P]\!] = \{S_0, S_1, S_2\}$ such that

$$S_0 = (\{ {}_{n_{\lambda}}{}^{@},\ {}_{m_{\gamma}}{}^{n_{\lambda}},\ {}_{k_{\mu}}{}^{@} \}, \{ {}^{n_{\lambda}}\mathtt{in}\, k_{\epsilon}.\, \mathtt{in}\, m_{\nu},\ {}^{k_{\mu}}\mathtt{open}\, n_{\beta} \})$$

$$S_1 = (\{ {}_{n_{\lambda}}{}^{k_{\mu}},\ {}_{m_{\gamma}}{}^{n_{\lambda}},\ {}_{k_{\mu}}{}^{@} \}, \{ {}^{n_{\lambda}}\mathtt{in}\, m_{\nu},\ {}^{k_{\mu}}\mathtt{open}\, n_{\beta} \})$$

$$S_2 = (\{ {}_{m_{\gamma}}{}^{k_{\mu}},\ {}_{k_{\mu}}{}^{@} \},\ {}^{k_{\mu}}\mathtt{in}\, m_{\nu}).$$

Notice that capability $\mathtt{in}\, m$ is not exercised inside n because, m is not a sibling of n when it is executable. Moreover, k acquires by opening n only ambient m and capability $\mathtt{in}\, m$. We have $\alpha(\{S_0, S_1, S_3\}) = (T^{\diamond}, C^{\diamond})$, where (to improve readability we use $\{\epsilon, \mu, \nu, \lambda, \beta, \gamma\}$ for the corresponding abstract labels with multiplicity one)

$$T^{\diamond} = \{ {}_{n_{\lambda}}{}^{@},\ {}_{m_{\gamma}}{}^{n_{\lambda}},\ {}_{k_{\mu}}{}^{@},\ {}_{n_{\lambda}}{}^{k_{\mu}},\ {}_{m_{\gamma}}{}^{k_{\mu}} \}$$

$$C^{\diamond} = \{ {}_{n_{\lambda}}{}^{@}\mathtt{in}\, k_{\epsilon}.\, \mathtt{in}\, m_{\nu},\ {}_{k_{\mu}}{}^{@}\mathtt{open}\, n_{\beta},\ {}_{k_{\mu}}{}^{@}\mathtt{in}\, m_{\nu},\ {}_{n_{\lambda}}{}^{k_{\mu}}\mathtt{in}\, m_{\nu} \}).$$

Let us compare the abstraction of the collecting semantics to the abstract collecting semantics. We have $\mathfrak{S}_{Coll^{\diamond}}[\![P]\!] = (T^{\diamond} \cup \{ {}_{n_{\lambda}}{}^{m_{\gamma}} \}, C^{\diamond})$. The abstract semantics is safe but not exact, i.e. $\alpha(\mathfrak{S}_{Coll}[\![P]\!]) \subset^{\diamond} \mathfrak{S}_{Coll^{\diamond}}[\![P]\!]$. The labels with multiplicity and the partial topology are essential to have no loss of information in the configuration. Since $\mathtt{in}\, k_{\epsilon}$ has multiplicity one, it is no longer executable after the movement of n in k. Therefore, $\mathtt{in}\, m_{\nu}$ is the only process which is contained in n when n is inside k, and which is acquired by k, when opening n.

The subtle point is that, in the abstract semantics, n moves inside m. In fact, we have that $\mathtt{in}\,m$ is executable, when n is inside k, and that k may contain ambient m. The abstract semantics cannot capture that k effectively contains ambient m only after the dissolution of n.

The following examples better explain the role of indexes in the abstraction. As we have discussed any labelling of a process P respecting the requirements of Definition 3.2 is enough to have a correct normal semantics of P. However, the choice of labels has drammatic consequences on the precision of the abstraction. Indeed, a typical schema to annotate processes is to keep all labels distinct also up to renaming of indexes so that only the copies produced by replication are identified by the abstraction.

