Information and Computation xxx (2010) xxx–xxx

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

Hybrid logical analyses of the ambient calculus

Thomas Bolander [a,*], René Rydhof Hansen [b]

[a] *Informatics and Mathematical Modelling, Technical University of Denmark, Denmark*
[b] *Department of Computer Science, Aalborg University, Denmark*

ARTICLE INFO

Article history:
Received 18 March 2008
Revised 17 January 2009
Available online xxxx

Keywords:
Hybrid logic
Static analysis
Mobile Ambients

ABSTRACT

In this paper, hybrid logic is used to formulate three control flow analyses for Mobile Ambients, a process calculus designed for modelling mobility. 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.

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* have 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 primary goal of this paper is to demonstrate that *hybrid logics* can be applied successfully to the area of static analysis and function as a cornerstone throughout all phases of specification, validation, and implementation of a static analysis. We do this by using hybrid logics to develop three increasingly precise analyses of the *Mobile Ambients* process calculus [1]. The ambient calculus was originally designed with modelling of high mobility distributed systems in mind.

The three analyses we develop are specified in the style of the Flow Logic approach to static analysis [2] and are partly inspired by the Flow Logic based analysis of the ambient calculus described in [3] (see Section 3 for more details). One of the major advantages of rooting our analyses firmly in hybrid logics, is that it enables us to use the *HyLoTab* theorem prover for hybrid logics [4] for proving the correctness of the analyses. Furthermore, by exploiting the model generation capabilities of *HyLoTab* we also automatically obtain prototype 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 manually proving the correctness of the implementation.

We further argue that hybrid logics are particularly well-suited for specifying analyses for the ambient calculus because the binary relational structure of models for hybrid logic formulae is similar to the structures underlying the semantics of Mobile Ambients. This leads to analysis specifications that are compact, straightforward, and to a large extent derived directly from the semantics. A further technical advantage is that nominals, intrinsic to hybrid logics, provides a convenient

[*] Corresponding author. Fax: +45 45 88 26 73.
Email addresses: tb@imm.dtu.dk (T. Bolander), rrh@cs.aau.dk (R.R. Hansen).

0890-5401/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2009.01.006

$$\begin{array}{lcll}
P, Q & ::= & (\nu n)P & \text{restriction} \\
 & | & \mathbf{0} & \text{inactivity} \\
 & | & P|Q & \text{composition} \\
 & | & !P & \text{replication} \\
 & | & n[P] & \text{ambient} \\
 & | & \texttt{in}\ n.P & \text{capability to enter } n \\
 & | & \texttt{out}\ n.P & \text{capability to exit } n \\
 & | & \texttt{open}\ n.P & \text{capability to open } n
\end{array}$$

Fig. 1. Ambient syntax.

way to represent names, occurring in the ambient calculus, in the analysis. Thereby avoiding additional technical complexity otherwise necessary to handle names.

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 useful only for demonstration purposes. Indeed the analysis described in Section 7 is comparable to the 0CFA analyses of [3,5] and the analysis defined in Section 8 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 not as precise as the "counting analysis" of [3] nor as the shape analysis of [6], it does have the advantage of retaining a simple specification that is straightforward to prove correct and easily implemented.

A preliminary version of this paper appeared as [7].

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 CCS, 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, formalised in terms of *containment* as explained in Section 3, 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 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 [1] 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 ambient calculus is defined in Fig. 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 $P|Q$; 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 $\texttt{in}\ n$ and $\texttt{out}\ n$ are used to move their enclosing ambients whereas $\texttt{open}\ n$ is used to dissolve the boundary of a sibling ambient; this will be made precise when we define the semantics below. We write $\text{fn}(P)$ for the free names of P. Trailing occurrences of the inactive process $\mathbf{0}$ will often be omitted.

Semantics

The semantics of the ambient calculus is defined as a straightforward *reduction semantics* (see Fig. 3) using a *congruence relation* (see Fig. 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 Fig. 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.

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

Fig. 2. Structural congruence.

$$m[\ \mathtt{in}\, n.P|Q\ ]|n[R] \to n[\ m[P|Q]|R\ ] \qquad (\mathtt{in})$$

$$n[\ m[\mathtt{out}\, n.P|Q]|R\ ] \to m[P|Q]|n[R] \qquad (\mathtt{out})$$

$$\mathtt{open}\, n.P|n[Q] \to P|Q \qquad (\mathtt{open})$$

$$\frac{P \to Q}{n[P] \to n[Q]}\ (\mathrm{amb}) \qquad \frac{P \to Q}{(\nu n)P \to (\nu n)Q}\ (\nu)$$

$$\frac{P \to Q}{P|R \to Q|R}\ (|) \qquad \frac{P' \equiv P \quad P \to Q \quad Q \equiv Q'}{P' \to Q'}\ (\equiv)$$

Fig. 3. Reduction relation.

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 *identified*. To allow our analyses to deal with this, we employ the method of [3]. We introduce the notion of *stable names*: assume the set of names is equipped with an equivalence relation $\equiv_\alpha$ such that each equivalence class $\{m \mid m \equiv_\alpha n\}$ is countably infinite and contains a unique representative n^*. We write SNam for the subset $\{n^* \mid n \in \mathsf{Nam}\}$ of representatives, called the *stable names*. We then restrict α-renaming by only allowing bound names to be renamed with names from the *same* equivalence class. This simplifies the technicalities of our analyses without affecting the generality of the semantics. In [8] a variant of the ambient calculus using *only* stable names is defined.

Processes can be composed in parallel and executed in an interleaved way, as shown by the $(|)$-rule. Finally, the $(\equiv)$-rule integrates the congruence relation defined in Fig. 2 into the reduction rules. We will not go into further detail with the semantics here but refer the reader to later sections and [1].

3. Analysing Mobile Ambients

As mentioned earlier, the most basic and essential property of an ambient program is how the *ambient hierarchy* dynamically develops and unfolds during execution since this is how computation is expressed in the ambient calculus. Here the ambient hierarchy is the tree structure determined by which ambients are *contained* in which other ambients. It is, in general, an undecidable problem to determine the exact tree structure, or rather sequence of tree structures, a given program will give rise to without actually running it. This is a straightforward consequence of the fact that the ambient calculus is Turing-complete, see [1].

By applying well-known standard techniques from the static analysis community, it is possible to side-step the above decidability issues by computing a *conservative over-approximation* of the set of all possible runtime ambient hierarchies. This approach to the ambient calculus was pioneered and developed in [9–11,3,5] where the developed static analyses are used to verify a number of security related properties. These analyses are comparable to computing control flow graphs for imperative languages or performing control flow analysis of a functional language and are therefore called *control flow analyses*.

In this paper, we shall only be concerned with control flow analyses, since they form the foundation upon which other, more specialised and advanced, analyses are built. For a recent example of a Flow Logic based control flow analysis of the ambient calculus, see [8]. In [3,6] more advanced static analysis techniques, such as *shape analysis*, are used to obtain very precise analyses. However, the increased precision comes at the cost of increased time complexity. Note that even the more advanced analyses compute or contain a control flow analysis component with varying degrees of precision. We conjecture

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

Fig. 4. Satisfaction relation for the hybrid logic HL(@, $^-$, ↓).

that the method used in this paper to develop analyses can be systematically extended to cover also the more advanced analyses.

Notably, the above analyses are all developed using the *Flow Logic* framework for static analysis. The Flow Logic framework [2] 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 results in a framework that is light-weight and well-suited for experimentation and rapid development of a large variety of analyses. The analyses developed in later sections of this paper are similar in spirit to the analyses of [3,5] and are specified in a manner 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 straightforward albeit 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.

