Validating Firewalls in Mobile Ambients

Flemming Nielson, Hanne Riis Nielson,
René Rydhof Hansen, Jacob Grydholt Jensen.

Department of Computer Science, Aarhus University,
Ny Munkegade, DK-8000 Aarhus C, Denmark.

E-mail: {fn,hrn,rrh,grydholt}@daimi.au.dk
Web: http://www.daimi.au.dk/~fn/FlowLogic.html

Abstract. The ambient calculus is a calculus of computation that allows active processes (mobile ambients) to move between sites. A firewall is said to be protective whenever it denies entry to attackers not possessing the required passwords. We devise a polynomial time algorithm for rejecting proposed firewalls that are not guaranteed to be protective. This is based on a control flow analysis for recording what processes may turn up inside what other processes; in particular, we develop a syntax-directed system for specifying the acceptability of an analysis, we prove that all acceptable analyses are semantically sound, and we demonstrate that each process admits a least analysis.

1 Introduction

The ambient calculus is a calculus of computation that allows active processes (called mobile ambients) to move between sites; it thereby extends the notion of mobility found in Java (e.g. [8]) where only passive code may move between sites. The untyped calculus was introduced in [5] and a type system for a polyadic variant was presented in [6]. The calculus is molded on traditional process algebras (such as the π-calculus) but rather than focusing on communication (of values, channels, or processes) it focuses on the movement of processes between different sites; the sites correspond to administrative domains and are modelled using a notion of ambients. We refer to Section 2 for a review of the ambient calculus.

Since processes may evolve when moving around, it becomes harder to analyse what processes may turn up inside what other processes. In Section 3 we show how to adapt techniques from the static analysis of functional programs to develop a control flow analysis [9] for the ambient calculus. This takes the form of a syntax-directed system for specifying when the analysis results are acceptable; we then prove that all acceptable analyses are semantically sound (by means of a subject-reduction result); finally we demonstrate that each process not only admits an analysis but in fact admits a least analysis (by means of a Moore-family result) and we discuss the existence of polynomial-time algorithms.

In [5] the ambient calculus is used to model and study a firewall where only agents knowing the required passwords are supposed to enter; indeed, it is shown that all agents in a special form will in fact enter. However, it is at least as important to ensure that an attacker not knowing the required passwords cannot enter, since this would present a useful technique for screening a system against attackers. In Section 4 we use our analysis to present a polynomial-time procedure for rejecting a class of non-protective firewalls that do not ban illegal access to the internals of the non-protective firewall; this is based on identifying an attacker that is as hard to protect against as any other attacker (somewhat in the manner of hard problems for a given complexity class).

Because of space limitations we only present the analysis for the (Turing complete) core fragment of the calculus where communicaton is now allowed; however, the control flow analysis has been designed so that it scales up to the full language.

2 Mobile Ambients

Syntax. We follow the presentation in [5]; we make a syntactic distinction between capabilities (M) and namings (N) as is also implicit in the type system of [6]. The syntax of processes $P \in \mathbf{Proc}$, capabilities $M \in \mathbf{Cap}$ and namings $N \in \mathbf{Nam}$ is given by:

$$
\begin{array}{llcll}
P ::= (\nu\, n^{\mu})P & \text{restriction} & \quad & M ::= \mathsf{in}^{l^t}\, N & \text{enter } N \\
\quad\mid\ \mathbf{0} & \text{inactivity} & & \quad\mid\ \mathsf{out}^{l^t}\, N & \text{exit } N \\
\quad\mid\ P \mid P' & \text{composition} & & \quad\mid\ \mathsf{open}^{l^t}\, N & \text{open } N \\
\quad\mid\ !P & \text{replication} & & & \\
\quad\mid\ N^{l^a}[P] & \text{ambient} & & N ::= n & \text{name} \\
\quad\mid\ M.P & \text{movement} & & &
\end{array}
$$

To allow the analysis to deal with the α-conversion that is part of the semantics we distinguish between the *name* n introduced by a restriction operator (and that may be α-renamed) and the corresponding *stable name* $\mu \in \mathbf{SNam}$ (that cannot be α-renamed). One way to understand this distinction is to think of a name as an internet address (e.g. `daimi.au.dk`) and to think of a stable name as the corresponding absolute address (e.g. `130.225.16.40`); clearly it is possible for the internet address to change (e.g. from `daimi.aau.dk` to `daimi.au.dk`) without a similar change in the absolute address (e.g. `130.225.16.40`). Another way to understand the distinction between names and stable names is to regard the stable names as static representations of the names arising dynamically.

As is customary for the Flow Logic approach to control flow analysis [3, 4] we have also placed labels $l^a \in \mathbf{Lab}^a$ on ambients and labels $l^t \in \mathbf{Lab}^t$ on transitions – this is merely a convenient way of indicating "program points" and is useful when developing the analysis. The sets of names, stable names and labels are

$P \equiv P$	$P \mid Q \equiv Q \mid P$
$P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R$	$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$
$P \equiv Q \Rightarrow Q \equiv P$	$!P \equiv P \mid !P$
$P \equiv Q \Rightarrow (\nu\, n^{\mu})P \equiv (\nu\, n^{\mu})Q$	$(\nu\, n^{\mu_n})(\nu\, m^{\mu_m})P \equiv (\nu\, m^{\mu_m})(\nu\, n^{\mu_n})P$
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	$\quad \underline{\text{if}}\ n \neq m$
$P \equiv Q \Rightarrow !P \equiv !Q$	$(\nu\, n^{\mu})(P \mid Q) \equiv P \mid (\nu\, n^{\mu})Q$
$P \equiv Q \Rightarrow N^{l}[P] \equiv N^{l}[Q]$	$\quad \underline{\text{if}}\ n \notin \underline{\text{fn}}(P)$
$P \equiv Q \Rightarrow M.\,P \equiv M.\,Q$	$(\nu\, n^{\mu})(m^{l}[P]) \equiv m^{l}[(\nu\, n^{\mu})P]$
$P \mid \mathbf{0} \equiv P$	$\quad \underline{\text{if}}\ n \neq m$
$(\nu\, n^{\mu})\mathbf{0} \equiv \mathbf{0}$	$(\nu\, n^{\mu})P \equiv (\nu\, m^{\mu})(P\{n \leftarrow m\})$
$!\mathbf{0} \equiv \mathbf{0}$	$\quad \underline{\text{if}}\ m \notin \underline{\text{fn}}(P) \qquad (\alpha\text{-renaming})$

Table 1. Structural congruence.

left unspecified but are assumed to be non-empty; it is not essential thay they be mutually disjoint and we occasionally write $l \in \mathbf{Lab} = \mathbf{Lab}^{a} \cup \mathbf{Lab}^{t}$.