Example 4.9. Consider the processes

$$P_1 = n_{\ell_1}[\mathtt{in}\,k_\mu] \mid n_{\ell_2}[\mathtt{in}\,m_\gamma] \mid m_\lambda[0]$$

$$P_2 = n_\beta[\mathtt{in}\,k_\mu] \mid n_\epsilon[\mathtt{in}\,m_\gamma] \mid m_\lambda[0]$$

where $\{\mu, \gamma, \lambda, \beta, \epsilon\}$ are distinct also up to renaming of indexes. We have

$$\mathfrak{S}_{Coll}[\![P_1]\!] = (\{\ {n_{\ell_1}}^{@},\ {n_{\ell_2}}^{@},\ {m_\lambda}^{@},\ {n_{\ell_2}}^{m_\lambda}\}, \{\ {}^{n_{\ell_1}}\mathtt{in}\,k_\mu,\ {}^{n_{\ell_2}}\mathtt{in}\,m_\gamma\})$$

$$\mathfrak{S}_{Coll}[\![P_2]\!] = (\{\ {n_\beta}^{@},\ {n_\epsilon}^{@},\ {m_\lambda}^{@},\ {n_\epsilon}^{m_\lambda}\}, \{\ {}^{n_\beta}\mathtt{in}\,k_\mu,\ {}^{n_\epsilon}\mathtt{in}\,m_\gamma\}).$$

The collecting semantics of P_1 and P_2 are equivalent up to renaming of labels. In the abstract semantics we have (for readability we use $\{\mu, \gamma, \lambda, \beta, \epsilon\}$ for the corresponding labels with multiplicity one)

$$\mathfrak{S}_{Coll^\diamond}[\![P_1]\!] = (\{\ {n_{\ell_\omega}}^{@},\ {m_\lambda}^{@},\ {n_{\ell_\omega}}^{m_\lambda}\}, \{\ {}^{{n_{\ell_\omega}}^{@}}\mathtt{in}\,k_\mu,\ {}^{{n_{\ell_\omega}}^{@}}\mathtt{in}\,m_\gamma\})$$

$$\mathfrak{S}_{Coll^\diamond}[\![P_2]\!] = (\{\ {n_\beta}^{@},\ {n_\epsilon}^{@},\ {m_\lambda}^{@},\ {n_\epsilon}^{m_\lambda}\}, \{\ {}^{{n_\beta}^{@}}\mathtt{in}\,k_\mu,\ {}^{{n_\epsilon}^{@}}\mathtt{in}\,m_\gamma\}).$$

Due to a different choice of labels the two copies of ambient n are identified by the abstract semantics of P_1, while they are keep distinct by the abstract semantics of P_2.

Example 4.10. Consider the process $Q = n_\lambda[\mathtt{in}\,n_\gamma]$ of Example 3.6, where $\lambda = \ell_1$ and $\gamma = \ell_1'$. We have for $\mu = \ell_\omega$ and $\epsilon = \ell_\omega'$,

$$\mathfrak{S}_{Coll^\diamond}[\![Q]\!] = (\ {n_\lambda}^{@},\ {}^{{n_\lambda}^{@}}\mathtt{in}\,n_\gamma)$$

$$\mathfrak{S}_{Coll^\diamond}[\![!Q]\!] = (\{\ {n_\mu}^{@},\ {n_\mu}^{n_\mu}\}, \{\ {}^{{n_\mu}^{@}}\mathtt{in}\,n_\epsilon,\ {}^{{n_\mu}^{n_\mu}}\mathtt{in}\,n_\epsilon,\ {}^{@}!Q\}).$$

The labels with multiplicity allows us to distinguish process $!Q$ from process Q. In the abstract semantics of Q the label of n is ℓ_1, which forbids the movement of n inside itself. Oppositely, in the abstract semantics of $!Q$ the unfolding of recursion produces a label ℓ_ω for n, which forces this movement. This result for $!Q$ is necessary to have a safe approximation of the concrete semantics, where the unfolding of replication produces multiple copies of n, which may interact with each other (see Example 3.6). In this example, the abstract semantics is exact, namely $\mathfrak{S}_{Coll^\diamond}[\![Q]\!] = \alpha(\mathfrak{S}_{Coll}[\![Q]\!])$ and $\mathfrak{S}_{Coll^\diamond}[\![!Q]\!] = \alpha(\mathfrak{S}_{Coll}[\![!Q]\!])$.

