Causal Type System for Ambient Movements*

Torben Amtoft**

Heriot-Watt University
tamtoft@cee.hw.ac.uk
www.cee.hw.ac.uk/~tamtoft

Abstract. The Ambient Calculus was developed by Cardelli and Gordon as a formal framework to study issues of mobility and migrant code. We present a type system for the calculus, parameterized by security constraints expressing where a given ambient may reside and where it may be dissolved. A subject reduction property then guarantees that a well-typed process never violates these constraints; additionally it ensures that communicating subprocesses agree on their "topic of conversation". Based on techniques borrowed from finite automata theory, type checking of type-annotated processes is decidable. Under certain quite natural restrictions, type inference is also possible.
The type system employs a notion of causality in that processes are assigned "behaviors". This significantly increases the precision of the analysis and compensates for the lack of "co-capabilities" (an otherwise increasingly popular extension to the ambient calculus); also it allows an ambient to hold multiple topics of conversation.

1 Introduction

The ambient calculus was developed by Cardelli & Gordon [CG98] as a framework for mobile computation where "ambients", containing active processes (and not just passive code), can move around—in and out of other ambients. In this view, the following question becomes a main security concern:

is it possible for a given ambient to end up inside another given ambient?

Various approaches have been taken to analyze the ambient calculus. First came the type system of [CG99], where the focus was on ensuring that within each ambient there is a well-defined "topic of conversation"; later this system was

* The appendices are not part of this extended abstract, limited to 15 pages. They are only attached as a convenience to the interested reader.

** The author is supported by the DART project (EC grant IST-2001-33477). Almost all of this work was carried out while he was at Boston University, supported by NSF EIA grant 9806745/9806746/9806747/9806835 and partially supported by Sun grant EDUD-7826-990410-US. He would like to thank Michele Bugliesi, Assaf Kfoury, and Santiago Pericas-Geertsen for inspiring discussions, and the latter two for commenting upon a draft of this paper.

extended [CGG99] so as to express also mobility properties. To give an (approximate) answer to the question above, many tools have been employed: flow analysis [NNHJ99], abstract interpretation [HJNN99,LM01], tree grammars [NN01], 3-valued logic [NNS00].

In this paper we address the issue using a type system where a process is assigned a so-called "behavior". A behavior[1] incorporates causality information, thus enabling one to express that one action precedes another. In [AKPG01] this notion was used to extend [CG99] so as to allow *consecutive* topics of conversation (akin to session types for the π-calculus [GH99]). Since behaviors can be viewed as finite automata, well-known techniques can be used for type checking and inference.

The original ambient calculus is quite liberal in that ambients may enter, exit, and even open (in effect dissolve), other ambients without their explicit permission. In particular the opening capability is problematic for security[2], as seen by the reduction rule

$$\mathsf{open}\ n.P \mid n[Q] \rightarrow P \mid Q$$

which shows that the process Q inside the opened ambient n will run on the same level, and therefore with the same control power, as the opening process P. Also for developing a precise analysis, the opening capability is problematic, as we essentially need to split the work of Q into two parts: what is done before n was opened (not influencing P), and what is done after n is opened (influencing P). As pointed out in [CGG99], conservatively assuming that all actions of Q *might* take place *after* n is opened prevents us from declaring immobile an ambient that opens an entering (hence mobile) ambient.

A solution, originating[3] in [LS00] and employed in [DCS00,AKPG01,GYY01], is to add an extra "co-open" construct enabling an ambient to specify exactly at which point of its computation it will allow itself to be dissolved.

It is possible, however, to stick with the original calculus, and still get a quite precise analysis. For that purpose, it is essential to keep track of how the location of an ambient changes over time, thus allowing an estimate concerning when it can be opened. This information can (cf. above) be achieved by a variety of methods [NN01,NNS00,LM01]; the main contribution of this paper is to show how it can be achieved using the technology of type systems and finite automata. A detailed comparison with other systems is left for future work.

Our type system is (implicitly) parameterized with respect to a set of security constraints, listing which interactions between ambients are allowed. One can view these constraints as prescriptive (thus provided by the user); establishing that a process is well-typed therefore verifies that no other interactions may

[1] The concept dates back at least to [NN94] where it was used in the context of Concurrent ML, the type system of which has a very different flavor from the one considered in this paper.

[2] For this reason, it is argued in [BCC01] that opening should be disallowed altogether.

[3] The *safe ambients* proposed in [LS00] additionally include the co-capabilities "co-in" and "co-out" which also have been used, e.g., in [BC01].

happen. Alternatively, one can take a descriptive view: a type inference algorithm might deduce the least set of constraints needed for typability.

As in [CGG00] we shall employ the notion of *groups*, with the intention that each ambient belongs to exactly one group $g \in \mathsf{Grps}$ (a finite set). We shall use G to range over sets of groups. *Dynamic* security constraints are then expressed using the predicate $\mathcal{O}(g_0, g)$ saying that an ambient of group g_0 is allowed to be opened while directly enclosed in an ambient of group g, whereas *static* security constraints are expressed using the predicate $\mathcal{I}(g_0, g)$ saying that an ambient of group g_0 is allowed to be directly enclosed in an ambient of group g. (We would expect that $\mathcal{O}(g_0, g)$ implies $\mathcal{I}(g_0, g)$.) We write $\mathcal{I}(g_0, G)$ to mean that $\mathcal{I}(g_0, g)$ for every g in G.

Our assumption is that the top-level process consists of a set of ambients put in parallel, and implicitly located within a "global" ambient of group @ (this property is preserved under reduction).

Example 1. Consider the "Trojan horse" from [BC01], which stripped of all co-capabilities, and with each ambient annotated with a unique group, looks like

$$a[\mathsf{open}\ b.\mathsf{in}\ c]^A \mid b[\mathsf{in}\ a.\mathsf{in}\ d]^B \mid c[P \mid d[Q]^D]^C$$

At run-time, b enters a where it is opened, leaving us with the configuration

$$a[\mathsf{in}\ c \mid \mathsf{in}\ d]^A \mid c[P \mid d[Q]^D]^C$$

Now a can enter c, and from there further enter d. This might not have been what the "owner" of c had in mind when "allowing" a to enter, as by looking only at the "code" of a there is no sign of a desire to enter d.

