URL: http://www.elsevier.nl/locate/entcs/volume62.html 19 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 consider a type discipline for the Ambient Calculus that associates ambients with security levels and constrains them to be traversed by or opened in ambients of higher security clearance only. We present a bottom-up algorithm that, given an untyped process P, computes a minimal set of constraints on security levels such that all actions during runs of P are performed without violating the security level priorities. Such an algorithm appears to be a prerequisite to use type systems to ensure security properties in the web scenario.

1 Introduction

The *Ambient Calculus* [CG98] has recently been successfully proposed was a model for the Internet and the sort of (mobile) computations that can take place on it. Ambients provide the abstraction for named locations: they may contain processes and sub-ambients, while π-calculus-like processes inside ambients are a natural representation of (concurrent) computations. A process may:

- communicate asynchronously with processes 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 be released into the parent ambient.

In order to enrich the algebraic theory of ambients, [LS00] introduced the Safe Ambient Calculus, where process activities are more finely controlled by

Work partially supported by MURST 40% TOSCA Project.

means of coactions. The basic idea here is that an ambient can be traversed or opened if a process inside it agrees.

A standard way of forbidding unwanted behaviours is to impose a *type discipline*. Different type disciplines have been proposed for the Ambient Calculus: types in [CG99] assure the correctness of communications, while the type system of [CGG99] in addition guarantees that only ambients declared as 'mobile' actually move and only ambients declared as 'openable' are opened. Adding subtyping as in [Zim00b] allows us to obtain a more flexible type discipline. Finally, using 'group names', [CGG00] allows the type of an ambient n to control the set of ambients n may cross and the set of ambients it may open. Moreover, dynamic private group names creation provides a flexible way of preventing unwanted names propagation.

A powerful type discipline for the Safe Ambient Calculus has been devised in [LS00] whose main features are the control of ambient mobility and the removing of *grave interferences*, i.e. of all non-deterministic choices between logically incompatible interactions. This is achieved using 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 behavioural 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 that is motivated essentially by its ensuring *security* properties. The type of an ambient name specifies a security level s and type correctness requires that an ambient at security level s can only be traversed or opened by ambients at security levels at least s. Two independent partial orders are defined on security levels to control respectively opening and movement rights. A general analysis of how to encode mandatory security policies inside the Ambient Calculus is carried out in [BCC01b]. Difficulties therein in finding a natural interpretation of basic notions like read and write access led the authors to introduce a variant of the Ambient Calculus, the Boxed Ambients [BCC01a].

Mobile ambients have been analysed also using proof systems [CG00], abstract interpretations [HJNN99], and flow analysis [DLB00].

This work presents a bottom-up algorithm for the type system introduced in [DCS00] that, given an untyped process P, computes a minimal set of constraints on security levels (i.e. a minimal partial order on security levels) such that all actions occurring during runs of P are performed without violating the security level priorities. Such an algorithm is proved to be sound and complete. We also show it consistent with the calculus reductions, i.e. the constraints computed on a process reduct are related to those computed on the process itself via an *embedding of constraints* relation. As we shall see, in a sense a reduction loosens the computed constraints. Lastly, we show that our algorithm gives information about group typing as defined in [CGG00].

In order to simplify the definition of the algorithm and its analysis, we ignore coactions, distinctions between movement and opening rights, and restrict communication to names only (rather than capabilities). Correctness, completeness and 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 consider the network as a whole, *globally* typed entity. This is an unrealistic 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 the rules of our type system. 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 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 reductions involving 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:

- $|$ 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 judgements 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 represents the absence of message exchanges. We shall use

$P, Q ::=$	**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

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 $\mathcal{U}$. Generic types will be indicated by $\mathsf{T}, \mathsf{T}', \ldots$ and Υ will denote the set of types.

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 will be indicated by $\mathcal{TS}$.

We formalise typing environments as partial functions from names to types. Reflecting the classification of types introduced above, we use E, $\mathcal{E}$, and E to denote elements of, respectively, $\Xi = \mathcal{N} \rightharpoonup \Upsilon$, $\mathcal{N} \rightharpoonup \mathcal{T}$, and $\mathcal{N} \rightharpoonup \mathcal{TS}$. Also, if $\mathsf{E}(n) = s[\mathsf{T}]$, then n_{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 $\mathsf{E}' = \{(n, \mathsf{T})\}$.