4. Hybrid logic

Hybrid logic is an extension of propositional modal logic. Both modal and hybrid logic are thoroughly introduced in [12]. 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 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 assume that the sets Nom and Prop are disjoint. The hybrid logic we consider has the following syntax:

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

where $p \in$ Prop and $a \in$ Nom. This hybrid logic is usually denoted HL(@, $^-$, ↓). Often this hybrid logic is presented with two sorts of nominals: nominal constants that cannot be bound by the downarrow binder and the quantifiers, and nominal variables that can. However, as in [13] we here choose to avoid the distinction. 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 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 in Fig. 4, where $\mathcal{M} = (W, R, V)$ is a model, w is an element of W, and ϕ is a formula of HL(@, $^-$, ↓).

By convention $\mathcal{M} \models \phi$ means $\mathcal{M}, w \models \phi$ for every world $w \in W$. A formula ϕ is *valid* if and 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 if 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.

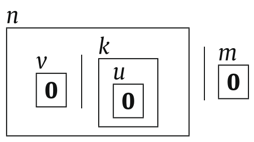


Fig. 5. Box visualisation of the process P_s.

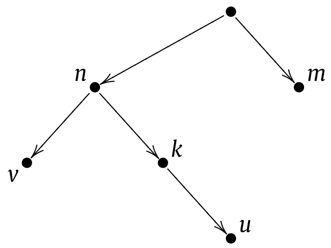


Fig. 6. Tree visualisation of the process P_s.

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

In later sections nominals are used to represent the (stable) names of the ambient calculus. To simplify matters we take the set of ambient calculus names, Nam, to be a subset of Nom, thus allowing names to be represented directly as nominals in a hybrid logical formula.

5. 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:

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

Consider the process P_s in this fragment given by

$$P_s = n[\ v[\mathbf{0}]|k[u[\mathbf{0}]]\ ]|m[\mathbf{0}].$$

The standard visualisation of P_s is given in Fig. 5. 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 we can obviously choose 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 Fig. 6. In [14], Cardelli and Gordon introduce a similar tree representation of ambient expressions, however they label the edges rather than the nodes by ambient names. We have chosen the labelling of nodes to allow a simple translation into hybrid logic. A labelled tree such as the one given Fig. 6 can be interpreted as a hybrid logical model with the ambient names interpreted as nominals. In fact, we can even describe the tree directly by the following 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 Fig. 6 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

$$\phi_s \wedge \bigwedge\{\neg @_a b \mid a, b \text{ are nominals in } \phi_s \text{ with } a \neq b\}.$$

Thus the formula ϕ_s can be considered as a syntactic representation of the tree in Fig. 6, which in turn represents the process P_s. Indeed the original process and the corresponding hybrid formula are quite similar:

$$\begin{array}{lllllllll} P_s & = & n\Big[& v[\mathbf{0}]\ | & k[u[\mathbf{0}]] & \Big] & | & m[\mathbf{0}] \\ \phi_s & = & \Diamond\Big(n \wedge & \Diamond v \wedge & \Diamond(k \wedge \Diamond u) & \Big) & \wedge & \Diamond m. \end{array}$$

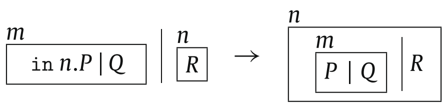


Fig. 7. Box visualisation of the (in) axiom.

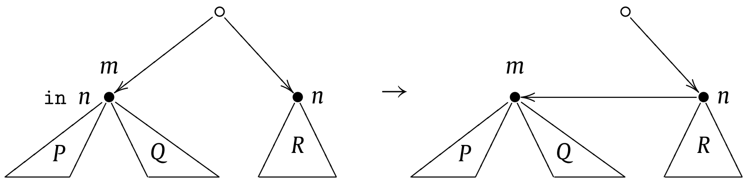


Fig. 8. Tree visualisation of the (in) axiom.

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 $\mathcal{T}$ recursively by:

$$\begin{aligned}\mathcal{T}(\mathbf{0}) &= \top\\ \mathcal{T}(P|Q) &= \mathcal{T}(P) \wedge \mathcal{T}(Q)\\ \mathcal{T}(n[P]) &= \Diamond(n \wedge \mathcal{T}(P)).\end{aligned}$$

It is easy to check that $\mathcal{T}(P_s)$ is logically equivalent to ϕ_s. The point is now that if we take a process P, translate it into the hybrid logical formula $\mathcal{T}(P)$, and calculate a model $\mathcal{M}$ of $\mathcal{T}(P)$ 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:

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

Capabilities are expressing actions that can be performed on the surrounding ambients. Consider the axiom (in) of Fig. 3. A standard visualisation of this axiom is given in Fig. 7. What happens in the reduction step of the axiom is that we "execute" the capability in $n.P$ which tells the surrounding ambient m to move inside the sibling ambient named n. Using trees instead of boxes, a simple representation of the (in) axiom could be as in Fig. 8. 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 'in n' can be seen as a node expressing "if n is my sibling, then add an edge from n to me" (compare Fig. 8). In hybrid logic this translates into: $\downarrow x.(\Diamond^{-}\Diamond n \to @_n\Diamond x)$. 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 n then there is an edge from n to x". Or simply: "If n is a sibling of the current node x then there is an edge from n to x". This suggests extending the translation $\mathcal{T}$ defined above to translate in capabilities in the following way:

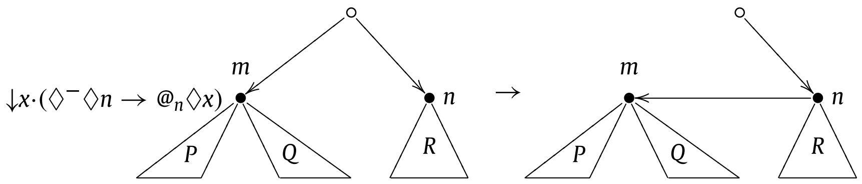
$$\mathcal{T}(\texttt{in}\, n.P) = \mathcal{T}(P) \wedge \downarrow x.(\Diamond^{-}\Diamond n \to @_n\Diamond x).$$

Graphically we can then replace the capability 'in n' in Fig. 8 by the hybrid logical formula $\downarrow x.(\Diamond^{-}\Diamond n \to @_n\Diamond x)$, as is done in Fig. 9. 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 Fig. 9, then it will also contain the tree on the right hand side. This is because the translated capability $\downarrow x.(\Diamond^{-}\Diamond n \to @_n\Diamond x)$ at m forces there to be an edge from n to m. We can make translations of the out and open capabilities that behave in a similar way. Hereby we obtain a translation $\mathcal{T}$ satisfying the following property:

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

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 $\mathcal{T}$ gives rise to a correct analysis of the calculus.

Note that even though we refer to the mapping $\mathcal{T}$ as a 'translation' it is not meaning preserving in any strict sense, and neither is it supposed to be. Rather, it is a mapping that transforms an ambient calculus process into a hybrid logical formula *encoding an approximation* of the process and its possible reductions. One of the approximations takes place in the translation of the composition operator. Given a process P, the processes P and $P|P$ are not in general congruent. However, their respective translations by $\mathcal{T}$ will always be logically equivalent, since the composition operator is translated into logical conjunction.



m
n
$\downarrow x\cdot(\Diamond^{-}\Diamond n \to @_n\Diamond x)$
P
Q
R
$\to$
m
n
P
Q
R

Fig. 9. Hybrid logical visualisation of the (`in`) axiom.

$$\begin{array}{lcl}
\mathcal{T}_0((\nu n)P) & = & \mathcal{T}_0(P) \\
\mathcal{T}_0(\mathbf{0}) & = & \top \\
\mathcal{T}_0(P|Q) & = & \mathcal{T}_0(P) \wedge \mathcal{T}_0(Q) \\
\mathcal{T}_0(!P) & = & \mathcal{T}_0(P) \\
\mathcal{T}_0(n[P]) & = & \Diamond(n^* \wedge \mathcal{T}_0(P)) \\
\mathcal{T}_0(\texttt{in}\ n.P) & = & \mathcal{T}_0(P)\wedge \downarrow x\cdot(\Diamond^{-}\Diamond n^* \to @_{n^*}\Diamond x) \\
\mathcal{T}_0(\texttt{out}\ n.P) & = & \mathcal{T}_0(P)\wedge \downarrow x\cdot(\Diamond^{-} n^* \to @_{n^*}\Box^{-}\Diamond x) \\
\mathcal{T}_0(\texttt{open}\ n.P) & = & \mathcal{T}_0(P) \wedge (\Diamond n^* \to n^*).
\end{array}$$

Fig. 10. Defining clauses for the translation $\mathcal{T}_0$.