We write $\underline{\text{fn}}(P)$ for the set of *free names* of P and similarly for M and N. The *programs* of interest are ambients in the form $n_{\star}^{l_{\star}}[P_{\star}]$ where $n_{\star} \notin \underline{\text{fn}}(P_{\star})$.

Example 1. Consider the following example from [5] for illustrating how an agent crosses a firewall using the prearranged passwords $\mathtt{k}$, $\mathtt{k}'$ and $\mathtt{k}''$:

$$\begin{array}{ll} \mathit{Firewall}: & (\nu\, \mathtt{w}^{w})\mathtt{w}^{\mathrm{A}}[\mathtt{k}^{\mathrm{B}}[\mathsf{out}^{1}\mathtt{w}.\,\mathsf{in}^{2}\mathtt{k}'.\,\mathsf{in}^{3}\mathtt{w}] \mid \mathsf{open}^{4}\mathtt{k}'.\,\mathsf{open}^{5}\mathtt{k}''.P] \\ \mathit{Agent}: & \mathtt{k}'^{\mathrm{C}}[\mathsf{open}^{6}\mathtt{k}.\,\mathtt{k}''^{\mathrm{D}}[Q]] \end{array}$$

The program of interest is $n_{\star}^{l_{\star}}[\mathit{Firewall} \mid \mathit{Agent}]$. We use typewriter font for names, italics for stable names, roman for ambient labels, and numbers for transition labels. □

Semantics. The semantics is given by a structural congruence relation $P \equiv Q$ and a reduction relation $P \rightarrow Q$ in the manner of the π-calculus. The congruence relation of Table 1 is a straightforward modification of a similar table in [5] with the exception that we have added the side condition "$\underline{\text{if}}\ n \neq m$" to the clause for $(\nu\, n^{\mu_n})(\nu\, m^{\mu_m})P$; in our setting it will be incorrect to have no side condition because the association between names and stable names must be maintained at all times. We write $P\{n \leftarrow m\}$ for the process that is as P but with all free occurrences of n replaced by m.

The reduction relation is given in Table 2 and is as in [5]; a pictorial representation of the three basic rules is given in Figure 1. It should be clear that the annotations in the syntax have no semantic consequences. We can make this precise as follows. Let $\mu_{\bullet}$ be a distinguished stable name, $l_{\bullet}^{a}$ a distinguished ambient label and $l_{\bullet}^{t}$ a distinguished transition label. Given a process P write $\lfloor P \rfloor$ for the process where all stable names are replaced by $\mu_{\bullet}$, all ambient labels by $l_{\bullet}^{a}$ and all transition labels by $l_{\bullet}^{t}$:

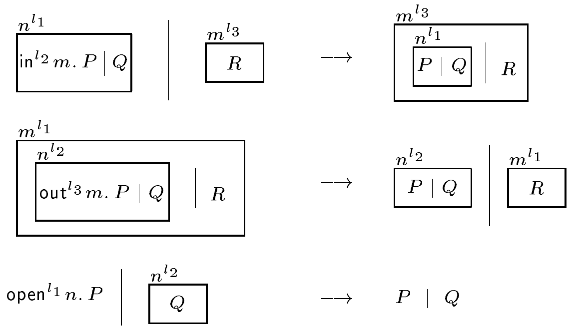

Fig. 1. Pictorial representation of the basic reduction rules.

Fact 1. $P \to^* Q \wedge \lfloor P \rfloor = \lfloor P' \rfloor \quad \Rightarrow \quad \exists Q' : P' \to^* Q' \wedge \lfloor Q \rfloor = \lfloor Q' \rfloor$.

The proof is by induction on the length of the derivation $P \to^* Q$; for the induction step $P \to^* R \to Q$ we proceed by induction on the shape of the inference of $R \to Q$. □

Example 2. We have the following sequence of reduction steps for $n^{l_\star}_\star[\mathit{Firewall} \mid \mathit{Agent}]$; in each step we have underlined the capability to be executed next and we have assumed that $\mathtt{w} \notin \underline{\mathrm{fn}}(Q)$.

$$
\begin{array}{l}
\quad n^{l_\star}_\star[(\nu\, \mathtt{w}^{w})\mathtt{w}^{\mathrm{A}}[\mathtt{k}^{\mathrm{B}}[\underline{\mathsf{out}^1\mathtt{w}}.\, \mathsf{in}^2\mathtt{k}'.\, \mathsf{in}^3\mathtt{w}] \mid \mathsf{open}^4\mathtt{k}'.\, \mathsf{open}^5\mathtt{k}''.\, P] \mid \mathtt{k}'^{\mathrm{C}}[\mathsf{open}^6\mathtt{k}.\, \mathtt{k}''^{\mathrm{D}}[Q]]] \\
\to n^{l_\star}_\star[(\nu\, \mathtt{w}^{w})(\mathtt{k}^{\mathrm{B}}[\underline{\mathsf{in}^2\mathtt{k}'}.\, \mathsf{in}^3\mathtt{w}] \mid \mathtt{w}^{\mathrm{A}}[\mathsf{open}^4\mathtt{k}'.\, \mathsf{open}^5\mathtt{k}''.\, P] \mid \mathtt{k}'^{\mathrm{C}}[\mathsf{open}^6\mathtt{k}.\, \mathtt{k}''^{\mathrm{D}}[Q]])] \\
\to n^{l_\star}_\star[(\nu\, \mathtt{w}^{w})(\mathtt{w}^{\mathrm{A}}[\mathsf{open}^4\mathtt{k}'.\, \mathsf{open}^5\mathtt{k}''.\, P] \mid \mathtt{k}'^{\mathrm{C}}[\mathtt{k}^{\mathrm{B}}[\mathsf{in}^3\mathtt{w}] \mid \underline{\mathsf{open}^6\mathtt{k}}.\, \mathtt{k}''^{\mathrm{D}}[Q]])] \\
\to n^{l_\star}_\star[(\nu\, \mathtt{w}^{w})(\mathtt{w}^{\mathrm{A}}[\mathsf{open}^4\mathtt{k}'.\, \mathsf{open}^5\mathtt{k}''.\, P] \mid \mathtt{k}'^{\mathrm{C}}[\underline{\mathsf{in}^3\mathtt{w}} \mid \mathtt{k}''^{\mathrm{D}}[Q]])] \\
\to n^{l_\star}_\star[(\nu\, \mathtt{w}^{w})\mathtt{w}^{\mathrm{A}}[\underline{\mathsf{open}^4\mathtt{k}'}.\, \mathsf{open}^5\mathtt{k}''.\, P \mid \mathtt{k}'^{\mathrm{C}}[\mathtt{k}''^{\mathrm{D}}[Q]]]] \\
\to n^{l_\star}_\star[(\nu\, \mathtt{w}^{w})\mathtt{w}^{\mathrm{A}}[\underline{\mathsf{open}^5\mathtt{k}''}.\, P \mid \mathtt{k}''^{\mathrm{D}}[Q]]] \\
\to n^{l_\star}_\star[(\nu\, \mathtt{w}^{w})\mathtt{w}^{\mathrm{A}}[P \mid Q]]
\end{array}
$$

