URL: http://www.elsevier.nl/locate/entcs/volume??.html 14 pages

A Type Inference Algorithm for Secure Ambients

Franco Barbanera,[a] Mariangiola Dezani-Ciancaglini,[b]
Ivano Salvo[b] and Vladimiro Sassone[c]

[a] *Dip. di Matematica e Informatica, Univ. di Catania*
e-mail: barba@dmi.unict.it

[b] *Dip. di Informatica, Univ. di Torino*
e-mail:{dezani, salvo}@di.unito.it

[c] *COGS, University of Sussex*
e-mail: vs@susx.ac.uk

Abstract

We present a bottom-up algorithm which, given an untyped process P, calculates the minimal set of constraints on security levels such that all the actions during a run of P can be performed without violating the security level priorities. Our algorithm appears as a preliminary step in order to use type systems to ensure security properties in the web scenario.

1 Introduction

The *Ambient Calculus* [CG98] has been recently successfully proposed as a model for the Web and the sort of (mobile) computations that can take place inside it. In such a calculus an ambient provides the abstraction for a named location: it may contain processes and sub-ambients, while π-calculus-like processes inside ambients are a natural representation of (concurrent) computations. A process may:

- communicate in an asynchronous way with a process in the same ambient;
- cause the enclosing ambient to move inside or outside other ambients;
- destroy the boundary of a sub-ambient, causing the contents of the sub-ambient to spill into the parent ambient.

In order to have a richer algebraic theory, in the Safe Ambient Calculus [LS00] the activity of processes is better controlled by means of coactions. The basic idea is that an ambient can be traversed or opened if at least one process inside it agrees.

A standard way of forbidding unwanted behaviors is to impose a *type discipline*. Different type disciplines have been proposed for the Ambient Calculus:

Work partially supported by MURST 40% TOSCA Project.

in [CG99] the types assure the correctness of communications. The type system of [CGG99] guarantees also that only ambients which are declared as mobile will move and only ambients which are declared as openable will be opened. Adding subtyping allows us to obtain a more flexible type discipline [Zim00b]. Lastly, by means of *group* names [CGG00], the type of an ambient n controls the set of ambients n may cross and the set of ambients n may open. Moreover the possibility of creating fresh group names gives a flexible way of statically preventing unwanted propagation of names.

A powerful type discipline for the Safe Ambient Calculus has been devised in [LS00]. The main features are the control of ambient mobility and the removing of all *grave interferences*, i.e. of all non-deterministic choices between logically incompatible interactions. This is achieved by means of types which can be derived only for *single-threaded* ambients, i.e. ambients which at every step offer at most one interaction with external or internal ambients. The secure safe ambient calculus of [BC01] is a typed variant of Safe Ambients in which ambient types are protection domains expressing behavioral invariants.

Security plays a crucial role in the theory and practice of distributed systems. In [DCS00] a type discipline is proposed for safe mobile ambients, which is essentially motivated by its ensuring *security* properties. The type of an ambient name specifies a security level s and it is required that an ambient at security level s can only be traversed or opened by ambients at security level at least s. Two independent partial orders are defined on security levels for opening and movement rights. A general analysis of how encoding the mandatory security policy inside Ambient Calculus is carried out in [BCC01b]: difficulties of finding a natural interpretation of basic notions like read and write access have led the authors to introduce a variant of Ambient Calculus, Boxed Ambients [BCC01a].

In this work, we present a bottom-up algorithm for the type system introduced in [DCS00], which, given an untyped process P, calculates the minimal set of constraints on security levels (i.e. the minimal partial order on security levels) such that all the actions during a run of P can be performed without violating the security level priorities. Such an algorithm is proved to be sound and complete. Moreover it is shown to be consistent with the reductions of the calculus, that is the constraints computed on a reduct of a process are related to those computed on the process itself through an *embedding of constraints* relation. In a sense, a reduction makes the computed constraints loosened. Lastly we show that our algorithm gives some information about group typing as defined in [CGG00].

In order to simplify the definition of the algorithm and its analysis, we drop coactions, the distinction between movement and opening rights, and we impose some restrictions on the Ambient Calculus syntax. The correctness and completeness (and its 'minimality') are proved with respect to a simplified version of the type system in [DCS00].

Except for the distributed system calculus defined in [BC01], type systems for ambients need to consider the network as a whole, *globally* typed object. This scenario is unrealistic as a model for the web, where agents typically interact with only partial knowledge of each other. And it is even more unrealistic when we try to model security properties, because attackers need not obey our type system's rules. Algorithms such as ours move a step towards an effective use of type systems to ensure security properties in the global computing scenario, as they can infer useful information even when partial or no type assumptions are available. More precisely, ambients can derive a minimal set of constraints on security levels necessary for an untyped process to be well-typed, and on the basis of such information, they can then decide whether or not to allow the process in, and with what privileges.

2 Calculus and Types

We focus on a version of the Ambient Calculus where only names can be communicated. Under this hypothesis the syntactic distinction between expressions and processes becomes redundant, and the syntax can be given as in Fig. 1, where n ranges over the set $\mathcal{N}$ of ambient names. It is worth remarking that our removing capability passing does not affect the expressiveness of the Ambient Calculus, as shown in [Zim00a] for even more severe restrictions. Reduction is the substitutive binary relation on terms generated by the rules in Fig. 1. We use (m **open** n) (resp. (m **in** n) and (m **out** n)) to indicate a (*Red* **open**)-reduction (resp. (*Red* **in**) and (*Red* **out**)) that targets ambient n and happens inside ambient m.

