Flow-sensitive Type Systems and the Ambient Calculus [1]

Torben Amtoft [2]

Department of Computing and Information Sciences, Kansas State University, Manhattan KS 66506, USA

Abstract

The Ambient Calculus was developed by Cardelli and Gordon as a formal framework to study issues of mobility and migrant code. Numerous analyses have been developed for numerous variants of that calculus. We take up the challenge of developing, in a *type-based setting*, a relatively precise "topology" analysis for the *original* version of the calculus. To compensate for the lack of "co-capabilities" (an otherwise increasingly popular extension), the analysis is flow-sensitive, with the actions of processes being summarized by "behaviors".

A subject reduction property guarantees that for a well-typed process, the location of any ambient is included in what is predicted by its type; 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.

1 Introduction

For program analysis, several frameworks have been devised, as described in the excellent textbook [NNH99]. One can construct isomorphisms between some of these [PP01,AT00,NP05] though abstract interpretation seems the most powerful [Cou97]. My focus has mostly been on type systems, in particular when augmented with "effects".

[1] Most of the research behind this work was carried out in 2001 while the author was at Boston University, supported by NSF EIA grant 9806745/9806746/9806747/9806835 and partially supported by Sun grant EDUD-7826-990410-US; also some part was done in 2002 when at Heriot-Watt University, supported by the DART project (EC grant IST-2001-33477).
[2] Supported by an AFOSR grant, and by NSF grant CCR-0296182.

Effect systems were proposed in [LG88] and further developed in, e.g., [TJ94]. In the context of analyzing Concurrent ML [Rep91], effects were augmented [NN93,NN94] with *temporal information.* The resulting *behaviors* enable a "flow-sensitive" analysis: for two statements S_1 and S_2, the analysis of $S_1; S_2$ is different from the analysis of $S_2; S_1$.

Together with the Nielson's, I developed the theory of types and behaviors for Concurrent ML, culminating with [ANN99]. As reported in [NAN98], the resulting tool greatly assisted in validating a number of safety properties for "realistic" concurrent systems, and also revealed[3] a subtle error in a system designed by formal methods.

Later, I focused on the ambient calculus (AC), 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, thus forming a dynamic tree structure. The model also features communication, in that values can be exchanged between neighboring processes. Over the years, numerous variants and extensions of the "classical" ambient calculus have been proposed: "safe ambients" [LS00,GYY00], "boxed ambients" [BCC01,BCMS02,BCDCG04], "controlled ambients" [TZH02], BioAmbients [NNP04], etc.

Early type systems for AC were flow-insensitive, initially [CG99] designed to ensure that each ambient has a unique "topic of conversation". This precludes configurations like

$$r[\langle 7\rangle \mid (z : \mathsf{int}).\mathsf{in}\ q.\langle z = 42\rangle] \mid q[\mathsf{open}\ r.(y : \mathsf{bool}).P] \tag{1}$$

where even though r allows both integers and booleans to be communicated, all values received are of the expected type, as seen by the fact that the only possible reduction sequence is

[3] One of the safety requirements was that a certain table must not be moved downward if it is in its lower position, and not moved upward if it is in its upper position. After automatically constructing the behavior for the table, and extracting the operations relevant for the given requirement, it became evident that wrt. these operations the program in question works by iterating a cycle which (i) moves the table upward; (ii) checks if the table is in its upper position; (iii) moves the table downward; (iv) checks if the table is in its lower position. This clearly violates the given requirement if the initial state of the table happens to be the upper position.

$$\begin{aligned}
&(1) \longrightarrow \text{(integer 7 is output and bound to } z)\\
r[\mathsf{in}\ q.\langle 7 = 42\rangle] \mid q[\mathsf{open}\ r.(y : \mathsf{bool}).P] &\longrightarrow \text{(ambient } r \text{ enters ambient } q)\\
q[r[\langle 7 = 42\rangle] \mid \mathsf{open}\ r.(y : \mathsf{bool}).P] &\longrightarrow \text{(ambient } r \text{ is dissolved inside } q)\\
q[\langle 7 = 42\rangle \mid (y : \mathsf{bool}).P] &\longrightarrow \text{(boolean is output and bound to } y)\\
q[P[y := \mathsf{false}]]&
\end{aligned}$$

To allow for multiple topics of conversation, in [AKPG01,AKPG02] we proposed a flow-sensitive type system, assigning behaviors to processes. (This is akin to the session types [GH99], or the graph types [Yos96], for the π-calculus.)

A key challenge for any analysis of AC, with potential applications for security, is to predict the shape of the dynamic tree structures:

$$\textbf{which ambients can end up where?} \tag{2}$$

This was first addressed by the flow-*in*sensitive analysis of [CGG00]. Note, however, that a precise answer to (2) requires flow-sensitivity, as illustrated by the two processes below which are equal modulo permutation of "capabilities":

$$p[\mathsf{open}\ q] \mid q[\mathsf{in}\ p.\mathsf{in}\ r.\mathsf{out}\ r] \mid r[\mathbf{0}] \tag{3}$$
$$p[\mathsf{open}\ q] \mid q[\mathsf{in}\ r.\mathsf{out}\ r.\mathsf{in}\ p] \mid r[\mathbf{0}] \tag{4}$$

