Hybrid Logical Analyses of the Ambient Calculus

Thomas Bolander[1] and René Rydhof Hansen[2]

[1] Informatics and Mathematical Modelling, Technical University of Denmark
tb@imm.dtu.dk
[2] Department of Computer Science, University of Copenhagen
rrhansen@diku.dk

Abstract. In this paper, hybrid logic is used to formulate a rational reconstruction of a previously published control flow analysis for the mobile ambients calculus and we further show how a more precise *flow-sensitive* analysis, that takes the ordering of action sequences into account, can be formulated in a natural way. We show that hybrid logic is very well suited to express the semantic structure of the ambient calculus and how features of hybrid logic can be exploited to reduce the "administrative overhead" of the analysis specification and thus simplify it. Finally, we use *HyLoTab*, a fully automated theorem prover for hybrid logic, both as a convenient platform for a prototype implementation as well as to formally prove the correctness of the analysis.

Keywords: Hybrid logic, static analysis, mobile ambients

1 Introduction

With the increased, and increasing, focus on making secure and reliable systems also comes an increased need for advanced tools and techniques that can verify important security and reliability properties of a system in all phases of design, modelling, and implementation. In recent years various forms of *static analysis* has gained popularity as basis for verification tools, in part because static analysis can be fully automated and in part because of the ability of static analysis to scale to even the largest systems. There has also been a growing interest in applying static analysis techniques to the verification of more or less formal system models, e.g., models formulated as UML diagrams or process calculi, to verify relevant safety and security properties before starting the actual implementation.

The main goal of this paper is to show how *hybrid logics* can be applied to the specification, validation, and implementation of static analyses. We do this by developing three increasingly precise analyses of the *mobile ambients* process calculus [6] which is used to model distributed systems with a high degree of mobility. The analyses are specified using hybrid logics, in a way that is inspired by the Flow Logic approach [10]; the correctness of the analyses are proved using the *HyLoTab* theorem prover for hybrid logics [11]; and by exploiting the model generation capabilities of *HyLoTab* we also automatically obtain implementations of the analyses directly from their respective specifications. In addition to the obvious convenience of getting an implementation for free, it also obviates the need for proving the correctness of the implementation.

Furthermore, we argue that hybrid logics are particularly well-suited for specifying analyses for the mobile ambients calculus because the structure of models for hybrid logic formulae is very similar to the structures underlying the semantics of mobile ambients. In addition, the nominals, intrinsic to hybrid logics, provides an easy way to handle the analysis of bound names in the ambient calculus and thereby avoid the additional technical complexity otherwise necessary, cf. [7, 8].

Finally, while the intention of this paper is to show some of the advantages of using hybrid logics for static analysis, the developed analyses are not mere "toy"-analyses only useful for demonstration purposes. Indeed the analysis described in Section 6 is comparable to the 0CFA analyses of [7, 8] and the analysis defined in Section 7 incorporates an element of *flow sensitivity*, by taking the ordering of action sequences into account, that markedly and non-trivially improves precision of the analysis. Although this flow-sensitive analysis is unlikely to be as precise as the "counting analysis" of [7] it does have the advantage of retaining a simple specification, that is straightforward to prove correct, as well as being easily implementable.

Related Work

The mobile ambients calculus has been studied in a number of contexts and several type systems and static analyses have been developed for it, see for example [4, 5, 7, 8, 2, 1, 9]. The analyses in this paper are mainly inspired by the control flow analyses developed in the Flow Logic framework [7, 8]. However, in none of these works are logics used so pervasively and essentially to obtain clear analysis specifications, computer assisted correctness proofs, and automated implementation. We are also not aware of any prior use of hybrid logics for static analysis.

2 Mobile Ambients

Process algebras have long been used for modelling concurrent and distributed systems. These models have been invaluable in studying and solving some of the many problems inherent in such systems such as deadlocking, synchronisation, fairness etc.

The *ambient calculus* is a process calculus specifically designed to model *mobility* of processes. This is in contrast to traditional process calculi, such as CSP and the π-calculus, that focus on *communication* between processes. In particular it is possible for an active process and its environment to move between sites. The entities that move in the calculus are called ambients and they may contain other active processes as well as other ambients. This gives rise to a tree structure that changes dynamically as ambients move.

It is exactly this tree structure the static analyses discussed in later sections are designed to approximate. The primary goal of the analyses is to ensure that *all possible* concrete tree structures that may occur during the execution of a program are represented in the analysis result. Since this is undecidable in general, the analyses have to *over-approximate* the set of actual tree structures

$P, Q ::= (\nu n)P$		restriction
	$\mid$ $\mathbf{0}$	inactivity
	$\mid$ $P \mid Q$	composition
	$\mid$ $!P$	replication
	$\mid$ $n[P]$	ambient
	$\mid$ `in` $n.P$	capability to enter n
	$\mid$ `out` $n.P$	capability to exit n
	$\mid$ `open` $n.P$	capability to open n

Fig. 1. Ambient Syntax.

and thus an analysis result may contain tree structures that will never actually occur in a program execution.

In this paper we focus on the core calculus and thus do not take communication into account. As shown in [6] the core calculus is Turing-complete.

Syntax

We assume the existence of a countably infinite set Nam of *names*. The metavariables k, l, m, n, and so on, range over names. The syntax of the mobile ambient calculus is defined in Figure 1. The restriction operator $(\nu n)P$ creates a new name n with scope P; the inactive process, i.e., a process that does nothing, is denoted by $\mathbf{0}$; processes P and Q running in parallel is represented by $P \mid Q$; replication, $!P$, is equivalent to an unbounded number of copies of P running in parallel, thereby providing a recursive operator.

By $n[P]$ we denote the ambient named n that has the process P running inside it. The capabilities `in` n and `out` n are used to move their enclosing ambients whereas `open` n is used to dissolve the boundary of a sibling ambient; this will be made precise when we define the semantics below. We write $\mathrm{fn}(P)$ for the free names of P. Trailing occurrences of the inactive process $\mathbf{0}$ will often be omitted.

Semantics

The semantics of the mobile ambients calculus is defined as a straightforward *reduction semantics* (see Figure 3) using a *congruence relation* (see Figure 2) as is common for process calculi.