We start by lifting functions from sets to ambient type schemes and to environments.

Definition 2.2 A map $\varphi\colon (dom(O) \to dom(O')) \cup (VarT \to \Upsilon)$ defines maps $\llcorner\varphi\lrcorner\colon \Upsilon \to \Upsilon$ and $\ulcorner\varphi\urcorner\colon \Xi \to \Xi$ as follows

$$\llcorner\varphi\lrcorner(\alpha) = \varphi(\alpha); \qquad \llcorner\varphi\lrcorner(s[\mathsf{T}]) = \varphi(s)[\llcorner\varphi\lrcorner(\mathsf{T})]$$
$$\ulcorner\varphi\urcorner(\mathsf{E}) = \llcorner\varphi\lrcorner \circ \mathsf{E}.$$

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

These elements will be employed in tuples either of the form $\langle \mathcal{E}, \mathcal{O}, S, T\rangle$ (for the type assignment system) or $\langle E, O, S, \tau\rangle$ (for the type inference algorithm). For the time being, we give the definition below using generic types so to capture all cases of interest at once.

Definition 2.3 (Relation $\lhd$) The binary relation $\lhd$ on tuples of the shape $\langle \mathsf{E}, O, S, \mathsf{T}\rangle$, with $S \subseteq dom(O)$ is defined by

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

if there exists $\varphi\colon (dom(O) \to dom(O')) \cup (VarT \to \Upsilon)$ such that

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

3 An assignment system for security level constraints

Our aim is to infer judgements on security policies expressed by permissions to cross boundaries and 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 judgement 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 by $\mathcal{E}$ 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 $\mathcal{E}, \mathcal{O}$, written $\mathcal{E}, \mathcal{O} \vdash_{ok} P$, if for all sequences of reductions $P \longrightarrow^*$ terminating with $(n\ \mathbf{in}\ m)$, $(n\ \mathbf{out}\ m)$, or $(n\ \mathbf{open}\ m)$ reductions, where m, n are free names in P, 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 check both $\mathcal{E}, \mathcal{O} \vdash_{ok} P$ and the soundness of name

exchanging. We thus derive more informative judgements, 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 a process type contains the security levels of the ambient names involved in capabilities possibly performed by the process; the second is the ambient type of the names it exchanges. Concerning the rules, (**in** $\vdash$), (**out** $\vdash$) and (**open** $\vdash$) 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, (**open** $\vdash$) checks that the type T of the ambient names exchanged by P and inside n coincide. Rule (Proc Amb $\vdash$), instead, verifies that:

- all security levels in 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

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 2.3.

Lemma 3.3 (Weakening) *If* $\langle \mathcal{E}, \mathcal{O}, S, T \rangle \lhd \langle \mathcal{E}', \mathcal{O}', S', T' \rangle$, *then*

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

Proof. We proceed by induction on the derivation of $\mathcal{E}, \mathcal{O} \vdash P : S, T$. We consider only one paradigmatic case.

If the last rule applied is (**in** $\vdash$) we have:

$$\frac{\mathcal{E}, \mathcal{O} \vdash P : S, T \quad n_{\mathcal{E}} = s_0 \quad \exists s_1 \in S. s_0 \leq_{\mathcal{O}} s_1}{\mathcal{E}, \mathcal{O} \vdash \mathbf{in}\ n.P : S, T}\ (\mathbf{in}\ \vdash)$$

By the induction hypothesis, we have $\mathcal{E}', \mathcal{O}' \vdash P : S', T'$. Let φ be the map which shows $\langle \mathcal{E}, \mathcal{O}, S, T \rangle \lhd \langle \mathcal{E}', \mathcal{O}', S', T' \rangle$. If $n_{\mathcal{E}'} = s'_0$, we get $\varphi(s_0) = s'_0$. Since $\forall s \in S\ \exists s' \in S'. \varphi(s) \leq_{\mathcal{O}'} s'$, we can find $s_2 \in S'$ such that $\varphi(s_1) \leq_{\mathcal{O}'} s_2$. Being φ monotone as a preorder map $\mathcal{O} \to \mathcal{O}'$ we get $s'_0 \leq_{\mathcal{O}'} s_2$. Therefore, we can infer, by rule (**in** $\vdash$), $\mathcal{E}', \mathcal{O}' \vdash \mathbf{in}\ n.P : S', T'$ as follows:

$$\frac{\mathcal{E}', \mathcal{O}' \vdash P : S', T' \quad n_{\mathcal{E}'} = s'_0 \quad \exists s_2 \in S'. s'_0 \leq_{\mathcal{O}'} s_2}{\mathcal{E}', \mathcal{O}' \vdash \mathbf{in}\ n.P : S', T'}\ (\mathbf{in}\ \vdash)$$

□

As is 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 induction on $\equiv$ and $\longrightarrow$. We only consider some interesting cases.

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

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

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

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

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

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

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

Fig. 2. Type Assignment System

Theorem 3.4 (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 3.5 (Subject Reduction of $\vdash$) *If $\mathcal{E},\mathcal{O}\vdash P:S,T$ and $P\longrightarrow Q$, then $\mathcal{E},\mathcal{O}\vdash Q:S,T$.*

Proof. The proof proceeds by induction on the derivation of $P\longrightarrow Q$. We present the cases for (*Red* **in**) and (*Red* **open**) in the details.

(*Red* **in**) $n[\mathbf{in}\ m.P\mid Q]\mid m[R]\longrightarrow m[n[P\mid Q]\mid R]$.

It is easy to check that a derivation for $\mathcal{E},\mathcal{O}\vdash n[\mathbf{in}\ m.P\mid Q]\mid m[R]:S,T$ must be of the shape

$$\text{(D1)}\ \frac{\vdots \qquad \dfrac{\mathcal{E},\mathcal{O}\vdash R:S'',T'' \quad \mathcal{E}(m)=s''[T''] \quad \forall s'\in S''.s'\leq_{\mathcal{O}} s''}{\mathcal{E},\mathcal{O}\vdash m[R]:S,T}}{\mathcal{E},\mathcal{O}\vdash n[\mathbf{in}\ m.P\mid Q]\mid m[R]:S,T}$$

where D1 is

$$\dfrac{\dfrac{\vdots}{\mathcal{E},\mathcal{O}\vdash P:S',T' \quad m_{\mathcal{E}}=s'' \quad \exists s'\in S'.s''\leq_{\mathcal{O}} s'} \qquad \vdots}{\dfrac{\mathcal{E},\mathcal{O}\vdash \mathbf{in}\ m.P:S',T' \qquad \mathcal{E},\mathcal{O}\vdash Q:S',T'}{\dfrac{\mathcal{E},\mathcal{O}\vdash \mathbf{in}\ m.P\mid Q:S',T' \quad \mathcal{E}(n)=s[T'] \quad \forall s'\in S'.s'\leq_{\mathcal{O}} s}{\mathcal{E},\mathcal{O}\vdash n[\mathbf{in}\ m.P\mid Q]:S,T}}}\ \text{(D1)}$$

We can now build a derivation for the judgement $\mathcal{E},\mathcal{O}\vdash m[n[P\mid Q]\mid R]:S,T$ as follows.

$$\dfrac{\dfrac{\dfrac{\vdots \qquad \vdots}{\mathcal{E},\mathcal{O}\vdash P:S',T' \quad \mathcal{E},\mathcal{O}\vdash Q:S',T'}}{\dfrac{\mathcal{E},\mathcal{O}\vdash P\mid Q:S',T' \quad \mathcal{E}(n)=s[T'] \quad \forall s'\in S'.s'\leq_{\mathcal{O}} s}{\mathcal{E},\mathcal{O}\vdash n[P\mid Q]:S'',T''}}\ \text{(D2)}}{\dfrac{\dfrac{\vdots}{\text{(D2)} \quad \mathcal{E},\mathcal{O}\vdash R:S'',T''}}{\mathcal{E},\mathcal{O}\vdash n[P\mid Q]\mid R:S'',T'' \quad \mathcal{E}(m)=s''[T''] \quad \forall s'\in S''.s'\leq_{\mathcal{O}} s''}}{\mathcal{E},\mathcal{O}\vdash m[n[P\mid Q]\mid R]:S,T}$$

(*Red* **open**) $\mathbf{open}\ n.P\mid n[Q]\longrightarrow P\mid Q$.