Our type system detects this attack, as in order for the process to be well-typed we have to make the type assignment

$$a : \mathsf{amb}^A_{@CD}$$

with the interpretation that a is of group A and may be immediately enclosed in[4] either @, C, or (the sign of danger) D. As we have seen, this is in fact an exact estimate. Similarly, assuming that P and Q do nothing of interest, we have

$$c : \mathsf{amb}^C_{@} \text{ and } d : \mathsf{amb}^D_C$$

The type assigned to b is more sophisticated:

$$b : \mathsf{amb}^B_{@A}[A :^C \mathsf{enter}(D)]$$

with the interpretation that b may be immediately enclosed in either @ or A, and that after being opened inside A the process thereby unleashed has behavior ${}^C\mathsf{enter}(D)$, that is it will steer its surrounding ambient A from being inside C into being inside D. We shall now briefly argue why the above is a valid typing for b, assuming the dynamic security constraint is given by $\mathcal{O}(B, A)$ and assuming

[4] Somewhat sloppily, we shall write $g_1 \ldots g_n$ for $\{g_1, \ldots, g_n\}$, and write "inside g" when we really mean "inside an ambient of group g".

an appropriate definition of the static security constraints. First note that in order for an ambient to enter a it must be a "sibling" of a and therefore inside a group that also a may be immediately enclosed in, that is (cf. the type of a) inside either @, C, or D. Therefore the capability $\mathsf{in}\ a$ can be given behavior ${}^{@_{CD}}\mathsf{enter}(A)$; and combined with similar reasoning for $\mathsf{in}\ d$ the process inside b can be assigned the behavior

$${}^{@_{CD}}\mathsf{enter}(A).{}^{C}\mathsf{enter}(D).$$

As b is initially within @, we read that b moves from @ to A and then suddenly shows up in $C\ldots$ the explanation being that b has been *opened* while inside A. We infer that b does in fact behave as predicted by its type: it can be enclosed in only @ or A, and after being opened[5] the process unleashed, to be executed by A, will behave as ${}^{C}\mathsf{enter}(D)$.

Example 2. Consider the variant of Example 1, also given in [BC01] (again we omit co-capabilities):

$$a[\mathsf{in}\ c]^A \mid b[\mathsf{in}\ a.\mathsf{out}\ a.\mathsf{in}\ d]^B \mid c[P \mid d[Q]^D]^C$$

Here one possible execution sequence is that b is carried by a into c, at which point b exits a and enters d—still, this might not be what the owner of c had in mind. The least typing assignment is given by

$$a : \mathsf{amb}^{A}_{@_C},\ b : \mathsf{amb}^{B}_{@_{ACD}},\ c : \mathsf{amb}^{C}_{@},\ d : \mathsf{amb}^{D}_{C}.$$

Thus the type of a does not reveal that it carries with it a process entering d; instead one has to examine the type of b which cannot be typed unless $\mathcal{I}(B, D)$ is allowed. This is in some sense similar to what [CGG00] would do, whereas the attack is immediately detected (but using co-capabilities) in [BC01].

Example 3. As a final example, consider the firewall first presented in [CG98]:

$$w[k[\mathsf{out}\ w.\mathsf{in}\ k'.\mathsf{in}\ w]^B \mid \mathsf{open}\ k'.\mathsf{open}\ k''.P]^A \mid k'[\mathsf{open}\ k.k''[Q]^D]^C$$

This process is deterministic: k will exit w and enter k' where it is dissolved; then k' will enter w where it is dissolved and afterwards k'' (carried into w by k') is also dissolved. We thus need the security constraints to allow:

$$\mathcal{O}(B, C),\ \mathcal{O}(C, A),\ \mathcal{O}(D, A)$$
$$\mathcal{I}(B, A),\ \mathcal{I}(D, C),\ \mathcal{I}(B, C),\ \mathcal{I}(C, A),\ \mathcal{I}(D, A),\ \mathcal{I}(A, @),\ \mathcal{I}(B, @),\ \mathcal{I}(C, @).$$

Assuming P and Q do nothing of interest, we can type the ambients as follows:

$$E(k) = \mathsf{amb}^{B}_{AC@}[C :^{@} \mathsf{enter}(A)] \qquad E(w) = \mathsf{amb}^{A}_{@}$$
$$E(k') = \mathsf{amb}^{C}_{A@}[A : \varepsilon] \qquad E(k'') = \mathsf{amb}^{D}_{AC}[A : \varepsilon]$$

[5] Which can only happen at one point, so we get exactly as precise information as if b had contained the process $\mathsf{in}\ a.\mathsf{co\text{-}open}\ b.\mathsf{in}\ d$.

Let $n, m \in \mathsf{Names}$ range over names.
Let $\xi, \chi \in \mathsf{Tags}$ range over tags.

Expressions $M \in \mathsf{Exp} ::= n \mid \mathsf{in}\ M \mid \mathsf{out}\ M \mid \mathsf{open}\ M \mid \epsilon \mid M_1.M_2$

Processes

$$P, Q, R \in \mathsf{Proc} ::= \mathbf{0} \mid P_1 \mid P_2 \mid !P \mid (\nu n : \tau).P \mid M.P \mid M[\![P]\!]^{\xi}$$
$$\mid \quad (n_1 \ldots n_k : \tau_1 \ldots \tau_k).P \mid \langle M_1 \ldots M_k \rangle \qquad (k \geqslant 0)$$

Fig. 1. Syntax of **AC**.

In the non-causal analysis of [NNHJ99], A as well as C can contain all of A, B, C, D. As several other analyses in the recent literature, we are thus much more precise.

The rest of this paper is organized as follows: Sect. 2 defines the ambient language and its semantics; Sect. 3 defines the entities used by the type system which is then presented in Sect. 4 and shown semantically sound in Sect. 5. Type checking is addressed in Sect. 6. For the interested reader, Appendix B sketches the more difficult problem of type inference.

2 The Language

Figure 1 shows the syntax of our language **AC**. A process $P \in \mathsf{Proc}$ is basically as in [CG99]: there are constructs for parallel composition ($P_1 \mid P_2$), replication ($!P$), restriction ($(\nu n : \tau).P$), k-ary input and output, and prefixing with capabilities ($M.P$); when there is no ambiguity we shall write M for $M.\mathbf{0}$ where $\mathbf{0}$ is the inactive process. (As in [AKPG01] it would be possible to add a functional core on the expression level, but to keep the exposition simple we have refrained from doing so.) Note that for the binding constructs, the name n being bound is annotated with a type (to be defined in Sect. 3).

Since the formulation of semantic soundness (Sect. 5) requires[6] us to distinguish between ambients with the same name (as is customary in approaches using flow logic, such as [NNHJ99]), we shall annotate each ambient $M[\![P]\!]^{\xi}$ with a *tag* ξ. This conforms with the notation used in the Introduction, assuming that there exists a function $\mathit{group} : \mathsf{Tags} \to \mathsf{Grps}$ allowing one to read the group of an ambient from its tag.

2.1 Operational Semantics

The semantics of **AC** is presented in Fig. 2. We write $P_1 \equiv P_2$ to denote that P_1 and P_2 are equivalent, modulo consistent renaming of bound names (which may

[6] This is to ensure that the behavior of the ambient "responsible" for a given reduction can evolve into a new behavior (essentially a suffix of the old) without affecting the behavior of ambients not participating in the reduction.

be needed to apply (Red Comm)) and modulo "syntactic rearrangement" (we have, e.g., that $P \mid \mathbf{0} \equiv P$ and $P \mid Q \equiv Q \mid P$). The definition is as in [CG99], except that (in order to establish "subject congruence", cf. Lemma 6) we omit the rule $!P \equiv P \mid !P$ and instead allow this "unfolding" to take place via the rule (Red Repl) (as also seen in, e.g., [NNS00]). To enable the reduct to be uniquely tagged, this rule actually permits $!P$ to be unfolded into $P' \mid !P$ where $P \equiv_{\mathrm{t}} P'$, that is P and P' are "equal modulo *group*" (for instance, $n\mathbf{[0]}^{\xi} \equiv_{\mathrm{t}} n\mathbf{[0]}^{\chi}$ if $\mathit{group}(\xi) = \mathit{group}(\chi)$).