5 A Second Abstraction

On top of the previous abstraction, we define a new coarser abstraction, aimed at computing more efficiently a safe approximation of the same property. The abstraction is obtained by simply dropping multiplicity from labels and partial topologies from processes. For example, consider the states shown at the beginning of Section 4, where $\lambda = \ell_1'$, $\mu = \ell_1$ and $\gamma = \ell_2$:

$$S = (\ {}_{n_\lambda}{}^{@},\ {}^{n_\lambda}\mathtt{open}\, m_\mu) \text{ and } S' = (\ {}_{n_\lambda}{}^{@}, \{\ {}^{n_\lambda}\mathtt{open}\, m_\mu,\ {}^{n_\lambda}\mathtt{open}\, m_\gamma\}).$$

In the new abstraction $S^\diamond$ and $S'^\diamond$ are represented by the same abstract state $(\ {}_{n_{\ell'}}{}^{@},\ {}^{n_{\ell'}}\mathtt{open}\, m_\ell)$, because the ability to distinguish one occurrence from multiple occurrences has been lost.

Abstract domain. Abstract labels are $\mathcal{L}$ and abstract names are $\mathcal{N} \cup \widehat{\mathcal{N}}$. Thus, the relation between names and labels is given by function $H_{\mathcal{L}} : \mathcal{L} \to \widehat{\mathcal{N}}$ (see Section 2). We use $\mathcal{A}^\circ$ (ranged over by $a^\circ, b^\circ, c^\circ \ldots$) for the set of abstract labelled names n_ℓ, such that $n \in \mathcal{N} \cup \widehat{\mathcal{N}}$ and $\ell \in \mathcal{L}$, augmented with the distinct symbol @. Abstract processes are built according to the syntax of Def. 2.1 over names $\mathcal{N} \cup \widehat{\mathcal{N}}$ and labels $\mathcal{L}$. We use $\mathcal{P}^\circ$ and $\mathcal{AP}^\circ$ for the set of abstract and active abstract processes.

Definition 5.1 (Abstract States). An *abstract state* S° is a pair (T°, C°) where

1. $T^\circ \in \wp(\mathcal{A}^\circ \times \mathcal{A}^\circ)$ (*abstract configuration*);
2. $C^\circ \in \wp(\mathcal{A}^\circ \times \mathcal{AP}^\circ)$ (*abstract configuration*).

All the previously defined notions on states and processes are adapted to abstract states and processes in the expected way. We use $\mathcal{S}^\circ$ for the set of abstract states.

The abstract domain is given by abstract states ordered by inclusion (we assume $\subseteq$ and $\cup$ defined component-wise).

Definition 5.2. The *abstract domain* is $\langle \mathcal{S}^\circ, \subseteq \rangle$.

In the following we may omit the over-script $-^\circ$ for any syntactic category, when the meaning is clear from the context.

The Galois connection. We present now the relation between the abstract domain of Def 4.3 and the new abstract domain establishing, as before, a Galois connection (see appendix A).

An abstract state is abstracted, as explained at the beginning of the section, by dropping the multiplicity from labels and the partial topology from processes. To this purpose, we use a renaming $\rho^\circ : \mathcal{L}^\diamond \to \mathcal{L}$, such that $\rho^\circ(\ell_1) = \rho^\circ(\ell_\omega) = \ell$.

Definition 5.3. Let $(T^\diamond, C^\diamond), S^\diamond \in \mathcal{S}^\diamond$, and $S^\circ \in \mathcal{S}^\circ$. We define $\alpha^\circ : \mathcal{S}^\diamond \to \mathcal{S}^\circ$ and $\gamma^\circ : \mathcal{S}^\circ \to \mathcal{S}^\diamond$ as follows

1. $\alpha^\circ((T^\diamond, C^\diamond)) = (T^\diamond, \{\ {}^{a}P \mid \ {}_{a}^{b}P \in C^\diamond\})\rho^\circ$;
2. $\gamma^\circ(S^\circ) = \bigcup^\diamond \{S^\diamond \mid \alpha^\circ(S^\diamond) \subseteq S^\circ\}$.

Lemma 5.4 (Galois insertion). *The pair of functions $(\alpha^\circ, \gamma^\circ)$ forms a Galois insertion between $\langle \mathcal{S}^\diamond, \subseteq^\diamond \rangle$ and $\langle \mathcal{S}^\circ, \subseteq \rangle$.*

Abstract semantics.