It is easy to check that a derivation for $\mathbf{open}\ n.P\mid n[Q]$ must be of the shape:

$$\dfrac{\dfrac{\vdots}{\mathcal{E},\mathcal{O}\vdash Q:S',T \quad \mathcal{E}(n)=s[T] \quad \forall s''\in S'.s''\leq_{\mathcal{O}} s}}{\text{(D3)} \qquad \mathcal{E},\mathcal{O}\vdash n[Q]:S,T}}{\mathcal{E},\mathcal{O}\vdash \mathbf{open}\ n.P\mid n[Q]:S,T}$$

with

$$\dfrac{\dfrac{\vdots}{\mathcal{E},\mathcal{O}\vdash P:S,T \quad \mathcal{E}(n)=s[T] \quad \exists s'\in S.s\leq_{\mathcal{O}} s'}}{\mathcal{E},\mathcal{O}\vdash \mathbf{open}\ n.P:S,T}\ \text{(D3)}$$

Conditions $\exists s'\in S.s\leq_{\mathcal{O}} s'$ and $\forall s''\in S'.s''\leq_{\mathcal{O}} s$ imply that $\forall s''\in S'\ \exists s'\in S.s''\leq_{\mathcal{O}} s'$. So we get $\langle\mathcal{E},\mathcal{O},S',T\rangle \ \triangleleft\ \langle\mathcal{E},\mathcal{O},S,T\rangle$ by choosing as φ the identity map. Then we can apply the Weakening Lemma (Lemma 3.3) in order to obtain $\mathcal{E},\mathcal{O}\vdash Q:S,T$. Finally we can build a derivation for $P\mid Q$ in the following way.

$$\dfrac{\dfrac{\vdots \qquad \vdots}{\mathcal{E},\mathcal{O}\vdash P:S,T \qquad \mathcal{E},\mathcal{O}\vdash Q:S,T}}{\mathcal{E},\mathcal{O}\vdash P\mid Q:S,T}$$

□

We close this section by stating formally that our type assignment system can type a process P with respect to a given environment and partial order only if P abides by them (according to Definition 3.1).

Theorem 3.6 (Soundness of $\vdash$) *If* $\mathcal{E}, \mathcal{O} \vdash P : S, T$ *then* $\mathcal{E}, \mathcal{O} \vdash_{ok} P$.

Proof. Let $\mathcal{E}, \mathcal{O} \vdash P : S, T$ and $P \longrightarrow^* Q \longrightarrow R$ where the last reduction step is $(n \ \mathbf{open} \ m)$ and m, n are free names in P. Then there is $Q' \equiv Q$ such that Q' has a subterm of the shape $n[\mathbf{open} \ m.S \mid m[U] \mid V]$. By Subject Reduction Theorem (Theorem 3.5) we get $\mathcal{E}, \mathcal{O} \vdash Q : S, T$, which implies $\mathcal{E}', \mathcal{O} \vdash n[\mathbf{open} \ m.S \mid m[U] \mid V] : S', T'$ for some $\mathcal{E}' \supseteq \mathcal{E}$, S', T'. Looking at the typing rules (**open** $\vdash$) and (Proc Amb $\vdash$), we get $n_{\mathcal{E}'} \geq_{\mathcal{O}} m_{\mathcal{E}'}$. We conclude $n_{\mathcal{E}} \geq_{\mathcal{O}} m_{\mathcal{E}}$ since by hypothesis m, n are free.
The proofs for the other cases are similar. □

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 $\langle E, O, S, \tau \rangle$ whose components are as described in the notational conventions of Section 2.

The bottom-up algorithm described in Fig. 3 enables to infer judgements 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 $\langle E, O, S, \tau \rangle$ 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 the details, let us observe that, although we finally want to express the constraints on P as a partial order of security levels, 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 $O \cup O'$.

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

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

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

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

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

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

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

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

Fig. 3. Minimal Constraints Algorithm

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

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

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

Definition 4.3 (Unification of Ambient Type Schemes) For $\alpha \neq \gamma$, 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

$s_i = s'_i$, $(1 \leq i \leq n)$, 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 security levels associated to the same ambient names. This is formalised 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.