The transition sequence shows that the firewall (which has the private name $\mathtt{w}$) sends out the pilot ambient named $\mathtt{k}$; since the agent knows the right passwords, and is in the right form, the pilot ambient can enter the agent and then guide it inside the firewall. □

$$
\begin{array}{ll}
P \to Q \Rightarrow (\nu\, n^\mu)P \to (\nu\, n^\mu)Q & n^{l_1}[\mathsf{in}^{l_2} m.\, P \mid Q] \mid m^{l_3}[R] \to m^{l_3}[n^{l_1}[P \mid Q] \mid R] \\
P \to Q \Rightarrow n^l[P] \to n^l[Q] & m^{l_1}[n^{l_2}[\mathsf{out}^{l_3} m.\, P \mid Q] \mid R] \to n^{l_2}[P \mid Q] \mid m^{l_1}[R] \\
P \to Q \Rightarrow P \mid R \to Q \mid R & \mathsf{open}^{l_1} n.\, P \mid n^{l_2}[Q] \to P \mid Q
\end{array}
$$

$$P \equiv P' \;\wedge\; P' \to Q' \;\wedge\; Q' \equiv Q \Rightarrow P \to Q$$

Table 2. Reduction relation.

3 Control Flow Analysis

Immediate constituents of ambients. The main aim of the analysis is to obtain the following information for each ambient: *(i)* which ambients *may* be immediately contained in it, and *(ii)* which transitions *may* it perform. An ambient will be identified by its label $l^a \in \mathbf{Lab}^a$ and a transition by its associated *stable capability* $\tilde{m} \in \mathbf{SCap}$; stable capabilities are given by

$$\tilde{m} ::= \mathsf{in}^{l^t}\,\mu \mid \mathsf{out}^{l^t}\,\mu \mid \mathsf{open}^{l^t}\,\mu$$

and correspond to capabilities except that names have been replaced by stable names. The analysis records this information in the following component:

$$I \in \mathbf{InAmb} = \mathbf{Lab}^a \to \mathcal{P}(\mathbf{Lab}^a \cup \mathbf{SCap})$$

When specifying the analysis we shall also use the "inverse" mapping I^{-1} : $(\mathbf{Lab}^a \cup \mathbf{SCap}) \to \mathcal{P}(\mathbf{Lab}^a)$ that returns the set of ambients in which the given ambient or transition might occur; formally $z \in I(l^a)$ if and only if $l^a \in I^{-1}(z)$. Later we shall write $I^+(l) \ni l'$ to mean that there exists $l_1, \cdots, l_n$ (for $n \geq 1$) such that $l = l_1$, $l' = l_n$, and $\forall i < n : I(l_i) \ni l_{i+1}$.

Stable names of ambients. Each occurrence of an ambient has a stable name and to keep track of this information the analysis also contains the following component:

$$H \in \mathbf{HNam} = \mathbf{Lab}^a \to \mathcal{P}(\mathbf{SNam})$$

Similarly to before we shall use the "inverse mapping" $H^{-1} : \mathbf{SNam} \to \mathcal{P}(\mathbf{Lab}^a)$ that returns the set of ambients that might have the given stable name; formally $\mu \in H(l^a)$ if and only if $l^a \in H^{-1}(\mu)$.

Naming environment. The association between free names and their stable names is expressed by a naming environment:

$$me \in \mathbf{MEnv} = \mathbf{Nam} \to_{\mathrm{fin}} \mathbf{SNam}$$

We shall write $me_\star$ for the initial naming environment for the program $n_\star^{l_\star}[P_\star]$ of interest and $dom(me_\star)$ for its finite domain.

Example 3. Consider the following analysis information (where the initial naming environment maps the names $\mathtt{k}$, $\mathtt{k}'$ and $\mathtt{k}''$ to k, k' and k'', respectively):

label	I	H
$l_\star$	$\{\mathrm{A}, \mathrm{B}, \mathrm{C}\}$	$\{\}$
A	$\{\mathsf{out}^1 w, \mathsf{in}^2 k', \mathsf{in}^3 w, \mathsf{open}^4 k', \mathsf{open}^5 k'', \mathsf{open}^6 k, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$	$\{w\}$
B	$\{\mathsf{out}^1 w, \mathsf{in}^2 k', \mathsf{in}^3 w\}$	$\{k\}$
C	$\{\mathsf{out}^1 w, \mathsf{in}^2 k', \mathsf{in}^3 w, \mathsf{open}^6 k, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\}$	$\{k'\}$
D	$\{\}$	$\{k''\}$

This shows that the ambient labelled A might perform transitions consuming any of the capabilities labelled 1–6 and that it might contain any of the ambients labelled A–D; in particular it might contain the ambient labelled C indicating that the agent might enter the firewall – and as shown in Example 2 this is indeed the case. ◻

3.1 The acceptability relation

The acceptability of the analysis is defined by the following four predicates (defined in Table 3 and explained below):

$(I, H) \models^{l}_{me} P$	for checking a process $P \in \mathbf{Proc}$;
$(I, H) \rhd_{me} M : \tilde{M}$	for translating a capability $M \in \mathbf{Cap}$ into a set $\tilde{M} \in \mathcal{P}(\mathbf{SCap})$ of stable capabilities;
$(I, H) \Vdash_{me} N : \tilde{N}$	for decoding a naming $N \in \mathbf{Nam}$ into a set $\tilde{N} \in \mathcal{P}(\mathbf{SNam})$ of stable names;
$(I, H) \equiv^{l} \tilde{m}$	for checking a stable capability $\tilde{m} \in \mathbf{SCap}$.

The first part of Table 3 gives a simple syntax-directed definition of what it means for an analysis result (I, H) to be acceptable for the process P. The predicate is defined relative to the current naming environment me and the current label l of the enclosing ambient. The naming environment is updated whenever we pass through a restriction operator and the label is updated whenever we pass inside a new ambient. Note that the analysis cannot distinguish between whether a process occurs only once or many times: $!P$ and P are analysed in the same way (as are $P \mid P$ and P).