Both processes are deterministic, with reduction sequences given by

$$\begin{array}{r|r}
(3) \longrightarrow & (4) \longrightarrow\\
p[q[\mathsf{in}\ r.\mathsf{out}\ r] \mid \mathsf{open}\ q] \mid r[\mathbf{0}] \longrightarrow & p[\mathsf{open}\ q] \mid r[q[\mathsf{out}\ r.\mathsf{in}\ p]] \longrightarrow\\
p[\mathsf{in}\ r.\mathsf{out}\ r] \mid r[\mathbf{0}] \longrightarrow & p[\mathsf{open}\ q] \mid q[\mathsf{in}\ p] \mid r[\mathbf{0}] \longrightarrow\\
r[p[\mathsf{out}\ r]] \longrightarrow & p[q[\mathbf{0}] \mid \mathsf{open}\ q] \mid r[\mathbf{0}] \longrightarrow\\
p[\mathbf{0}] \mid r[\mathbf{0}] & p[\mathbf{0}] \mid r[\mathbf{0}]
\end{array}$$

We see that (3) allows p to be located inside r, whereas (4) does not. The difference is that when p opens q in (3), q has an in r "capability" left, whereas when p opens q in (4), that capability has already been consumed. The implications are that *when analyzing the occurrence of* open q *in* p, *we must know which of the capabilities of* q *are still left*, as those will be the ones "unleashed" within p. This cannot be done in a flow insensitive setting, a fact that was early recognized as a general obstacle to a precise analysis, cf. the discussion in [CGG99] where a naive analysis cannot declare immobile an ambient that opens a packet which *has* moved.

The above conundrum can be resolved at the *language level*, most radically by removing open from the language, as done in boxed ambients [BCC01] (at the price of allowing communication not just between siblings but also between parent and child), and in M^3 [CDCGS03]. The most popular approach, however, is to let the execution of capabilities be multilateral rather than unilateral. In particular, in order for p to execute $\mathsf{open}\ q$, the ambient q must be ready to execute a corresponding "$\mathsf{co\text{-}open}$". This device was first employed in *safe ambients* [LS00] and later employed, e.g., in [DCS00,AKPG01,GYY01,Lev03,LB04]. (Safe ambients additionally include the co-capabilities $\mathsf{co\text{-}in}$ and $\mathsf{co\text{-}out}$ which also have been used, e.g., in [BC01].) In the language of [AKPG01], the process (1) becomes

$$r[\langle 7\rangle \mid (z : \mathsf{int}).\mathsf{in}\ q.\mathsf{co\text{-}open}\ r.\langle z = 42\rangle] \mid q[\mathsf{open}\ r.(y : \mathsf{bool}).P]$$

and it is now easy to give r the type $\mathsf{amb}[\mathsf{put}(\mathsf{bool})]$ saying that when r is opened, it will unleash the writing of a boolean—and nothing more. Therefore it is safe for q, expecting a boolean, to open r, even though inside r there is a subprocess writing an integer.

To address (2), we could apply the same device and, e.g., require the process (3) to be rewritten into

$$p[\mathsf{open}\ q] \mid q[\mathsf{in}\ p.\mathsf{co\text{-}open}\ q.\mathsf{in}\ r.\mathsf{out}\ r] \mid r[\mathbf{0}]$$

from which it is easy to see what is unleashed when q is opened. But rather than requiring the hand to fit the glove, I shall deem it more appropriate to design a glove that fits the hand, and **aim at an analysis which is able to address (2) for classical** AC.

Several such analyses have been proposed, based on various paradigms. *Control flow analysis* is employed in [NNHJ99,BCF02]. *Abstract interpretation* is employed in [HJNN99], and in [LM04] where one analysis keeps track of the context one level up; this is sufficient to achieve a quite precise analysis, yet is "only" polynomial (n^7). [NN01] employs *tree grammars*, computing a set of grammars such that at any step in the reduction process, the current tree structure can be described by one of these grammars—the method is very precise, but potentially also very expensive. A *3-valued logic* is employed in [NNS00] to estimate the possible shapes of the tree structure; the framework allows for trade-offs wrt. precision versus costs.

Due to my background, I took up the challenge of solving (2) in the setting of *type systems*. That quest succeeded during the summer of 2001, as reported and detailed in [Amt02], and to be highlighted in this paper. The **main contributions of that work** are that *(i)* it provides the first type-based system which keeps track of how the location of an ambient changes over time, thus allowing a precise estimate concerning when it can be opened; *(ii)* the type system smoothly incorporates a communication analysis which allows for multiple topics of conversation, whereas the analyses listed in the previous paragraph are for a communication-free subset of classical AC. A key feature of the approach is that behaviors are finite automata, allowing (as in [AKPG01]) type checking to use techniques from automaton theory.

In 2002, I took part in the early development of PolyA, an even more general type system [AMW04]. Here a process is given a type which is essentially an upper bound of all shapes it can evolve to (much as what is done in [NNP04] for BioAmbients). The analysis is very precise, and also smoothly incorporates communication; a spin-off is the generic tool Poly [MW05] which can handle a very broad range of mobile calculi. Still, in my humble opinion, this approach is not in the proper "spirit" of type systems [4], since types are almost indistinguishable from processes (as is also the case in [CDC02]).

The upshot is that, in my view, [Amt02] is still the solution to Question 2 that best conforms to the paradigm of type and effect systems. In subsequent sections, we shall motivate and describe the design of that system, referring to [Amt02] for further technical details.