Thus the difference between P and $P|P$ is not captured in the translation. For a more precise translation, the composition operator would have to be translated into a contraction-free connective in a suitable substructural logic. However, this would introduce a number of other technical issues and complexities outside the scope of the present paper and therefore we do not consider it further here, but leave it for future work.

Note also that the translation $\mathcal{T}$ merges all occurrences of ambients with the same name, i.e., the translation is unable to distinguish between two different syntactic occurrences 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, it 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. [3,5].

6. The naïve analysis

We now develop the ideas introduced above in detail. Our first analysis is rather simple, and we thus call it the 'the naïve analysis'. The naïve analysis is based on a simple translation $\mathcal{T}_0$ taking a process P and returning a hybrid logical formula $\mathcal{T}_0(P)$. The translation is defined inductively by the clauses given in Fig. 10. Note that in the translation $\mathcal{T}_0$, all ambient calculus names n are replaced by their respective stable names n^*. This is done in order to ensure that the translation is invariant under the restricted form of α-renaming employed in the semantics of the ambient calculus (cf. Section 2 above).

The intuition behind the translations of the `out` and `open` capabilities is probably best illustrated through visualisations of the corresponding axioms. The standard box visualisation of the (`out`) axiom is given in Fig. 11. A corresponding tree visualisation is given in Fig. 12. In the latter figure we see that when the capability `out` n at m is executed the edge from n to m is being replaced by an edge from the root to m. Ignoring the edge being removed, the capability thus expresses: "at the present node x, if there is an edge from n to x then add an edge from the predecessor(s) of n to x". In terms of hybrid logic this can be expressed by the following formula: $\downarrow x\cdot(\Diamond^{-} n \to @_n\Box^{-}\Diamond x)$. This is exactly the formula used in $\mathcal{T}_0$ to translate the `out` n capability. Consider now the `open` capability. The standard box visualisation of the (`open`) axiom is given in Fig. 13 and the corresponding tree visualisation in Fig. 14. From the tree visualisation we see that executing the `open` capability essentially amounts to performing an edge contraction. This is very simple to express in hybrid logic, as we can identify the two end points of an edge by letting the same nominal hold at both points. This leads us to translate the capability `open` n by the hybrid logical formula $\Diamond n \to n$, which expresses: "if there is an edge from here to n, then n is here".

For the translation operator $\mathcal{T}_0$ we have the following results.

Lemma 3. *For all processes P and Q of the ambient calculus, if $P \equiv Q$ then $\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$.*

Proof. We need to show that for each of the rules of Fig. 2 we get a true statement if we replace each congruence $P \equiv Q$ appearing in the rule by $\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$. In Fig. 15 we have shown that this indeed holds (compare Fig. 15 with Fig. 2). The remark in parentheses given after each statement in the figure explains why the statement is true. We also need to check that if P is α-equivalent to Q then $\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$. So suppose P and Q are α-equivalent. By convention, P and Q then become syntactically identical when all occurring names are replaced by their corresponding stable names. Since the translation $\mathcal{T}_0$ replaces all names by their corresponding stable names, we immediately get $\mathcal{T}_0(P) = \mathcal{T}_0(Q)$ and thus $\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$. $\Box$

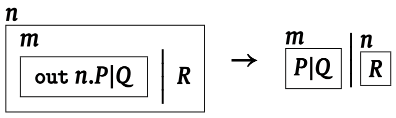


Fig. 11. Box visualisation of the (out) axiom.

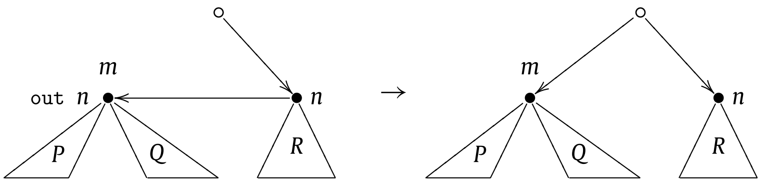


Fig. 12. Tree visualisation of the (out) axiom.

Theorem 4 (Subject reduction for $\mathcal{T}_0$)**.** *For all processes U and V of the ambient calculus, if* $U \rightarrow V$ *then* $\models \mathcal{T}_0(U) \rightarrow \mathcal{T}_0(V)$.

Proof. The proof proceeds by induction on the depth of the proof tree for $U \rightarrow V$. 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 $U \rightarrow V$ is an instance of the (in) axiom then there must be processes P, Q, R and names n, m such that:

$$U = m[\ \texttt{in}\ n.P|Q\ ]|n[R]$$
$$V = n[\ m[P|Q]|R\ ].$$

This implies

$$\mathcal{T}_0(U) = \Diamond\Big(m^* \wedge \mathcal{T}_0(P) \wedge \mathcal{T}_0(Q) \wedge \downarrow x.(\Diamond^- \Diamond n^* \rightarrow @_{n^*}\Diamond x)\Big) \wedge \Diamond\Big(n^* \wedge \mathcal{T}_0(R)\Big)$$
$$\mathcal{T}_0(V) = \Diamond\Big(n^* \wedge \mathcal{T}_0(R) \wedge \Diamond(m^* \wedge \mathcal{T}_0(P) \wedge \mathcal{T}_0(Q))\Big).$$

We now need to prove $\models \mathcal{T}_0(U) \rightarrow \mathcal{T}_0(V)$. By Lemma 2, it suffices to prove validity of the following formula:

$$\psi = \Big(\mathcal{T}_0(U) \rightarrow \mathcal{T}_0(V)\Big)[p/\mathcal{T}_0(P), q/\mathcal{T}_0(Q), r/\mathcal{T}_0(R)]$$