We write $P_1 \xrightarrow{\ell} P_2$ if P_1 reduces in one step to P_2 by performing "an action described by ℓ". Note that the definition of this relation deviates from the standard in that in (Red Open) and (Red Comm) we provide the surrounding ambient so as to record in ℓ *where* the opening or communication has taken place. We use a notion of "process evaluation contexts" to succinctly describe the place in a process where a subprocess (Red PctxtP) is reduced. Note that $P \xrightarrow{\ell} Q$ does not imply that $M.P \xrightarrow{\ell} M.Q$ since the capability M must be executed before P can be activated.

3 Types and Behaviors

Our type system assigns types ($\tau \in \mathsf{Typ}$) to expressions and behaviors ($b \in \mathsf{Beh}$) to processes; these entities are recursively defined in Fig. 3 using auxiliary notions such as *actions* and *behavior contexts*.

In the definition of actions, we use H to range over *upwards closed* sets of groups, where G is upwards closed if $g \in G$ and $\mathcal{O}(g, g')$ implies $g' \in G$. The intuition is that if an ambient n can be directly enclosed in g, and g' can open g, then n might also be directly enclosed in g'. We let $g^{\uparrow}$ denote the least upwards closed set containing g. In Example 1, we have $B^{\uparrow} = \{A, B\}$.

Rather than defining behaviors syntactically, using certain constructors (as in [AKPG01]), we shall take a more semantic approach and define a behavior as a regular (possibly infinite) and non-empty[7] set of finite traces. This gives the user more freedom in specification, and fits well with type checking which (Sect. 6) is carried out using finite automata. Nevertheless, it is still convenient to define certain semantic operators on behaviors:

$$\begin{aligned} \varepsilon &= \{\bullet\} \\ a.b &= \{a \diamond tr \mid tr \in b\} \\ b_1 \mid b_2 &= \textstyle\bigcup_{tr_1 \in b_1, tr_2 \in b_2} tr_1 \parallel tr_2 \end{aligned}$$

Here $\bullet$ denotes the empty sequence, $a \diamond tr$ denotes concatenation, and $tr_1 \parallel tr_2$ denotes all traces that can be formed by arbitrarily interleaving tr_1 with tr_2. The operator $\mid$ is associative and commutative, with $\{\bullet\}$ as neutral element. When there is no ambiguity, we write a for $a.\varepsilon$.

An ambient n has a type of the form $\mathsf{amb}^{g}_{H}[\{g_i : b_i\}_{i \in I}]$, which (cf. the Introduction) should be read as follows: it has group g, can be directly enclosed

[7] Since for semantic soundness, we need behaviors to be inhabited.

Reduction Labels

$\ell ::= \epsilon$ only "internal computation" is performed
$\mid\ \xi : \mathsf{enter}\ \chi$ the ambient ξ is steered into the ambient χ
$\mid\ \xi : \mathsf{exit}\ \chi$ the ambient ξ is steered out of the ambient χ
$\mid\ \xi : \mathsf{open}\ \chi$ a process controlling the ambient ξ opens the ambient χ
$\mid\ \xi : \mathsf{comm}\ \sigma$ inside the ambient ξ values of type σ are communicated

Process Evaluation Contexts

$\mathcal{PC} ::= \Box \mid \mathcal{PC} \mid P \mid (\nu n : \tau).\mathcal{PC} \mid n[\![\mathcal{PC}]\!]^{\xi}$

Notation: $\mathcal{PC}\lfloor P \rfloor$ is the process resulting from replacing $\Box$ by P in $\mathcal{PC}$.

Reduction Rules

In (Red Comm), $P[n_i := M_i]$ denotes the result of substituting $M_1 \ldots M_k$ for $n_1 \ldots n_k$ in P. We assume that the bound names have been renamed so as to avoid capture of free names.

$$m[\![\mathsf{in}\ n.P \mid Q]\!]^{\xi} \mid n[\![R]\!]^{\chi} \xrightarrow{\xi:\mathsf{enter}\ \chi} n[\![m[\![P \mid Q]\!]^{\xi} \mid R]\!]^{\chi} \qquad (\mathsf{Red\ In})$$

$$n[\![m[\![\mathsf{out}\ n.P \mid Q]\!]^{\xi} \mid R]\!]^{\chi} \xrightarrow{\xi:\mathsf{exit}\ \chi} m[\![P \mid Q]\!]^{\xi} \mid n[\![R]\!]^{\chi} \qquad (\mathsf{Red\ Out})$$

$$m[\![\mathsf{open}\ n.P \mid n[\![Q]\!]^{\chi} \mid R]\!]^{\xi} \xrightarrow{\xi:\mathsf{open}\ \chi} m[\![P \mid Q \mid R]\!]^{\xi} \qquad (\mathsf{Red\ Open})$$

$$m[\![(n_1 \ldots n_k : \tau_1 \ldots \tau_k).P \mid \langle M_1 \ldots M_k \rangle \mid Q]\!]^{\xi} \xrightarrow{\xi:\mathsf{comm}\ \times(\tau_1, \ldots, \tau_k)} m[\![P[n_i := M_i] \mid Q]\!]^{\xi} \qquad (\mathsf{Red\ Comm})$$

$$!P \xrightarrow{\epsilon} P' \mid !P \quad \text{if } P' \equiv_{\mathrm{t}} P \qquad (\mathsf{Red\ Repl})$$

$$\text{If } P \xrightarrow{\ell} Q \text{ then } \mathcal{PC}\lfloor P \rfloor \xrightarrow{\ell} \mathcal{PC}\lfloor Q \rfloor \qquad (\mathsf{Red\ PctxtP})$$

$$\text{If } P' \equiv P, P \xrightarrow{\ell} Q, Q \equiv Q' \text{ then } P' \xrightarrow{\ell} Q' \qquad (\mathsf{Red}\ \equiv)$$

Fig. 2. Operational Semantics.

inside ambients of groups belonging to H, and after being opened inside g_i it behaves as b_i.

A capability has a type of the form $\mathsf{cap}[B]$ where B is a behavior context, that is a "behavior with a hole inside" (a convenient device which was introduced in [AKPG01] and used also in [GYY01]). The idea can best be illustrated by an example: if n has type $\mathsf{amb}^{g}_{H}[g_0 : b_0]$ and P has behavior b, then the process $\mathsf{open}\ n.P$ will open g and then run b_0 in parallel with b. And in fact, referring to Fig. 4, ${}_{g_0}\mathsf{open}(g).(b_0 \mid b)$ is the result of (Proc Action) plugging[8] b into $\mathsf{cap}[{}_{g_0}\mathsf{open}(g).(b_0 \mid \Box)]$ which by (Exp Open) is the type of $\mathsf{open}\ n$.