The essential idea is that the ambient hierarchy, i.e., the structure determining which ambients are inside which other ambients, can be changed dynamically by ambients entering or exiting other ambients or even by dissolving another ambient. These notions are formalised as the (`in`), (`out`), and (`open`) reduction shown in Figure 3. In later sections these rules will be visualised and explained in more detail. The (`amb`) rule allows processes to be active even when moved around in the ambient hierarchy. The (ν) rule defines a scope for the name bound at the ν in much the same way that λ acts as a name binder in the λ-calculus. In the ambient calculus α-equivalent processes, i.e., processes that are identical modulo α-renaming of bound names, are considered to be *identical*.

$$
\begin{array}{ll}
P \equiv P & P \mid Q \equiv Q \mid P \\
P \equiv Q \Rightarrow Q \equiv P & (P \mid Q) \mid R \equiv P \mid (Q \mid R) \\
P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R & !P \equiv P \mid !P \\
 & \\
P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q & P \equiv Q \Rightarrow \texttt{in}\ n.P \equiv \texttt{in}\ n.Q \\
P \equiv Q \Rightarrow P \mid R \equiv Q \mid R & P \equiv Q \Rightarrow \texttt{out}\ n.P \equiv \texttt{out}\ n.Q \\
P \equiv Q \Rightarrow\ !P \equiv !Q & P \equiv Q \Rightarrow \texttt{open}\ n.P \equiv \texttt{open}\ n.Q \\
P \equiv Q \Rightarrow n[P] \equiv n[Q] & \\
 & (\nu n)(\nu m)P \equiv (\nu m)(\nu n)P \\
P \mid \mathbf{0} \equiv P & (\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \\
(\nu n)\mathbf{0} \equiv \mathbf{0} & \quad \text{if } n \notin \mathrm{fn}(P) \\
!\mathbf{0} \equiv \mathbf{0} & (\nu n)(m[P]) \equiv m[(\nu n)P] \\
 & \quad \text{if } n \neq m
\end{array}
$$

Fig. 2. Structural congruence.

$$n[\ \texttt{in}\ m.P \mid Q\ ] \mid m[R] \rightarrow m[\ n[P \mid Q] \mid R\ ] \qquad (\texttt{in})$$

$$m[\ n[\texttt{out}\ m.P \mid Q] \mid R\ ] \rightarrow n[P \mid Q] \mid m[R] \qquad (\texttt{out})$$

$$\texttt{open}\ n.P \mid n[Q] \rightarrow P \mid Q \qquad (\texttt{open})$$

$$\frac{P \rightarrow Q}{n[P] \rightarrow n[Q]}\ (\text{amb}) \qquad \frac{P \rightarrow Q}{(\nu n)P \rightarrow (\nu n)Q}\ (\nu)$$

$$\frac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R}\ (\mid) \qquad \frac{P' \equiv P \qquad P \rightarrow Q \qquad Q \equiv Q'}{P' \rightarrow Q'}\ (\equiv)$$

Fig. 3. Reduction relation.

This equivalence often gives rise to technical difficulties in the design of static analyses and heavy technical machinery, like annotating the syntax of processes with labels or markers that remain stable under α-conversion, frequently has to be employed to overcome the inherent difficulties in handling α-equivalence. In a later section we show how such difficulties can be completely avoided by encoding ambient names as nominals in hybrid logic and thereby significantly simplify the technical developments needed for an analysis of the ambient calculus. Processes can be composed in parallel and executed in an interleaved way, as shown by the $(\mid)$-rule. Finally, the $(\equiv)$-rule integrates the congruence relation defined in Figure 2 into the reduction rules. We will not go into further detail with the semantics here but refer the reader to later sections and [6].

Analysing Ambients

As mentioned earlier, the essential and most basic property to analyse in the ambient calculus, is how the ambient hierarchy, i.e., the tree structure determined by which ambients are contained inside which other ambients, may dynamically develop during execution. The analyses described in this paper all compute over-approximations of this dynamically evolving structure. Such analyses are comparable to computing control flow graphs for imperative languages or performing

control flow analysis of a functional language. Indeed, in keeping with [7, 8], we call such analyses of the ambient calculus *control flow analyses*. In [7, 8] the Flow Logic framework is used to define a number of control flow analyses for the ambient calculus. The Flow Logic framework [10] is a specification-oriented framework for defining static analyses where specification of the analysis is clearly separated from the computation of the analysis result.

In the Flow Logic framework analyses are defined by specifying what it means for a proposed analysis result to be correct (rather than how to compute it). In this way an analysis designer can focus on high-level aspects such as what properties to analyse and what correctness means without having to take low-level implementation issues into account. This approach gives a framework that is light-weight and well-suited for rapidly developing a large variety of analyses. The analyses described in later sections are developed in a way that is inspired by the Flow Logic framework.

An analysis specified using Flow Logic is typically implemented by systematically transforming the analysis specification into a constraint generator and then use an appropriate constraint solver to perform the actual (fixed-point) computation of the analysis result. It is often a trivial but tedious task to prove that the constraint generator correctly implements the analysis specification. In this paper we show how the model-generating capabilities of the *HyLoTab* theorem prover for hybrid logics enables us to automatically obtain an implementation for performing the necessary fixed-point computations directly from the high-level analysis specification and thereby obviating the need for proving correctness of the implementation: it is correct by construction.

3 Hybrid Logic

Hybrid logic is an extension of propositional modal logic. Both modal and hybrid logic are thoroughly introduced in [3]. Hybrid logic is obtained from propositional modal logic by adding a second sort of propositional symbols, called *nominals*. We assume that a countably infinite set Nom of nominals is given. The metavariables a, b, c, and so on, range over nominals. The semantic difference between ordinary propositional symbols and nominals is that nominals are required to be true at *exactly one* world; that is, a nominal "points to a unique world". In addition to the nominals we have a countable set of ordinary *propositional symbols*, Prop. We use the metavariables p, q, r, and so on, to range over propositional symbols. We assume that the sets Nom and Prop are disjoint. The hybrid logic we consider has the following syntax:

$$\phi ::= \top \mid p \mid a \mid \neg\phi \mid \phi \wedge \phi \mid \Diamond\phi \mid \Diamond^{-}\phi \mid @_a\phi \mid \downarrow a.\phi$$

where $p \in \mathsf{Prop}$ and $a \in \mathsf{Nom}$. This hybrid logic is usually denoted $HL(@, {}^{-}, \downarrow)$. The dual modal operators $\Box$ and $\Box^{-}$, and the propositional connectives not taken as primitive are defined as usual. We now define models.