The abstract normalization function $\delta^\circ : \mathcal{A}^\circ \times \mathcal{P}^\circ \to \mathcal{S}^\circ$ is equal to the concrete one (shown in Tab 1) with ℓ in place λ and $H_{\mathcal{L}}$ in place of $H_{\mathcal{L}_I}$. The abstract transitions are obtained by the rules of Tab. 5. The unfolding of replication simply creates a copy of the replicated process without modifying the labels. The rules for movements and opening are similar to the corresponding rules of the abstract semantics (see Tab. 4) restricted to multiplicity ω. There is only one difference: the topology is used instead of the partial topology. For instance, a capability $\mathtt{in}\, m$ is executed inside an ambient a, if a and an ambient named m have a common father in the topology.

$$\frac{{}^{a}!P \in C}{(T, C) \mapsto^\circ \delta^\circ\ {}^{a}P \ \cup\ (T, C)}$$

$$\frac{{}^{a}\mathtt{in}\, m_\ell . P \in C \qquad {}_{a}^{b},\ {}_{m_{\ell'}}^{b} \in T}{(T, C) \mapsto^\circ \delta^\circ\ {}^{a}P \ \cup\ (T \cup \{\ {}_{a}^{m_{\ell'}}\}, C)}$$

$$\frac{{}^{a}\mathtt{out}\, m_\ell . P \in C \qquad {}_{a}^{m_{\ell'}} \in T}{(T, C) \mapsto^\circ \delta^\circ\ {}^{a}P \ \cup\ (T \ \cup\ T[\ {}_{a}^{b} /\ {}_{m_{\ell'}}^{b}], C)}$$

$$\frac{{}^{a}\mathtt{open}\, m_\ell . P \in C \qquad {}_{m_{\ell'}}^{a} \in T}{(T, C) \mapsto^\circ \delta^\circ\ {}^{a}P \ \cup\ (T \ \cup\ T[\ {}_{b}^{a} /\ {}_{b}^{m_{\ell'}}], C \ \cup\ C[\ {}^{a}Q /\ {}^{m_{\ell'}}Q])}$$

Table 5. Abstract transitions $\mapsto^\circ$

Definition 5.5 (The abstract collecting semantics). Let $S^\circ \in \mathcal{S}^\circ$ and let $P \in \mathcal{P}$. We define $\mathfrak{S}_{Coll^\circ}[\![P]\!] = \operatorname{lfp} \Psi^\circ(\{\alpha^\circ(\alpha([\delta\ {}^{@}P]))\ \})$ where $\Psi^\circ(S^\circ) = \bigcup S'^\circ$ such that $S^\circ \mapsto^\circ S'^\circ$.

The new abstract collecting semantics is a safe approximation of the abstract collecting semantics of Def. 4.6.

Theorem 5.6 (Safeness). *Let $S^\diamond \in \mathcal{S}^\diamond$ and $P \in \mathcal{P}$. We have $\alpha^\circ(\Psi^\diamond(S^\diamond)) \subseteq \Psi^\circ(\alpha^\circ(S^\diamond))$ and $\alpha^\circ(\mathfrak{S}_{Coll^\diamond}[\![P]\!]) \subseteq \mathfrak{S}_{Coll^\circ}[\![P]\!]$.*

It is a well-known results of abstract interpretation that Galois insertions are closed under composition. Therefore, Theorem 5.6 implies that the new abstract collecting semantics is a safe approximation of the concrete one[1].

Examples. There are several interesting examples showing the differences with the abstraction of Section 4. For instance this abstraction doesn't distinguish between one or more than one occurrence. Thus, the processes Q and $!Q$ of Example 4.10 are identified. Another loss of information is due to the removal of the partial topology, as the following example explains.

Example 5.7. Consider the process $P = P_1 \mid P_2 \mid P_3$, where $P_1 = !n_\lambda[\mathtt{in}\, m_\mu.\, \mathtt{in}\, k_\nu]$, $P_2 = !m_\beta[0]$ and $P_3 = !k_\gamma[0]$. We assume that labels $\{\lambda, \mu, \nu, \beta, \gamma\}$ are distinct also up to renaming of indexes. In the first abstraction we have (for readability we use $\{\lambda, \mu, \nu, \beta, \gamma\}$ for the corresponding abstract labels with multiplicity ω), $\mathfrak{S}_{Coll^\diamond}[\![P]\!] = (T^\diamond, C^\diamond)$ where

$$T^\diamond = \{\ {}_{n_\lambda}{}^{@},\ {}_{k_\gamma}{}^{@},\ {}_{m_\beta}{}^{@},\ {}_{n_\lambda}{}^{m_\beta}\}$$

$$C^\diamond = \{\ {}^{{}_{n_\lambda}{}^{@}}\mathtt{in}\, m_\mu.\, \mathtt{in}\, k_\nu,\ {}^{{}_{n_\lambda}{}^{m_\beta}}\mathtt{in}\, m_\mu.\, \mathtt{in}\, k_\nu,\ {}^{{}_{n_\lambda}{}^{m_\beta}}\mathtt{in}\, k_\nu,\ {}^{@}P_1,\ {}^{@}P_2,\ {}^{@}P_3\}.$$