We shall employ the notion of *level*: an entity has level i if i is an upper bound of the depth of nested occurrences of $\mathsf{amb}\_[\_]$ or $\mathsf{cap}[\_]$ within it. (We

[8] To be more precise, given B and b we define the behavior $B\lfloor b \rfloor$ by (recursively) stipulating $\Box\lfloor b \rfloor = b$ and $(a.B)\lfloor b \rfloor = a.B\lfloor b \rfloor$ and $(b_0 \mid B)\lfloor b \rfloor = b_0 \mid B\lfloor b \rfloor$. (Note that the operators on the left hand sides are syntactic entities, whereas the ones on the right hand sides are the semantic operators defined above.) Similarly, given B and B_1 we define the behavior context $B\lfloor B_1 \rfloor$.

Actions

$$
\begin{array}{rlll}
a \in \mathsf{Act} ::= & {}^{H}\mathsf{enter}(g) & & \text{steers an ambient from } H \text{ to } g \\
| & {}^{g}\mathsf{exit}(H) & & \text{steers an ambient from } g \text{ to } H \\
| & {}_{G}\mathsf{open}(g) & & \text{if executed in } G, \text{ opens } g \\
| & \mathsf{put}(\sigma) & & \text{output tuple of type } \sigma \\
| & \mathsf{get}(\sigma) & & \text{input tuple of type } \sigma
\end{array}
$$

Traces

$$tr \in \mathsf{Trace} ::= a^* \qquad \text{finite sequence of actions}$$

Behaviors

$$b \in \mathsf{Beh} \subseteq \mathcal{P}(\mathsf{Trace}) \qquad \text{non-empty regular set of traces}$$

Behavior Contexts

$$B \in \mathsf{BehCont} ::= \Box \mid a.B \mid (b \mid B)$$

Behavior Rows

$$br \in \mathsf{BehRows} ::= \{g_i : b_i\}_{i \in I} \qquad \text{behaves as } b_i \text{ when opened in } g_i$$

Tuples

$$\sigma \in \mathsf{Tuple} ::= \times(\tau_1, \ldots, \tau_k) \qquad (k \geqslant 0)$$

When there is no ambiguity, we write τ_1 if $k = 1$.

Types

$$
\begin{array}{rlll}
\tau \in \mathsf{Typ} ::= & \mathsf{amb}^{g}_{H}[br] & & \text{type of ambient name} \\
| & \mathsf{cap}[B] & & \text{type of capability} \\
| & \cdots & & \text{base types (optional)}
\end{array}
$$

In $\mathsf{amb}^{g}_{H}[\{g_i : b_i\}_{i \in I}]$, we demand that $H \neq \emptyset$ and $\mathcal{I}(g, H)$ and $\forall i \in I$: $\mathcal{O}(g, g_i)$. If $I = \emptyset$ we may write amb^{g}_{H}; if I is a singleton $\{i\}$ we may write $\mathsf{amb}^{g}_{H}[g_i : b_i]$.

Fig. 3. Syntax of Types and Behaviors.

use "_" to stand for an arbitrary entity of the appropriate kind.) Example: $\mathsf{put}(\mathsf{cap}[\{\mathsf{put}(\mathsf{amb}^{g}_{H})\} \mid \Box])$ has level two.

3.1 Ordering Relations

In the subsequent paragraphs we define relations $b_1 \leqslant b_2$ and $\tau_1 \leqslant \tau_2$. The definitions are mutually recursive, yet well-defined: a relation on level i types induces (via its pointwise extension to level i tuples) a relation on level i actions which induces a relation on level i behaviors which induces a relation on level i behavior rows and contexts which in turn induces a relation on level $i+1$ types.

The relation $a_1 \leqslant a_2$, with the intuitive interpretation that a_2 is more "permissive" than a_1, is defined by stating that the constructors have the polarity[9]

[9] That is, ${}^{H_1}\mathsf{enter}(g_1) \leqslant {}^{H_2}\mathsf{enter}(g_2)$ iff $H_1 \subseteq H_2$ and $g_1 = g_2$, etc.

$^{\oplus}\mathsf{enter}(=)$ $\quad$ $^{=}\mathsf{exit}(\oplus)$ $\quad$ $_{\ominus}\mathsf{open}(=)$ $\quad$ $\mathsf{put}(\oplus)$ $\quad$ $\mathsf{get}(\ominus)$

Concerning the polarity of $^{g}\mathsf{exit}(H)$, the intuition is that if the ambient as a result of leaving g will enter an ambient whose group is in H, this group also belongs to any set containing H; similarly for $^{H}\mathsf{enter}(g)$. For $_{G}\mathsf{open}(g)$, the intuition is that if a process will open g provided it is in an ambient of group in G, it will also open g whenever it is in an ambient of a group belonging to a subset of G. Concerning the actions for communication, we have (assuming that int and real are types with $\mathsf{int} \leqslant \mathsf{real}$) that $\mathsf{put}(\mathsf{int}) \leqslant \mathsf{put}(\mathsf{real})$, since a process that sends an integer thereby also sends a real number, and $\mathsf{get}(\mathsf{real}) \leqslant \mathsf{get}(\mathsf{int})$, since a process that accepts a real number also will accept an integer. Thus output is covariant and input is contravariant, while in other systems found in the literature it is the other way round—the reason for this discrepancy is that we take a *descriptive* rather than a *prescriptive* point of view[10].

The relation $b_1 \leqslant b_2$, with the intuitive interpretation that b_2 is more "permissive" than b_1, is defined by stipulating that $b_1 \leqslant b_2$ iff for all $tr_1 \in b_1$ there exists $tr_2 \in b_2$ such that $tr_1 \leqslant tr_2$. Here the relation $tr_1 \leqslant tr_2$ is the pointwise extension of the relation $a_1 \leqslant a_2$ (note that if $tr_1 \leqslant tr_2$ then tr_1 and tr_2 have the same length).

The relation $\leqslant$ on behavior rows is defined by stipulating that, with $br = \{g_i : b_i\}_{i \in I}$ and $br' = \{g'_j : b'_j\}_{j \in J}$, $br \leqslant br'$ holds iff for all $j \in J$ there exists $i \in I$ such that $g_i = g'_j$ and $b_i \leqslant b'_j$.

The relation $\leqslant$ on behavior contexts is defined as follows:

Definition 1. *$B_1 \leqslant B_2$ holds iff for all level 0 behaviors b: $B_1\lfloor b \rfloor \leqslant B_2\lfloor b \rfloor$.*

The restriction to level 0 behaviors is formally needed, in order for the relation on level i behavior contexts to be determined by the relation on level i behaviors, but is not essential: Lemma 9 (in Appendix A.2) will tell us that if $B_1 \leqslant B_2$ then $B_1\lfloor b \rfloor \leqslant B_2\lfloor b \rfloor$ holds for all b (moreover, for all B_1, B_2 there exists b_0 such that testing $B_1 \leqslant B_2$ amounts to testing $B_1\lfloor b_0 \rfloor \leqslant B_2\lfloor b_0 \rfloor$).

The relation $\tau_1 \leqslant \tau_2$, with the intuitive interpretation that an expression of type τ_1 also has type τ_2, is defined as the least reflexive and transitive relation with the property that $B_1 \leqslant B_2$ implies $\mathsf{cap}[B_1] \leqslant \mathsf{cap}[B_2]$, that is the polarity is $\mathsf{cap}[\oplus]$.