Definition 1. *A* model *is a tuple* (W, R, V) *where*

1. *W is a non-empty set, whose elements are usually called* worlds.

2. *R is a binary relation on W called the* accessibility relation *of the model. If $(w, v) \in R$ we say that v is* accessible *from w.*
3. *For each propositional symbol p, $V(p)$ is a subset of W. For each nominal a, $V(a)$ is an element of W. $V(a)$ is called the world* denoted *by a.*

Given a model $\mathcal{M} = (W, R, V)$, a world $w \in W$, and a nominal a, we will use $\mathcal{M}^a_w$ to refer to the model which is identical to $\mathcal{M}$ except V maps a to w.

The relation $\mathcal{M}, w \models \phi$ is defined inductively, where $\mathcal{M} = (W, R, V)$ is a model, w is an element of W, and ϕ is a formula of $HL(@, ^-, \downarrow)$.

$$
\begin{aligned}
\mathcal{M}, w &\models \top \text{ iff true} \\
\mathcal{M}, w &\models p \text{ iff } w \in V(p) \\
\mathcal{M}, w &\models a \text{ iff } V(a) = w \\
\mathcal{M}, w &\models \neg\phi \text{ iff not } \mathcal{M}, w \models \phi \\
\mathcal{M}, w &\models \phi \wedge \psi \text{ iff } \mathcal{M}, w \models \phi \text{ and } \mathcal{M}, w \models \psi \\
\mathcal{M}, w &\models \Diamond\phi \text{ iff for some } v \in W,\ (w, v) \in R \text{ and } \mathcal{M}, v \models \phi \\
\mathcal{M}, w &\models \Diamond^-\phi \text{ iff for some } v \in W, (v, w) \in R \text{ and } \mathcal{M}, v \models \phi \\
\mathcal{M}, w &\models @_a\phi \text{ iff } \mathcal{M}, V(a) \models \phi \\
\mathcal{M}, w &\models \downarrow a.\phi \text{ iff } \mathcal{M}^a_w, w \models \phi.
\end{aligned}
$$

By convention $\mathcal{M} \models \phi$ means $\mathcal{M}, w \models \phi$ for every world $w \in W$. A formula ϕ is *valid* if an only if $\mathcal{M} \models \phi$ for any model $\mathcal{M}$. In this case we simply write $\models \phi$. A formula ϕ is *satisfiable* if and only if $\neg\phi$ is not valid, that is, if and only if there exists a model $\mathcal{M}$ and a world w such that $\mathcal{M}, w \models \phi$. A formula ϕ is *satisfiable in a model $\mathcal{M} = (W, R, V)$* if and only there is a world $w \in W$ such that $\mathcal{M}, w \models \phi$. When ϕ is satisfiable in a model $\mathcal{M}$ we also say that $\mathcal{M}$ *satisfies* ϕ.

If ϕ and ψ are formulae and χ is a subformula of ϕ, we use $\phi[\psi/\chi]$ to denote the formula obtained from ϕ by replacing all occurrences of χ by ψ. Later we will need the following well-known and basic result which we state without proof.

Lemma 1. *Let ϕ and ψ be formulae of hybrid logic, and let p be propositional symbol occurring in ϕ. If $\models \phi$ then $\models \phi[\psi/p]$.*

In later sections nominals are used to represent the names of the ambient calculus. To simplify matters we take Nam to be a subset of Nom thus allowing ambient names to be represented directly as nominals in a hybrid logic formula. This further has the great advantage that α-renaming in the ambient calculus can be directly modelled by α-renaming in hybrid logic, and thereby dispense with the need for more technically involved solutions.

4 Modelling Ambients in Hybrid Logic

Our inspiration for modelling ambients using hybrid logic comes from the way expressions in the ambient calculus are usually visualised. To make things simple, let us start by considering only the following fragment of the ambient calculus:

$$P, Q \ ::= \ \mathbf{0} \ \mid \ P|Q \ \mid \ n[P]. \tag{1}$$

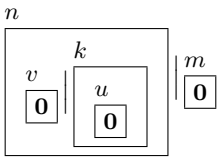

Fig. 4. Box visualisation of the process P_s.

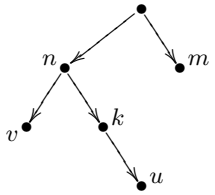

Fig. 5. Tree visualisation of the process P_s.

Consider the process P_s in this fragment given by $P_s = n[\ v[\mathbf{0}] \mid k[u[\mathbf{0}]]\ ] \mid m[\mathbf{0}]$. The standard visualisation of P_s is given in Figure 4. The idea is here to visualise each ambient expression $n[P]$ as a box, using the name n of the ambient as a label on the box, and P as the content of the box. If the process P itself contains ambients, these will be visualised as boxes nested inside the box labelled n. This gives the system of boxes nested within each other a tree-like structure, and it seems obvious to try to visualise this structure as a tree instead of a system of boxes. Ambient names can then be used to label the nodes of the tree, so the process P_s would become represented by the tree given in Figure 5. Such a tree can be interpreted as a hybrid logical model with the ambient names interpreted as nominals. In fact, we can even describe the tree of Figure 5 directly a hybrid logical formula, $\phi_s = \Diamond\Big(n \wedge \Diamond v \wedge \Diamond(k \wedge \Diamond u)\Big) \wedge \Diamond m$. The formula ϕ_s represents the tree in the sense that any hybrid logical model $\mathcal{M}$ in which ϕ_s is satisfiable will contain the tree of Figure 5 as a subtree (unless some of the nominals are mapped into identical worlds, but if needed this possibility can be excluded by replacing ϕ_s by $\phi_s \wedge \bigwedge_{x \neq y} \neg @_x y$). Thus the formula ϕ_s can be considered as a syntactic representation of the tree in Figure 5, which in turn represents the process P_s. Indeed the original process and the corresponding hybrid formula are quite similar:

$$\begin{aligned} P_s &= n\big[\quad v[\mathbf{0}] \mid k[u[\mathbf{0}]] \quad\big]\ \mid\ m[\mathbf{0}]. \\ \phi_s &= \Diamond\big(n \wedge \Diamond v \wedge\ \Diamond(k \wedge \Diamond u)\ \big) \wedge \Diamond m. \end{aligned}$$

Inspired by this similarity we can define a direct translation from processes of the fragment (1) into hybrid logical formulae. We define such a translation $(\cdot)^*$ recursively by:

$$\begin{aligned} \mathbf{0}^* &= \top \\ (P \mid Q)^* &= P^* \wedge Q^* \\ (n[P])^* &= \Diamond(n \wedge P^*). \end{aligned}$$

It is easy to check that P_s^* is logically equivalent to ϕ_s. The point is now that if we take a process P, translate it into the hybrid logical formula P^*, and calculate a model $\mathcal{M}$ of P^* then $\mathcal{M}$ will actually be an analysis of P. This is because the accessibility relation of $\mathcal{M}$ will be encoding the containment relation between the ambients appearing in Q.