The clause for ambients $N^{l^a}[P]$ first checks the subprocess P using the appropriate naming environment and label. It then demands that the label of the ambient is recorded as being inside the current label. Finally, it demands that the stable name of the ambient is recorded as being a name of the ambient: Intuitively, $\tilde{N}$ is the singleton $\{me(n)\}$ when N is n; this is made precise by the third part of the table. As in Prolog, any free identifier on the righthandsides (like $\tilde{N}$) is assumed to be existentially quantified.

The clause for movement $M. P$ first checks the subprocess P using the appropriate naming environment and label. It then translates the capability M into the set of stable capabilities $\tilde{M}$ by replacing names with stable names: Intuitively $\tilde{M}$ is the singleton $\{\mathsf{in}^{l^t}\, me(n)\}$ when M is $\mathsf{in}^{l^t}\, n$ and similarly for the other capabilities; this is made precise by the second part of the table. Finally, each stable capability is analysed in turn as explained below.

The last part of Table 3 shows how to check stable capabilities against the analysis result (I, H). Figure 2 illustrates these clauses pictorially; the similarity

$(I,H) \models^{l}_{me} (\nu\, n^{\mu})P$	iff $(I,H) \models^{l}_{me[n\mapsto\mu]} P$
$(I,H) \models^{l}_{me} \mathbf{0}$	iff true
$(I,H) \models^{l}_{me} P \mid P'$	iff $(I,H) \models^{l}_{me} P \ \wedge\ (I,H) \models^{l}_{me} P'$
$(I,H) \models^{l}_{me} !P$	iff $(I,H) \models^{l}_{me} P$
$(I,H) \models^{l}_{me} N^{l^a}[P]$	iff $(I,H) \models^{l^a}_{me} P \ \wedge\ l^a \in I(l) \ \wedge$ $(I,H) \Vert\!\!\equiv_{me} N : \tilde{N} \ \wedge\ \tilde{N} \subseteq H(l^a)$
$(I,H) \models^{l}_{me} M.\,P$	iff $(I,H) \models^{l}_{me} P \ \wedge$ $(I,H) \rhd_{me} M : \tilde{M} \ \wedge\ \forall \tilde{m} \in \tilde{M} : (I,H) \mid\!\!\equiv^{l} \tilde{m}$
$(I,H) \rhd_{me} \mathsf{in}^{l^t} N : \tilde{M}$	iff $(I,H) \Vert\!\!\equiv_{me} N : \tilde{N} \ \wedge\ \tilde{M} \supseteq \{\mathsf{in}^{l^t} \mu \mid \mu \in \tilde{N}\}$
$(I,H) \rhd_{me} \mathsf{out}^{l^t} N : \tilde{M}$	iff $(I,H) \Vert\!\!\equiv_{me} N : \tilde{N} \ \wedge\ \tilde{M} \supseteq \{\mathsf{out}^{l^t} \mu \mid \mu \in \tilde{N}\}$
$(I,H) \rhd_{me} \mathsf{open}^{l^t} N : \tilde{M}$	iff $(I,H) \Vert\!\!\equiv_{me} N : \tilde{N} \ \wedge\ \tilde{M} \supseteq \{\mathsf{open}^{l^t} \mu \mid \mu \in \tilde{N}\}$
$(I,H) \Vert\!\!\equiv_{me} n : \tilde{N}$	iff $\tilde{N} \supseteq \{me(n)\}$
$(I,H) \mid\!\!\equiv^{l} \mathsf{in}^{l^t} \mu$	iff $\mathsf{in}^{l^t} \mu \in I(l) \ \wedge$ $\forall l^a \in I^{-1}(\mathsf{in}^{l^t} \mu) : \forall l^{a'} \in I^{-1}(l^a) :$ $\forall l^{a''} \in I(l^{a'}) \cap H^{-1}(\mu) : l^a \in I(l^{a''})$
$(I,H) \mid\!\!\equiv^{l} \mathsf{out}^{l^t} \mu$	iff $\mathsf{out}^{l^t} \mu \in I(l) \ \wedge$ $\forall l^a \in I^{-1}(\mathsf{out}^{l^t} \mu) : \forall l^{a'} \in I^{-1}(l^a) \cap H^{-1}(\mu) :$ $\forall l^{a''} \in I^{-1}(l^{a'}) : l^a \in I(l^{a''})$
$(I,H) \mid\!\!\equiv^{l} \mathsf{open}^{l^t} \mu$	iff $\mathsf{open}^{l^t} \mu \in I(l) \ \wedge$ $\forall l^a \in I^{-1}(\mathsf{open}^{l^t} \mu) : \forall l^{a'} \in I(l^a) \cap H^{-1}(\mu) :$ $\forall l' \in I(l^{a'}) : l' \in I(l^a)$

Table 3. Control flow analysis.

between Figures 2 and 1 stresses the systematic way in which a control flow analysis may be developed from a formal semantics.

The clause for $\mathsf{in}^{l^t} \mu$ first ensures that the stable capability is properly recorded as part of the current ambient l. Then it ensures that all contexts l^a in which the capability could occur (and this clearly includes l) are properly recorded as being possible subambients of all sibling ambients $l^{a''}$ having the stable name μ. This involves quantifying over all possible parent ambients $l^{a'}$ and using the component H to obtain the stable name of $l^{a''}$.

The clause for $\mathsf{out}^{l^t} \mu$ follows a similar pattern. First it ensures that the stable capability is recorded as part of the current ambient l. Next it ensures that all contexts l^a in which the capability could occur (and again this includes l) are

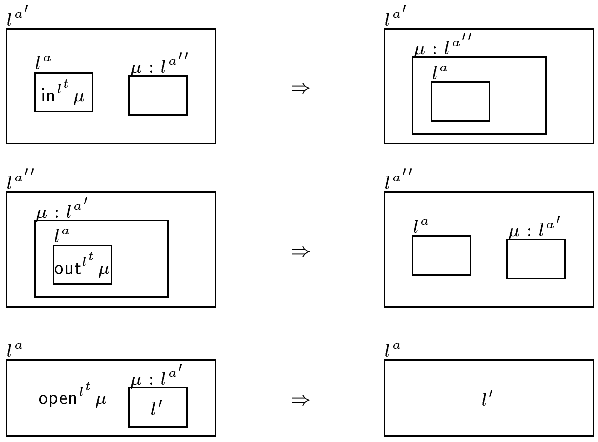

Fig. 2. Pictorial representation of the analysis of stable capabilities.

properly recorded as being possible ambients in all the possible grand parents $l^{a''}$ provided that the parent $l^{a'}$ has the stable name μ.