Concerning the polarity of the type $\mathsf{amb}^{g}_{H}[br]$, it turns out that the proof of subject reduction reveals that this type must be covariant as well as contravariant in H and br. We could thus (as in [AKPG01]) play the trick of [Zim00] (akin to the "split types" of [BPG00] for an object-oriented calculus) and split each of these arguments into two: one contravariant and the other covariant. But to keep things simple, we refrain from doing so.

It is straightforward to verify that the relations $\leqslant$ thus defined are reflexive and transitive, and that the operators "$\mid$" and "." on behaviors respect the relation $\leqslant$.

[10] From a prescriptive point of view, a channel that allows the writing of real numbers also allows the writing of integers, and a channel that allows the reading of integers also allows the reading of real numbers.

3.2 Predicates on Traces

Given an ambient n, controlled by a process P and residing within an ambient whose group belongs to H, we want to estimate the destination of n after (partial) execution of P. For this purpose, we define the upwards closed sets $Dest(H, a)$ and $Dest(H, tr)$ as follows:

$$\begin{aligned} Dest(H, {}^{H_0}\mathsf{enter}(g)) &= \text{if } H \cap H_0 \neq \emptyset \text{ then } g^{\uparrow} \text{ else } \emptyset \\ Dest(H, {}^{g}\mathsf{exit}(H_0)) &= \text{if } g \in H \text{ then } H_0 \text{ else } \emptyset \\ Dest(H, a) &= H \text{ otherwise} \\ Dest(H, \bullet) &= H \\ Dest(H, a \diamond tr) &= Dest(Dest(H, a), tr) \end{aligned}$$

The idea behind the first line is that an ambient residing inside H can only enter an ambient g if they are "siblings", a necessary condition for which is that $H \cap H_0 \neq \emptyset$ where H_0 are the possible surroundings for g. Note that $Dest(H, tr)$ is monotone in H as well as in tr (due to the polarities ${}^{\oplus}\mathsf{enter}(=)$ and ${}^{=}\mathsf{exit}(\oplus)$), and that $Dest(\emptyset, tr) = \emptyset$ for all tr.

We shall have particular interest in *feasible* traces, the intuition being that such a trace can execute "on its own"[11].

Definition 2. *We say that a trace tr is* feasible *from H if* (i) $Dest(H, tr) \neq \emptyset$, *and* (ii) *all occurrences of* $\mathsf{put}(\_)$ *and* $\mathsf{get}(\_)$ *in tr come in pairs, with* $\mathsf{put}(\_)$ *immediately preceding* $\mathsf{get}(\_)$.

Note that if $H \neq \emptyset$ then $\bullet$ is feasible from H.

Example 4. The trace $tr = {}^{@_{CD}}\mathsf{enter}(A)\ {}^{C}\mathsf{enter}(D)$ is not feasible from $\{@\}$ (cf. Example 1). For $Dest(@, tr) = Dest(Dest(@, {}^{@_{CD}}\mathsf{enter}(A)), {}^{C}\mathsf{enter}(D)) = Dest(A, {}^{C}\mathsf{enter}(D)) = \emptyset$.

We now define a predicate, crucial for our type system, telling whether an ambient initially residing within H_0, and controlled by a process with behavior b, can be assigned the type $\mathsf{amb}^{g}_{H}[\{g_i : b_i\}_{i \in I}]$. For this to be the case, H must be an upper approximation of where the ambient can travel "on its own", and if opened when inside g_i then b_i must approximate the unleashed behavior; moreover, no type errors must arise during communication, and the "condition" of an opening action must be satisfied. Formally, we have

Definition 3 (The trace correctness predicate). *The relation*

$$(H_0, b) \overset{g}{\rightsquigarrow} (H, \{g_i : b_i\}_{i \in I})$$

holds iff for all $tr_1 \diamond tr_2 \in b$ such that tr_1 is feasible from H_0 the following properties hold with $H_1 = Dest(H_0, tr_1)$:

1. *$H_1 \subseteq H$, and*

[11] A non-feasible trace is still useful, since interleaving it with another trace may produce feasible traces.

2. *for all* $i \in I$*:* $g_i \in H_1$ *implies* $\{tr_2\} \leqslant b_i$*, and*
3. *if* tr_2 *takes the form* $\mathsf{put}(\sigma_1)\,\mathsf{get}(\sigma_2) \diamond tr_3$ *then* $\sigma_1 \leqslant \sigma_2$*, and*
4. *if* tr_2 *takes the form* ${}_G\mathsf{open}(\_) \diamond tr_3$ *then* $g \in G$*.*

Example 5. Continuing Example 1, with $b = {}^{@CD}\mathsf{enter}(A).{}^{C}\mathsf{enter}(D)$ we have

$$(\{@\}, b) \overset{B}{\rightsquigarrow} (@A, \{A : {}^{C}\mathsf{enter}(D)\})$$

since if $tr_1 \diamond tr_2$ belongs to b with tr_1 feasible from $\{@\}$ then tr_1 will be of length zero or one... but never of length two.

Example 6. Continuing Example 3, with $b = {}^{A}\mathsf{exit}(@).{}^{A@}\mathsf{enter}(C).{}^{@}\mathsf{enter}(A)$ (the process inside k) we have $(\{A\}, b) \overset{B}{\rightsquigarrow} (AC@, \{C : {}^{@}\mathsf{enter}(A)\})$.

Some useful results about the trace correctness predicate:

Lemma 1. *If* $(H_0, b) \overset{g}{\rightsquigarrow} (H, \{g_i : b_i\}_{i\in I})$ *then* $H_0 \subseteq H$*, and for all* $i \in I$ *with* $g_i \in H_0$ *we have* $b \leqslant b_i$*.*

Lemma 2. *The predicate* $(H_0, b) \overset{g}{\rightsquigarrow} (H, br)$ *is anti-monotonic in* H_0 *and* b*, and monotonic in* H *and* br*.*

Lemma 3. *Let* tr *be of the form either* $\text{-}\mathsf{enter}(\_)$*,* $\text{-}\mathsf{exit}(\_)$*,* $\_\mathsf{open}(\_)$*, or* $\mathsf{put}(\_)\mathsf{get}(\_)$*.*
Assume that $(H_0, tr.b) \overset{g}{\rightsquigarrow} (H, br)$*.*
Then with $H_1 = Dest(H_0, tr)$ *we also have* $(H_1, b) \overset{g}{\rightsquigarrow} (H, br)$*.*

4 The Type System

Figure 4 defines judgments $E \vdash M : \tau$ and $\Delta, E \vdash_g P : b$, where E is an environment mapping names into types (we write $E, n : \tau$ for the environment E' that behaves as E except that it maps n into τ), where the *behavior pool* Δ maps tags into behaviors, and where g is the group of the ambient in which P is initially situated.