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 [15]):

```
-((<>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, m^* has been replaced by c0, n^* 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 $U \rightarrow V$ is the root of a proof tree of depth k, and suppose that for any reduction step $U' \rightarrow V'$ with a proof tree of depth less than k we have $\models \mathcal{T}_0(U') \rightarrow \mathcal{T}_0(V')$. 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 $U \rightarrow V$. Consider first the case of the (amb) rule. If $U \rightarrow V$ has been produced by (amb) then there must exist processes P, Q and a name n such that $U = n[P]$, $V = n[Q]$, and there is a proof tree of depth $k-1$ for the reduction step $P \rightarrow Q$. By induction hypothesis we then get $\models \mathcal{T}_0(P) \rightarrow \mathcal{T}_0(Q)$. Furthermore, by the definition of $\mathcal{T}_0$ we get $\mathcal{T}_0(U) = \Diamond(n^* \wedge \mathcal{T}_0(P))$ and $\mathcal{T}_0(V) = \Diamond(n^* \wedge \mathcal{T}_0(Q))$. Since $\mathcal{T}_0(P) \rightarrow \mathcal{T}_0(Q)$ is valid it is easy to see that $\mathcal{T}_0(U) \rightarrow \mathcal{T}_0(V)$ 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 3 immediately gives the required conclusion. □

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

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

$$\texttt{open}\, n.P \;\Big|\; \overset{n}{\boxed{Q}} \;\rightarrow\; P|Q$$

Fig. 13. Box visualisation of the (open) axiom.

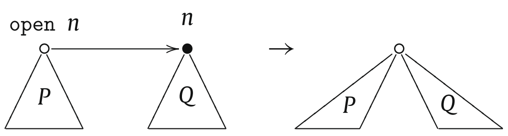


Fig. 14. Tree visualisation of the (open) axiom.

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 $m \downarrow_P n$, if there exists a subprocess P' of P satisfying $P' \equiv m[\ Q|n[R]\ ]$. 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 $M \subseteq \mathsf{SNam}^2$ satisfying:

For any process Q and any pair of names m and n, if $P \rightarrow^* Q$ and $m \downarrow_Q n$ 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 5 (Correctness). *Given any process P and any model $\mathcal{M}$ satisfying $\mathcal{T}_0(P)$, the nominal accessibility relation of $\mathcal{M}$ is a correct analysis of P.*

Proof. First it is proved that the following holds:

For all processes Q, all ambient names m, n, and all models $\mathcal{M}$,
if $m \downarrow_Q n$ and $\mathcal{M} = (W, R, V)$ satisfies $\mathcal{T}_0(Q)$ then
$(V(m^*), V(n^*)) \in R$. (3)

The property is proved by induction on the syntactic structure of Q. The base case $Q = \mathbf{0}$, and the induction steps $Q = !U$ and $Q = (\nu k)U$ are trivial. Now consider the induction step where $Q = \texttt{in}\, k.U$ for some k and U. Suppose $m \downarrow_Q n$ and $\mathcal{M}$ satisfies $\mathcal{T}_0(Q)$. By definition of $\mathcal{T}_0$ we have $\mathcal{T}_0(Q) = \mathcal{T}_0(U) \wedge \psi$ for some formula ψ. Thus since $\mathcal{M}$ satisfies $\mathcal{T}_0(Q)$ it must also satisfy $\mathcal{T}_0(U)$. Since $Q = \texttt{in}\, k.U$ and $m \downarrow_Q n$, we must also have $m \downarrow_U n$. Thus by induction hypothesis we now get $(V(m^*), V(n^*)) \in R$, as required. The cases of the out and open capabilities are completely similar to the case of the in capability. Consider now the case where $Q = U|V$ for some U and V. Suppose $m \downarrow_Q n$ and $\mathcal{M}$ satisfies $\mathcal{T}_0(Q)$. Then $\mathcal{M}$ must satisfy both $\mathcal{T}_0(U)$ and $\mathcal{T}_0(V)$, by definition of $\mathcal{T}_0$. Since $m \downarrow_Q n$, either $m \downarrow_U n$ or $m \downarrow_V n$. In either case the induction hypothesis implies $(V(m^*), V(n^*))$, as needed. Consider finally the case where $Q = k[U]$ for some k and U. Suppose $m \downarrow_Q n$ and $\mathcal{M}$ satisfies $\mathcal{T}_0(Q)$. Since $\mathcal{M}$ satisfies $\mathcal{T}_0(Q)$ there must exist a world w such that $\mathcal{M}, w \models \mathcal{T}_0(Q)$, that is, $\mathcal{M}, w \models \Diamond(k^* \wedge \mathcal{T}_0(U))$. This implies $\mathcal{M}, V(k^*) \models \mathcal{T}_0(U)$. Since $Q = k[U]$ and $m \downarrow_Q n$ we must have either $m \downarrow_U n$ or $k = m$ and $U \equiv V|n[W]$ for some processes V and W. Assume first $m \downarrow_U n$. Since $\mathcal{M}, V(k^*) \models \mathcal{T}_0(U)$ we have that $\mathcal{M}$ satisfies $\mathcal{T}_0(U)$ and thus the induction hypothesis immediately gives the needed conclusion. Assume now that $k = m$ and $U \equiv V|n[W]$. Since $\mathcal{M}, V(k^*) \models \mathcal{T}_0(U)$, Lemma 3 implies $\mathcal{M}, V(k^*) \models \mathcal{T}_0(V) \wedge \Diamond(n^* \wedge \mathcal{T}_0(W))$. In particular, we get $\mathcal{M}, V(k^*) \models \Diamond n^*$, and since $k = m$ this implies $(V(m^*), V(n^*)) \in R$, as needed. This completes the proof of (3). Now assume P is a process and $\mathcal{M} = (W, R, V)$ is a model satisfying $\mathcal{T}_0(P)$. What we need to prove is the following:

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

Let thus a process Q be given such that $P \rightarrow^* Q$ and $m \downarrow_Q n$. By subject reduction, Theorem 4, we get $\models \mathcal{T}_0(P) \rightarrow \mathcal{T}_0(Q)$. Since $\mathcal{M}$ satisfies $\mathcal{T}_0(P)$ it must thus also satisfy $\mathcal{T}_0(Q)$. 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 $\mathcal{T}_0(P)$. 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.

$\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(P)$
(reflexivity of $\leftrightarrow$)
$\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$ implies $\models \mathcal{T}_0(Q) \leftrightarrow \mathcal{T}_0(P)$
(symmetry of $\leftrightarrow$)
$\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$ and $\models \mathcal{T}_0(Q) \leftrightarrow \mathcal{T}_0(R)$
implies $\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(R)$
(transitivity of $\leftrightarrow$)

$\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$ implies
$\models \mathcal{T}_0((\nu n)P) \leftrightarrow \mathcal{T}_0((\nu n)Q)$
(trivial)
$\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$ implies
$\models \mathcal{T}_0(P|R) \leftrightarrow \mathcal{T}_0(Q|R)$
(substitution principle)
$\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$ implies $\models \mathcal{T}_0(!P) \leftrightarrow \mathcal{T}_0(!Q)$
(trivial)
$\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$ implies $\mathcal{T}_0(n[P]) \leftrightarrow \mathcal{T}_0(n[Q])$
(substitution principle)

$\models \mathcal{T}_0(P|\mathbf{0}) \leftrightarrow \mathcal{T}_0(P)$
($\top$ is identity wrt. $\wedge$)
$\models \mathcal{T}_0((\nu n)\mathbf{0}) \leftrightarrow \mathcal{T}_0(\mathbf{0})$
(trivial)
$\models \mathcal{T}_0(!\mathbf{0}) \leftrightarrow \mathcal{T}_0(\mathbf{0})$
(trivial)

$\models \mathcal{T}_0(P|Q) \leftrightarrow \mathcal{T}_0(Q|P)$
(symmetry of $\wedge$)
$\models \mathcal{T}_0((P|Q)|R) \leftrightarrow \mathcal{T}_0(P|(Q|R))$
(associativity of $\wedge$)
$\models \mathcal{T}_0(!P) \leftrightarrow \mathcal{T}_0(P|\,!P)$
(contraction of $\wedge$)