The structural congruence $\equiv$ is defined as the least congruence such that:

- $\mid$ and $\mathbf{0}$ form a commutative monoid up to $\equiv$;
- $!P \equiv !P \mid P$; $(\nu n)\mathbf{0} \equiv \mathbf{0}$; $(\nu n)(P \mid Q) \equiv (\nu n)P \mid Q$ (for n not free in Q); and $(\nu n)(\nu m)P \equiv (\nu m)(\nu n)P$.

Although it is possible to consider refined notions of structural congruence, as e.g. in [CG00], their adoption here is orthogonal to our development.

As mentioned earlier, we shall present a type assignment system that enables to infer judgments on processes expressing their abiding to constraints on permissions to open or cross boundaries. Then we shall present an algorithm to compute the minimal set of constraints.

We consider a set of type-level names, the *security levels*, $\mathcal{U}$, and a set of type variables $VarT$, ranged over by $\alpha, \alpha', \ldots$, together with the constant type *Shh* that represent the absence of message exchanges. We shall use metavariables S, O, and $\mathcal{O}$ to denote, respectively, subsets, preorders and partial orders on $\mathcal{U}$, and write $s \leq_O s'$ for $(s, s') \in O$.

Definition 2.1 (Ambient Types and Schemes) Types are generated by

$$\mathsf{T} ::= \delta \mid s[\mathsf{T}],$$

where δ ranges over $VarT \cup \{Shh\}$, and s over $\mathcal{U}$.

Ambient types, denoted by $T, T', \ldots$, are types that contain no elements of $VarT$. The set of ambient types will be indicated by $\mathcal{T}$.

Ambient type schemes, denoted by $\tau, \sigma, \ldots$, are types in which *Shh* does not occur. The set of Ambient type schemes is indicated by $\mathcal{TS}$.

$P, Q ::=$	**Processes**		
$\mathbf{0}$	(inactivity)	$\mathbf{in}\ n.P$	(can enter into n)
$\mathbf{out}\ n.P$	(can exit out of n)	$\mathbf{open}\ n.P$	(can open n)
$n[P]$	(ambient)	$(\nu n)P$	(restriction)
$P \mid Q$	(parallel)	$!P$	(replication)
$(x).P$	(input action)	$\langle n \rangle$	(output action)

$n[\mathbf{in}\ m.Q \mid Q'] \mid m[R] \longrightarrow m[n[Q \mid Q'] \mid R]$	(*Red* **in**)
$m[n[\mathbf{out}\ m.Q \mid Q'] \mid R] \longrightarrow n[Q \mid Q'] \mid m[R]$	(*Red* **out**)
$\mathbf{open}\ n.Q \mid n[Q'] \longrightarrow Q \mid Q'$	(*Red* **open**)
$\langle n \rangle \mid (x).P \longrightarrow P\{x := n\}$	(*Red I/O*)
$P \longrightarrow Q \quad \Rightarrow \quad P \mid R \longrightarrow Q \mid R$	(*Red par*)
$P \longrightarrow Q \quad \Rightarrow \quad (\nu n)P \longrightarrow (\nu n)Q$	(*Red res*)
$P \longrightarrow Q \quad \Rightarrow \quad n[P] \longrightarrow n[Q]$	(*Red amb*)
$P' \equiv P \quad P \longrightarrow Q \quad Q \equiv Q' \quad \Rightarrow \quad P' \longrightarrow Q'$	(*Red* $\equiv$)

Fig. 1. Processes and Reduction

Typing environments will be formalised as partial functions from names to types. Reflecting the classification of types introduced above, we shall use E, E, and $\mathcal{E}$ to denote, respectively, $\mathcal{N} \rightharpoonup \mathsf{T}$, $\mathcal{N} \rightharpoonup \mathcal{TS}$, and $\mathcal{N} \rightharpoonup \mathcal{T}$. Also, if $\mathcal{E}(n) = s[\mathsf{T}]$, then $n_{\mathcal{E}}$ will denote s; and if E and E' have disjoint domains, then $\mathsf{E} \cdot \mathsf{E}'$ stands for partial function formed as their union. In particular, we write $\mathsf{E} \cdot n{:}\, \mathsf{T}$ when $\mathsf{E}' = \{n, \mathsf{T}\}$.

In the following we shall give precise meaning and use the above elements tuples either in the form $\langle \mathcal{E}, S, \mathcal{O}, T \rangle$ (for the type assignment system) or $\langle E, S, O, \tau \rangle$ (for the type inference algorithm). For the time being, we give the definition below on a general sort of tuples that capture all cases of interest.

Definition 2.2 (Relation $\triangleleft$) The binary relation $\triangleleft$ on tuples of the shape $\langle \mathsf{E}, S, O, \mathsf{T} \rangle$, with $S \subseteq dom(O)$ is defined by

$$\langle \mathsf{E}, S, O, \mathsf{T} \rangle \quad \triangleleft \quad \langle \mathsf{E}', S', O', \mathsf{T}' \rangle$$

if there exists $\varphi\colon (dom(O) \to dom(O')) \cup (\mathit{VarT} \to \mathsf{T})$ such that

- $\varphi(\mathsf{E}) \subseteq \mathsf{E}'$;
- $\forall s \in S\ \exists s' \in S'.\varphi(s) \leq_{O'} s'$;
- φ is monotone as a preorder map $O \to O'$;
- $\varphi(\mathsf{T}) = \mathsf{T}'$.