Already in Sect. 3 we saw how the rule (Exp Open) is designed to interact with the rule (Proc Action), similarly for the rules (Exp In) and (Exp Out). The side condition for the rule (Proc Repl) ensures that the behavior of a replicated process is in fact invariant under replication. The crux of the type system is the rule (Proc Amb), where the situation is that inside g we have an ambient M with tag ξ containing a process P. We must then be able to assign P the behavior $b = \Delta(\xi)$, when analyzed inside the group $g_0 = group(\xi)$. Moreover, it must hold that $(g^{\uparrow}, b) \overset{g_0}{\rightsquigarrow} (H, br)$ where $\mathsf{amb}^{g_0}_H[br]$ is the type of M, cf. the motivation for the trace correctness predicate given in Sect. 3.2.

For the static security constraint $\mathcal{I}(\_, \_)$, we have the following lemma:

Lemma 4. *Assume that inside* P *an ambient of group* g_0 *is directly enclosed in an ambient of group* g*. If* P *is typable then* $\mathcal{I}(g_0, g)$ *does hold.*

Non-structural Rules

(Proc Subsumption)
$$\frac{\Delta, E \vdash_g P : b}{\Delta, E \vdash_g P : b'} \quad (b \leqslant b')$$

(Exp Subsumption)
$$\frac{E \vdash M : \tau}{E \vdash M : \tau'} \quad (\tau \leqslant \tau')$$

Expressions

(Exp n)
$$\frac{E(n) = \tau}{E \vdash n : \tau}$$

(Exp ϵ)
$$\frac{}{E \vdash \epsilon : \mathsf{cap}[\Box]}$$

(Exp Action)
$$\frac{E \vdash M_1 : \mathsf{cap}[B_1] \quad E \vdash M_2 : \mathsf{cap}[B_2]}{E \vdash M_1.M_2 : \mathsf{cap}[B_1 \lfloor B_2 \rfloor]}$$

(Exp In)
$$\frac{E \vdash M : \mathsf{amb}^g_H[br]}{E \vdash \mathsf{in}\ M : \mathsf{cap}[^H\mathsf{enter}(g).\Box]}$$

(Exp Out)
$$\frac{E \vdash M : \mathsf{amb}^g_H[br]}{E \vdash \mathsf{out}\ M : \mathsf{cap}[^g\mathsf{exit}(H).\Box]}$$

(Exp Open)
$$\frac{E \vdash M : \mathsf{amb}^g_H[\{g_i : b_i\}_{i \in I}]}{E \vdash \mathsf{open}\ M : \mathsf{cap}[_G\mathsf{open}(g).(b \mid \Box)]} \quad \text{if} \quad \begin{array}{l} \forall g' \in G : \exists i \in I : \\ g' = g_i \text{ and } b_i \leqslant b \end{array}$$

Processes

(Proc Zero)
$$\frac{}{\Delta, E \vdash_g \mathbf{0} : \varepsilon}$$

(Proc Par)
$$\frac{\Delta, E \vdash_g P_1 : b_1 \quad \Delta, E \vdash_g P_2 : b_2}{\Delta, E \vdash_g P_1 \mid P_2 : b_1 \mid b_2}$$

(Proc Repl)
$$\frac{\Delta, E \vdash_g P : b}{\Delta, E \vdash_g !P : b} \quad \text{if } (b \mid b) \leqslant b$$

(Proc Res)
$$\frac{\Delta, E, n : \mathsf{amb}^{g_0}_H[br] \vdash_g P : b}{\Delta, E \vdash_g (\nu n : \mathsf{amb}^{g_0}_H[br]).P : b}$$

(Proc Action)
$$\frac{E \vdash M : \mathsf{cap}[B] \qquad \Delta, E \vdash_g P : b}{\Delta, E \vdash_g M.P : B \lfloor b \rfloor}$$

(Proc Amb)
$$\frac{E \vdash M : \mathsf{amb}^{g_0}_H[br] \quad \Delta, E \vdash_{g_0} P : b}{\Delta, E \vdash_g M[\![P]\!]^\xi : \varepsilon} \quad \text{if} \quad \begin{array}{l} (g^\uparrow, b) \overset{g_0}{\rightsquigarrow} (H, br) \\ \mathit{group}(\xi) = g_0 \\ \Delta(\xi) = b \end{array}$$

(Proc Input)
$$\frac{\Delta, E, n_1 : \tau_1, \cdots, n_k : \tau_k \vdash_g P : b}{\Delta, E \vdash_g (n_1 \ldots n_k : \tau_1 \ldots \tau_k).P : \mathsf{get}(\sigma).b}$$

(Proc Output)
$$\frac{\forall i \in \{1 \ldots k\} : E \vdash M_i : \tau_i}{\Delta, E \vdash_g \langle M_1 \ldots M_k \rangle : \mathsf{put}(\sigma)}$$

In (Proc Input) and (Proc Output), $\sigma = \times(\tau_1, \ldots, \tau_k)$ and $k \geqslant 0$.

Fig. 4. Typing Rules.

Proof. From Lemma 1 we see that for the rule (Proc Amb) it holds that $g^\uparrow \subseteq H$, in particular that $g \in H$. From the requirement to an $\mathsf{amb}^{\_}_{\_}[\_]$ type we have $\mathcal{I}(g_0, H)$, in particular $\mathcal{I}(g_0, g)$.

For the dynamic security constraint $\mathcal{O}(\_, \_)$, see the formulation of semantic soundness (Theorem 1).

5 Semantic Soundness

We shall now show that our type system is semantically sound. This property is formulated as a subject reduction result, intuitively stating that "well-typed processes behave according to their behavior" and also stating that "well-typed processes never evolve into ill-typed processes". For a succinct version of this result, we introduce a relation $\Delta \xrightarrow{\ell} \Delta'$, expressing that the behavior pool Δ evolves into Δ' as predicted by the reduction label ℓ. Formally, the relation $\Delta \xrightarrow{\ell} \Delta'$ holds iff

- Δ' agrees with Δ on $\mathrm{dom}(\Delta) \setminus dom(\ell)$
  (where $dom(\epsilon) = \emptyset$, $dom(\xi : \mathsf{enter}\ \chi) = \{\xi\}$, etc.);
- if $\ell = \xi : \mathsf{enter}\ \chi$ then ${}^{\emptyset}\mathsf{enter}(group(\chi)).\Delta'(\xi) \leqslant \Delta(\xi)$;
- if $\ell = \xi : \mathsf{exit}\ \chi$ then ${}^{group(\chi)}\mathsf{exit}(\emptyset).\Delta'(\xi) \leqslant \Delta(\xi)$;
- if $\ell = \xi : \mathsf{open}\ \chi$ then ${}_{\mathsf{Grps}}\mathsf{open}(group(\chi)).\Delta'(\xi) \leqslant \Delta(\xi)$;
- if $\ell = \xi : \mathsf{comm}\ \sigma$ then $\exists \sigma' \leqslant \sigma\colon \mathsf{put}(\sigma').\mathsf{get}(\sigma).\Delta'(\xi) \leqslant \Delta(\xi)$.

Example 7. If $\ell = \xi : \mathsf{enter}\ \chi$ and Δ maps ξ into ${}^{A}\mathsf{enter}(B).{}^{B}\mathsf{exit}(A)$, then $\Delta \xrightarrow{\ell} \Delta'$ holds if $group(\chi) = B$ (the ambient tagged ξ is in fact steered into an ambient of group B) and Δ' maps ξ into ${}^{B}\mathsf{exit}(A)$ (the remaining action).