For the stable capability $\mathsf{open}^{l^t}\mu$ we once again start by ensuring that it is properly recorded as part of the current ambient l. Then we consider all contexts l^a in which the capability could occur (and once more this includes l) and find all subambients $l^{a'}$ having the stable name μ; these are opened by ensuring that whatever is included in the subambient $l^{a'}$ also occurs in the parent ambient l^a.

It is crucial to observe that we need to consult all possible contexts l^a in which the capability could occur and not just the obvious candidate l. This is because, in order to establish semantic soundness, the analysis has to take into account that the current ambient might be dissolved by an open capability. (This fine point was the main difficulty that needed to be overcome when developing the analysis.)

Example 4. Let us check the condition $(I,H) \models^{\mathrm{A}}_{me} \mathtt{k}^{\mathrm{B}}[\mathsf{out}^1\mathtt{w}.\,\mathsf{in}^2\mathtt{k'}.\,\mathsf{in}^3\mathtt{w}]$ that arises when checking that the analysis information (I,H) of Example 3 correctly validates the program $n^{l_\star}_\star[\mathit{Firewall} \mid \mathit{Agent}]$ of Example 1; here the naming environment me maps $\mathtt{k}$, $\mathtt{k'}$, $\mathtt{k''}$ and $\mathtt{w}$ to k, k', k'' and w, respectively. First we decide to let $\tilde{N}$ be $\{k\}$. We then need to check that $(I,H) \Vert\!\equiv_{me} \mathtt{k} : \{k\}$ (which follows from the choice of me), that $\{k\} \subseteq H(\mathrm{B})$ (which follows from Example 3), that $\mathrm{B} \in I(\mathrm{A})$ (which once more follows from Example 3) and that $(I,H) \models^{\mathrm{B}}_{me} \mathsf{out}^1\mathtt{w}.\,\mathsf{in}^2\mathtt{k'}.\,\mathsf{in}^3\mathtt{w}$ (see below).

To check that $(I,H) \models^{\mathrm{B}}_{me} \mathsf{out}^1\mathtt{w}.\,\mathsf{in}^2\mathtt{k'}.\,\mathsf{in}^3\mathtt{w}$ we first decide to let $\tilde{M}$ be $\{\mathsf{out}^1 w\}$. We then need to check that $(I,H) \rhd_{me} \mathsf{out}^1\mathtt{w} : \{\mathsf{out}^1 w\}$ (which follows from $(I,H) \Vert\!\equiv_{me} \mathtt{w} : \{w\}$), that $(I,H) \models^{\mathrm{B}} \mathsf{out}^1 w$ (see below) and that $(I,H) \models^{\mathrm{B}}_{me} \mathsf{in}^2\mathtt{k'}.\,\mathsf{in}^3\mathtt{w}$ (which amounts to twice repeating the checking illustrated for $\mathsf{out}^1\mathtt{w}$).

Finally, let us check that $(I,H) \models^{\mathrm{B}} \mathsf{out}^1 w$. First we check that $\mathsf{out}^1 w \in I(\mathrm{B})$ (using Example 3). For the second condition we have $l^a \in I^{-1}(\mathsf{out}^1 w) = \{\mathrm{A},\mathrm{B},\mathrm{C}\}$ and for each of the choices for l^a we have $l^{a'} \in I^{-1}(l^a) \cap H^{-1}(w) = \{\mathrm{A},\mathrm{C}\} \cap \{\mathrm{A}\} = \{\mathrm{A}\}$ so the parent ambient $l^{a'}$ of l^a will always be A. The grand parent of l^a is $l^{a''} \in I^{-1}(\mathrm{A}) = \{\mathrm{A},\mathrm{C}\}$ so the second condition amounts to checking that all of A,B and C are elements of both $I(\mathrm{A})$ and $I(\mathrm{C})$ and clearly this is the case. □

3.2 Properties of the analysis

In the terminology of data flow analysis [9] the above analysis is *flow-insensitive* since we ignore the order in which the capabilities occur; also it is *context-insensitive* (or *monovariant*) since a capability is analysed in the same way for all contexts in which it occurs.

Semantic correctness. Having specified what it means for an analysis result (I,H) to be acceptable the next step is to show that the notion of acceptability is semantically meaningful. We begin by establishing some auxiliary properties.

Fact 2. The analysis enjoys the following monotonicity properties:

(i) If $(I,H) \models_{me}^{l_1} P$ and $I(l_1) \subseteq I(l_2)$ then $(I,H) \models_{me}^{l_2} P$.
(ii) If $(I,H) \rhd_{me} M : \tilde{M}_1$ and $\tilde{M}_1 \subseteq \tilde{M}_2$ then $(I,H) \rhd_{me} M : \tilde{M}_2$.
(iii) If $(I,H) \Vdash_{me} N : \tilde{N}_1$ and $\tilde{N}_1 \subseteq \tilde{N}_2$ then $(I,H) \Vdash_{me} N : \tilde{N}_2$.
(iv) If $(I,H) \models^{l_1} \tilde{m}$ and $I(l_1) \subseteq I(l_2)$ then $(I,H) \models^{l_2} \tilde{m}$.

The proofs of *(ii)*, *(iii)* and *(iv)* are immediate; the proof of *(i)* is by structural induction. □

To express the next fact we shall write $me_1 =_P me_2$ to mean that me_1 and me_2 are equal on the free names of P and similarly for M and N.

Fact 3. The analysis only depends on the stable free names:

(i) If $me_1 =_P me_2$ and $(I,H) \models_{me_1}^{l} P$ then $(I,H) \models_{me_2}^{l} P$.
(ii) If $me_1 =_M me_2$ and $(I,H) \rhd_{me_1} M : \tilde{M}$ then $(I,H) \rhd_{me_2} M : \tilde{M}$.
(iii) If $me_1 =_N me_2$ and $(I,H) \Vdash_{me_1} N : \tilde{N}$ then $(I,H) \Vdash_{me_2} N : \tilde{N}$.

The proofs of *(iii)* and then *(ii)* are immediate; the proof of *(i)* is by structural induction. □

Lemma 1. If $P \equiv Q$ then $(I,H) \models_{me}^{l} P$ if and only if $(I,H) \models_{me}^{l} Q$.
The proof is by induction on the proof of $P \equiv Q$ and relies on Fact 3. □

We shall follow the approach from type systems and express the semantic correctness result as a *Subject Reduction Result*:

Theorem 1. If $(I,H) \models_{me}^{l} P$ and $P \to Q$ then $(I,H) \models_{me}^{l} Q$.