3 An assignment system for security levels constraints

Our aim is to infer judgments on security policies expressed by permissions to cross boundaries. These are granted according to security levels associated with ambients. We use a mapping $\mathcal{E}$ to describe the type associated to each ambient n. Ambient types describe the security level of ambients and, recursively, the security levels of the names that can be exchanged inside n. The information provided by $\mathcal{E}$ is complemented by a partial order $\mathcal{O}$ that describes the priority relations between security levels that the given security policy dictates. The central judgment in our assignment system sounds as '*P abides by the constraints $\mathcal{E}$ and $\mathcal{O}$*', whose meaning can be informally explained as: if, in any possible reduction sequence of a process abiding $\mathcal{E}$ and $\mathcal{O}$, a reduction makes an ambient n (resp. a process inside n) enter or exit (resp. open) an ambient m, then the security level associated to n is not less than the one associated to m, that is $n_{\mathcal{E}} \geq_{\mathcal{O}} m_{\mathcal{E}}$.

Definition 3.1 Process P abides by the constraints $\mathcal{E}, \mathcal{O}$, writtend $\mathcal{E}, \mathcal{O} \vdash_{ok} P$, if for all sequences $P \longrightarrow^{+}$ terminating with $(n\ \mathbf{in}\ m)$, $(n\ \mathbf{out}\ m)$, or $(n\ \mathbf{open}\ m)$ reductions, we have $n_{\mathcal{E}} \geq_{\mathcal{O}} m_{\mathcal{E}}$.

The system proposed in Fig. 2 is essentially a simplification of the system of [DCS00]. Its purpose is to infer all possible judgments $\mathcal{E}, \mathcal{O} \vdash_{ok} P$. As a matter of fact, we derive judgments that are more informative, viz. $\mathcal{E}, \mathcal{O} \vdash P : S, T$, where $S \subseteq dom(\mathcal{O})$ and S, T is a process type as defined below.

Definition 3.2 (Process Types) A process type $\mathcal{F}$ is a pair (S, T), with $S \subseteq \mathcal{U}$ and $T \in \mathcal{T}$.

The first component of process types contains the security levels of ambient names involved in capabilities possibly performed by a process; the second is the ambient type of the names exchanged by processes. Concerning the rules, $(\mathbf{in}\ n\ \vdash)$, $(\mathbf{out}\ n\ \vdash)$ and $(\mathbf{open}\ n\ \vdash)$ verify that the security level s of the target ambient name n is bounded by at least one security level in S. Further to that, $(\mathbf{open}\ n\ \vdash)$ checks that the type T of the ambient names exchanged by P and inside n coincide. Rule (Proc Amb $\vdash$), instead, verifies that:

- all the security levels of S (which are those of the capabilities possibly performed by P) are bounded by the security level s of the ambient n;
- the type T of the ambient names exchanged by P and inside n coincide.

3.1 Properties

As to be expected, system $\vdash$ assigns the same type to structurally congruent processes, and such types are preserved under reduction. The proofs here proceed by rule induction on $\equiv$ and $\longrightarrow$.

Lemma 3.3 (Subject Congruence of $\vdash$) *If $P \equiv Q$ and $\mathcal{E}, \mathcal{O} \vdash P : \mathcal{F}$, then $\mathcal{E}, \mathcal{O} \vdash Q : \mathcal{F}$.*

Theorem 3.4 (Subject Reduction of $\vdash$) *If $\mathcal{E}, \mathcal{O} \vdash P : S, T$ and $P \longrightarrow Q$, then $\mathcal{E}, \mathcal{O} \vdash Q : S, T$.*

$$\frac{}{\mathcal{E},\mathcal{O} \vdash \mathbf{0} : S,T}\ (\text{Proc } \mathbf{0}\ \vdash) \qquad \frac{\mathcal{E}\cdot n : T',\mathcal{O} \vdash P : S,T}{\mathcal{E},\mathcal{O} \vdash (\nu n)P : S,T}\ (\text{Proc Res} \vdash)$$

$$\frac{\mathcal{E},\mathcal{O} \vdash P : S,T \quad n_{\mathcal{E}} = s \quad \exists s' \in S.s \leq_{\mathcal{O}} s'}{\mathcal{E},\mathcal{O} \vdash \mathbf{in}\ n.P : S,T}\ (\mathbf{in}\ n\ \vdash)$$

$$\frac{\mathcal{E},\mathcal{O} \vdash P : S,T \quad n_{\mathcal{E}} = s \quad \exists s' \in S.s \leq_{\mathcal{O}} s'}{\mathcal{E},\mathcal{O} \vdash \mathbf{out}\ n.P : S,T}\ (\mathbf{out}\ n\ \vdash)$$

$$\frac{\mathcal{E},\mathcal{O} \vdash P : S,T \quad \mathcal{E}(n) = s[T] \quad \exists s' \in S.s \leq_{\mathcal{O}} s'}{\mathcal{E},\mathcal{O} \vdash \mathbf{open}\ n.P : S,T}\ (\mathbf{open}\ n\ \vdash)$$

$$\frac{\mathcal{E},\mathcal{O} \vdash P : S,T \quad \mathcal{E}(n) = s[T] \quad s' \leq_{\mathcal{O}} s,\ \text{for all } s' \in S}{\mathcal{E},\mathcal{O} \vdash n[P] : S',T'}\ (\text{Proc Amb} \vdash)$$

$$\frac{\mathcal{E},\mathcal{O} \vdash P : S,T \quad \mathcal{E},\mathcal{O} \vdash Q : S,T}{\mathcal{E},\mathcal{O} \vdash P \mid Q : S,T}\ (\text{Proc Par} \vdash) \qquad \frac{\mathcal{E},\mathcal{O} \vdash P : S,T}{\mathcal{E},\mathcal{O} \vdash !P : S,T}\ (\text{Proc Repl} \vdash)$$

$$\frac{\mathcal{E}\cdot x : T,\mathcal{O} \vdash P : S,T}{\mathcal{E},\mathcal{O} \vdash (x).P : S,T}\ (\text{Proc Input} \vdash) \qquad \frac{}{\mathcal{E}\cdot n : T,\mathcal{O} \vdash \langle n\rangle : S,T}\ (\text{Proc Output} \vdash)$$

Fig. 2. Type Assignment System

To relate the type assignment system to a type inference algorithm, we find it useful to derive a weakening lemma that uses the relation $\lhd$ introduced in Definition 2.2. The proof is by induction on deductions.