$\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$ implies
$\models \mathcal{T}_0(\mathtt{in}\ n.P) \leftrightarrow \mathcal{T}_0(\mathtt{in}\ n.Q)$
(substitution principle)
$\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$ implies
$\models \mathcal{T}_0(\mathtt{out}\ n.P) \leftrightarrow \mathcal{T}_0(\mathtt{out}\ n.Q)$
(substitution principle)
$\models \mathcal{T}_0(P) \leftrightarrow \mathcal{T}_0(Q)$ implies
$\models \mathcal{T}_0(\mathtt{open}\ n.P) \leftrightarrow \mathcal{T}_0(\mathtt{open}\ n.Q)$
(substitution principle)

$\models \mathcal{T}_0((\nu n)(\nu m)P) \leftrightarrow \mathcal{T}_0((\nu m)(\nu n)P)$
(trivial)
$\models \mathcal{T}_0((\nu n)(P|Q)) \leftrightarrow \mathcal{T}_0(P|(\nu n)Q)$
if $n \notin \mathrm{fn}(P)$
(trivial)
$\models \mathcal{T}_0((\nu \mathrm{n})(\mathrm{m}[P])) \leftrightarrow \mathcal{T}_0(\mathrm{m}[(\nu \mathrm{n})P])$
if $n \neq m$
(trivial)

Fig. 15. Congruences translated into logical equivalences.

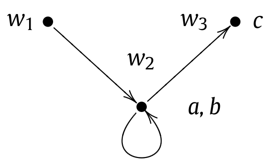


Fig. 16. Model of $\mathcal{T}_0(P)$.

Example 6 (A simple process). Consider the simple process P given by

$$P = a[\ \mathtt{open}\ b.c[\mathbf{0}]|b[\mathbf{0}]\ ].$$

For simplicity we will assume that a, b, and c are stable names. It is seen that b is contained in a in this process. Applying the $(\mathtt{open})$ axiom and the (amb) rule to the process we get

$$P \rightarrow a[\ c[\mathbf{0}]|\mathbf{0}\ ].$$

Thus P reduces to a process in which c is contained in a. Therefore 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 5, the nominal accessibility relation of any model of $\mathcal{T}_0(P)$ will be a correct analysis. We can use *HyLoTab* to compute a model of $\mathcal{T}_0(P)$ 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 Fig. 16. From the figure we see that the nominal accessibility relation of $\mathcal{M}$ is $\{a, b\} \times \{a, b, c\}$. This set constitutes, by Theorem 5, a correct analysis of the process P. We see that the analysis contains the pairs (a, b) and

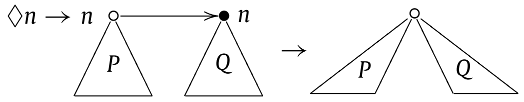


Fig. 17. Hybrid logical visualisation of the (`open`) axiom using $\mathcal{T}_0$.

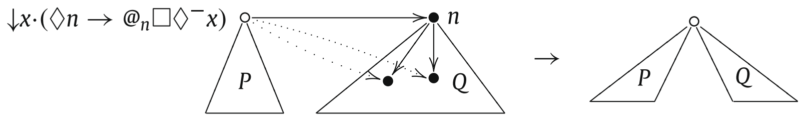

$\downarrow x{\cdot}(\Diamond n \rightarrow @_n\Box\Diamond^- x)$
n
P
Q
$\rightarrow$
P
Q

Fig. 18. A better hybrid logical visualisation of the (`open`) axiom.

(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 $\mathcal{T}_0$ is a bit too simplistic.

Let us take a closer look at how $\mathcal{T}_0$ translates the `open` capability. A tree visualisation of the (`open`) axiom was given in Fig. 14. Replacing the capability `open` n in Fig. 14 by its translation $\Diamond n \rightarrow n$ we get the hybrid logical visualisation given in Fig. 17 (n is assumed to be a stable name). Consider the left-hand tree in this figure. Since the formula $\Diamond n \rightarrow n$ 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: $\downarrow x{\cdot}(\Diamond n \rightarrow @_n\Box\Diamond^- x)$. If we translate the `open` capability by this formula rather than simply $\Diamond n \rightarrow n$ then we get the hybrid logical visualisation in Fig. 18. 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 6.

7. The not-so-naïve analysis

We now construct a slightly more precise analysis. It is based on a translation $\mathcal{T}_1$ from processes into hybrid logical formulae given by the clauses in Fig. 19. Note that the first five clauses in the definition of $\mathcal{T}_1$ are identical to the corresponding five clauses for $\mathcal{T}_0$. Furthermore, the clauses for the `in` and `out` capabilities only differ from the corresponding $\mathcal{T}_0$ clauses by adding $\Diamond\Box^-$ in front of the downarrow binder. The addition of $\Diamond\Box^-$ ensures that the body Q of an ambient $n[Q]$ will be translated into formulas holding at proper descendants of the node named n. This is necessary to be sure that everything in Q will be moved into the right place if the boundary of the ambient $n[Q]$ is dissolved by an `open` n capability (cf. Fig. 18).

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

$$\mathcal{T}_1(P) \wedge \Diamond\Box^- \downarrow x{\cdot}(An^* \rightarrow @_{n^*}Bx)$$

where A and B are compound 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 $\mathcal{T}_1$ as we did for $\mathcal{T}_0$.

Lemma 7. *For all processes P and Q of the ambient calculus, if* $P \equiv Q$ *then* $\models \mathcal{T}_1(P) \leftrightarrow \mathcal{T}_1(Q)$.

$$
\begin{array}{lcl}
\mathcal{T}_1((\nu n)P) & = & \mathcal{T}_1(P) \\
\mathcal{T}_1(\mathbf{0}) & = & \top \\
\mathcal{T}_1(P|Q) & = & \mathcal{T}_1(P) \wedge \mathcal{T}_1(Q) \\
\mathcal{T}_1(!P) & = & \mathcal{T}_1(P) \\
\mathcal{T}_1(n[P]) & = & \Diamond(n^* \wedge \mathcal{T}_1(P)) \\
\mathcal{T}_1(\texttt{in}\ n.P) & = & \mathcal{T}_1(P) \wedge \Diamond\Box^- \downarrow x.(\Diamond^-\Diamond n^* \to @_{n^*}\Diamond x) \\
\mathcal{T}_1(\texttt{out}\ n.P) & = & \mathcal{T}_1(P) \wedge \Diamond\Box^- \downarrow x.(\Diamond^- n^* \to @_{n^*}\Box^-\Diamond x) \\
\mathcal{T}_1(\texttt{open}\ n.P) & = & \mathcal{T}_1(P) \wedge \Diamond\Box^- \downarrow x.(\Diamond n^* \to @_{n^*}\Box\Diamond^- x).
\end{array}
$$

Fig. 19. Defining clauses for the translation $\mathcal{T}_1$.

Proof. The proof is essentially the same as the proof of Lemma 3: simply replace $\mathcal{T}_0$ by $\mathcal{T}_1$ everywhere. □

Theorem 8 (Subject reduction for $\mathcal{T}_1$)**.** *For all processes U and V of the ambient calculus, if* $U \to V$ *then* $\models \mathcal{T}_1(U) \to \mathcal{T}_1(V)$.

Proof. Since the first five clauses are identical for $\mathcal{T}_0$ and $\mathcal{T}_1$, we can reuse most of the proof of the subject reduction result for $\mathcal{T}_0$, Theorem 4. We only need to check the cases where $P \to Q$ has been produced by one of the rules (in), (out) or (open). The cases of (in) and (out) are treated exactly as in the proof of Theorem 4, so we will here only cover the case of (open). In the case of (open) there exist processes P, Q and a name n such that:

$$
\begin{aligned}
U &= \texttt{open}\ n.P|n[Q] \\
V &= P|Q.
\end{aligned}
$$

This gives

$$
\begin{aligned}
\mathcal{T}_1(U) &= \mathcal{T}_1(P) \wedge \Diamond\Box^- \downarrow x.(\Diamond n^* \to @_{n^*}\Box\Diamond^- x) \wedge \Diamond(n^* \wedge \mathcal{T}_1(Q)) \\
\mathcal{T}_1(V) &= \mathcal{T}_1(P) \wedge \mathcal{T}_1(Q).
\end{aligned}
$$

Unfortunately, the formula

$$
\Big(\mathcal{T}_1(U) \to \mathcal{T}_1(V)\Big)[p/\mathcal{T}_1(P), q/\mathcal{T}_1(Q)]
$$

is not valid. So to prove the validity of $\mathcal{T}_1(U) \to \mathcal{T}_1(V)$ we need to take a closer look at the structure of $\mathcal{T}_1(P)$ and $\mathcal{T}_1(Q)$. From the definition of $\mathcal{T}_1$ it can be seen that any process W will be translated into a formula $\mathcal{T}_1(W)$ on the form $\psi_1 \wedge \psi_2 \wedge \cdots \psi_k$, where each ψ_i is either the formula $\top$ or a formula on the form $\Diamond\chi$. Thus for any process W, $\mathcal{T}_1(W)$ will be logically equivalent to a formula of the form $\Diamond\chi_1 \wedge \cdots \Diamond\chi_l$. In particular, $\mathcal{T}_1(P)$ and $\mathcal{T}_1(Q)$ will be logically equivalent to formulas on this form. To prove the validity of $\mathcal{T}_1(U) \to \mathcal{T}_1(V)$ it thus suffices to prove the validity of the following formula:

$$
\Big(\mathcal{T}_1(U) \to \mathcal{T}_1(V)\Big)[\Diamond p/\mathcal{T}_1(P), \Diamond q/\mathcal{T}_1(Q)].
$$

This latter formula has been fed to *HyLoTab* and, as required, its validity was verified. The fact that we need to use $\Diamond p$ and $\Diamond q$ instead of p and q to obtain a valid formula shows that the extra $\Diamond$ in the translation of the in and out capabilities is indeed required, as also noted in the informal discussion following the definition of $\mathcal{T}_1$. □

Theorem 9 (Correctness)**.** *Given any process P and any model* $\mathcal{M}$ *satisfying* $\mathcal{T}_1(P)$, *the nominal accessibility relation of* $\mathcal{M}$ *is a correct analysis of P.*

Proof. The proof is essentially identical to the proof of Lemma 5: simply replace $\mathcal{T}_0$ by $\mathcal{T}_1$ everywhere. □

Example 10 (A simple process)**.** In Example 6 we considered the process P given by $P = a[\ \texttt{open}\ b.c[\mathbf{0}]|b[\mathbf{0}]\ ]$. Let us see how our new analysis based on the translation $\mathcal{T}_1$ behaves with respect to P. Giving $\mathcal{T}_1(P)$ as input formula to *HyLoTab* it produces the model $\mathcal{M} = (W, R, V)$ given by

$$
\begin{aligned}
W &= \{w_1, w_2, w_3, w_4, w_5\} \\
R &= \{(w_1, w_2), (w_2, w_3), (w_2, w_4), (w_2, w_5)\} \\
V(a) &= w_2, V(b) = w_3, V(c) = w_4.
\end{aligned}
$$

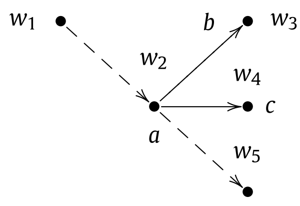


Fig. 20. Model of $\mathcal{T}_1(P)$.

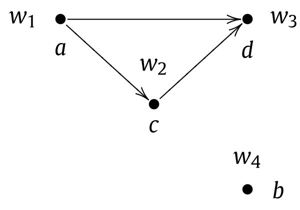


Fig. 21. Model of $\mathcal{T}_1(Q)$.

This model is illustrated in Fig. 20. The dashed edges are the edges that either 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 9, this set is a correct analysis of P. Since the set is a proper subset of the analysis we obtained in Example 6, 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 6.

Example 11 (Nested capabilities). Let us consider a slightly more complex example. Let Q be the process given by $Q = a[\ \texttt{open}\ b.\texttt{open}\ c.\mathbf{0}|c[\ d[\mathbf{0}]\ ]\ ]$. We will assume that a, b, c and d are stable names. Using $\mathcal{T}_1(Q)$ as input to *HyLoTab* we get the model $\mathcal{M}$ illustrated in Fig. 21. Its nominal accessibility relation is $\{(a, c), (a, d), (c, d)\}$. This is the same analysis as provided by the 0CFA analysis of [3]. 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 [3] 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 open b and 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:

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

This implies that each of the three individual capabilities in k, out l and 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 in $k.P$ into a formula of the form $\mathcal{T}_1(P) \wedge \Diamond\psi$ we could translate it into a formula of the form $\Diamond(\mathcal{T}_1(P) \wedge \psi)$. 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.

8. A better analysis

We finally construct an even more precise analysis where the nesting of not only ambients but also of capabilities is taken into account. The resulting analysis is a so-called *flow sensitive* analysis because it takes the order in which capabilities can be exercised into account. The analysis is based on a translation $\mathcal{T}_2$ from processes into hybrid logical formulae given by the clauses in Fig. 22.

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

$$\Diamond(\mathcal{T}_2(P) \wedge \downarrow y.\Box^- \downarrow x{\cdot}(An^* \rightarrow @_{n^*}By \wedge @_x y))$$

$$
\begin{array}{lcl}
\mathcal{T}_2((\nu n)P) & = & \mathcal{T}_2(P) \\
\mathcal{T}_2(\mathbf{0}) & = & \top \\
\mathcal{T}_2(P|Q) & = & \mathcal{T}_2(P) \wedge \mathcal{T}_2(Q) \\
\mathcal{T}_2(!P) & = & \mathcal{T}_2(P) \\
\mathcal{T}_2(n[P]) & = & \Diamond(n^* \wedge \mathcal{T}_2(P)) \\
\mathcal{T}_2(\texttt{in}\, n.P) & = & \Diamond(\mathcal{T}_2(P) \wedge \downarrow y.\Box^- \downarrow x\cdot(\Diamond^-\Diamond n^* \to @_{n^*}\Diamond x \wedge @_x y)) \\
\mathcal{T}_2(\texttt{out}\, n.P) & = & \Diamond(\mathcal{T}_2(P) \wedge \downarrow y.\Box^- \downarrow x\cdot(\Diamond^- n^* \to @_{n^*}\Box^-\Diamond x \wedge @_x y)) \\
\mathcal{T}_2(\texttt{open}\, n.P) & = & \Diamond(\mathcal{T}_2(P) \wedge \downarrow y.\Box^- \downarrow x\cdot(\Diamond n^* \to @_{n^*}\Box\Diamond^- x \wedge @_x y)).
\end{array}
$$

Fig. 22. Defining clauses for the translation $\mathcal{T}_2$.

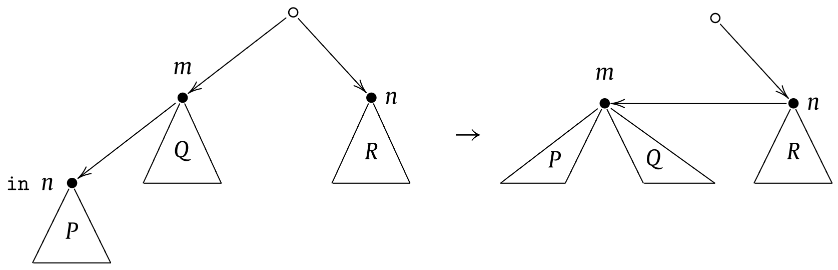


Fig. 23. Modified tree visualisation of the (in) axiom.

where A and B are the same compound modal operators as for $\mathcal{T}_1$. As before, we will try to illustrate the intuition behind the translations of the capabilities by tree visualisations. The previously given tree visualisations need to be modified, since now our trees will not only encode the nesting of ambients, but also the nesting of capabilities. Consider again the standard box visualisation of the (in) axiom given in Fig. 7. In the modified tree visualisation, the capability in n will no longer be put at the same level as the ambient m, but will generate a new level below m. This results in the tree visualisation presented in Fig. 23. The visualised reduction step now involves two changes to the tree structure: (1) the edge from the root to m is replaced by an edge from n to m; (2) the edge from m to the node labelled in n is contracted. In the translation $\mathcal{T}_2$ the edge replacement is treated the same way as in $\mathcal{T}_1$. The edge contraction is dealt with by the addition of the subformula $@_x y$ in the definition of $\mathcal{T}_2(\texttt{in}\, n.P)$. The translations of the out and open capabilities by $\mathcal{T}_2$ are similarly obtained from $\mathcal{T}_1$ by the addition of a subformula $@_x y$ taking care of the edge contraction.

Lemma 12. *For all processes P and Q of the ambient calculus, if* $P \equiv Q$ *then* $\models \mathcal{T}_2(P) \leftrightarrow \mathcal{T}_2(Q)$.

Proof. The proof is essentially the same as the proof of Lemma 3: simply replace $\mathcal{T}_0$ by $\mathcal{T}_2$ everywhere. □

Theorem 13 (Subject reduction for $\mathcal{T}_2$)**.** *For all processes U and V of the ambient calculus, if* $U \to V$ *then* $\models \mathcal{T}_2(U) \to \mathcal{T}_2(V)$.

Proof. Since the first five clauses are identical for $\mathcal{T}_0$, $\mathcal{T}_1$ and $\mathcal{T}_2$, we again only need to consider the rules (in), (out) and (open). In each case we need to prove the validity of a formula $\mathcal{T}_2(U) \to \mathcal{T}_2(V)$. As in the proof of Theorem 8, this is done using *HyLoTab*. The only complication is that as in the proof of Theorem 8 we need to replace the translated subprocesses by formulas on the form $\Diamond p$ rather than simply propositional symbols. We leave out the details, as the proof is essentially the same as the proof of Theorem 8. □