Consider now the following extended fragment of the ambient calculus including the three capabilities `in`, `out` and `open`:

$$P, Q\ ::=\ \mathbf{0}\ \mid\ P|Q\ \mid\ n[P]\ \mid\ \texttt{in}\ n.P\ \mid\ \texttt{out}\ n.P\ \mid\ \texttt{open}\ n.P.$$

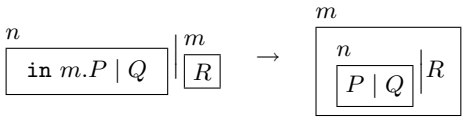

Fig. 6. Box visualisation of the (in) axiom.

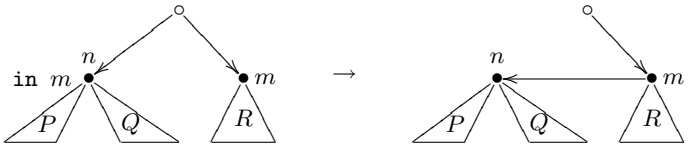

Fig. 7. Tree visualisation of the (in) axiom.

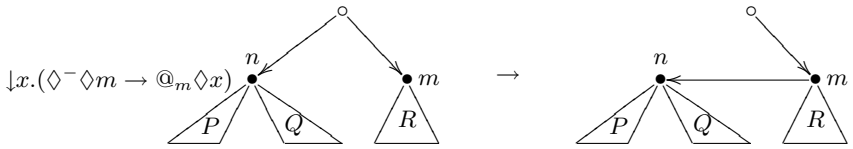

Fig. 8. Hybrid logical visualisation of the (in) axiom.

Capabilities are expressing actions that can be performed on the surrounding ambients. Consider the axiom (in) of Figure 3. A standard visualisation of this axiom is given in Figure 6. What happens in the reduction step of the axiom is that we "execute" the capability in $m.P$ which tells the surrounding ambient n to move inside the sibling ambient named m. Using trees instead of boxes, a simple representation of the (in) axiom could be as in Figure 7. In the figure the root is marked by $\circ$. We will use this as a general convention in the following. From the figure we see that the capability essentially removes one edge and adds another. Since our analyses of the ambient calculus are going to be *over-approximations*, we will concentrate on edges that are added and ignore edges that are removed. Ignoring the removed edge, the capability '$\texttt{in}\ m$' can be seen as expressing "if m is my sibling, then add an edge from m to me" (compare Figure 7). In hybrid logic this translates into: $\downarrow x.(\Diamond^{-}\Diamond m \rightarrow @_m\Diamond x)$. The hybrid logical formula expresses: "At the current node x, if it is possible to go one step backwards and then one step forwards to find the node m then there is an edge from m to x". Or simply: "If m is a sibling of the current node x then there is an edge from m to x". This suggests extending the translation $(\cdot)^*$ defined above to translate in-capabilities in the following way:

$$(\texttt{in}\ n.P)^* = P^* \wedge \downarrow x.(\Diamond^{-}\Diamond n \rightarrow @_n\Diamond x).$$

Graphically we can then replace the capability '$\texttt{in}\ m$' in Figure 7 by the hybrid logical formula $\downarrow x.(\Diamond^{-}\Diamond m \rightarrow @_m\Diamond x)$, as is done in Figure 8. The point is now that if we have a hybrid logical model containing the tree on the left hand side of the reduction arrow in Figure 8, then it will also contain the tree on the right hand side. This is because the translated capability $\downarrow x.(\Diamond^{-}\Diamond m \rightarrow @_m\Diamond x)$ at n

forces there to be an edge from m to n. We can make translations of the `out` and `open` capabilities that behave in a similar way. In the end we obtain a translation $(\cdot)^*$ satisfying the following property:

> Let P and Q be processes. If a hybrid logical model $\mathcal{M}$ satisfies P^* and if P can be reduced to Q, then $\mathcal{M}$ also satisfies Q^*.

This is a *subject reduction* result, which is a central property of any analysis based in Flow Logic. As we shall see in the following section, it follows directly from this subject reduction result that the translation $(\cdot)^*$ gives rise to a correct analysis of the calculus.

Note that the above translation of ambient calculus processes into hybrid logic formulae merges all occurrences of ambients with the same name, i.e., the translation is unable to distinguish between two different syntactic occurences of the same name. This results in a slight imprecision (when analysing processes with non-bound names) in the analyses described in later sections. However, this does not affect the correctness of the analyses and it is easily remedied in the rare cases where the extra precision is needed, e.g., by adding more structure to ambient names or even by annotating ambients with labels, cf. [7, 8].

5 The Naive Analysis

We now develop the ideas introduced above in detail. Our first analysis is rather simple, and we thus call it the 'the naive analysis'. The naive analysis is based on a simple operator $(\cdot)_0$ taking a process P and returning a hybrid logical formula P_0. The operator is defined inductively by the following clauses.

$$\begin{array}{ll}
((\nu n)P)_0 & = P_0 \\
\mathbf{0}_0 & = \top \\
(P \mid Q)_0 & = P_0 \wedge Q_0 \\
(!P)_0 & = P_0 \\
(n[P])_0 & = \Diamond(n \wedge P_0) \\
(\texttt{in}\ n.P)_0 & = P_0 \wedge \downarrow x.(\Diamond^- \Diamond n \rightarrow @_n \Diamond x) \\
(\texttt{out}\ n.P)_0 & = P_0 \wedge \downarrow x.(\Diamond^- n \rightarrow @_n \Box^- \Diamond x) \\
(\texttt{open}\ n.P)_0 & = P_0 \wedge (\Diamond n \rightarrow n).
\end{array}$$

For the translation operator $(\cdot)_0$ we have the following results.

Lemma 2. *For all processes P and Q, if $P \equiv Q$ then $\models P_0 \leftrightarrow Q_0$.*

Proof (Sketch). We need to show that for each of the rules of Figure 2 we get a true statement if we replace each congruence $P \equiv Q$ appearing in the rule by $\models P_0 \leftrightarrow Q_0$. It is easy to check that this is indeed the case using the basic properties of the logical connectives $\leftrightarrow$ and $\wedge$.