2 Tracking Locations

We now illustrate how to keep track of the location of an ambient, so as to estimate when it is opened. For the process in (4), we observe that q starts being at top-level, which we write as "q is in $\#$" where $\#$ is a "global" ambient. After executing $\mathsf{in}\ r$, q will be enclosed [5] by r. After executing $\mathsf{out}\ r$, it is not immediately clear where q will be, since r might have moved while containing q. This motivates that the type of an ambient should tell where the ambient may be located (cf. the type system for boxed ambients in [MS02]). In this case, the information that r is enclosed by $\#$ only, will enable us to infer that q will again be enclosed by $\#$. Finally, q gets enclosed by p after executing $\mathsf{in}\ p$.

We conclude that q may be enclosed by either of r, p, or $\#$, enabling us to give q the type $\mathsf{amb}_{rp\#}$. Moreover, we see that when q is enclosed by p, no capabilities are left, enabling us to further give q the type $\mathsf{amb}_{rp\#}[\{p : \varepsilon\}]$ where ε denotes the "empty" behavior. The $\{p : \varepsilon\}$ component expresses that when opened inside p, the empty behavior is "unleashed". Thus p doesn't gain any capabilities by opening q, so we can give p the type $\mathsf{amb}_{\#}$ which in particular shows that p *is never enclosed by* r.

Now look at (3), where the process within q may be given the behavior

[4] Anticipating that criticism, [MW05] invites skeptical reader to instead use term "versatile program analysis framework"!
[5] When writing "enclosed", we always mean "directly enclosed".

$$^{?}\mathsf{enter}(p).\,^{\#}\mathsf{enter}(r).\,^{r}\mathsf{exit}(\#) \tag{5}$$

where we have exploited that r has type $\mathsf{amb}_{\#}$ (so that r can only be entered by an ambient which is enclosed by $\#$). This behavior may seem contradictory: how can q first enter p, and next enter r *from $\#$?* The explanation is that q has been *opened* when in p, justifying that we give q the type

$$\mathsf{amb}_{p\#}[\{p : {}^{\#}\mathsf{enter}(r).\,^{r}\mathsf{exit}(\#)\}]$$

From this typing we see that p by opening q gets a capability to enter r, so if amb_H is a typing for p then H has to include r. As expected, the analysis does not rule out that p may be enclosed in r.

Example 1 *As a larger example, consider the firewall first presented in [CG98]:*

$$w[k[\mathsf{out}\ w.\mathsf{in}\ k'.\mathsf{in}\ w] \mid \mathsf{open}\ k'.\mathsf{open}\ k''.P] \mid k'[\mathsf{open}\ k.k''[Q]]$$

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

$$\begin{array}{ll} k\ : \mathsf{amb}_{wk'\#}[k' : {}^{\#}\mathsf{enter}(w)] & w\ : \mathsf{amb}_{\#} \\ k'\ : \mathsf{amb}_{w\#}[w : \varepsilon] & k''\ : \mathsf{amb}_{wk'}[w : \varepsilon] \end{array}$$

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

A problem with the above typing is that the "secret" name w appears in the typings of k' and k''. This motivates the introduction of *groups*, as in [CGG00], with the intention that each ambient belongs to exactly one group $g \in \mathsf{Grps}$ (a finite set). With W the group of w, etc, we have, e.g., $k' : \mathsf{amb}^{K'}_{W\#}[W : \varepsilon]$

Our type system is (implicitly) parameterized with respect to a set of dynamic "security constraints" of the form $\mathcal{O}(g_0, g)$, saying that an ambient of group g_0 is allowed to be opened while enclosed in an ambient of group g, One can view these constraints as prescriptive (thus provided by the user); establishing that a process is well-typed then verifies that no other interactions may happen. Alternatively, one can take a descriptive view: a type inference algorithm might deduce the least set of constraints needed for typability. In Example 1 we have $\mathcal{O}(K, K')$, $\mathcal{O}(K', W)$, $\mathcal{O}(K'', W)$

We shall use G to range over sets of groups, and 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

$$
\begin{array}{rcll}
a \in \mathsf{Actions} & ::= & {}^{H}\mathsf{enter}(g) & \text{steers an ambient from } H \text{ to } g \\
 & \mid & {}^{g}\mathsf{exit}(H) & \text{steers an ambient from } g \text{ to } H \\
 & \mid & {}_{G}\mathsf{open}(g) & \text{if executed in } G\text{, opens } g \\
 & \mid & \mathsf{put}(\sigma) & \text{output tuple of type } \sigma \\
 & \mid & \mathsf{get}(\sigma) & \text{input tuple of type } \sigma \\
tr \in \mathsf{Traces} & ::= & a^{*} & \text{finite sequence of actions} \\
b \in \mathsf{Behaviors} & \subseteq & \mathcal{P}(\mathsf{Traces}) & \text{non-empty regular set of traces} \\
B \in \mathsf{BehContxt} & ::= & \square \mid a.B \mid (b \mathbin{\boldsymbol{|}} B) & \\
br \in \mathsf{BehRows} & ::= & \{g_i : b_i\}_{i \in I} & \text{behaves as } b_i \text{ when opened in } g_i \\
\sigma \in \mathsf{Tuples} & ::= & \times(\tau_1, \ldots, \tau_k) & \text{we write } \tau_1 \text{ if } k = 1 \\
\tau \in \mathsf{Types} & ::= & \mathsf{amb}^{g}_{H}[br] & \text{type of ambient name} \\
 & \mid & \mathsf{cap}[B] & \text{type of capability} \\
 & \mid & \mathsf{int} \mid \mathsf{real} \mid \mathsf{bool} & \text{base types (optional)}
\end{array}
$$

Fig. 1. Syntax of Types and Behaviors.