Theorem 14 (Correctness)**.** *Given any process P and any model* $\mathcal{M}$ *satisfying* $\mathcal{T}_2(P)$, *the nominal accessibility relation of* $\mathcal{M}$ *is a correct analysis of P.*

Proof. Essentially identical to the proof of Lemma 3: simply replace $\mathcal{T}_0$ by $\mathcal{T}_2$ everywhere. □

Example 15 (Nested capabilities). In Example 11 we considered the process Q given by $Q = a[\ \texttt{open}\, b.\texttt{open}\, c.\mathbf{0}|c[\ d[\mathbf{0}]\ ]\ ]$. As mentioned in Example 11, the translation $\mathcal{T}_1$ gave us an analysis of Q which was not completely precise. Let us see whether the translation $\mathcal{T}_2$ performs better. Giving $\mathcal{T}_2(Q)$ as input to *HyLoTab* we obtain the model $\mathcal{M}$ presented in Fig. 24. This model is exactly as the model of $\mathcal{T}_1(Q)$ 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 $\mathcal{T}_1$ and the 0CFA analysis of [3]. This improved precision is gained by making the analysis of capability sequences flow sensitive (cf. the concluding discussion at the end of Section 7).

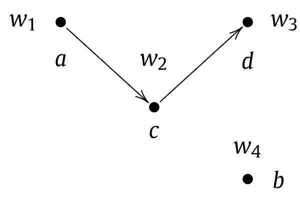


Fig. 24. Model of $\mathcal{T}_2(Q)$.

Flow sensitive analyses

The increased precision of the $\mathcal{T}_2$ analysis developed in this section derives from the encoding of not only the ambient hierarchy but also the *capability hierarchy*. Intuitively this allows the analysis to take the temporal ordering of capabilities into account when analysing an ambient program. Such analyses are called *flow sensitive* analyses and are, in general, more precise (and often much more expensive) than the corresponding flow *in*sensitive analysis. In [16] a flow sensitive control flow analysis of the ambient calculus is developed and extended into a so-called *boundary nesting analysis* that is used to verify that certain confidential information is not leaked. We believe that the precision of the boundary nesting analysis in [16] is comparable to that of our $\mathcal{T}_2$ analysis. Furthermore, the advantages of choosing a logical foundation for our analyses that is very close to the semantics of the ambient calculus become evident: the hybrid logic formulation remains succinct and does not require any major additions to the specification. In comparison, the flow sensitive analysis of [16] requires significant extensions of and additions to the underlying control flow analysis in order to accomodate the increased precision. A thorough and formal comparison of the two analyses is the topic of future work.

9. Complexity issues

So far we have not been concerned with the important issues of computability and complexity. Since the full hybrid logic HL($@, ^-, \downarrow$) is undecidable, it is not immediately obvious that models of translated processes can always be computed. Fortunately, the fragment of HL($@, ^-, \downarrow$) utilised in this paper is decidable, and *HyLoTab* decides it. This means that when *HyLoTab* is given the translation of a process P as input it will eventually terminate and return a model of the translated process, that is, an analysis of P. We will prove this in the following.

A formula of HL($@, ^-, \downarrow$) is said to be in *negation normal form* (*NNF*) if all negation symbols are immediately in front of nominals or propositional symbols. A simple procedure for putting an arbitrary hybrid formula into negation normal form is given in [17]. Let ϕ denote a formula in negation normal form. ϕ is said to be *non-existential* if for all subformulas of the form $\Diamond\psi$ or $\Diamond^-\psi$, ψ is a nominal [18]. ϕ is said to *semi-non-existential* if for all subformulas of the form $\Box\psi$ or $\Box^-\psi$, ψ is non-existential. In other words, ϕ is semi-non-existential if and only if each $\Diamond$ and $\Diamond^-$ in the scope of a $\Box$ or $\Box^-$ is applied directly to a nominal. A formula *not* in negation normal form is said to be (semi-)non-existential if and only if it becomes (semi-)non-existential when put in negation normal form. We now have the following result.

Theorem 16. *HyLoTab decides the satisfiability problem for the semi-non-existential formulae of HL*($@, ^-, \downarrow$).

Proof. Let ϕ be a semi-non-existential formula of HL($@, ^-, \downarrow$). We need to prove that *HyLoTab* terminates on input ϕ. To prove this we simply need to prove that any tableau with root ϕ in the calculus employed by *HyLoTab* is finite (see [4] for details on the tableau calculus). It suffices to prove that any tableau *branch* with root ϕ is finite. To prove this, let Γ denote such a tableau branch, and let n denote the total number of connectives occurring in ϕ. Each node on Γ is a formula of the form $@_m\psi$, where m is a nominal and ψ is a quasi-subformula of ϕ (a *quasi-subformula* is either a subformula or the negation of a subformula). The number of nodes on Γ is thus bounded by the product of the number of nominals occurring on Γ and the number of quasi-subformulas of ϕ. The number of quasi-subformulas of ϕ is obviously in $O(n)$. Thus to prove that Γ is of finite length, we only need to prove that the number of nominals occurring on Γ is finite. This number is equal to the sum of the nominals in the input formula and the number of fresh nominals introduced on the branch. In the tableau calculus employed by *HyLoTab*, only the rules for $\Diamond$ and $\Diamond^-$ can introduce fresh nominals to the branch. The occurrences of $\Diamond$ and $\Diamond^-$ in ϕ that are not in the scope of a $\Box$ or a $\Box^-$ can only produce one fresh nominal each. The occurrences of $\Diamond$ or $\Diamond^-$ in the scope of a $\Box$ or a $\Box^-$ cannot produce fresh nominals, as these are by assumption only applied directly to nominals. Thus the number of fresh nominals introduced to Γ must be in $O(n)$, and therefore the total number of nominals on the branch must also be in $O(n)$. This is the required conclusion. □

It is easy to check that each of the translations $\mathcal{T}_0$, $\mathcal{T}_1$, and $\mathcal{T}_2$ only produce semi-non-existential formulas. Thus the theorem above guarantees that all three translations give rise to computable analyses.

We will not provide a more detailed complexity analysis in this paper, as it would require a much more detailed analysis of the workings of the *HyLoTab* prover. The generality of the *HyLoTab* prover makes it unlikely that *HyLoTab*-based analyses will reach the state-of-the-art complexities for analyses of the ambient calculus. To obtain better complexities one should make specialised algorithms for model generation of formulas of the type generated by T_0, T_1, and T_2. We have not analysed the complexities that would be obtained by constructing such specialised algorithms.