Capability $\mathtt{in}\, k$ is not executed inside n, because it becomes executable only after the execution of $\mathtt{in}\, m$, and k cannot be contained in m.

In the new abstraction we have (for readability we use $\{\lambda, \mu, \nu, \beta, \gamma\}$ for the corresponding abstract labels without indexes),

$$\mathfrak{S}_{Coll^\circ}[\![P]\!] = (T^\diamond \cup \{\ {}_{n_\lambda}{}^{k_\gamma}\}, \{\ {}^{n_\lambda}\mathtt{in}\, m_\mu.\, \mathtt{in}\, k_\nu,\ {}^{n_\lambda}\mathtt{in}\, k_\nu,\ {}^{@}P_1,\ {}^{@}P_2,\ {}^{@}P_3\}).$$

Now $\mathtt{in}\, k$ is executed, because n and k have @ as a common father. The abstract semantics cannot capture that $\mathtt{in}\, k$ becomes executable inside n only after the movement inside m.

6 Conclusions

The main contribution of the paper is the definition of an abstract interpretation framework for MA, based on the normal semantics. The normal semantics uses an explicit representation of the topological structure of processes, in terms of topology and configuration, which is more viable for abstraction than the standard reduction semantics. We have shown two safe abstractions establishing a

[1] $\alpha^\circ(\alpha(\mathfrak{S}_{Coll}[\![P]\!])) \subseteq \mathfrak{S}_{Coll^\circ}[\![P]\!]$.

property concerning the run-time topological structure of processes: the proposed abstract semantics are effectively program analyzers. By restricting the attention to a process P of size n, in the first case the topology of the greatest state contains at most $O(n^2)$ elements and the configuration at most $O(n^3)$ elements. Hence, the iterations before reaching the fixed-point are at most $O(n^3)$. Any iteration has complexity $O(n^4)$, because it requires to check at most $O(n)$ conditions for any element of the configuration. Similarly, in the second case we have at most $O(n^2)$ iterations, where any iteration has complexity $O(n^3)$. Therefore, it is not difficult to devise a naive implementation of the first analysis in $O(n^7)$ and of the second one in $O(n^5)$ by using standard algorithms.

There are several CFA [12, 13, 9], in Flow Logics style, which compute safe approximations of the same property. More in detail, the analysis of [12] could be obtained in our framework from the abstract transition rules of Tab. 5, by weakening the conditions to be checked for the execution of the continuations of capabilities. Our second analysis is therefore more precise than the one of [12], which can be implemented in $O(n^3)$ using more sophisticated techniques [16]. The CFA of [13] is a very powerful exponential counting analysis that refines the approach of [12] essentially by using multiplicity and by considering sets of abstract states rather than abstract states. The relation between the two analyses is demonstrated by a formal comparison in abstract interpretation style. We could obtain this analysis by removing the partial topology from the domain of our first analysis and by a standard "lift" to the power-set. Our first analysis combines together the idea of multiplicity from [13] and in some sense the idea of using contextual information like the partial topology from the CFA of [9] for Safe Ambients [10]. The integration of these two aspects allows us to have a simple polynomial analysis collecting non-trivial information. For instance as shown by Example 4.8 of Section 4 the analysis is able to capture that capabilities with multiplicity one can be used only once altough it does not employ sets of abstract states as in [13].

In our opinion the semantic-based abstract interpretation approach demonstrates several advantages over the "ad-hoc" syntax-based approaches. We have shown that existing analyses and new analyses can be specified in a common framework, where the analyses have a simpler formulation, are related to a formal definition of the property we want to approximate, and are safe and directly compared in terms of precision by construction. The obtained analyses (in particular the one of [12]) are more precise without affecting the complexity from an algorithmic point of view. Moreover, we believe that within this framework new analyses could be derived by simply modifying the abstract domain of properties according to standard abstract interpretation techniques.