$g' \in G$. 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 $K'^{\uparrow} = \{K', W\}$.

3 Types and Behaviors

We have hinted at the form of types τ and behaviors b, to be defined mutually recursively in Fig. 1.

Concerning behaviors, we have seen the need for constants like ε, and we clearly also need operators like $b_1 \mathbin{\boldsymbol{|}} b_2$ (to model parallel composition) and $a.b$ (to model sequential prefixing). Ultimately, a behavior must approximate sets of traces, so why not let (unlike what we did in [AKPG01]) a behavior *be* a nonempty regular set of finite traces (cf. [RV97] where types are graphs), so as to give the user more freedom in specification? We can then *define* the above operators: ε as $\{\bullet\}$ where $\bullet$ is the empty sequence; $a.b$ as $\{a \diamond tr \mid tr \in b\}$ where $\diamond$ denotes concatenation; and $b_1 \mathbin{\boldsymbol{|}} b_2$ as $\bigcup_{tr_1 \in b_1, tr_2 \in b_2} tr_1 \parallel tr_2$ where $tr_1 \parallel tr_2$

denotes all traces that can be formed by arbitrarily interleaving tr_1 with tr_2.

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

Concerning types of the form $\mathsf{cap}[B]$, they are for the typing of capabilities (like $\mathsf{open}\ n$). Here B is a *behavior context*, that is a "behavior with a hole inside"; we write $B\lfloor b \rfloor$ for the result of "plugging" b into the hole of B. The notion of behavior contexts was introduced in [AKPG01] and used also in, e.g., [GYY01]); it conveniently expresses the result of prefixing, as illustrated by the situation where 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. 2, ${}_{g_0}\mathsf{open}(g).(b_0 \mid b)$ is the result of (Proc Action) plugging b into $\mathsf{cap}[{}_{g_0}\mathsf{open}(g).(b_0 \mid \Box)]$ which by (Exp Open) is the type of $\mathsf{open}\ n$.

4 Ordering Relations

The type system (Fig. 2) uses subsumption rules, based on an ordering $\tau_1 \leqslant \tau_2$ on types (subtyping), and an ordering $b_1 \leqslant b_2$ on behaviors (subbehaviors). These orderings are defined in a mutually recursive way, together with orderings on actions, traces, etc. To ensure that this is well-defined, 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 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. Then, as detailed in the subsequent paragraphs, a relation on level i types induces a relation on level i tuples which induces a relation on level i actions which induces a relation on level i traces 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, can be summarized by stating that the constructors have the following polarity:

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

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 ${}^{H}\mathsf{enter}(g)$. Concerning the actions for communication, we have 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. 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.

Concerning the relation $b_1 \leqslant b_2$, with the intuitive interpretation that b_2 is more "permissive" than b_1, we would expect, e.g., that $a_1.a_2 \leqslant a_1 \mid a_2$ since the right hand side does not prescribe which action comes first. It might be tempting to come up with a set of axioms, but it is much more fruitful to take a semantic approach (cf. the observations in [CC91] about recursive types) and stipulate 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 $a_1 \leqslant a_2$ (if $tr_1 \leqslant tr_2$ then tr_1 and tr_2 have the same length). Note that $b_1 \leqslant b_2$ and $b_2 \leqslant b_1$ does not necessarily imply $b_1 = b_2$ (for instance, let $b_1 = \{a_1\}$ and let $b_2 = \{a_1, a_2\}$ with $a_2 \leqslant a_1$).

The relation $\leqslant$ on behavior rows is defined as follows: 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 by stipulating that $B_1 \leqslant B_2$ holds iff for all (level 0) behaviors b we have $B_1\lfloor b \rfloor \leqslant B_2\lfloor b \rfloor$. 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 one can show it to be superfluous: if $B_1 \leqslant B_2$ then $B_1\lfloor b \rfloor \leqslant B_2\lfloor b \rfloor$ holds for all b.

The relation $\tau_1 \leqslant \tau_2$, with the intuitive interpretation that an expression of type τ_1 also has type τ_2, can be summarized by stating that $\mathsf{int} \leqslant \mathsf{real}$ and that we have the polarity $\mathsf{cap}[\oplus]$. Concerning the polarity of the type $\mathsf{amb}^g_H[br]$, it turns out that the proof of subject reduction for the type system of Fig. 2 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.

5 Type System

Our type system is defined in Fig. 2, where E is an environment mapping variables to types. For expressions M, defined as in classical AC, the judgements are of the form $E \vdash M : \tau$; we have already motivated the rule ($\mathsf{Exp\ Open}$) and the other rules are similar or simpler.