The proof is by induction on the transition $P \to Q$ and relies on Lemma 1 and Fact 2. $\square$

As a consequence, if (I, H) is an acceptable analysis result for the program $n_\star^{l_\star}[P_\star]$ of interest (with respect to the initial naming environment $me_\star$) then it will continue being so for all the derivatives of the program.

Existence of analysis results. So far we have only shown how to check that a given pair (I, H) is indeed an acceptable analysis result; we have not studied *(i)* whether or not acceptable analysis results always exist, and if they do, *(ii)* whether or not there always is a least analysis result.

To obtain these results we shall show that the set of acceptable analysis results constitutes a *Moore family* (or has a model intersection property):

> A subset Y of a complete lattice $(L, \sqsubseteq)$ is a Moore family whenever $Y' \subseteq Y$ implies that $\sqcap Y' \in Y$.

By taking $Y' = \emptyset$ we see that a Moore family Y cannot be empty and by taking $Y' = Y$ we see that it always contains a least element; this will be essential for answering *(i)* and *(ii)* in the affirmative.

In our setting the complete lattice of interest is the set $\mathbf{InAmb} \times \mathbf{HNam}$ of pairs of mappings (I, H) and the ordering is the pointwise extension of the subset ordering. We then have:

Theorem 2. $\{(I, H) \mid (I, H) \models^{l}_{me} P\}$ is a Moore family for all l, me and P.
The proof shows that all of the sets

$$\{(I, H, \tilde{N}) \mid (I, H) \Vert\!\!\equiv_{me} N : \tilde{N}\}$$
$$\{(I, H) \mid (I, H) \models^{l} \tilde{m}\},$$
$$\{(I, H, \tilde{M}) \mid (I, H) \rhd_{me} M : \tilde{M}\}$$

are Moore families and then proceeds by structural induction. $\square$

By restricting the attention to a given program $n_\star^{l_\star}[P_\star]$ of size s one can devise an $O(s^5)$ algorithm for computing the least solution. Roughly the idea is as follows [7]. There are $O(s)$ places where conditions needs to be checked. Each condition can have length $O(s^3)$ because there are at most three nested quantifiers each ranging over $O(s)$ entities. Hence at most $O(s^4)$ basic conditions need to be checked. Since the height of each set of values is $O(s)$ this can be implemented in $O(s^5)$ basic steps using standard worklist algorithms. (We conjecture that a more sophisticated implementation will be able to achieve $O(s^4)$.)

4 Validating Firewalls

In the examples we have studied a notion of firewall given by its private name `w` and the passwords `k`, `k'` and `k''` used for entering it. One aspect of being a firewall is that agents in the *approved* form must be allowed to enter. For the firewall proposed in Example 1 the approved form is $\mathtt{k}'^{\mathrm{C}}[\mathsf{open}^6\mathtt{k}.\,\mathtt{k}''^{\mathrm{D}}[Q]]$ and in Example 2 we showed that agents in this form can indeed enter the firewall: $\mathit{Firewall} \mid \mathit{Agent} \rightarrow^* (\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[P \mid Q]$ (assuming that $\mathtt{w} \notin \underline{\mathrm{fn}}(Q)$). As in [5] this can be strengthened to establish that $\mathit{Firewall} \mid \mathit{Agent}$ is observationally equivalent to $(\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[P \mid Q]$ (assuming that $\mathtt{w} \notin \underline{\mathrm{fn}}(Q)$).

Another aspect of being a firewall, not dealt with in [5], is to ensure that processes not knowing the right passwords cannot enter. Due to the power of the ambient calculus this is not as trivial as it might appear at first sight. As an example, a process that does not initially know the passwords might nonetheless learn them by other means. As another example, the firewall might contain a trapdoor through which processes might be able to enter (see Example 5 below).

We define a process U to be *ignorant* whenever $\underline{\mathrm{fn}}(U) \cap \{\mathtt{k}, \mathtt{k}', \mathtt{k}''\} = \emptyset$. We then define a proposed firewall F to be *protective* whenever the semantics of Section 2 prevents it from allowing any ignorant process to enter.

Example 5. Consider the proposed firewall

$$\begin{aligned}\mathit{Firewall}' : (\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[&\mathtt{k}^{\mathrm{B}}[\mathsf{out}^1\mathtt{w}.\,\mathsf{in}^2\mathtt{k}'.\,\mathsf{in}^3\mathtt{w}] \mid \mathsf{open}^4\mathtt{k}'.\,\mathsf{open}^5\mathtt{k}''.P\\ &\mid \mathtt{t}^{\mathrm{E}}[\mathsf{out}^7\mathtt{w}.\,\mathsf{in}^8\mathtt{w}.\,\mathsf{open}^9\mathtt{q}] \mid \mathsf{open}^{10}\mathtt{t}]\end{aligned}$$

that additionally contains a trapdoor $\mathtt{t}$. It is easy to check that

$$\mathit{Firewall}' \mid \mathit{Agent} \rightarrow^* (\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[\cdots \mid P \mid Q]$$

using *Agent* of Example 1 (assuming that $\mathtt{w} \notin \underline{\mathrm{fn}}(Q)$). But now the ignorant process $\mathtt{q}^{\mathrm{F}}[\mathsf{in}^{11}\mathtt{t}.\,Q]$ can also enter as is shown by

$$\mathit{Firewall}' \mid \mathtt{q}^{\mathrm{F}}[\mathsf{in}^{11}\mathtt{t}.\,Q] \rightarrow^* (\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[\cdots \mid P \mid Q]$$

(assuming that $\mathtt{w} \notin \underline{\mathrm{fn}}(Q)$) unlike what was intended. This means that $\mathit{Firewall}'$ is not a protective firewall because it can be entered by a process not knowing the right passwords. □

The control flow analysis can be used to devise a test for whether or not a proposed firewall F is protective; the test is displayed in Table 4 and will be explained below. Since the control flow analysis is approximate also the test for protectiveness will be approximate; however, we shall ensure that *whenever the test is passed then no ignorant processes can enter.* We believe this to be typical of applications where software developed by subcontractors is *validated* before being embedded in the software system under construction.

Let us fix the distinct stable names w, k, k', k'', $\mu_\circ$, and $\mu_\diamond$, the distinct labels $l^a_\diamond$ and $l^t_\diamond$, and the distinct names $n_\circ$ and $n_\diamond$. Thanks to Fact 1 we may without loss of generality assume that a proposed firewall $F = (\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F']$ does not contain any of these distinguished symbols in the subprocess $\mathtt{w}^{\mathrm{A}}[F']$.