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

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

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 $n \in dom(E) \cap dom(E')$.

Getting back to Fig. 3, we can now discuss the details of the formal definition of the algorithm. At the beginning, as formalised in the (Proc **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. Capabilities **in** n and **out** n 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.

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

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

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

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

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

4.1 Properties

As for the type assignment system, our algorithm enjoys the expected properties: it infers *isomorphic* effects for structural congruent processes and computes *looser* effects after reductions. Reduction in fact loosens the minimal constraints on a process, because these depend on the potential execution of capabilities that disappear under reduction. In the following definitions, we formalise the notions of isomorphic effects and loosened constraints.

Definition 4.4 (Partial Order Collapse) For $\mathsf{F} = \langle E, O, S, \tau \rangle$ an effect, its partial order collapse is the effect $\mathsf{F}_{\equiv} = \langle E/_{\equiv}, O/_{\equiv}, S/_{\equiv}, \tau/_{\equiv} \rangle$ where $\_/_{\equiv}$ is a quotient map [1] for the equivalence relation $\equiv =_{def} \leq_O \cap \geq_O$. To improve proof readability, we occasionally denote the tuple $\langle E/_{\equiv}, O/_{\equiv}, S/_{\equiv}, \tau/_{\equiv} \rangle$ as $\langle \overline{E}, \overline{O}, \overline{S}, \overline{\tau} \rangle$.

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 $\mathsf{F}_{\equiv}$.

[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.

Definition 4.5 (The Relation $\bowtie$ on Ordered Effects) Let $\mathsf{F} = \langle E, O, S, \tau\rangle$ and $\mathsf{F}' = \langle E', O', S', \tau'\rangle$ be ordered effects. We say that $\mathsf{F} \bowtie \mathsf{F}'$ if $\mathsf{F} \lhd \mathsf{F}'$ (as defined in Definition 2.3) via a map φ which is an isomorphism between S and S' and between O and O'.

Definition 4.6 (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 $\mathsf{F}_{\equiv} \bowtie \mathsf{F}'_{\equiv}$.

We can prove that $\cong$ is a congruence with respect to all the effect transformations used to define our minimal constraints algorithm.

Lemma 4.7 *Let* $\mathsf{F} \cong \mathsf{F}'$, $\Vdash P : \mathsf{F}$ *and* $\Vdash P : \mathsf{F}'$. *If* $\Vdash P : \mathsf{F}$ *is a premise and* $\Vdash Q : \mathsf{F}''$ *is the conclusion of an arbitrary rule of the minimal constraints algorithm, then if we apply the same rule using as premise* $\Vdash P : \mathsf{F}'$ *instead of* $\Vdash P : \mathsf{F}$, *we get as conclusion* $\Vdash Q : \mathsf{F}'''$ *with* $\mathsf{F}'' \cong \mathsf{F}'''$.

As formally stated by the lemma below, the system $\Vdash$ infers an effect for each well-formed process: at worst, all security levels will be equated, making our security policy void. The proof by structural induction on processes using previous lemma is standard.

Lemma 4.8 (Termination) *For every* P *there exists* F, *unique up to* $\cong$, *such that* $\Vdash P : \mathsf{F}$.

In analogy to Theorems 3.4 and 3.5 and we get:

Theorem 4.9 (Subject Congruence of $\Vdash$) *If* $P \equiv Q$, $\Vdash P : \mathsf{F}$ *and* $\Vdash Q : \mathsf{F}'$, *then* $\mathsf{F} \cong \mathsf{F}'$.

Proof. The proof by induction on $\equiv$ using Lemmas 4.7 and 4.8 is easy. □

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). Let $(\cdot)^*$ be the map replacing *Shh* to type variables defined in Definition 5.1.