Processes P are as in classical AC, except that in order for the semantics to express *which* of identically named ambients that make a move, each ambient is given a unique *tag* (as is customary in flow logic [NNHJ99]). We use ξ, χ to range over tags. Judgements are of the form $\Delta, E \vdash_g P : b$, where Δ maps an ambient tag into the behavior of the process inside that ambient, and where g is the group of the enclosing ambient (to be used in the rule ($\mathsf{Proc\ Amb}$)). We have already motivated the rule ($\mathsf{Proc\ Action}$). Concerning the rule ($\mathsf{Proc\ Repl}$), the side condition ensures that the behavior of a replicated process is in fact invariant under replication.

The crux of the type system is the rule ($\mathsf{Proc\ Amb}$), where the situation is that an ambient tagged ξ and with type $\mathsf{amb}^{g_0}_{H}[br]$ contains a process P and is enclosed by an ambient of group g. We need to check that H contains (the groups of) all ambients that may possibly contain ξ, and therefore need to know the initial location of ξ which is $g^{\uparrow}$. (It is for that purpose that the judgments carry a g component.) When typing P, the enclosing ambient is g_0, the group of ξ, and we must be able to assign P the behavior $b = \Delta(\xi)$. Moreover, it must hold that $(g^{\uparrow}, b) \overset{g_0}{\leadsto} (H, br)$ which is a shorthand for the following (informally stated) complex property:

Assumption 1: Let tr belong to b, of the form $tr_1 \diamond tr_2$.
Assumption 2: tr_1 can be executed "without environment interaction", i.e.,
- all occurrences of $\mathsf{put}(\_)$ and $\mathsf{get}(\_)$ in tr_1 come in pairs, with $\mathsf{put}(\_)$ immediately preceding $\mathsf{get}(\_)$;
- there is no "jump of location" in tr_1. *Example*: the trace depicted in (5) has a jump of location, after its first action.

Assumption 3: Let H_1 be the "final destination" of tr_1. *Example:* the final destination of ${}^{g_1}\mathsf{exit}(g_2) \diamond {}^{g_2}\mathsf{enter}(g_3)$ is ${g_3}^{\uparrow}$.
Consequence 1: H_1 must be a subset of H (so that H indeed is an upper bound of where ξ may end up).
Consequence 2: If tr_2 starts with $\mathsf{put}(\sigma_1)\,\mathsf{get}(\sigma_2)$ then $\sigma_1 \leqslant \sigma_2$ (well-defined topic of conversation)
Consequence 3: If tr_2 starts with ${}_G\mathsf{open}(\_)$ then $g_0 \in G$.
Consequence 4: With $br = \{g_i : b_i\}_{i \in I}$, for all $i \in I$: if $g_i \in H_1$ then $\{tr_2\} \leqslant b_i$. That is, if ξ after executing tr_1 can be inside g_i, then b_i, approximating what happens after ξ is opened within g_i, better contain tr_2.

Non-structural Rules

$$\text{(Proc Subsumption)}\quad \frac{\Delta, E \vdash_g P : b}{\Delta, E \vdash_g P : b'}\ (b \leqslant b') \qquad\qquad \text{(Exp Subsumption)}\quad \frac{E \vdash M : \tau}{E \vdash M : \tau'}\ (\tau \leqslant \tau')$$

Expressions

$$\text{(Exp } n\text{)}\quad \frac{E(n) = \tau}{E \vdash n : \tau} \qquad \text{(Exp } \epsilon\text{)}\quad \frac{}{E \vdash \epsilon : \mathsf{cap}[\Box]} \qquad \text{(Exp Action)}\quad \frac{E \vdash M_1 : \mathsf{cap}[B_1] \quad E \vdash M_2 : \mathsf{cap}[B_2]}{E \vdash M_1.M_2 : \mathsf{cap}[B_1\lfloor B_2\rfloor]}$$

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

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

Processes

$$\text{(Proc Zero)}\quad \frac{}{\Delta, E \vdash_g \mathbf{0} : \varepsilon} \qquad \text{(Proc Par)}\quad \frac{\Delta, E \vdash_g P_1 : b_1 \qquad \Delta, E \vdash_g P_2 : b_2}{\Delta, E \vdash_g P_1 \mid P_2 : b_1 \mid b_2} \qquad \text{(Proc Repl)}\quad \frac{\Delta, E \vdash_g P : b}{\Delta, E \vdash_g !P : b} \quad \text{if } (b \mid b) \leqslant b$$

$$\text{(Proc Res)}\quad \frac{\Delta, E, n : \mathsf{amb}^{g_0}_H[br] \vdash_g P : b}{\Delta, E \vdash_g (\nu n : \mathsf{amb}^{g_0}_H[br]).P : b} \qquad \text{(Proc Action)}\quad \frac{E \vdash M : \mathsf{cap}[B] \qquad \Delta, E \vdash_g P : b}{\Delta, E \vdash_g M.P : B\lfloor b\rfloor}$$

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

$$\text{(Proc Input)}\quad \frac{\Delta, E, n_1 : \tau_1, \cdots, n_k : \tau_k \vdash_g P : b}{\Delta, E \vdash_g (n_1 \ldots n_k : \tau_1 \ldots \tau_k).P : \mathsf{get}(\sigma).b} \quad \sigma = \times(\tau_1, \ldots, \tau_k) \quad \text{(Proc Output)}\quad \frac{\forall i \in \{1 \ldots k\} : E \vdash M_i : \tau_i}{\Delta, E \vdash_g \langle M_1 \ldots M_k\rangle : \mathsf{put}(\sigma)}$$

Fig. 2. Typing Rules.