Theorem 1 (Subject reduction for processes). *Suppose that*

$$P \xrightarrow{\ell} Q$$

where P and Q are uniquely tagged, and where all tags occurring in Q but not in P do not occur in $\mathrm{dom}(\Delta)$*. Further assume that*

$$\Delta, E \vdash_g P : b.$$

Then there exists Δ' such that

$$\Delta \xrightarrow{\ell} \Delta'$$

$$\Delta', E \vdash_g Q : b$$

*and additionally (***Safety of opening***): if $\ell = \xi : \mathsf{open}\ \chi$ then $\mathcal{O}(group(\chi), group(\xi))$.*

The proof of this theorem makes heavy use of Lemmas 2 and 3. To deal with the case (Red Open), we also need

Lemma 5. *Suppose that $\Delta, E \vdash_g P : b$. If $\mathcal{O}(g, g')$ then also $\Delta, E \vdash_{g'} P : b$.*

Proof. Structural induction on the derivation, where the only non-trivial case is (Proc Amb). Noting that $g' \in g^{\uparrow}$ and therefore ${g'}^{\uparrow} \subseteq g^{\uparrow}$, Lemma 2 will ensure that the side condition still holds.

To deal with the case (Red Comm), we also need a standard substitution lemma. To deal with (Red $\equiv$), we need a "subject congruence" result (proved using standard lemmas for "swapping", "weakening", and "strengthening" the environment):

Lemma 6. *If $P \equiv Q$ then $\Delta, E \vdash_g P : b$ iff $\Delta, E \vdash_g Q : b$.*

6 Type Checking

In this section we show that given a complete type derivation for some process P, we can check its validity according to the rules from Fig. 4.

Lemma 7. *The relations $\leqslant$ defined in Sect. 3.1 are decidable.*

The decision procedures are defined mutually recursively: given a procedure for deciding $\leqslant$ on level i types we can construct a procedure for deciding $\leqslant$ on level i actions (obvious); given a procedure for deciding $\leqslant$ on level i actions we can construct a procedure for deciding $\leqslant$ on level i behaviors (see Appendix A.1); given a procedure for deciding $\leqslant$ on level i behaviors we can construct a procedure for deciding $\leqslant$ on level i behavior contexts (see Appendix A.2) and a procedure for deciding $\leqslant$ on level i behavior rows (obvious); given a procedure for deciding $\leqslant$ on level i behavior contexts/rows we can construct a procedure for deciding $\leqslant$ on level $i+1$ types (obvious).

The decision procedures are thus potentially very expensive for entities of high level, but this in itself is no reason for discomfiture. Such entities are probably rare in practice; for instance, in the communication-free ambient calculus, types are of level 1 and all other entities are of nesting 0. Also, hope for efficiency in practice is supported by other similar situations. For example, ML type-inference shows an extreme disparity between worst-case performance in theory (exponential time) and actual performance in practice (very fast).

Lemma 8. *The relation $(H_0, b) \overset{g}{\rightsquigarrow} (H, br)$ is decidable.*

For a proof sketch, see Appendix A.3. Together with Lemma 7 this shows

Theorem 2 (Decidability of type checking). *Given a purported derivation $\Delta, E \vdash_g P : b$, we can effectively check its validity.*

7 Conclusion

We have demonstrated that it is possible to use *type-based* methods (also employing techniques from finite automata) to develop a precise analysis of the *original* ambient calculus. Future work includes implementing the inference algorithm, and measuring its actual performance.

References

[AKPG01] T. Amtoft, A. J. Kfoury, and S. M. Pericas-Geertsen. What are polymorphically-typed ambients? In D. Sands, ed., *ESOP 2001, Genova*, vol. 2028 of *LNCS*, pp. 206–220. Springer-Verlag, Apr. 2001. An extended version appears as Technical Report BUCS-TR-2000-021, Comp.Sci. Department, Boston University, 2000.

[BC01] M. Bugliesi and G. Castagna. Secure safe ambients. In *Conf. Rec. POPL '01: 28th ACM Symp. Princ. of Prog. Langs.*, pp. 222–235, 2001.

[BCC01] M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in mobile ambients. In *CONCUR 2001*, vol. 2154 of *LNCS*, pp. 102–120. Springer-Verlag, 2001.

[BPG00] M. Bugliesi and S. M. Pericas-Geertsen. Depth subtyping and type inference for object calculi. In *Proc. Seventh Workshop on Foundations of Object-Oriented Languages*, Boston, Mass., U.S.A., 2000.

[CG98] L. Cardelli and A. D. Gordon. Mobile ambients. In M. Nivat, ed., *FoSSaCS'98*, vol. 1378 of *LNCS*, pp. 140–155. Springer-Verlag, 1998.

[CG99] L. Cardelli and A. D. Gordon. Types for mobile ambients. In *POPL'99, San Antonio, Texas*, pp. 79–92. ACM Press, Jan. 1999.

[CGG99] L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility types for mobile ambients. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds., *ICALP'99*, vol. 1644 of *LNCS*, pp. 230–239. Springer-Verlag, July 1999. Extended version appears as Microsoft Research Technical Report MSR-TR-99-32, 1999.

[CGG00] L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient groups and mobility types. In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000), Tohoku University, Sendai, Japan*, vol. 1872 of *LNCS*, pp. 333–347. Springer-Verlag, Aug. 2000.

[DCS00] M. Dezani-Ciancaglini and I. Salvo. Security types for mobile safe ambients. In *ASIAN Computing Science Conference - ASIAN'00*, vol. 1961 of *LNCS*, pp. 215–236. Springer, 2000.

[GH99] S. Gay and M. Hole. Types and subtypes for client-server interactions. In *Proc. European Symp. on Programming*, vol. 1576 of *LNCS*, pp. 74–90. Springer-Verlag, 1999.

[GYY01] X. Guan, Y. Yang, and J. You. Typing evolving ambients. *Information Processing Letters*, 80(5):265–270, Nov. 2001.

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

[LM01] F. Levi and S. Maffeis. An abstract interpretation framework for analysing mobile ambients. In *SAS'01*, vol. 2126 of *LNCS*, pp. 395–411. Springer-Verlag, 2001.

[LS00] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *POPL'00, Boston, Massachusetts*, pp. 352–364. ACM Press, Jan. 2000.

[NN94] H. R. Nielson and F. Nielson. Higher-order concurrent programs with finite communication topology. In *Conf. Rec. 21st Ann. ACM Symp. Princ. of Prog. Langs.*, pp. 84–97, 1994.

[NN01] H. R. Nielson and F. Nielson. Shape analysis for mobile ambients. *Nordic Journal of Computing*, 8:233–275, 2001. A preliminary version appeared at POPL'00.

[NNHJ99] F. Nielson, H. R. Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in mobile ambients. In *Proc. CONCUR'99*, vol. 1664 of *LNCS*, pp. 463–477. Springer-Verlag, 1999.