Theorem 1 (Subject Reduction for $(\cdot)_0$). *For all processes P and Q, if $P \rightarrow Q$ then $\models P_0 \rightarrow Q_0$.*

Proof. The proof proceeds by induction on the depth of the proof tree for $P \rightarrow Q$. In the base case we consider proof trees of depth 0 corresponding to the axioms

(`in`), (`out`) and (`open`). Consider first the case of the (`in`) axiom. If $P \to Q$ is an instance of the (`in`) axiom then there must be processes P', Q', R' and names n, m such that $P = n[\ \texttt{in}\ m.P' \mid Q'\ ] \mid m[R']$ and $Q = m[\ n[P' \mid Q'] \mid R'\ ]$. This implies

$$P_0 = \Diamond\Big(n \wedge P'_0 \wedge Q'_0 \wedge \downarrow x.(\Diamond^- \Diamond m \to @_m \Diamond x)\Big) \wedge \Diamond\Big(m \wedge R'_0\Big)$$
$$Q_0 = \Diamond\Big(m \wedge R'_0 \wedge \Diamond(n \wedge P'_0 \wedge Q'_0)\Big).$$

We now need to prove $\models P_0 \to Q_0$. By Lemma 1, it suffices to prove validity of the formula $\psi = (P_0 \to Q_0)[p/P'_0, q/Q'_0, r/R'_0]$, where p, q and r are arbitrarily chosen propositional symbols. The theorem prover *HyLoTab* can be used to verify the validity of ψ in the following way. First ψ is negated, and then the negated formula is translated into the syntactic form expected by *HyLoTab* (see [12]):

```
-((<>conj(c0,p0,p1,Dx.(<~><>c1 -> @c1 <>x))
& <>conj(c1,p2)) -> <>conj(c1,p2,<>conj(c0,p0,p1))).
```
(2)

To conform to the *HyLoTab* syntax, n has been replaced by `c0`, m by `c1`, p by `p0`, q by `p1` and r by `p2`. Now *HyLoTab* is run with the formula (2) as input, and it is checked that *HyLoTab* terminates with output "not satisfiable". Since *HyLoTab* thus proves $\neg\psi$ to be non-satisfiable, ψ must be valid, as required. This concludes the case of the (`in`) axiom. The cases of the (`out`) and (`open`) axioms are similar, and have been verified by *HyLoTab* as well. Thus the base cases are completed. We now turn to the induction step. Assume that $P \to Q$ is the root of a proof tree of depth n, and suppose that for any reduction step $P' \to Q'$ with a proof tree of depth less than n we have $\models P'_0 \to Q'_0$. There are now four cases to consider, one for each of the four possible proof rules (amb), (ν), (|) and ($\equiv$) that could have produced the reduction step $P \to Q$. Consider first the case of the (amb) rule. If $P \to Q$ has been produced by (amb) then there must exist processes P', Q' and a name n such that $P = n[P']$, $Q = n[Q']$, and there is a proof tree of depth $n - 1$ for the reduction step $P' \to Q'$. By induction hypothesis we then get $\models P'_0 \to Q'_0$. Furthermore, by the definition of $(\cdot)_0$ we get $P_0 = \Diamond(n \wedge P'_0)$ and $Q_0 = \Diamond(n \wedge Q'_0)$. Since $P'_0 \to Q'_0$ is valid it is easy to see that $P_0 \to Q_0$ must be valid as well, as required. This concludes the case of the (amb) rule. The cases of the three remaining rules, (ν), (|) and ($\equiv$), are equally simple. In the case of the ($\equiv$) rule, Lemma 2 immediately gives the required conclusion.

We will now show how the translation $(\cdot)_0$ gives rise to a correct analysis of the ambient calculus. First we need a few new notions. Let $\mathcal{M} = (W, R, V)$ be a model. The *nominal accessibility relation* of $\mathcal{M}$ is the following relation:

$$\{(m, n) \in \mathsf{Nom}^2 \mid (V(m), V(n)) \in R\}.$$

Thus the nominal accessibility relation contains the set of pairs of nominals (m, n) for which the world denoted by n is accessible from the world denoted by m. It is simply the accessibility relation of $\mathcal{M}$ translated into a relation on nominals. Let P be a process and let m and n be names occurring in P. We say that m *contains* n in P, written $m >_P n$, if there exists a subprocess P' of P

satisfying $P' \equiv m[\ Q \mid n[R]\ ]$. The intuition here is that 'm contains n' means that the ambient named n runs immediately inside the ambient named m. A *correct analysis* of a process P is a set M of pairs of ambient names satisfying:

> For any process Q and any pair of ambient names m and n, if $P \rightarrow^* Q$ and $m >_Q n$ then $(m, n) \in M$.

Thus if M is a correct analysis of a process P and if (m, n) is a pair of names *not* belonging to M, then we know that it is impossible for n to end up running immediately inside m. This is what allows one to prove security properties of e.g. firewalls using correct analyses: If a correct analysis of a process P does not contain a pair (m, n) then we have verified that n can never enter m.

Theorem 2 (Correctness). *Given any process P and any model $\mathcal{M}$ satisfying P_0, the nominal accessibility relation of $\mathcal{M}$ is a correct analysis of P.*

Proof (Sketch). First it is proved that the following holds:

$$\text{For all processes } Q, \text{ all ambient names } m, n, \text{ and all models } \mathcal{M}, \\ \text{if } m >_Q n \text{ and } \mathcal{M} = (W, R, V) \text{ satisfies } Q_0 \text{ then } (V(m), V(n)) \in R. \tag{3}$$

This property is proved by induction on the syntactic structure of Q. We leave out the details, as all cases in the induction proof are easily carried out using the relevant clauses in the definition of $(\cdot)_0$. Now assume P is a process and $\mathcal{M} = (W, R, V)$ is a model satisfying P_0. What we need to prove is the following:

> If Q is a process such that $P \rightarrow^* Q$ and $m >_Q n$ then $(V(m), V(n)) \in R$.

Let thus a process Q be given such that $P \rightarrow^* Q$ and $m >_Q n$. By subject reduction, Theorem 1, we get $\models P_0 \rightarrow Q_0$. Since $\mathcal{M}$ satisfies P_0 it must also satisfy Q_0. We can now apply (3) to conclude $(V(m), V(n)) \in R$, as required. This concludes the proof.