This work is part of a project aimed at studying the relationship among abstract interpretation, CFA and types. We intend, as a short term goal, to specify also the types of MA [5, 2] within the proposed abstract interpretation framework. We believe that the formalization of CFA[2] and types in a common

[2] Notice that the proposed analyses could be extended to the full language with communication.

setting would allow us to finally compare their expressive power, integrate them, understand the pros and cons of each approach, and possibly for which class of properties one method is more adequate than another.

References

1. M. Bugliesi and G. Castagna *Secure Safe Ambients.* Proceedinds of POPL '01, pages 222-235. ACM Press 2001.
2. L. Cardelli, G. Ghelli and A. Gordon *Mobility types for mobile ambients.* Proceedings of ICALP' 99, LNCS 1644, pages 230-239. Springer-Verlag, 1999.
3. 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.
4. L. Cardelli and G. Ghelli and A. Gordon. *Ambient Groups and Mobility Types.* Proceedings of IFIP TCS 2000, LNCS 1872, pages 333-347. Springer-Verlag, 2000.
5. L. Cardelli and A. Gordon. *Mobile Ambients* Proceedings of FoSSaCS' 98, LNCS 1378, pages 140-155. Springer-Verlag, 1998.
6. P. Cousot and R. Cousot, *Systematic Design of Program Analysis Frameworks*, Proceedings of POPL '79, pages 269–282, ACM Press, 1979.
7. P. Cousot and R. Cousot, *Abstract Interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, Proceedings of POPL '77, pages 238–252, ACM Press, 1977.
8. M. Dezani-Ciancaglini and I. Salvo *Security Types for Mobile Safe Ambients.* Proceedings of ASIAN'00, LNCS 1961, pages 215-236, Springer-Verlag, 2000.
9. P. Degano and F. Levi and C. Bodei *Safe Ambients: Control Flow Analysis and Security.* Proceedings of ASIAN'00, LNCS 1961, pages 199-214, Springer-Verlag, 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. F. Levi and S. Maffeis *An Abstract Interpretation Framework for Analysing Mobile Ambients (Extended Version).*
    Avalaible at http://www.di.unipi.it/~levifran/papers.html
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.
16. F. Nielson and H. Seidl, Control-Flow Analysis in Cubic Time To appear in Proc. of ESOP'01.

A Some background on abstract interpretation

We briefly recall the basic concepts of abstract interpretation. The main idea is that of establishing a formal relationship, using Galois connections, between the concrete domain $\langle C, \leq \rangle$ of computation and a simpler abstract domain $\langle A, \leq^{\alpha} \rangle$ modelling the property we want to approximate. The ordering $\leq^{\alpha}$ is intended to model precision so that $a \leq^{\alpha} a'$ means that a' is a safe approximation of a.

Definition A.1 (Galois connection). Let (α, γ) be a pair of functions, such that $\alpha : C \to A$ (abstraction) and $\gamma : A \to C$ (concretization). The pair (α, γ) is a *Galois connection* iff $\forall c \in C, \forall a \in A, \alpha(c) \leq^{\alpha} a \Leftrightarrow c \leq \gamma(a)$. If also $\alpha(\gamma(a)) = a$, then (α, γ) is called a Galois insertion.

Given a semantics $\mathcal{S}$, computed as the least fixed-point of a semantic function F over the concrete domain, $\alpha(\mathcal{S})$ gives the exact abstract property corresponding to $\mathcal{S}$. Therefore, safeness of an approximate semantics $\mathcal{S}^{\alpha}$ over the abstract domain is guaranteed by the simple condition $\alpha(\mathcal{S}) \leq^{\alpha} \mathcal{S}^{\alpha}$. One of the main results is that a safe approximate semantics $\mathcal{S}^{\alpha}$ can be computed as the least fixed-point of an abstract semantic function F^{α} as stated by the folowing theorem.

Theorem A.2 (Safeness). *Let (α, γ) be a Galois connection between $\langle C, \leq \rangle$ and $\langle A, \leq^{\alpha} \rangle$. Moreover, let $F : C \to C$ and $F^{\alpha} : A \to A$ be monotonic functions. If $\alpha(F(c)) \leq^{\alpha} F^{\alpha}(\alpha(c))$, for each $c \in C$, then $\alpha(\mathit{lfp}\ F) \leq^{\alpha} \mathit{lfp}\ F^{\alpha}$.*