[NNS00] F. Nielson, H. R. Nielson, and M. Sagiv. A Kleene analysis of mobile ambients. In *Programming Languages & Systems, 9th European Symp. Programming*, vol. 1782 of *LNCS*, pp. 305–319. Springer-Verlag, 2000.

[Zim00] P. Zimmer. Subtyping and typing algorithms for mobile ambients. In *FOSSACS 2000, Berlin*, vol. 1784 of *LNCS*, pp. 375–390. Springer-Verlag, 2000.

A Type Checking

A.1 Behaviors

We now describe how to construct, given a procedure for deciding $\leqslant$ on level i actions, a procedure for deciding $\leqslant$ on level i behaviors. For that purpose, we construct ϵ-transition-free automata A_1 and A_2 recognizing b_1 and b_2, and then construct their "difference automaton" $A_1 \setminus A_2$ (the states of which are of the form (q_1, Q_2) with q_1 a state in A_1 and Q_2 a set of states in A_2, with $(\iota_1, \{\iota_2\})$ being initial if ι_1 and ι_2 are initial, with (q_1, Q_2) being final if q_1 is final and Q_2 does not contain any final states, and with an a-transition from (q_1, Q_2) to (q_1', Q_2') if A_1 has an a-transition from q_1 to q_1' and Q_2' is the set of states q_2' in A_2 for which there exists $q_2 \in Q_2$ and an a^+ transition from q_2 to q_2' where $a \leqslant a^+$). Deciding $b_1 \leqslant b_2$ now amounts to checking whether $A_1 \setminus A_2$ rejects all inputs.

A.2 Behavior Contexts

We now describe how to construct, given a procedure for deciding $\leqslant$ on level i behaviors, a procedure for deciding $\leqslant$ on level i behavior contexts. Looking at Definition 1, deciding $B_1 \leqslant B_2$ seems to involve plugging in an infinite number of behaviors. Fortunately, it suffices to plug in a single (level 0) behavior $\{\mathsf{test}\}$, provided test *tests* B_1 and B_2, that is: it is incomparable with all actions occurring in B_1 and B_2. We can easily pick up such test, for instance as ${}^{\emptyset}\mathsf{enter}(g^*)$ where g^* is a group not occurring elsewhere.

Lemma 9. *Given B_1 and B_2 which are tested by* test*, the following conditions are equivalent:*

(a) $B_1 \leqslant B_2$
(b) $B_1\lfloor b \rfloor \leqslant B_2\lfloor b \rfloor$ *for all b (regardless of level)*
(c) $B_1\lfloor \{\mathsf{test}\} \rfloor \leqslant B_2\lfloor \{\mathsf{test}\} \rfloor$.

The non-trivial part of this lemma (showing that (c) implies (b)) is proved using an auxiliary result (which is proved by structural induction in B): $tr \in B\lfloor b \rfloor$ if and only if there exists $tr_1 \diamond \mathsf{test} \diamond tr_2 \in B\lfloor \{\mathsf{test}\} \rfloor$ and $tr_0 \in \{tr_2\} \mid b$ such that $tr = tr_1 \diamond tr_0$.

A.3 Trace Correctness

The procedure for checking $(H_0, b) \overset{g}{\rightsquigarrow} (H, br)$ works as follows: with A an automaton (without "garbage" states) recognizing b, we annotate each state q with a set of groups H_q such that *(i)* with ι the initial state, $H_0 \subseteq H_\iota$; and *(ii)* if there is an a-transition from q_1 to q_2 with a not of the form $\mathsf{put}(\_)$ or $\mathsf{get}(\_)$, then $Dest(H_{q_1}, a) \subseteq H_{q_2}$; and *(iii)* if there is a $\mathsf{put}(\_)$ transition from q_1 to q_3 and a $\mathsf{get}(\_)$ transition from q_3 to q_2, then $H_{q_1} \subseteq H_{q_2}$. Clearly we can compute the least such annotation; and it is not difficult to see that $H_q \neq \emptyset$ if and only if q is the end point of a feasible trace. Then for all states q check that *(i)* $H_q \leqslant H$; that *(ii)* if $H_q \neq \emptyset$ and q is the source of an a-transition with a of the form ${}_G\mathsf{open}(\_)$ then $g \in G$; and that *(iii)* if $H_q \neq \emptyset$ and q is the source of a path of the form $\mathsf{put}(\sigma_1)\mathsf{get}(\sigma_2)$ then $\sigma_1 \leqslant \sigma_2$. This takes care of all but item 2 in Definition 3. For this item, let $br = \{g_i : b_i\}_{i \in I}$ and for each $i \in I$ proceed as follows: construct an automaton A_i whose initial state has ϵ-transitions to the states q for which $g_i \in H_q$, and which otherwise is as A. Now check (using the above results) that $A_i \leqslant b_i$.

B Type Inference

We now briefly sketch how one might perform type *inference*, where ideally the user, in addition to the security constraints, provides only the group of each ν-bound ambient name. It seems that we need to impose several restrictions on the form of the input, none of which appears to exclude most commonly met processes:

1. only *ambients* can be replicated (as given $b_0 \neq \varepsilon$, it is not obvious how to find b with $b_0 \leqslant b$ such that $b \mid b \leqslant b$);
2. for each group g, there exists at most one g_0 such that $\mathcal{O}(g, g_0)$ (to help dealing with the non-determinism in (Exp Open));
3. no ambient names are communicated, and if capabilities are communicated then the user has to provide their full type;
4. it is possible to define a total order $\prec$ among the groups such that whenever an ambient of group g contains the expression $\mathsf{open}\ n$ where n has group g' then $g' \prec g$.

Then a type reconstruction algorithm could proceed as follows:

1. For each ν-bound ambient n with group g, define $H_g = \{g' \mid \mathcal{I}(g, g')\}$ and assign to n the (preliminary) type $\mathsf{amb}^g_{H_g}$. (Alternatively, to gain greater precision, one could proceed in an iterative way: initiate H_g to $\emptyset$, and if step 2 fails then increment H_g appropriately and start all over again.)
2. Starting with the smallest g, we find a behavior b such that if a process P is enclosed within an ambient of group g then P has behavior b. Note that occurrences of $\mathsf{open}\ n$ in P will not cause trouble, since n has a smaller group (thanks to restriction 4) and hence has been given a type already. With g_0 the group (cf. restriction 2) such that $\mathcal{O}(g, g_0)$, we now find b_0 (using techniques similar to the ones used for establishing Lemma 8) such that $(\_, b) \overset{g}{\rightsquigarrow} (H_g, \{g_0 : b_0\})$ (this might fail, for instance if H_g is too small). Now globally update the type of all ambients of group g into $\mathsf{amb}^g_{H_g}[g_0 : b_0]$.

One can optimize the procedure sketched above in various ways, and alleviate certain of the restrictions. For example, it would be useful to allow for an ambient n to contain $\mathsf{open}\ n$ (cf. Gonthier's coalescing encoding of channels mentioned in [CG99]) which will give rise to equations of the form $(\_, b) \overset{g}{\rightsquigarrow} (H_g, \{g_0 : \beta_0\})$ where β_0 occurs in b. In certain cases, these can be solved (by creating a "backwards loop" in the automaton).