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

$$\begin{array}{ccc} \Vdash P\colon \langle E, O, S, \tau\rangle & & \Vdash Q\colon \langle E', O', S', \tau'\rangle \\ \text{(by Th. 5.2)}\ \Downarrow & & \Uparrow\ \text{(by Th. 5.3)} \\ \overline{E}^*, \overline{O} \vdash P : \overline{S}, \overline{\tau}^* & \xRightarrow[\text{(by Th. 3.5)}]{} & \overline{E}^*, \overline{O} \vdash Q : \overline{S}, \overline{\tau}^* \end{array}$$

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

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 to the conditions of Definition 2.3 the requirement of φ to be *injective*. This, however, turns out to be

impossible, again because by performing a reduction we make the number of capabilities in a process decrease. Certainly, the fewer the capabilities in a process, the fewer the pairs in the preorder inferred by our algorithm. However, as the quotient-order construction is non monotone, a smaller preorder does not imply a smaller corresponding partial order. This is well illustrated by process P below, where reduction gets us a smaller preorder, but a bigger partial order.

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

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 Fig. 4). If we reduce P via a (*Red* **open**) reduction, we get

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

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

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 4.6) any possible typing in the type assignment system. To connect the the algorithm with the type assignment system we need to replace type variables by *Shh*.

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

Theorem 5.2 (Correctness**)** *Let P be a process. IF $\Vdash P : \langle E, O, S, \tau\rangle$ then $\overline{E}^*, \overline{O} \vdash P : \overline{S}, \overline{\tau}^*$.*

Proof. The proof is by induction on derivations. The details for rule (Proc Par $\Vdash$) are as follows.

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

By induction hypothesis, we have $\overline{E}^*, \overline{O} \vdash P : \overline{S}, \overline{\tau}^*$ and $\overline{E'}^*, \overline{O'} \vdash Q : \overline{S'}, \overline{\sigma}^*$. By weakening (Lemma 3.3), we have:

$$\overline{((E \cdot E')\{\sigma = \tau\})}^*, \overline{(O \bigoplus_{E,E'} O')[\sigma = \tau]} \vdash P : \overline{S \cup S'}, \overline{max(\sigma,\tau)}^*$$

and

$$\overline{((E \cdot E')\{\sigma = \tau\})}^*, \overline{(O \bigoplus_{E,E'} O')[\sigma = \tau]} \vdash Q : \overline{S \cup S'}, \overline{max(\sigma,\tau)}^*.$$

The statement is proved just by applying the (Proc Par $\vdash$) rule. $\square$

The order 'computed' by the algorithm is minimal in the following sense.

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

Proof. By induction on derivations. All cases are quite simple, apart from those of rules (Proc Par) and (Proc Amb), which require some technical ingenuity. We first consider the (Proc Par) rules.

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

$$\frac{\mathcal{E}, \mathcal{O} \vdash P : S'', T \quad \mathcal{E}, \mathcal{O} \vdash Q : S'', T}{\mathcal{E}, \mathcal{O} \vdash P \mid Q : S'', T}\ (\text{Proc Par } \vdash)$$

Notice that by construction $S \cap S' = \varnothing$ and $FV(E, \tau) \cap FV(E', \sigma) = \varnothing$, where FV gives the set of (free) type variables in ambient type schemes and environments. Therefore also $\overline{S} \cap \overline{S'} = \varnothing$ and $FV(\overline{E}, \overline{\tau}) \cap FV(\overline{E'}, \overline{\sigma}) = \varnothing$. By induction there are φ_P and φ_Q such that:

- $\varphi_P(\overline{E}) \subseteq \mathcal{E}, \varphi_Q(\overline{E'}) \subseteq \mathcal{E}$;
- $\forall s \in \overline{S}\ \exists s' \in S''. \varphi_P(s) \leq_{\mathcal{O}} s'$;
- $\forall s \in \overline{S'}\ \exists s' \in S''. \varphi_Q(s) \leq_{\mathcal{O}} s'$;
- φ_P is monotone as a preorder map $\overline{O} \to \mathcal{O}$;
- φ_Q is monotone as a preorder map $\overline{O'} \to \mathcal{O}$;
- $\varphi_P(\overline{\tau}) = \varphi_Q(\overline{\sigma}) = T$.