10. Conclusions

The ambient calculus has been studied in many variations and contexts and numerous type systems and static analyses have been developed for it. Much of this work is aimed at particular application areas or focussed on providing powerful analyses [19,14,3,5,20,21,6], most of which are variations over or specialisations of the basic notion of control flow analysis. In this paper we have been more concerned with the theory underlying the analyses and in particular to examine the use of hybrid logics as a platform for static analysis of the ambient calculus. This is especially evident in the way the analyses were derived, almost mechanically, from the semantics of the ambient calculus leading to compact yet clear analyses that are very close to the ambient semantics.

We have argued, persuasively we hope, that hybrid logics do indeed provide a very elegant and useful platform for developing concrete analyses in a systematic and incremental way, with strong tool support throughout all phases of development, but also as a convenient platform for experimenting with and reasoning about many aspects of static analysis.

Related work

While various logics have been used extensively for program analysis, we are not aware of any prior use of hybrid logics in this context. A number of specialised logics specifically for reasoning about the ambient calculus and similarly structured data have been proposed in the literature [14,22]. These logics are very expressive and focussed on exact and powerful reasoning about ambient programs. However, this power comes at a price: the full logics are either undecidable or have very high time complexities. In program analysis the focus is on balancing expressiveness and precision with implementability and efficiency. This leads to the adoption of simpler logics or fragments of more powerful logics for analysis purposes. In future work we intend to continue investigating the merits of different (fragments of) hybrid or modal logics for program analysis.

The analyses in this paper are mainly inspired by the control flow analyses developed in the Flow Logic framework [3,5]. In none of these works are logics used so pervasively and essentially to obtain clear analysis specifications, computer assisted correctness proofs, and automated implementation.

Future work

Based on the positive experiences with formulating program analyses using hybrid logics described in this paper, we will continue exploring the design space and in particular work on making more precise analyses by exploiting the close connection between the ambient semantics and hybrid logics. We believe that the possibilities of using hybrid logic for precise and yet compact as well as clear analyses of the ambient calculus are yet far from exhausted.

References

[1] L. Cardelli, A.D. Gordon, Mobile ambients, Theoretical Computer Science, 240 (2000) 177–213.
[2] H.R. Nielson, F. Nielson, Flow logic: a multi-paradigmatic approach to static analysis, The Essence of Computation: Complexity, Analysis, Transformation, Lecture Notes in Computer Science, vol. 2566, Springer-Verlag, 2002, pp. 223–244.
[3] F. Nielson, R.R. Hansen, H.R. Nielson, Abstract interpretation of mobile ambients, Science of Computer Programming 47 (2–3) (2003) 145–175.
[4] J. van Eijck, Constraint Tableaux for Hybrid Logics, Manuscript, CWI, Amsterdam, 2002.
[5] F. Nielson, H.R. Nielson, R.R. Hansen, Validating firewalls using flow logics, Theoretical Computer Science 283 (2) (2002) 381–418.
[6] H.R. Nielson, F. Nielson, Shape analysis for mobile ambients, in: Conference Record of the Annual ACM Symposium on Principles of Programming Languages, POPL'00, ACM Press, 2000, pp. 142–154.
[7] T. Bolander, R.R. Hansen, Hybrid logical analyses of the ambient calculus, in: D. Leivant, R. de Queiroz (Eds.), Workshop on Logic, Language, Information and Computation (WoLLIC'07), Lecture Notes in Computer Science, vol. 4576, Springer-Verlag, Rio de Janeiro, Brazil, 2007, pp. 83–100, doi: 10.1007/978-3-540-73445-1.
[8] H.R. Nielson, F. Nielson, M. Buchholtz, Security for mobility, in: R. Focardi, R. Gorrieri (Eds.), FOSAD, Lecture Notes in Computer Science, vol. 2946, Springer, 2004, pp. 207–265.
[9] R.R. Hansen, J.G. Jensen, Flow logics for mobile ambients, Master's Thesis, Aarhus University, 1999.
[10] R.R. Hansen, J.G. Jensen, F. Nielson, H.R. Nielson, Abstract interpretation of mobile ambients, in: A. Cortesi, G. Filé (Eds.), Proceedings of the Static Analysis Symposium, SAS'99, Lecture Notes in Computer Science, vol. 1694, Springer-Verlag, Venice, Italy, 1999, pp. 134–148.
[11] F. Nielson, H.R. Nielson, R.R. Hansen, J.G. Jensen, Validating firewalls in mobile ambients, in: International Conference on Concurrency Theory (CONCUR'99), 1999, pp. 463–477. Available from: <citeseer.ist.psu.edu/nielson99validating.html>.
[12] P. Blackburn, M. de Rijke, Y. Venema, Modal logic, Cambridge Tracts in Theoretical Computer Science, vol. 53, Cambridge University Press, Cambridge, UK, 2001.
[13] T. Braüner, Natural deduction for hybrid logic, Journal of Logic and Computation 14 (3) (2004) 329–353. Available from: <http://dx.doi.org/10.1093/logcom/14.3.329>.
[14] L. Cardelli, A.D. Gordon, Anytime, anywhere: modal logics for mobile ambients, Conference Record of the Annual ACM Symposium on Principles of Programming Languages, POPL'00, ACM Press, 2000, pp. 365–377.

[15] J. van Eijck, HyLoTab—Tableau-based Theorem Proving for Hybrid Logics, Manuscript, CWI, Amsterdam, 2002.
[16] C. Braghin, Static analysis of security properties in mobile ambients, Ph.D. Thesis, Università Ca' Foscari di Venezia, TD-2005-1, 2005. Available from: <http://www.unive.it/media/dipInformatica/phd/CBraghin_thesis_hyperlinks3.pdf>.
[17] B. ten Cate, M. Franceschet, On the complexity of hybrid logics with binders, Proceedings of Computer Science Logic 2005, Lecture Notes in Computer Science, vol. 3634, Springer-Verlag, 2005, pp. 339–354.
[18] T. Bolander, P. Blackburn, Terminating tableau calculi for hybrid logics extending K, in: Proceedings of Methods for Modalities 5 (M4M-5), 2007.
[19] L. Cardelli, A.D. Gordon, Types for mobile ambients, Conference Record of the Annual ACM Symposium on Principles of Programming Languages, POPL'99, ACM Press, 1999, pp. 79–92.
[20] T. Amtoft, H. Makholm, J.B. Wells, Polya: true type polymorphism for mobile ambients, in: J.-J. Levy, E.W. Mayr, J.C. Mitchell (Eds.), TCS 2004 (Third IFIP International Conference on Theoretical Computer Science), Toulouse, France, August 2004, Kluwer Academic Publishers, 2004, pp. 591–604.
[21] T. Amtoft, A.J. Kfoury, S.M. Pericas-Geertsen, What are polymorphically-typed ambients?, in: D. Sands (Ed.), ESOP 2001, Genova, LNCS, vol. 2028, Springer-Verlag, 2001, pp. 206–220.
[22] C. Calcagno, P. Gardner, U. Zarfaty, Context logic and tree update, in: Conference Record of the Annual ACM Symposium on Principles of Programming Languages, POPL'05, ACM, 2005, pp. 271–282.