Lemma 3.5 (Weakening) *If* $\langle \mathcal{E},\mathcal{O},S,T\rangle \lhd \langle \mathcal{E}',\mathcal{O}',S',T'\rangle$, *then*

$$\mathcal{E},\mathcal{O} \vdash P : S,T \quad \Rightarrow \quad \mathcal{E}',\mathcal{O}' \vdash P : S',T'.$$

We close this section by stating formally that our type assignment system can type a process P w.r.t. given environment and partial order if and only if P abides by them (according to Definition 3.1). The proof of the 'if' part is by induction on reductions and that of the 'only if' part by induction on derivations.

Theorem 3.6 (Soundness and Completeness of $\vdash$) $\mathcal{E},\mathcal{O} \vdash_{ok} P$ *if and only if there exist* S,T *such that* $\mathcal{E},\mathcal{O} \vdash P : S,T$.

4 A minimal constraints algorithm

In this section, we address the problem of finding, for a given, untyped process P, a 'minimal' set of constraints that P abides by.

Definition 4.1 (Minimal Effects) A *minimal effect*, F is a tuple $\langle E, S, O, \tau\rangle$ whose components are as described in the notational conventions of Section 2.

The bottom-up algorithm described in Fig. 3 enables to infer judgments of the shape

$$\Vdash P : \mathsf{F}$$

where P is a process, and F is a minimal effect. If the algorithm computes a minimal effect $\langle E, S, O, \tau\rangle$ for a process P, this will be such that:

- E contains the inferred type assumptions about free names of P,
- S includes security levels associated to the names involved in the capabilities possibly performed by P,
- O represents the minimal order relation on security levels for P to be well-typed, and
- τ represents the ambient type scheme of the names exchanged by P.

Before discussing the rules in detail, let us remark that, even though we finally express the constraints that make P typeable as a partial order, we find it convenient to work with preorders, as they simplify considerably the rules of Fig. 3. In particular, they allow us to define the following operations on effects independently of their environments.

Definition 4.2 (Operations on Preorders) Let $O, O' \subseteq \mathcal{U} \times \mathcal{U}$ be preorders of security levels.

We define $O \oplus O'$ to be the preorder induced by the union of O and O', i.e. the transitive closure of $O \cup O'$.

For S a set of security levels, and s a security level, $S^{\uparrow s}$ is the preorder where s is added to S as the top element. Formally:

$$S^{\uparrow s} = \{(s,s)\} \cup \{(s',s) \mid s' \in S\} \cup \{(s',s') \mid s' \in S\}$$

We use $O[s \le s']$ for $O \oplus \{(s,s')\}$ and $O[s = s']$ for $O[s \le s'][s' \le s]$.

Let $\sigma = s_1[\ldots s_n[\alpha]\ldots]$ and $\tau = s'_1[\ldots s'_m[\gamma]\ldots]$ be ambient type schemes. We define

$$\max(\sigma,\tau) = \begin{cases} \sigma \text{ if } n \ge m \\ \tau \text{ if } n < m \end{cases}$$

Definition 4.3 (Unification of Ambient Type Schemes) For $\alpha \ne \gamma$, let $\sigma = s_1[\ldots s_n[\alpha]\ldots]$ and $\tau = s'_1[\ldots s'_m[\gamma]\ldots]$ be ambient type schemes, and assume $\max(\sigma,\tau) = \tau$. The *unification* of σ and τ produces the set of equations $s_i = s'_i$, and the substitution $\{\alpha \leftarrow s'_{n+1}[\ldots s'_m[\gamma]\ldots]\}$. We shall use:

- $E\{\sigma = \tau\}$ to denote the environment obtained from E, by replacing all occurrences of α with $s'_{n+1}[\ldots s'_m[\gamma]\ldots]$.
- $O[\sigma = \tau]$ to denote the preorder $O[s_1 = s'_1]\cdots[s_n = s'_n]$.

The unification of ambient type schemes is central in our construction to express the constraints involved in parallel composition of processes – rule (Proc Par) in Fig. 3 – where we need to compose preorders equating the

$$\frac{\alpha \text{ fresh}}{\Vdash \mathbf{0} : \langle \varnothing, \varnothing, \varnothing, \alpha \rangle}\ (\text{Proc } \mathbf{0} \Vdash) \qquad \frac{\Vdash P : \langle E, S, O, \tau \rangle}{\Vdash (\nu n)P : \langle E \searrow n, S, O, \tau \rangle}\ (\text{Proc Res} \Vdash)$$

$$\frac{\Vdash P : \langle E, S, O, \tau \rangle \quad X(n) = s[\sigma] \quad X = E^n}{\Vdash \mathbf{in}\ n.P : \langle X, S \cup \{s\}, O[s = s], \tau \rangle}\ (\mathbf{in}\ n \Vdash)$$

$$\frac{\Vdash P : \langle E, S, O, \tau \rangle \quad X(n) = s[\sigma] \quad X = E^n}{\Vdash \mathbf{out}\ n.P : \langle X, S \cup \{s\}, O[s = s], \tau \rangle}\ (\mathbf{out}\ n \Vdash)$$