INPUT:	a proposed firewall $F = (\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F']$ without distinguished symbols in $\mathtt{w}^{\mathrm{A}}[F']$
OUTPUT:	"accept" or "reject"
METHOD:	construct $\breve{F}$ (see the text) construct $me_0 = me_\star \&(\breve{F}, \mu_\circ)$ construct $\hat{T}$ (see the text) find the least (I, H) such that $(I, H) \models^{l_\star}_{me_0} \breve{F} \mid \hat{T}$ if $\exists l : l^a_\diamond \in I^+(l) \wedge w \in H(l)$ then "reject" else "accept"

Table 4. Testing for protectiveness.

Given a process Q we shall write $\hat{Q}$ for the process where all stable names are replaced by $\mu_\diamond$, all ambient labels by $l^a_\diamond$, and all transition labels by $l^t_\diamond$. Write B for the process

$$\mathsf{in}^{l^t_\diamond} n_\diamond \mid \mathsf{out}^{l^t_\diamond} n_\diamond \mid \mathsf{open}^{l^t_\diamond} n_\diamond \mid \mathsf{in}^{l^t_\diamond} n_\circ \mid \mathsf{out}^{l^t_\diamond} n_\circ \mid \mathsf{open}^{l^t_\diamond} n_\circ$$

and define T to be

$$B \mid n_\diamond^{l^a_\diamond}[\ B \mid n_\circ^{l^a_\diamond}[\mathbf{0}]\ ]$$

and note that this defines an ignorant process with $T = \hat{T}$.

Define the naming environment $me_\star$ by $me_\star(\mathtt{k}) = k$, $me_\star(\mathtt{k}') = k'$, $me_\star(\mathtt{k}'') = k''$, $me_\star(n_\circ) = \mu_\circ$, and $me_\star(n_\diamond) = \mu_\diamond$. For a naming environment me, a set X of names and a stable name μ define the naming environment $me\&(X, \mu)$ by

$$(me\&(X,\mu))(n) = \begin{cases} me(n) & \text{if } n \in dom(me) \\ \mu & \text{if } n \in X \setminus dom(me) \\ \text{undefined} & \text{if } n \notin X \cup dom(me) \end{cases}$$

and note that $(me\&(X,\mu))(n) = me(n)$ whenever $me(n)$ is defined. We shall allow to write $me\&(P,\mu)$ for $me\&(\underline{\mathrm{fn}}(P),\mu)$.

Given a proposed firewall $F = (\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F']$ we shall write $\breve{F}$ for the process $(\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F'']$ where F'' is like F' except that all stable names have been replaced by $\mu_\circ$. We have now defined all the notation used in the test displayed in Table 4. It operates on a proposed firewall F and outputs "accept" or "reject". It is clearly deterministic and given that the least (I, H) can be found in polynomial time it operates in polynomial time itself (in the size of F).

The correctness of the test hinges on the following key result; it shows that, from the point of view of the analysis, it is as hard to protect a firewall F against the process T as it is to protect the firewall F against any other ignorant process U:

Lemma 2. Let $F = (\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F']$ be a proposed firewall as demanded in Table 4 and let (I, H) be as in Table 4. If U is an ignorant process then

$$(I, H) \models^{l_\star}_{me'} \breve{F} \mid \hat{U}$$

where $me' = (me_\star \& (\breve{F}, \mu_\circ)) \& (\hat{U}, \mu_\diamond)$.

Proof. Write $me_0 = me_\star \& (\breve{F}, \mu_\circ)$ as in Table 4. By construction of (I, H) we have $(I, H) \models^{l_\star}_{me_0} \breve{F} \mid \hat{T}$ and using Fact 3 we get $(I, H) \models^{l_\star}_{me'} \breve{F} \mid \hat{T}$ from which $(I, H) \models^{l_\star}_{me'} \breve{F}$ and $(I, H) \models^{l_\star}_{me'} \hat{T}$ follows. By expansion of the latter we get

$$(I, H) \models\!\!\equiv^l \mathsf{in}^{l^t_\diamond}\mu, \qquad (I, H) \models\!\!\equiv^l \mathsf{out}^{l^t_\diamond}\mu, \qquad (I, H) \models\!\!\equiv^l \mathsf{open}^{l^t_\diamond}\mu \tag{1}$$

for all $l \in \{l_\star, l^a_\diamond\}$ and $\mu \in \{\mu_\circ, \mu_\diamond\}$; we also obtain

$$l^a_\diamond \in I(l_\star), \qquad l^a_\diamond \in I(l^a_\diamond), \qquad \mu_\diamond \in H(l^a_\diamond), \qquad \mu_\circ \in H(l^a_\diamond) \tag{2}$$

To conclude that $(I, H) \models^{l_\star}_{me'} \breve{F} \mid \hat{U}$ we need to prove that $(I, H) \models^{l_\star}_{me'} \hat{U}$ holds. Since U is ignorant this follows by Fact 3 from the following auxiliary result holding for arbitrary processes R:

$$(I, H) \models^l_{me} \hat{R}$$
$$\text{for all } l,\ me \text{ satisfying } l \in \{l_\star, l^a_\diamond\} \ \wedge\ \forall n \in \underline{\mathrm{fn}}(R) : me(n) \in \{\mu_\circ, \mu_\diamond\}$$

The proof of the auxiliary result is by structural induction on R and most of the cases are immediate so let us only consider the two interesting ones.

The case $\hat{R} = N^{l^a_\diamond}[\hat{R}_0]$: That $(I, H) \models^{l^a_\diamond}_{me} \hat{R}_0$ follows from the induction hypothesis; that $l^a_\diamond \in I(l)$ follows from (2); taking $\tilde{N} = \{\mu_\circ, \mu_\diamond\}$ we have $\tilde{N} \subseteq H(l^a_\diamond)$ from (2) and $(I, H) \,\|\!\!\equiv_{me} N : \tilde{N}$ is immediate.