We define a map φ as follows:

$$\varphi(s) = \begin{cases} \varphi_P(s) \text{ if } s \in \overline{S} \\ \varphi_Q(s) \text{ if } s \in \overline{S'} \end{cases}$$

$$\varphi(\alpha) = \begin{cases} \varphi_P(\alpha) \text{ if } \alpha \in FV(\overline{E}, \overline{\tau}) \\ \varphi_Q(\alpha) \text{ if } \alpha \in FV(\overline{E'}, \overline{\sigma}) \end{cases}$$

Then

- $\varphi(\overline{(E \cdot E')\{\sigma = \tau\}}) \subseteq \mathcal{E}$;
- $\forall s \in \overline{S \cup S'}\ \exists s' \in S''. \varphi(s) \leq_{\mathcal{O}} s'$;
- φ is monotone as a preorder map $(\overline{O \bigoplus_{E,E'} O')[\sigma = \tau]} \to \mathcal{O}$;
- $\varphi(\overline{max(\sigma, \tau)}) = T$.

For the (Proc Amb) rules, we proceed as follows

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

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

By induction hypothesis, there is φ_P such that:

- $\varphi_P(\overline{E}) \subseteq \mathcal{E}$;
- $\forall s \in \overline{S}\ \exists s''' \in S''.\varphi_P(s) \leq_{\mathcal{O}''} s'''$;
- φ_P is monotone as a preorder map $\overline{O} \to \mathcal{O}$;
- $\varphi_P(\overline{\tau}) = T$.

We define

$$\varphi(s') = \begin{cases} \varphi_P(s') \text{ if } s' \in \overline{S} \\ s'' \qquad \text{otherwise} \end{cases}$$

$$\varphi(\beta) = \begin{cases} \varphi_P(\beta) \text{ if } \beta \in FV(\overline{E}, \overline{\tau}) \\ T' \qquad \text{if } \beta \equiv \alpha \\ T \qquad \text{otherwise} \end{cases}$$

If $n \in dom(E)$, from $\varphi_P(\overline{E}) \subseteq \mathcal{E}$ and $\varphi_P(\overline{\tau}) = T$, we get $\varphi_P(s) = s''$, and $\varphi_P(\overline{\sigma}) = T$. Otherwise, $s \notin \overline{S}$ implies $\varphi(s) = s''$ and σ is a type variable that does not belong to $FV(\overline{E}, \overline{\tau}) \cup \{\alpha\}$ and hence, by definition of φ, we have that $\varphi(\overline{\sigma}) = T$. In both cases we obtain:

- $\varphi(\overline{X\{\sigma = \tau\}}) \subseteq \mathcal{E}$;
- φ is monotone as a preorder map $\overline{(S^{\uparrow s} \oplus O)[\sigma = \tau]} \to \mathcal{O}$;
- $\varphi(\alpha) = T'$.

This concludes our proof, being the remaining clause trivially satisfied for the empty set. □

6 Relations with group types

In this section, we draft a relationship 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

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

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

$$\frac{\Gamma \vdash_G P : S,T \quad n_\Gamma = s \quad s \in S}{\Gamma \vdash_G \mathbf{in}\ n.P : S,T}\ (\mathbf{in}\ \ \vdash_G)$$

$$\frac{\Gamma \vdash_G P : S,T \quad n_\Gamma = s \quad s \in S}{\Gamma \vdash_G \mathbf{out}\ n.P : S,T}\ (\mathbf{out}\ \ \vdash_G)$$

$$\frac{\Gamma \vdash_G P : S,T \quad \Gamma(n) = s[S,T] \quad s \in S}{\Gamma \vdash_G \mathbf{open}\ n.P : S,T}\ (\mathbf{open}\ \ \vdash_G)$$

$$\frac{\Gamma \vdash_G P : S,T \quad \Gamma(n) = s[S,T]}{\Gamma \vdash_G n[P] : S',T'}\ (\text{Proc Amb} \vdash_G)$$

$$\frac{\Gamma \vdash_G P : S,T \quad \Gamma \vdash_G Q : S,T}{\Gamma \vdash_G P \mid Q : S,T}\ (\text{Proc Par} \vdash_G) \qquad \frac{\Gamma \vdash_G P : S,T}{\Gamma \vdash_G !P : S,T}\ (\text{Proc Repl} \vdash_G)$$

$$\frac{\Gamma \cdot x : T \vdash_G P : S,T}{\Gamma \vdash_G (x).P : S,T}\ (\text{Proc Input} \vdash_G) \qquad \frac{}{\Gamma \cdot n : T \vdash_G \langle n \rangle : S,T}\ (\text{Proc Output} \vdash_G)$$

Fig. 5. Type Assignment System for Group Types

Let $\mathcal{GT}$ stand for the set of group types. The typing rules of [CGG00] with the types in $\mathcal{GT}$ are given in Fig. 5 using the current notational conventions. There is no surprise here, 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 its downward closure as

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

and the mapping $\mu_\mathcal{O}\colon \Upsilon \to \mathcal{GT}$ such that

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

We lift $\mu_\mathcal{O}$ to environments as usual.