$$\frac{\Vdash P : \langle E, S, O, \tau \rangle \quad X(n) = s[\sigma] \quad X = E^n}{\Vdash \mathbf{open}\ n.P : \langle X[\sigma = \tau], S \cup \{s\}, O[s = s][\sigma = \tau], \max(\sigma, \tau) \rangle}\ (\mathbf{open}\ n \Vdash)$$

$$\frac{\Vdash P : \langle E, S, O, \tau \rangle \quad X(n) = s[\sigma] \quad X = E^n \quad \alpha \text{ fresh}}{\Vdash n[P] : \langle X[\sigma = \tau], \varnothing, (S^{\uparrow s} \oplus O)[\sigma = \tau], \alpha \rangle}\ (\text{Proc Amb} \Vdash)$$

$$\frac{\Vdash P : \langle E, S, O, \tau \rangle \quad \Vdash Q : \langle E', S', O', \sigma \rangle}{\Vdash P \mid Q : \langle (E \cdot E')\{\sigma = \tau\}, S \cup S', (O \bigoplus_{E,E'} O')[\sigma = \tau], \max(\sigma, \tau) \rangle}\ (\text{Proc Par} \Vdash)$$

$$\frac{\Vdash P : \mathsf{F}}{\Vdash !P : \mathsf{F}}\ (\text{Proc Repl} \Vdash) \qquad \frac{\alpha \text{ fresh}}{\Vdash \langle n \rangle : \langle n : \alpha, \varnothing, \varnothing, \alpha \rangle}\ (\text{Proc Output} \Vdash)$$

$$\frac{\Vdash P : \langle E, S, O, \tau \rangle \quad E^x(x) = \sigma}{\Vdash (x).P : \langle (E \searrow x)\{\sigma = \tau\}, S, O[\sigma = \tau], \max(\sigma, \tau) \rangle}\ (\text{Proc Input} \Vdash)$$

Fig. 3. Minimal Constraints Algorithm

security levels associated to the same ambient names. This is formalized as:

$$O \bigoplus_{E,E'} O' = (O \oplus O')[E(n) = E'(n)]_{n \in dom(E) \cap dom(E')}$$

Finally, we will be building environments by means of the operations below.

$$E^n = \begin{cases} E & \text{if } n \in dom(E) \\ E, n : s[\alpha] & \text{if } n \notin dom(E) \quad \text{with } s, \alpha \text{ fresh} \end{cases}$$

$$E \searrow n = \begin{cases} E \setminus \{n : E(n)\} & \text{if } n \in dom(E) \\ E & \text{if } n \notin dom(E) \end{cases}$$

Moreover, we extend the notation $E \cdot E'$ to non necessarily disjoint E and E' by defining $(E \cdot E')(n) = \max(E(n), E'(n))$ if $n \in dom(E) \cap dom(E')$.

$$\dfrac{\dfrac{}{\Vdash \mathbf{0} : \langle \emptyset, \emptyset, \emptyset, \alpha_0 \rangle}}{\Vdash \mathbf{open}\ m.\mathbf{0} : \langle \{m : s_2[\alpha_2]\}, \{(s_2, s_2)\}, \{s_2\}, \alpha_0 \rangle}\ \text{(D1)}$$

$$\dfrac{\dfrac{\dfrac{}{\Vdash \mathbf{0} : \langle \emptyset, \emptyset, \emptyset, \alpha_4 \rangle}}{\Vdash \mathbf{out}\ n.\mathbf{0} : \langle \{n : s_1[\alpha_1]\}, \{(s_1, s_1)\}, \{s_1\}, \alpha_4 \rangle}}{\Vdash m[\mathbf{out}\ n.\mathbf{0}] : \langle \{n : s_1[\alpha_1], m : s_3[\alpha_3]\}, \{(s_1, s_1), (s_3, s_3), (s_2, s_3)\}, \emptyset, \alpha_5 \rangle}\ \text{(D2)}$$

$$\dfrac{\dfrac{(D1) \qquad\qquad (D2)}{\Vdash P \mid Q : \langle \{n : s_1[\alpha_1], m : s_2[\alpha_2]\}, \{(s_1, s_1), (s_2, s_2), (s_1, s_2)\}, \{s_2\}, \alpha_0 \rangle}}{\Vdash n[P \mid Q] : \langle \{n : s_1[\alpha_1], m : s_2[\alpha_2]\}, \{(s_1, s_1), (s_2, s_2), (s_1, s_2), (s_2, s_1)\}, \emptyset, \alpha_6 \rangle}$$

Fig. 4. Example of type inference ($P \equiv \mathbf{open}\ m.\mathbf{0}$ and $Q \equiv m[\mathbf{out}\ n.\mathbf{0}]$)

Getting back to Fig. 3, we can now discuss the details of the formal definition of the algorithm. At the beginning, as formalized in the (Proc $\mathbf{0}$) rule, no relation among security levels exists. The environment is empty and there is no information about names exchanged or involved in capabilities potentially performed by the processes. A **in** n or a **out** n capability cause the name n to be inserted in the environment, if not already present, and the security level associated to n to be inserted in the set of effects S. (Notice the use of X in the rules to indicate that the same level is associated to n both in premises and in the conclusions.) In the typing of a process of the shape **open** $n.P$, we have to take care that, after the opening of the ambient n, processes running inside n will run in parallel with P, and hence we have to unify the type exchanged by P with the type exchanged inside n. According to our security policy, to infer the type of a process of the shape $n[P]$, we need to impose that the security level of n is greater or equal to those of names involved in capabilities possibly performed by P. When two processes P and Q are put in parallel, we need to unify types exchanged by P and Q, and compute an order among security levels, which contains those calculated for P and Q. In the typing of an input action, $(x).P$, we need to unify the type of the expected value with the type of messages exchanged by P. The algorithm has to take care of scope rules of the calculus, removing from the environment names which become bound because of an input action or a restriction.