The theorem above shows that to make an analysis of a process P we simply need to find a model of P_0. This can be done e.g. by using *HyLoTab*, since *HyLoTab* is a tableau-based theorem prover, and given a satisfiable formula as input it will return a model of that formula. Let us consider an example.

Example 1 (A simple process). Consider the simple process P given by $P = a[\ \texttt{open}\ b.c[\mathbf{0}] \mid b[\mathbf{0}]\ ]$. We see that b is contained in a in this process. Applying the (`open`) axiom and the (amb) rule to the process we get $P \rightarrow a[\ c[\mathbf{0}] \mid \mathbf{0}\ ]$. So P reduces to a process in which c is contained in a. Thus a correct analysis of P must contain at least the pairs (a, b) and (a, c)—where the first pair corresponds to the fact that a contains b in P, and the second pair corresponds to the fact that P can be reduced to a process in which a contains c. We will now try to apply the machinery above to construct an analysis of P. By the correctness theorem above, Theorem 2, the nominal accessibility relation of any model of P_0 will be a correct analysis. We can use *HyLoTab* to compute a model of P_0 by simply giving the formula as input to *HyLoTab*. *HyLoTab* then returns the model $\mathcal{M} = (W, R, V)$ given by: $W = \{w_1, w_2, w_3\}$; $R = \{(w_1, w_2), (w_2, w_2), (w_2, w_3)\}$; $V(a) = w_2, V(b) = w_2, V(c) = w_3$. This model is illustrated in Figure 9. From the figure we see that the nominal accessibility relation of $\mathcal{M}$ is $\{a, b\} \times \{a, b, c\}$.

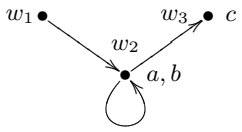

Fig. 9. Model of P_0.

Fig. 10. Tree visualisation of the `open` axiom.

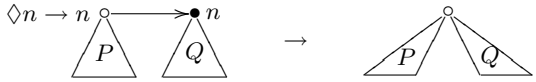

Fig. 11. Hybrid logical visualisation of the `open` axiom using $(\cdot)_0$.

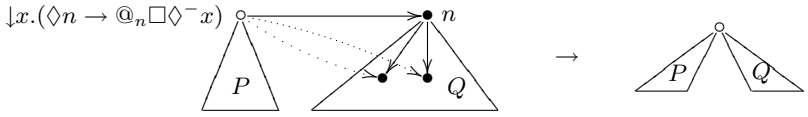

Fig. 12. A better hybrid logical visualisation of the `open` axiom.

This set constitutes, by Theorem 2, a correct analysis of the process P. We see that the analysis contains the pairs (a, b) and (a, c) as expected. However, it also contains the pair (b, a) which actually does not correspond to a possible configuration, since it is easily seen that no sequence of reduction steps can reduce P to a process where b contains a. This is because our treatment of the `open` capability in the translation $(\cdot)_0$ is a bit too simplistic.

Let us take a closer look at how we translate the `open` capability. A tree visualisation of the `open` axiom ($\texttt{open}\ n.P \mid n[Q] \to P \mid Q$) is given in Figure 10. Replacing the capability $\texttt{open}\ n$ in Figure 10 by its translation $\Diamond n \to n$ we get the hybrid logical visualisation given in Figure 11. Consider the left-hand tree in this figure. Since the formula $\Diamond n \to n$ holds at the root of this tree, and since the node labelled n is accessible from the root, the nominal n must also hold at the root of the tree. Since nominals are true at unique worlds, the two nodes must be identical. In other words, the two nodes are collapsed into one, corresponding to the right-hand tree in the figure. This shows that the translation of the `open` capability behaves as expected. However, there is one complication. The collapsed node will obviously be labelled by n, but the root of the right-hand tree is not labelled by n. Since a correct analysis is an *over-approximation*, adding n as label to some node can never affect correctness, but it can affect the precision of the analysis. What we would like is a more precise translation of the open capability that does not add n as label to the root node. That is, we seek a translation that 'copies' the subtree Q at the node labelled n to the root node,

but does not also copy the label n. Such a 'copying' of Q can be performed by the following process: For every node u in Q, if there is an edge from n to u then add an edge from the root to u. We can also express this process as a hybrid logical formula at the root node: $\downarrow x.(\Diamond n \to @_n \Box \Diamond^- x)$. If we translate the `open` capability by this formula rather than simply $\Diamond n \to n$ then we get the hybrid logical visualisation in Figure 12. The dashed edges correspond to the edges that are added by the new translation. In the following section we show that using this more complicated translation of the `open` capability actually yields a more precise analysis. In particular, it will solve the imprecision problem discussed in Example 1.

6 The not-so-Naive Analysis

We now try to construct a slightly more precise analysis. It is based on a translation $(\cdot)_1$ from processes into hybrid logical formulae given by the following clauses.

$$\begin{array}{ll}
((\nu n)P)_1 & = P_1 \\
\mathbf{0}_1 & = \top \\
(P \mid Q)_1 & = P_1 \wedge Q_1 \\
(!P)_1 & = P_1 \\
(n[P])_1 & = \Diamond(n \wedge P_1) \\
(\texttt{in}\ n.P)_1 & = P_1 \wedge \Diamond\Box^- \downarrow x.(\Diamond^- \Diamond n \to @_n \Diamond x) \\
(\texttt{out}\ n.P)_1 & = P_1 \wedge \Diamond\Box^- \downarrow x.(\Diamond^- n \to @_n \Box^- \Diamond x) \\
(\texttt{open}\ n.P)_1 & = P_1 \wedge \Diamond\Box^- \downarrow x.(\Diamond n \to @_n \Box \Diamond^- x).
\end{array}$$

Note that the first five clauses in the definition of $(\cdot)_1$ are identical to the corresponding five clauses for $(\cdot)_0$. Furthermore, the clauses for the `in` and `out` capabilities only differ from the corresponding $(\cdot)_0$ clauses by adding $\Diamond\Box^-$ in front of the downarrow binder. The clauses for the `in`, `out` and `open` capabilities are now very similar. The translations of `in` $n.P$, `out` $n.P$ and `open` $n.P$ are all on the form

$$P_1 \wedge \Diamond\Box^- \downarrow x.(An \to @_n Bx)$$

where A and B are combined modal operators given by:

formula	value of A	value of B
`in` $n.P$	$\Diamond^-\Diamond$	$\Diamond$
`out` $n.P$	$\Diamond^-$	$\Box^-\Diamond$
`open` $n.P$	$\Diamond$	$\Box\Diamond^-$

Note in particular the duality between the translations of `out` $n.P$ and `open` $n.P$.