We can prove that $\Vdash P : \langle E, O, S, \tau \rangle$ 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 $\vdash$. We only work out the paradigmatic case when the last rule applied is (Proc Amb $\vdash$). Let $\mathcal{E}, \mathcal{O} \vdash n[P] : S', T'$

be the conclusion. This means that, for some S and T, we have derived the judgement $\mathcal{E}, \mathcal{O} \vdash P : S, T$ and that $E(n) = s[T]$, with 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 the Weakening Lemma (Lemma 3.3), we can derive the judgement $\mathcal{E}, \mathcal{O} \vdash P : \mathcal{S}_{\mathcal{O}}(s), T$. Applying the induction hypothesis, the judgement $\mu_{\mathcal{O}}(E) \vdash_G P : \mathcal{S}_{\mathcal{O}}(s), \mu_{\mathcal{O}}(T)$ is derivable and $\mu_{\mathcal{O}}(E)(n) = s[\mathcal{S}_{\mathcal{O}}(s), \mu_{\mathcal{O}}(T)]$. We can conclude $\mu_{\mathcal{O}}(E) \vdash_G n[P] : \mathcal{S}_{\mathcal{O}}(S'), \mu_{\mathcal{O}}(T')$. □

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

Proof. Simply by the correctness of the type inference algorithm with respect to the type assignment system (Theorem 5.2) using the previous lemma. □

7 Conclusions and Future Works

In the scenario we considered an ambient can enter another when the former has a greater priority. Although this is quite a reasonable assumption, one has to be aware that once the ambient has entered its greater security clearance enables it to do, so to speak, whatever it likes. This is to be avoided if we are to yield a safer, a more realistic, and thoroughly desirable situation in which an ambient successfully controls 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 *POPL'01*, pages 222–235. ACM Press, 2001.

[BCC01a] Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Boxed ambients. In *TACS'01*, volume 2215 of *Lecture Notes in Computer Science*, pages 38–63. Springer-Verlag, 2001.

[BCC01b] Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Reasoning about security in mobile ambients. In *CONCUR'01*, volume 2154 of *Lecture Notes in Computer Science*, pages 102–120. Springer-Verlag, 2001.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *FoSSaCS'98*, 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. ACM Press, 1999.

[CG00] Luca Cardelli and Andrew D. Gordon. Anytime, anywhere. modal logics for mobile ambients. In *POPL'00*, pages 365–377. ACM Press, 2000.

[CGG99] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Mobility types for mobile ambients. In *ICALP'99*, 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 *TCS'00*, 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 *ASIAN'00*, volume 1961 of *Lecture Notes in Computer Science*, pages 215–236. Springer-Verlag, 2000.

[DLB00] Pierpaolo Degano, Francesca Levi, and Chiara Bodei. Safe ambients: Control flow analysis and security. In *ASIAN'00*, volume 1961 of *Lecture Notes in Computer Science*, pages 199–214. Springer-Verlag, 2000.

[HJNN99] R. R. Hansen, J. G. Jensen, F. Nielson, and H. Riis Nielson. Abstract interpretation of mobile ambients. In *SAS'99*, volume 1694 of *LNCS*, pages 134–148. Springer-Verlag, 1999.

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

[Zim00a] Pascal Zimmer. On the expressiveness of pure mobile ambients. In *EXPRESS'00*, volume 39 of *Electronic Notes in Theoretical Computer Science*. Elsevier, 2000.

[Zim00b] Pascal Zimmer. Subtyping and typing algorithms for mobile ambients. In *FoSSaCS'00*, volume 1784 of *Lecture Notes in Computer Science*, pages 375–390. Springer-Verlag, 2000.