The case $\hat{R} = \hat{M}.\,\hat{R}_0$: That $(I, H) \models^l_{me} \hat{R}_0$ follows from the induction hypothesis; taking $\tilde{M} = \{\mathsf{in}^{l^t_\diamond}\mu_\circ, \mathsf{out}^{l^t_\diamond}\mu_\circ, \mathsf{open}^{l^t_\diamond}\mu_\circ, \mathsf{in}^{l^t_\diamond}\mu_\diamond, \mathsf{out}^{l^t_\diamond}\mu_\diamond, \mathsf{open}^{l^t_\diamond}\mu_\diamond\}$ we have $\forall \tilde{m} \in \tilde{M} : (I, H) \models\!\!\equiv^l \tilde{m}$ from (1) and $(I, H) \,|\!\!\rhd_{me} M : \tilde{M}$ is immediate. □

When F passes the test and U is an ignorant process we want to show that no subambient of U ever passes inside `w`. Informally, this will take the form of assuming that $F \mid U \to^* R$ and guaranteeing that R contains no subambient $\mathtt{w}^{l^a_1}[\cdots \mathtt{u}^{l^a_2}[\cdots]\cdots]$ where `u` comes from U. Formalising this is somewhat tricky and we shall therefore avail ourselves of Fact 1 that allows us to arrange the labelling to suit our needs. Indeed, if $F \mid U \to^* R$ then $\breve{F} \mid \hat{U} \to^* R'$ for some R' such that $\lfloor R \rfloor = \lfloor R' \rfloor$.

Theorem 3. If F passes the test of Table 4 and U is an ignorant process and if $\breve{F} \mid \hat{U} \to^* R$ then R contains no subterm $n_1^{l^a_1}[\cdots n_2^{l^a_2}[\cdots]\cdots]$ where n_1 has stable name w and l^a_2 is $l^a_\diamond$.

Proof. Setting $me' = (me_\star \& (\breve{F}, \mu_\circ)) \& (\hat{U}, \mu_\diamond)$ and letting (I, H) be as in Table 4, it follows from Lemma 2 that $(I, H) \models^{l_\star}_{me'} \breve{F} \mid \hat{U}$. By Theorem 1 we also have $(I, H) \models^{l_\star}_{me'} R$. Suppose for the sake of contradiction that R does contain $n_1^{l^a_1}[\cdots n_2^{l^a_2}[\cdots]\cdots]$ where n_1 has stable name w and l^a_2 is $l^a_\diamond$. Then it follows from $(I, H) \models^{l_\star}_{me'} R$ that $l^a_\diamond \in I^+(l^a_1) \wedge w \in H(l^a_1)$ showing that the test could not have been passed. □

In summary, we have succeeded in using the control flow analysis to devise a polynomial time algorithm for ensuring that a proposed firewall is indeed protective; a web-based implementation is accessible via the Flow Logic webpage `http://www.daimi.au.dk/~fn/FlowLogic.html`.

Example 6. To test *Firewall* from Example 1 and *Firewall'* from Example 5 we need to be more precise about the subprocess P; in our tests we have used

$$!\mathsf{p}[\mathsf{in}\ \mathsf{p} \mid \mathsf{out}\ \mathsf{p} \mid \mathsf{open}\ \mathsf{p} \mid \mathsf{p}[\mathbf{0}]]$$

(omitting labels) as an example of an unrestricted internal process. Then *Firewall* passes the test because $H^{-1}(w) = \{\mathrm{A}\}$ and $I^{+}(\mathrm{A}) \not\ni l^{a}_{\diamond}$ but *Firewall'* fails the test because $H^{-1}(w) = \{\mathrm{A}\}$ and $I^{+}(\mathrm{A}) \ni l^{a}_{\diamond}$. □

5 Conclusion

It is well known that static techniques are needed for determining whether or not programs always evaluate in a permissible manner. Type systems have already been extensively used to study the properties of web-based languages and related calculi (e.g. [1,6]) but more "traditional" approaches [9] to static analysis have much to offer as well. In this paper we developed a control flow analysis for the ambient calculus building on recent developments for the pi-calculus [3,4].

The interplay between type systems and control flow analyses is not yet fully understood. While both type systems and control flow analyses can be proved semantically sound using a subject-reduction result, it would seem that only approaches based on control flow analyses admit least analyses for all processes. Indeed, often type systems (e.g. [1,6]) lack the corresponding notion of principal type, thereby making them harder to use in practice as there may be exponentially many types to consider before any conclusions can be drawn. In a subsequent paper we hope to use state-of-the-art techniques from data flow analysis to present an even stronger analysis than the control flow analysis developed here.

More importantly we demonstrated how a careful exploitation of the detailed operation of the control flow analysis allowed us to construct an attacker that was as hard to protect against as any other attacker; this is somewhat reminiscent of the identification of hard problems in a given complexity class. This allowed us to predict the operation of the firewall in conjunction with all ignorant attackers based on its operation in conjunction with the hard attacker; if it successfully protects against the hard attacker it will also protect against all other ignorant attackers.

This is a novel approach to the validation of software systems and we expect it to scale up to other calculi. Indeed, the basic machinery of the control flow analysis has already been developed for a number of calculi. Furhermore, by considering more powerful analyses expressed in the form of Flow Logics it is likely that one can reduce the gap between processes that know some of the passwords (hence are not ignorant) but still do not display them in the approved form.

References

1. M.Abadi: Secrecy by typing in security protocols. In Proceedings of Theoretical Aspects of Computer Software, volume 1281 of Lecture Notes in Computer Science, pages 611–638, Springer Verlag, 1997.
2. A.Aiken: Set constraints: Results, applications and future directions. In Proceedings of the Second Workshop on the Principles and Practices of Concurrent Programming, volume 874 of Lecture Notes in Computer Science, pages 171–179, Springer Verlag, 1994.
3. C.Bodei, P.Degano, F.Nielson, H.R.Nielson: Control Flow Analysis for the π-calculus. In Proceedings CONCUR'98, volume 1466 of Lecture Notes in Computer Science, pages 84–98, Springer Verlag, 1998.
4. C.Bodei, P.Degano, F.Nielson, H.R.Nielson: Static analysis of processes for no read-up and no write-down. In Proceedings FoSSaCS'99, volume 1578 of Lecture Notes in Computer Science, pages 120–134, Springer Verlag, 1999.
5. L.Cardelli, A.D.Gordon: Mobile Ambients. In Proceedings FoSSaCS'98, volume 1378 of Lecture Notes in Computer Science, pages 140–155, Springer Verlag, 1998.
6. L.Cardelli, A.D.Gordon: Types for Mobile Ambients. In Proceedings POPL'99, pages 79–92, ACM Press, 1999.
7. R.R.Hansen, J.G.Jensen: Flow Logics for Mobile Ambients. M.Sc.-thesis, 1999.
8. T.Jensen, D.LeMétayer, T.Thorn: Security and Dynamic Class Loading in Java: a formalisation. Report IRISA, Rennes, 1997.
9. F.Nielson, H.R.Nielson, C.Hankin: *Principles of Program Analysis*, Springer Verlag, 1999.