We can now prove the same results for $(\cdot)_1$ as we did for $(\cdot)_0$.

Lemma 3. *For all processes P and Q, if $P \equiv Q$ then $\models P_1 \leftrightarrow Q_1$.*

Proof. Identical to the proof of Lemma 2 (simply replace 0 by 1 everywhere).

Theorem 3 (Subject Reduction for $(\cdot)_1$). *For all processes P and Q, if $P \to Q$ then $\models P_1 \to Q_1$.*

Proof (Sketch). Since the first five clauses are identical for $(\cdot)_0$ and $(\cdot)_1$, we can reuse most of the proof of the subject reduction result for $(\cdot)_0$, Theorem 1. We only need to check the cases where $P \to Q$ has been produced by one of the rules (`in`), (`out`) or (`open`). In each case we again end up with an implication $P_1 \to Q_1$ that we need to prove valid, and again we use *HyLoTab* to verify the validity.

Theorem 4 (Correctness). *Given any process P and any model $\mathcal{M}$ satisfying P_1, the nominal accessibility relation of $\mathcal{M}$ is a correct analysis of P.*

Proof. Identical to the proof of Theorem 2 (simply replace 0 by 1 everywhere).

Example 2 (A simple process). In Example 1 we considered the process P given by $P = a[\ \texttt{open}\ b.c[\mathbf{0}] \mid b[\mathbf{0}]\ ]$. Let us see how our new analysis based on the translation $(\cdot)_1$ behaves with respect to P. Giving P_1 as input formula to *HyLoTab* it produces the model $\mathcal{M} = (W, R, V)$ given by $W = \{w_1, w_2, w_3, w_4, w_5\}$; $R = \{(w_1, w_2), (w_2, w_3), (w_2, w_4), (w_2, w_5)\}$; $V(a) = w_2, V(b) = w_3, V(c) = w_4$. This model is illustrated in Figure 13. The dashed edges are the edges that either

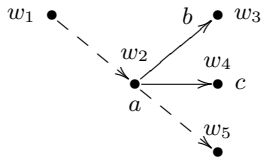

Fig. 13. Model of P_1.

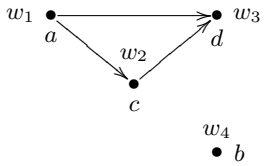

Fig. 14. Model of Q_1.

start or end in a node not labelled by any nominal. Such edges do not affect the nominal accessibility relation, and from now on we simply omit them. From the figure we see that the nominal accessibility relation of $\mathcal{M}$ is $\{(a, b), (a, c)\}$. By Theorem 4, this set is a correct analysis of P. Since the set is a proper subset of the analysis we obtained in Example 1, we have now indeed obtained a more precise analysis. Actually, the present analysis of P contains only the two pairs that any correct analysis of P is required to contain, by Example 1.

Example 3 (Nested capabilities). Let us consider a slightly more complex example. Let Q be the process given by $Q = a[\ \texttt{open}\ b.\texttt{open}\ c.\mathbf{0} \mid c[\, d[\mathbf{0}]\,]\ ]$. Using Q_1 as input to *HyLoTab* we get the model $\mathcal{M}$ illustrated in Figure 14. Its nominal accessibility relation is $\{(a, c), (a, d), (c, d)\}$. This is the same analysis as provided by the 0CFA analysis of [7]. However, the analysis is not completely precise. There are no non-trivial reductions that can be made on Q since there is no ambient named b that the outermost `open` capability can be applied to. Thus we would expect a precise analysis to only give the pairs (a, c) and (c, d) corresponding to the fact that a contains c and c contains d in Q. The reason that the present analysis and the 0CFA of [7] do not give this result is that they both ignore the nesting order on the capabilities. In the case of Q this means

that the relative ordering of the two capabilities $\texttt{open}\, b$ and $\texttt{open}\, c$ is ignored. Using the hybrid logical machinery we have developed it is fortunately quite easy to improve the analysis and obtain one which takes the ordering of capabilities into account.

Consider the following sequence of capabilities: $\texttt{in}\ k.\texttt{out}\ l.\texttt{open}\ m.\mathbf{0}$. Using the present translation we get:

$$(\texttt{in}\ k.\texttt{out}\ l.\texttt{open}\ m.\mathbf{0})_1 = (\texttt{in}\ k.\mathbf{0})_1 \wedge (\texttt{out}\ l.\mathbf{0})_1 \wedge (\texttt{open}\ m.\mathbf{0})_1.$$

This implies that each of the three individual capabilities $\texttt{in}\ k$, $\texttt{out}\ l$ and $\texttt{open}\ m$ will be 'executed' in the same world of the Kripke model induced by the translated formula. This of course means that the translation is flow-*in*sensitive: The ordering of the capabilities is not preserved under the translation. However, since we are working with graphs, there is a simple way to solve this problem. Instead of translating a capability like $\texttt{in}\ k.P$ into a formula of the form $P_1 \wedge \psi$ we could translate it into a formula of the form $\Diamond(P_1 \wedge \psi)$. Then each capability will add a new edge to the graph because of the $\Diamond$ in front of the formula, and the body of the capability will be 'executed' at the world that this new edge leads to. Thus sequences of capabilities will be translated into paths, and the ordering of them will thus be preserved. In the following section we show how to construct such a flow-sensitive translation.