6 Semantic Soundness

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". For space reasons, we omit a complete listing of the rules, and only list a few examples (for (Red Comm), we assume that a unary tuple is communicated):

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

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

$$m[(n:\tau).P \mid \langle V \rangle \mid Q]^{\xi} \xrightarrow{\xi:\mathsf{comm}\ \tau} m[P[n:=V] \mid Q]^{\xi} \qquad \text{(Red Comm)}$$

$$!P \xrightarrow{\epsilon} P' \mid !P \quad \text{if } P' \text{ and } P \text{ are equal except for the tags} \qquad \text{(Red Repl)}$$

Note that (Red Open), as does (Red Comm), deviates from the "standard" semantics in that we provide the enclosing ambient so as to record in ℓ *where* the opening (or communication) has taken place. Another deviation from the standard semantics (but also seen in, e.g., [NNS00]) is that rather than having a rule $!P \equiv P \mid !P$ (along with non-controversial congruence rules like $P \mid \mathbf{0} \equiv P$ and $P \mid Q \equiv Q \mid P$), we allow this "unfolding" to take place via the rule (Red Repl). The reason is that otherwise it seems hard to establish "subject congruence". Note that (Red Repl) is designed to permit also the reduct to be uniquely tagged.

That our type system is semantically sound can now be stated using a subject reduction result, intuitively stating that "well-typed processes never evolve into ill-typed processes" and also stating that "well-typed processes behave according to their behavior". One case of this result is as follows (other cases are similar):

Theorem 2 *Suppose that $P \xrightarrow{\xi:\mathsf{comm}\ \sigma} Q$ where P and Q are uniquely tagged. Further assume that*

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

Then there exists Δ' which agrees with Δ on $\mathrm{dom}(\Delta) \setminus \{\xi\}$ such that

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

and such that there exists $\sigma' \leqslant \sigma$ with $\mathsf{put}(\sigma').\mathsf{get}(\sigma).\Delta'(\xi) \leqslant \Delta(\xi)$.

7 Type checking

Given a complete type derivation for some process P, we can check its validity according to the rules from Fig. 2. To see this, first observe that behaviors can be represented as finite automata (cf. the regular types of [Nie93]), with transitions labeled by actions.

Next observe that the relations $\leqslant$ defined in Sect. 4 are decidable, with the decision procedures defined mutually recursively; the only non-trivial issues are

(1) given a procedure for deciding $\leqslant$ on level i actions, construct a procedure for deciding $\leqslant$ on level i behaviors;
(2) given a procedure for deciding $\leqslant$ on level i behaviors, construct a procedure for deciding $\leqslant$ on level i behavior contexts.

We first address 1. Given b_1 and b_2, recognized by ϵ-transition-free automata A_1 and A_2, we must decide whether $b_1 \leqslant b_2$. For that purpose, we construct[6] a "difference automaton" $A_1 \setminus A_2$, and checks whether $A_1 \setminus A_2$ rejects all inputs.

We next address 2. Given B_1 and B_2, we must decide whether $B_1 \leqslant B_2$, that is whether $B_1\lfloor b \rfloor \leqslant B_2\lfloor b \rfloor$ for all behaviors b. This might seem infeasible to check, but fortunately, we can show (much as in [AKPG01]) that it is sufficient to check whether $B_1\lfloor \mathsf{test} \rfloor \leqslant B_2\lfloor \mathsf{test} \rfloor$, where test is chosen so as to be incomparable with all actions occurring in B_1 and B_2.

Finally, to check whether $(g^{\uparrow}, b) \overset{g_0}{\rightsquigarrow} (H, br)$ holds, we annotate each state in the automaton for b with the sets of groups that may enclose g_0 at the given point. The detailed construction is rather involved; again, we refer to [Amt02] for the details.

The decision procedures are thus potentially very expensive for entities of high level, but 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. Another concern is that the size of the automaton for a behavior is exponential in the number of parallel constructs; in practice, however, there may never be more than a few processes running in parallel. In general, 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).

In order to come up with an algorithm for type *reconstruction*, it seems that we would at least need to impose several restrictions (none of which appears to exclude most commonly met processes) on the form of the input: *(i)* only *ambients* can be replicated (as given $b_0 \neq \varepsilon$, it is not obvious how to find b

[6] The states of $A_1 \setminus A_2$ 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 $a \leqslant a^+$.

with $b_0 \leqslant b$ such that $b \mid b \leqslant b$); *(ii)* no ambient names are communicated, and if capabilities are communicated then the user has to provide their full type; *(iii)* 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 $g' \prec g$.

8 Conclusion

We have demonstrated that it is possible to use *type-based* methods to develop a precise analysis of the *classical* ambient calculus, by presenting a *flow-sensitive* type system where *behaviors* summarize the actions of ambients. Behaviors are regular sets of traces; therefore finite automata can be used for *type checking*. We encourage implementing the type checking algorithm and measuring its actual performance, as well as taking steps towards type reconstruction.

Acknowledgements

The author would like to thank Michele Bugliesi, Assaf Kfoury, and Santiago Pericas-Geertsen for inspiring discussions, and the latter two for commenting upon an early draft of [Amt02].