At the end, we obtain associations between free ambient names and type schemes in the environment and a preorder relation among security levels. Figure 4 gives an example of derivation.

4.1 Properties

As for the type assignment system, our algorithm enjoys some of the expected properties: it infers *isomorphic* effects for structural congruent processes; by reduction, the minimal constraints of a process get *loosened*, because order constraints depend on the potential execution of capabilities, and performed capabilities disappear during reduction. In the following definitions we formalize the notion of isomorphic effects and loosened constraints. We start by lifting functions from sets to environments and ambient type scheme.

Definition 4.4 For $S, S' \subseteq \mathcal{U}$, a map $\varphi\colon (S \to S') \cup (\mathit{VarT} \to \mathcal{TS})$ defines maps $\llcorner\varphi\lrcorner\colon \mathcal{TS} \to \mathcal{TS}$ and $\ulcorner\varphi\urcorner\colon \mathit{Env} \to \mathit{Env}$ as follows

$$\llcorner\varphi\lrcorner(\alpha) = \varphi(\alpha); \qquad \llcorner\varphi\lrcorner(s[\tau]) = \varphi(s)[\llcorner\varphi\lrcorner(\tau)]$$
$$\ulcorner\varphi\urcorner(E) = \llcorner\varphi\lrcorner \circ E.$$

In the following we shall omit explicit mention of $\llcorner\_\lrcorner$ and $\ulcorner\_\urcorner$.

Definition 4.5 (Partial Order Collapse) For $\mathsf{F} = \langle E, S, O, \tau\rangle$ an effect, its partial order collapse is the effect $\mathsf{F}_{\equiv} = \langle E/_{\equiv}, S/_{\equiv}, O/_{\equiv}, \tau/_{\equiv}\rangle$ where $\_/_{\equiv}$ is a quotient map[1] for the equivalence relation $\equiv =_{def} \leq_O \cap \geq_O$. For the sake of readability we will denote $\mathsf{F}_{\equiv}$ with $\overline{\mathsf{F}}$ and the tuple $\langle E/_{\equiv}, S/_{\equiv}, O/_{\equiv}, \tau/_{\equiv}\rangle$ with $\langle \overline{E}, \overline{S}, \overline{O}, \overline{\tau}\rangle$.

Note that partial order collapses are a special kind of effects, since their preorder component is actually a partial order. We call them *ordered effects.* Ordered effects are relevant to our development because, as it will be evident shortly, they represent the essence of effects and provide a bridge to ambient types. In particular, if our rules allow to infer $\Vdash P : \mathsf{F}$, then the essential information given by our algorithm is contained in $\mathsf{F}_{\equiv}$.

Definition 4.6 (The Relation $\bowtie$ on Ordered Effects) Let $\mathsf{F} = \langle E, S, O, \tau\rangle$ and $\mathsf{F}' = \langle E', S', O', \tau'\rangle$ be ordered effects. We say that $\mathsf{F} \bowtie \mathsf{F}'$ if $\mathsf{F} \lhd \mathsf{F}'$ (as defined in Definition 2.2) via a map φ which is an isomorphism between S and S' and between O and O'.

Definition 4.7 (Effects Inequalities and Isomorphism) Effect F is *refined* by effect F', notation $\mathsf{F} \prec \mathsf{F}'$, if $\mathsf{F}_{\equiv} \lhd \mathsf{F}'_{\equiv}$. Effect F is *isomorphic* to effect F', notation $\mathsf{F} \cong \mathsf{F}'$, if $\mathsf{F}_{\equiv} \bowtie \mathsf{F}'_{\equiv}$.

Theorem 4.8 (Congruence) *If $P \equiv Q$, $\Vdash P : \mathsf{F}$ and $\Vdash Q : \mathsf{F}'$, then $\mathsf{F} \cong \mathsf{F}'$.*

As stated formally by the following theorem, the system $\Vdash$ infers an effect for each well-formed process: at worst, all security levels could be equated, making our security policy not relevant.

Lemma 4.9 (Termination) *For every P there exists F, unique up to $\cong$, such that $\Vdash P : \mathsf{F}$.*

Theorem 4.10 (Loosening by reduction) *If $\Vdash P : \mathsf{F}$, and $P \longrightarrow Q$, and $\Vdash Q : \mathsf{F}'$, then $\mathsf{F} \prec \mathsf{F}'$.*

Proof. We give a concise proof which stems from relationships between systems $\vdash$ and $\Vdash$, discussed in the next section (Theorems 5.2 and 5.3).

[1] We convene that $\_/_{\equiv}$ chooses a canonical representative for each class and maps security levels accordingly, as the canonical quotient map $x \mapsto [x]/_{\equiv}$ would not yield an effect for trivial reasons.