7 A Better Analysis

We now try to construct an even more precise analysis. It is based on a translation $(\cdot)_2$ from processes into hybrid logical formulae given by the following clauses.

$$\begin{array}{ll}
((\nu n)P)_2 & = P_2 \\
\mathbf{0}_2 & = \top \\
(P \mid Q)_2 & = P_2 \wedge Q_2 \\
(!P)_2 & = P_2 \\
(n[P])_2 & = \Diamond(n \wedge P_2) \\
(\texttt{in}\ n.P)_2 & = \Diamond(P_2 \wedge \downarrow y.\Box^- \downarrow x.(\Diamond^-\Diamond n \rightarrow @_n\Diamond x \wedge @_y\Box\Diamond^- x)) \\
(\texttt{out}\ n.P)_2 & = \Diamond(P_2 \wedge \downarrow y.\Box^- \downarrow x.(\Diamond^- n \rightarrow @_n\Box^-\Diamond x \wedge @_y\Box\Diamond^- x)) \\
(\texttt{open}\ n.P)_2 & = \Diamond(P_2 \wedge \downarrow y.\Box^- \downarrow x.(\Diamond n \rightarrow @_n\Box\Diamond^- x \wedge @_y\Box\Diamond^- x)).
\end{array}$$

Note that the first five clauses in the definition of $(\cdot)_2$ are identical to the corresponding five clauses for $(\cdot)_0$ and $(\cdot)_1$. Furthermore, the clauses for the $\texttt{in}$, $\texttt{out}$ and $\texttt{open}$ capabilities are quite similar to the corresponding clauses for $(\cdot)_1$. The translation of $\texttt{in}\ n.P$, $\texttt{out}\ n.P$ and $\texttt{open}\ n.P$ are now all on the form

$$\Diamond(P_2 \wedge \downarrow y.\Box^- \downarrow x.(An \rightarrow @_n By \wedge @_x\Box\Diamond^- y))$$

where A and B are the same combined modal operators as for $(\cdot)_1$.

Lemma 4. *For all processes P and Q, if $P \equiv Q$ then $\models P_2 \leftrightarrow Q_2$.*

Proof. Identical to the proof of Lemma 2 (simply replace 0 by 2 everywhere).

Theorem 5 (Subject Reduction for $(\cdot)_2$). *For all processes P and Q, if $P \to Q$ then $\models P_2 \to Q_2$.*

Proof. Since the first five clauses are identical for $(\cdot)_0$, $(\cdot)_1$ and $(\cdot)_2$, we again only need to consider the rules (`in`), (`out`) and (`open`). These cases are treated similarly to the corresponding cases in the proof of subject reduction for $(\cdot)_1$ and $(\cdot)_0$.

Theorem 6 (Correctness). *Given any process P and any model $\mathcal{M}$ satisfying P_2, the nominal accessibility relation of $\mathcal{M}$ is a correct analysis of P.*

Proof. Identical to the proof of Lemma 2 (simply replace 0 by 2 everywhere).

Example 4 (Nested capabilities). In Example 3 we considered the process Q given by $Q = a[\ \mathtt{open}\ b.\mathtt{open}\ c.\mathbf{0} \mid c[\ d[\mathbf{0}]\ ]\ ]$. As mentioned in Example 3, the translation $(\cdot)_1$ gave us an analysis of Q which was not completely precise. Let us see whether the translation $(\cdot)_2$ performs better. Giving Q_2 as input to *HyLoTab* we obtain the model $\mathcal{M}$ presented in Figure 15. This model is exactly as the model of Q_1

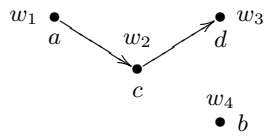

Fig. 15. Model of Q_2.

except that the edge from w_1 to w_3 is no longer present. Thus we instead get the following nominal accessibility relation: $\{(a, c), (c, d)\}$. This relation is the analysis we expected and it is seen to be more precise than both our analysis based on $(\cdot)_1$ and the 0CFA analysis of [7]. This improved precision is gained by making the analysis of capability sequences flow sensitive (cf. the concluding discussion at the end of Section 6).

8 Complexity Issues and Future Work

So far in this paper we have not dealt with questions concerning computability and complexity. Since the full hybrid logic $HL(@, ^-, \downarrow)$ is actually undecidable, it is not immediately obvious that the formulae generated by our translations are always computable. Fortunately, the translations are designed in such a way that they translate into a decidable fragment of hybrid logics called the *innocent fragment* [11]. A formula ϕ is called *innocent* if every binder subformula $\downarrow x.\psi$ is only outscoped by $\Diamond$'s being in the scope of an even number of negation operators (when counting negation operators it is assumed that ϕ uses only the primitive operators, that is, not $\to$, $\Box$ or $\Box^-$). It is easily seen that our translations all fall within the innocent fragment. In [11] van Eijck proves that *HyLoTab* decides the innocent fragment. Thus given *any* process P, *HyLoTab* will necessarily

terminate on any translation of P resulting in a model for the translation and thereby an analysis of P.

Furthermore, it appears that the fragment of hybrid logic used in our translations is sufficiently simple that it is possible to obtain a complexity bound on the model generation that is comparable to existing analysis tools for the ambient calculus. However, we leave this investigation for future work. Another line of future work we are interested in is to try to make even better analyses than the one provided by $(\cdot)_2$. We believe that the possibilities of using hybrid logic for precise and yet compact as well as clear analysis of the ambient calculus are yet far from exhausted.

References

1. 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.
2. 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.
3. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2001.
4. Luca Cardelli and Andrew D. gordon. Types for Mobile Ambients. In *Conference Record of the Annual ACM Symposium on Principles of Programming Languages, POPL'99*, pages 79–92. ACM Press, 1999.
5. Luca Cardelli and Andrew D. Gordon. Anytime, Anywhere: Modal logics for mobile ambients. In *Conference Record of the Annual ACM Symposium on Principles of Programming Languages, POPL'00*, pages 365–377. ACM Press, 2000.
6. Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240:177–213, 2000.
7. Flemming Nielson, René Rydhof Hansen, and Hanne Riis Nielson. Abstract Interpretation of Mobile Ambients. *Science of Computer Programming*, 47(2–3):145–175, 2003.
8. Flemming Nielson, Hanne Riis Nielson, and René Rydhof Hansen. Validating Firewalls using Flow Logics. *Theoretical Computer Science*, 283(2):381–418, 2002.
9. Hanne Riis Nielson and Flemming Nielson. Shape analysis for mobile ambients. In *Conference Record of the Annual ACM Symposium on Principles of Programming Languages, POPL'00*, pages 142–154. ACM Press, 2000.
10. Hanne Riis Nielson and Flemming Nielson. Flow Logic: a multi-paradigmatic approach to static analysis. In *The Essence of Computation: Complexity, Analysis, Transformation*, volume 2566 of *Lecture Notes in Computer Science*, pages 223–244. Springer Verlag, 2002.
11. Jan van Eijck. Constraint tableaux for hybrid logics. Manuscript, CWI, Amsterdam, 2002.
12. Jan van Eijck. Hylotab—tableau-based theorem proving for hybrid logics. Manuscript, CWI, Amsterdam, 2002.