References

[AKPG01] Torben Amtoft, Assaf J. Kfoury, and Santiago M. Pericas-Geertsen. What are polymorphically-typed ambients? In David Sands, editor, *ESOP 2001, Genova*, volume 2028 of *LNCS*, pages 206–220. Springer-Verlag, April 2001.

[AKPG02] Torben Amtoft, Assaf J. Kfoury, and Santiago M. Pericas-Geertsen. Orderly communication in the ambient calculus. *Computer Languages, Systems & Structures*, 28:29–60, 2002.

[Amt02] Torben Amtoft. Causal type system for ambient movements. Technical Report 2002-04, Department of Computing and Information Sciences, Kansas State University, 2002.

[AMW04] Torben Amtoft, Henning Makholm, and J. B. Wells. PolyA: True type polymorphism for mobile ambients. In J.-J. Levy, E. W. Mayr, and J. C. Mitchell, editors, *TCS 2004 (3rd IFIP International Conference on*

Theoretical Computer Science), Toulouse, France, August 2004, pages 591–604. Kluwer Academic Publishers, 2004.

[ANN99] Torben Amtoft, Flemming Nielson, and Hanne Riis Nielson. *Type and Effect Systems: Behaviours for Concurrency.* Imperial College Press, 1999.

[AT00] Torben Amtoft and Franklyn Turbak. Faithful translations between polyvariant flows and polymorphic types. In *ESOP 2000, Berlin*, volume 1782 of *LNCS*, pages 26–40. Springer-Verlag, 2000.

[BC01] Michele Bugliesi and Giuseppe Castagna. Secure safe ambients. In *POPL 2001, London*, pages 222–235. ACM Press, 2001.

[BCC01] Michele Bugliesi, Giuseppe Castagna, and Silvia Crafa. Boxed ambients. In *4th International Conference on Theoretical Aspects of Computer Science (TACS'01)*, volume 2215 of *LNCS*, pages 38–63. Springer-Verlag, 2001.

[BCDCG04] Eduardo Bonelli, Adriana Compagnoni, Mariangiola Dezani-Ciancaglini, and Pablo Garralda. Boxed ambients with communication interfaces. In *29th International Symposium on Mathematical Foundations of Computer Science (MFCS 2004)*, volume 3153 of *LNCS*, pages 119–148. Springer-Verlag, 2004.

[BCF02] Chiara Braghin, Agostino Cortesi, and Riccardo Focardi. Security boundaries in mobile ambients. *Computer Languages*, 28(1):101–127, November 2002.

[BCMS02] Michele Bugliesi, Silvia Crafa, Massimo Merro, and Vladimiro Sassone. Communication interference in mobile boxed ambients. In *FST & TCS 2002*, volume 2556 of *LNCS*, pages 71–84. Springer-Verlag, 2002.

[BPG00] Michele Bugliesi and Santiago Pericas-Geertsen. Depth subtyping and type inference for object calculi. In *Proc. of the 7th Int. Workshop on Foundations of Object Oriented Languages (FOOL'7)*, 2000. Electronic Proceedings: `http://www.cis.upenn.edu/~bcpierce/FOOL//FOOL7.html`.

[CC91] Felice Cardone and Mario Coppo. Type inference with recursive types: Syntax and semantics. *Information and Computation*, 92:48–80, 1991.

[CDC02] M. Coppo and M. Dezani-Ciancaglini. A fully abstract model for higher-order mobile ambients. In *VMCAI'02*, volume 2294 of *LNCS*, pages 255–271. Springer-Verlag, 2002.

[CDCGS03] Mario Coppo, Mariangiola Dezani-Ciancaglini, Elio Giovannetti, and Ivano Salvo. M3: Mobility types for mobile processes in mobile ambients. In *CATS 2003*, volume 78 of *Electronic Notes in Theoretical Computer Science*, 2003.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In Maurice Nivat, editor, *Foundations of Software Science and Computational Structures (FOSSACS'98)*, volume 1378 of *LNCS*, pages 140–155. Springer-Verlag, 1998.

[CG99] Luca Cardelli and Andrew D. Gordon. Types for mobile ambients. In *POPL'99, San Antonio, Texas*, pages 79–92. ACM Press, 1999.

[CGG99] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Mobility types for mobile ambients. In Jiri Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP'99*, volume 1644 of *LNCS*, pages 230–239. Springer-Verlag, 1999. Extended version appears as Microsoft Research Technical Report MSR-TR-99-32, 1999.

[CGG00] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Ambient groups and mobility types. In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000), Tohoku University, Sendai, Japan*, volume 1872 of *LNCS*, pages 333–347. Springer-Verlag, August 2000.

[Cou97] Patrick Cousot. Types as abstract interpretations. In *POPL'97, Paris*, pages 316–331. ACM Press, 1997.

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

[GH99] Simon Gay and Malcolm Hole. Types and subtypes for client-server interactions. In *ESOP'99*, volume 1576 of *LNCS*, pages 74–90. Springer-Verlag, 1999.

[GYY00] Xudong Guan, Yiling Yang, and Jinyuan You. Making ambients more robust. In *ICS2000, Beijing, China*, 2000.

[GYY01] Xudong Guan, Yiling Yang, and Jinyuan You. Typing evolving ambients. *Information Processing Letters*, 80(5):265–270, November 2001.