Let $\mathsf{F} = \langle E, S, O, \tau\rangle$ and $\mathsf{F}' = \langle E', S', O', \tau'\rangle$. We have that

$$\begin{tikzcd}
\Vdash P\colon \langle E, S, O, \tau\rangle \arrow[d, Rightarrow, "\text{(by Th. 5.2)}"'] & & \Vdash Q\colon \langle E', S', O', \tau'\rangle \\
\overline{E}^*, \overline{O} \vdash P : \overline{S}, \overline{\tau}^* \arrow[rr, Rightarrow, "\text{(by Th. 3.4)}"'] & & \overline{E}^*, \overline{O} \vdash Q : \overline{S}, \overline{\tau}^* \arrow[u, Rightarrow, "\text{(by Th. 5.3)}"']
\end{tikzcd}$$

where, according to Theorem 5.3, $\langle \overline{E'}^*, \overline{S'}, \overline{O'}, \overline{\tau'}^*\rangle \lhd \langle \overline{E}^*, \overline{S}, \overline{O}, \overline{\tau}^*\rangle$. It easily follows that $\langle E, S, O, \tau\rangle \prec \langle E', S', O', \tau'\rangle$. $\square$

One could argue that it would be quite natural to expect the statement of the Loosening by reduction Theorem to hold for a stronger version of the relation $\prec$, namely the one obtained by adding, in the second condition on φ of Definition 4.6, the requirement of φ to be *injective*. This, however, turns out to be impossible, intuitively because by performing a reduction we make the number of capabilities in a process decrease. It is obvious that the fewer the capabilities in a process, the fewer the pairs in the preorder inferred by our algorithm. However, to have a smaller preorder does not imply to get a smaller corresponding partial order. Then, roughly, by reduction we get smaller preorders, but not necessarily smaller corresponding partial orders. This is better explained by the following example. Let us consider the process

$$P = n[\mathbf{open}\ m.\mathbf{0} \mid m[\mathbf{out}\ n.\mathbf{0}]].$$

The partial order of constraints for P is simply $\{(s, s)\}$, with $n_{\mathcal{E}} = s$ and $m_{\mathcal{E}} = s$, whereas the algorithm computes $\{(s_1, s_1), (s_2, s_2), (s_1, s_2), (s_2, s_1)\}$, with $n_{\mathcal{E}} = s_1$ and $m_{\mathcal{E}} = s_2$ (see Figure 4). If we reduce P via a (Red **open**) reduction, we get

$$n[\mathbf{out}\ n.\mathbf{0}]$$

It is not difficult to see that the partial order of constraints is bigger for this process than for P. In fact it is $\{(s_1, s_1), (s_2, s_2), (s_1, s_2)\}$, coming out of a preorder computed by the algorithm as $\{(s_1, s_2), (s_1, s_1), (s_2, s_2)\}$.

5 Correctness and Minimality

In this section, we show that the algorithm is correct and complete with respect to the type assignment system. Moreover, Theorem 5.3 asserts that the effect computed by the algorithm *refines* (in the sense of Definition 4.7) any possible typing in the type assignment system.

Definition 5.1 Let $(\cdot)^*\colon \mathit{VarT} \to \mathcal{T}$ be the map such that $\alpha^* = \mathit{Shh}$, for all $\alpha \in \mathit{VarT}$. We lift $(\cdot)^*$ to type schemes and environments in the obvious way.

Theorem 5.2 (Correctness) *If, for a process P, we have:* $\Vdash P : \langle E, S, O, \tau\rangle$ *then* $\overline{E}^*, \overline{O} \vdash P : \overline{S}, \overline{\tau}^*$*, where* $\langle \overline{E}, \overline{S}, \overline{O}, \overline{\tau}\rangle = \langle E, S, O, \tau\rangle_{\equiv}$.

The order 'calculated' by the algorithm is miminal in the following sense.

Theorem 5.3 (Minimality) *Let* $\Vdash P : \langle E, S, O, \tau\rangle$ *and* $\mathcal{E}, \mathcal{O} \vdash P : S', T$*. Then* $\langle \overline{E}^*, \overline{S}, \overline{O}, \overline{\tau}^*\rangle \lhd \langle \mathcal{E}, S', \mathcal{O}, T\rangle$.

Both theorems can be proved by induction on derivations. All cases are quite simple, apart from that of rule (Proc Amb $\Vdash$), which requires so technical ingenuity.

6 Relations with group types

In this section we experiment about the relationships between types as considered here and the group types of [CGG00]. We consider security levels as group names, and do not distinguish between groups for opening and for moving ambients. The syntax of group types is then

$$T := Shh \mid s[S,T].$$

Let $\mathcal{GT}$ stand for the set of group types. Due to lack of space we cannot reformulated here the typing rules of [CGG00] with the types in $\mathcal{GT}$; suffices it to say that there is no surprise in that, and the rules are as expected. We denote by $\vdash_G$ the type system so obtained.

Definition 6.1 Given a partial order $\mathcal{O}$ we define

$$\mathcal{S}_{\mathcal{O}}(s) = \{s' \mid s' \leq_{\mathcal{O}} s\}, \qquad \mathcal{S}_{\mathcal{O}}(S) = \bigcup_{s \in S} \mathcal{S}_{\mathcal{O}}(s)$$

and the mappings $\mu_{\mathcal{O}} : \mathcal{TS} \to \mathcal{GT}$, $\nu_{\mathcal{O}} : T \to \mathcal{GT}$ such that

$$\mu_{\mathcal{O}}(Shh) = Shh, \quad \mu_{\mathcal{O}}(s[\tau]) = s[\mathcal{S}_{\mathcal{O}}(s), \mu_{\mathcal{O}}(\tau)],$$
$$\nu_{\mathcal{O}}(\alpha) = Shh, \quad \nu_{\mathcal{O}}(s[\tau]) = s[\mathcal{S}_{\mathcal{O}}(s), \nu_{\mathcal{O}}(\tau)].$$

We lift $\mu_{\mathcal{O}}$ and $\nu_{\mathcal{O}}$ to environment as usual.

We can prove that $\vdash P : \langle E, S, O, \tau\rangle$ implies $\mu_{\mathcal{O}}(E) \vdash_G P : \mathcal{S}_{\mathcal{O}}(S), \mu_{\mathcal{O}}(\tau)$ where $\mathcal{O}$ is the partial order induced by the preorder O.