[HJNN99] Rene Rydhof Hansen, Jacob Grydholt Jensen, Flemming Nielson, and Hanne Riis Nielson. Abstract interpretation of mobile ambients. In *SAS'99*, volume 1694 of *LNCS*, pages 134–148. Springer-Verlag, 1999.

[LB04] Francesca Levi and Chiara Bodei. A control flow analysis for safe and boxed ambients. In *ESOP'04 (European Symposion on Programming)*, volume 2986 of *LNCS*, pages 188–203. Springer-Verlag, 2004.

[Lev03] Francesca Levi. Types for evolving communication in safe ambients. In L.D. Zuck, P.C. Attie, A. Cortesi, and S. Mukhopadhyay, editors, *4th. Int. Conference on Verification Model Checking and Abstract Interpretation (VMCAI'03)*, volume 2575 of *LNCS*, pages 102–115. Springer-Verlag, 2003.

[LG88] John M. Lucassen and David K. Gifford. Polymorphic effect systems. In *ACM Symposium on Principles of Programming Languages*, pages 47–57. ACM Press, 1988.

[LM04] Francesca Levi and Sergio Maffeis. On abstract interpretation of mobile ambients. *Information and Computation*, 188(2):179–240, January 2004.

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

[MS02] Massimo Merro and Vladimiro Sassone. Typing and subtyping mobility in boxed ambients. In *CONCUR'02*, volume 2421 of *LNCS*, pages 304–320. Springer-Verlag, 2002.

[MW05] Henning Makholm and J.B. Wells. Instant polymorphic type systems for mobile process calculi: Just add reduction rules and close. In Mooly Sagiv, editor, *ESOP 2005, Edinburgh*, volume 3444 of *LNCS*, pages 389–407. Springer-Verlag, April 2005.

[NAN98] Hanne Riis Nielson, Torben Amtoft, and Flemming Nielson. Behaviour analysis and safety conditions: a case study in CML. In *FASE'98 (part of ETAPS'98)*, volume 1382 of *LNCS*, pages 255–269. Springer-Verlag, 1998. Also appears as technical report DAIMI PB-528, October 1997.

[Nie93] Oscar Nierstrasz. Regular types for active objects. In *OOPSLA '93*, volume 28 of *ACM SIGPLAN Notices*, pages 1–15, 1993.

[NN93] Flemming Nielson and Hanne Riis Nielson. From CML to process algebras. In *CONCUR '93*, volume 715 of *LNCS*, pages 493–508. Springer-Verlag, 1993. Full version appears in TCS, 155(1):179–219, 1996.

[NN94] Hanne Riis Nielson and Flemming Nielson. Higher-order concurrent programs with finite communication topology. In *ACM Symposium on Principles of Programming Languages*, pages 84–97. ACM Press, January 1994.

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

[NNH99] Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of Program Analysis.* Springer-Verlag, 1999. Web page at `www.imm.dtu.dk/~riis/PPA/ppa.html`.

[NNHJ99] Flemming Nielson, Hanne Riis Nielson, Rene Rydhof Hansen, and Jacob Grydholt Jensen. Validating firewalls in mobile ambients. In *Proc. CONCUR'99*, volume 1664 of *LNCS*, pages 463–477. Springer-Verlag, 1999.

[NNP04] Hanne Riis Nielson, Flemming Nielson, and Henrik Pilegaard. Spatial analysis of BioAmbients. In R. Giacobazzi, editor, *SAS 2004 (11th Static Analysis Symposium)*, volume 3148 of *LNCS*, pages 69–83. Springer-Verlag, 2004.

[NNS00] Flemming Nielson, Hanne Riis Nielson, and Mooly Sagiv. A Kleene analysis of mobile ambients. In *ESOP 2000, Berlin*, volume 1782 of *LNCS*, pages 305–319. Springer-Verlag, 2000.

[NP05] Mayur Naik and Jens Palsberg. A type system equivalent to a model checker. In Mooly Sagiv, editor, *ESOP 2005, Edinburgh*, volume 3444 of *LNCS*, pages 374–388. Springer-Verlag, April 2005.

[PP01] Jens Palsberg and Christina Pavlopoulou. From polyvariant flow information to intersection and union types. *Journal of Functional Programming*, 11(3):263–317, May 2001. A preliminary version appeared in POPL'98, pages 197–208.

[Rep91] John H. Reppy. CML: A higher-order concurrent language. In *SIGPLAN'91 Conference on Programming Language Design and Implementation*, June 1991.

[RV97] Antnio Ravara and Vasco T. Vasconcelos. Behavioural types for a calculus of concurrent objects. In *Euro-Par'97*, volume 1300 of *LNCS*, pages 554–561. Springer-Verlag, 1997. Extended version in Technical Report 97-6, Dep. of Mathematics, Technical University of Lisbon, 1997.

[TJ94] Jean-Pierre Talpin and Pierre Jouvelot. The type and effect discipline. *Information and Computation*, 111:245–296, 1994.

[TZH02] David Teller, Pascal Zimmer, and Daniel Hirschkoff. Using ambients to control resources. In *CONCUR'02*, volume 2421 of *LNCS*, pages 288–303. Springer-Verlag, 2002.

[Yos96] Nobuko Yoshida. Graph types for monadic mobile processes. In *16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'96)*, volume 1180 of *LNCS*, pages 371–386. Springer-Verlag, 1996.

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