Lemma 6.2 *$\mathcal{E}, \mathcal{O} \vdash P : S, T$ implies $\mu_{\mathcal{O}}(E) \vdash_G P : \mathcal{S}_{\mathcal{O}}(S), \mu_{\mathcal{O}}(T)$.*

Proof. By induction on type derivations of $\vdash$. If the last rule applied is (Proc Amb $\vdash$) let $\mathcal{E}, \mathcal{O} \vdash n[P] : S'', T''$ be the conclusion. This implies that for some S and T, we have derived the judgment $\mathcal{E}, \mathcal{O} \vdash P : S, T$ and that $E(n) = s[T']$, with $T = T'$ and s greater than every security level in S. Under such conditions and by definition of $\mathcal{S}_{\mathcal{O}}$, we have that $S \subseteq \mathcal{S}_{\mathcal{O}}(S) \subseteq \mathcal{S}_{\mathcal{O}}(s)$. By weakening, we can derive the judgment $\mathcal{E}, \mathcal{O} \vdash P : \mathcal{S}_{\mathcal{O}}(s), T$. Applying the induction hypothesis, the judgment $\mu_{\mathcal{O}}(E) \vdash_G P : \mathcal{S}_{\mathcal{O}}(s), \mu_{\mathcal{O}}(T)$ is derivable and $\mu_{\mathcal{O}}(E)(n) = s[\mathcal{S}_{\mathcal{O}}(s), \mu_{\mathcal{O}}(T')]$ are derivable. Since $T = T'$, clearly implies $\mu_{\mathcal{O}}(T) = \mu_{\mathcal{O}}(T')$, we can conclude $\mu_{\mathcal{O}}(E) \vdash_G n[P] : \mathcal{S}_{\mathcal{O}}(S''), \mu_{\mathcal{O}}(T'')$. □

Theorem 6.3 *$\Vdash P : \langle E, S, O, T\rangle$ implies $\nu_{\mathcal{O}}(\overline{E}) \vdash_G P : \mathcal{S}_{\mathcal{O}}(\overline{S}), \nu_{\mathcal{O}}(\overline{T})$, where $\mathcal{O}$ is the partial order induced by the preorder O.*

Proof. Just by applying the correctness of the type inference algorithm with respect to the type assignment system and the previous lemma. □

7 Conclusions and Future Works

In the scenario we are considering an ambient can enter another when the former has a greater priority. Although this is quite reasonable, one has to be aware that once the ambient has entered its greater priority could enable it to do, so to speak, whatever it likes. This is to be avoided if we wish to have a more realistic, safer and thoroughly desirable scenario in which an ambient manages to control what happens inside itself.

Reflecting on this, we are naturally led to consider the possibility for an ambient to assign a safe security level to ambients crossing its boundaries and, in general, to all its sub-ambients. Moreover the security level of an ambient can increase when it received an 'audit' certificate or decrease when it crosses an unsafe ambient. A distributed version of our type assignment and algorithm can provide the right framework to address an extension of the Ambient Calculus with capabilities allowing security levels modifications as the ones suggested above.

References

[BC01] Michele Bugliesi and Giuseppe Castagna. Secure safe ambients. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages*, pages 222–235. ACM Press, 2001.

[BCC01a] Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Boxed ambients. In Benjamin Pierce, editor, *Proceedings of Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science. Springer-Verlag, 2001. To appear.

[BCC01b] Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Reasoning about security in mobile ambients. In Kim G. Larsen and Mogens Nielsen, editors, *Proceedings of CONCUR 2001*, volume 2154 of *Lecture Notes in Computer Science*, pages 102–120. Springer-Verlag, 2001.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In Maurice Nivat, editor, *FoSSaCS 1998*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer-Verlag, 1998.

[CG99] Luca Cardelli and Andrew D. Gordon. Types for mobile ambients. In *POPL'99*, pages 79–92, New York, NY, USA, 1999. ACM Press.

[CG00] Luca Cardelli and Andrew Gordon. Anytime, anywhere. modal logics for mobile ambients. In *Proceedings of the 27th ACM Symposium on Principles of Programming Languages*. ACM Press, 2000.

[CGG99] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Mobility types for mobile ambients. In Jiří Wiederman, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP 1999*, volume 1644 of *Lecture Notes in Computer Science*, pages 230–239. Springer-Verlag, 1999.

[CGG00] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Ambient groups and mobility types. In Jan van Leeuwen et al., editor, *Theoretical Computer Science: Exploring New Frontiers in Theoretical Informatics*, volume 1872 of *Lecture Notes in Computer Science*, pages 333–347. Springer-Verlag, 2000.

[DCS00] Mariangiola Dezani-Ciancaglini and Ivano Salvo. Security types for safe mobile ambients. In He Jifeng and Masahiko Sato, editors, *Proceedings of ASIAN 2000*, volume 1961 of *Lecture Notes in Computer Science*, pages 215–236. Springer-Verlag, 2000.

[LS00] Francesca Levi and Davide Sangiorgi. Controlling interference in ambients. In *POPL'00*, pages 352–364, New York, NY, USA, 2000. ACM Press.

[Zim00a] Pascal Zimmer. On the expressiveness of pure mobile ambients. In *EXPRESS00, the 7th International Workshop on Expressiveness in Concurrency*, Electronic Notes in Theoretical Computer Science. Elsevier, 2000.

[Zim00b] Pascal Zimmer. Subtyping and typing algorithms for mobile ambients. In Jerzy Tiuryn, editor, *FoSSaCS 2000*, volume 1784 of *Lecture Notes in Computer Science*, pages 375–390. Springer-Verlag, 2000.