Validating Firewalls using Flow Logics

Flemming Nielson, Hanne Riis Nielson, René Rydhof Hansen.

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

E-mail: {fn,hrn,rrh}@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 to communicate and to move between sites. A site is said to be a protective firewall whenever it denies entry to all attackers not possessing the required passwords. We devise a computationally sound test for validating the protectiveness of a proposed firewall and show how to perform the test in polynomial time.

The first step is the definition of a flow logic for analysing the flow of control in mobile ambients; it amounts to a syntax-directed specification of the acceptability of a control flow estimate. The second step is to define a hardest attacker and to determine whether or not there exists a control flow estimate that shows the inability of the hardest attacker to enter; if such an estimate exists, then none of the infinitely many attackers can enter unless they contain at least one of the passwords, and consequently the firewall cannot contain any trap doors.

1 Introduction

The ambient calculus is a calculus of computation that is based on traditional process algebras (such as the π-calculus [15]). The main focus is not on communication, however, but on the ability of active processes to move between sites representing administrative domains; the calculus thereby extends the notion of mobility found in Java [13] where only passive code can be moved between sites. Both processes and sites are modelled as ambients; their ability to move around is governed by the capabilities possessed. The calculus was introduced in [7] and has been studied extensively [6, 8, 9, 14, 18, 19, 22]. We refer to Section 2 for a review of the ambient calculus.

Since processes may evolve when moving around, the structure of a system of ambients is very dynamic. In Section 3 we therefore develop a control flow analysis [17] for predicting the set of processes that may turn up inside a given ambient. This takes the form of defining a flow logic [16] for checking whether or not a control flow estimate (as might have been produced by a control flow analysis) is indeed acceptable; in the absence of higher-order features in the ambient calculus, this amounts to a syntax-directed definition of a number of judgements. The analysis combines the ability to handle communication (in the manner of analyses for the π-calculus [2, 3]) with the ability to handle movement (in the manner of an analysis for the communication-free fragment [18]). Semantic correctness is established by proving that all acceptable analyses are semantically sound (by means of a subject-reduction result in the manner of type systems).

On the algorithmic side we show that there always is a least control flow estimate and that it can be computed in polynomial time; this takes the form of generating a set of constraints that is then solved by a worklist algorithm [17].

In [7] the communication-free fragment of the ambient calculus is used to model and study a firewall where only agents knowing the required passwords are supposed to enter; indeed, assuming fairness, it is shown that all agents making correct use of all the passwords will in fact enter. However, in the interest of security and safety of systems, it is at least as important to ensure that an attacker not knowing any of the passwords cannot possibly enter; we shall say that the firewall is protective when this is the case. As an example, a protective firewall cannot contain trap doors or other ways of circumventing the protection offered by the passwords.

The difficulty of course is, that there are an infinity of attackers that do not know the passwords, and that it seems infeasible (and indeed undecidable) to perform automatic tests that will guard against all of these. To overcome these problems we change in Section 4 the "level of granularity" of our observations to coincide with those of the control flow analysis. We then prove that there is a process, called a hardest attacker, such that:

> If there exists a control flow estimate that shows the inability of the hardest attacker to enter, then none of the infinitely many attackers can enter unless they contain at least one of the passwords.

The ability to identify hardest attackers is perhaps comparable to the ability to identify hardest problems in given complexity classes. To argue the case in less technical jargon, consider the following "folk theorem":

> Testing[1] can prove the presence of bugs but never their absence.

Unfortunately, this has lead to the wide-spread belief that no experimentation with software can be used for formally validating software. The technical results presented here, generalising those of [18], provide a concrete instance of the rather different, and more useful, insight:

> Testing[2] can prove the absence of bugs but never their presence.

Expanding the area of applicability of this insight will likely lead to fundamental changes in the validation of software used in security oriented applications. The ability to extend the analysis to the existing software base, perhaps involving legacy code of "unknown" origin, offers a level of guarantee well above that of other formal approaches.

2 Mobile Ambients

Syntax. The presentation of the ambient calculus as given in [7] actually defines a "pre-syntax" for ambients. One aspect of this is that not all the defined ambient expressions are meaningful and hence a type system [8] is needed to rule out the undesired elements; the other aspect is that some of the clarifications made in the type system could in fact equally well be performed in the abstract syntax. To

[1] Testing in the sense of *dynamically running* a program on a number of inputs.
[2] Testing in the sense of *statically analysing* a program on a number of inputs.

$P ::= (\nu\, n^{\mu})P$	restriction	$M ::= \mathsf{in}^{l^i} N$	enter N
$\mid\ \mathbf{0}$	inactivity	$\mid\ \mathsf{out}^{l^o} N$	exit N
$\mid\ P \mid P'$	composition	$\mid\ \mathsf{open}^{l^p} N$	open N
$\mid\ !P$	replication	$\mid\ x$	variable
$\mid\ N^{l^a}[P]$	ambient	$\mid\ \epsilon$	null
$\mid\ M.\, P$	movement	$\mid\ M.M'$	path
$\mid\ \langle M \rangle^{l^c}$	output of capability		
$\mid\ \langle\!\langle N \rangle\!\rangle^{l^n}$	output of name	$N ::= n$	name
$\mid\ (x^{\beta^c}).\, P$	input of capability	$\mid\ u$	variable
$\mid\ (\!(u^{\beta^n})\!).\, P$	input of name		

Table 1. Abstract syntax.

avoid the artificial problem of devising semantics and static analysis for blatantly meaningless expressions we shall use a slightly more refined syntax that makes some of the distinctions of the type system.

The syntax of ambients in Table 1 is built around three syntactic categories: a class of processes, ranged over by $P \in \mathbf{Proc}$, a class of capabilities, ranged over by $M \in \mathbf{Cap}$, and a class of namings, ranged over by $N \in \mathbf{Nam}$. We follow [7] in distinguishing between names (introduced by the restriction operator known from process algebras) and variables (introduced by input statements); we also distinguish between variables used for holding capabilities, ranged over by $x \in \mathbf{Var}^c$, and variables used for holding names, ranged over by $u \in \mathbf{Var}^n$. We explain the constructs below.

First we consider processes (ignoring the superscript annotations in Table 1). Borrowing from the π-calculus [15] local scope is managed using the restriction operator. Also there is the inactive process, the parallel composition of two processes and a replicated process that is allowed to unfold to arbitrarily many ("infintely many") copies of the process. The next two constructs are unique to the ambient calculus. An ambient is a process operating inside a named border. Movement of ambients is governed by capabilities (explained below) and includes the ability for an ambient to move out of an enclosing ambient and for an ambient to move into a sibling ambient. Output of capabilities and names is much as in the π-calculus except that the channel is implicit and embedded in the enclosing ambient itself. Similarly for input of capabilities and names. As usual, trailing inactive actions will often be omitted from examples.

Next we consider capabilities and namings (once more ignoring the superscript annotations in Table 1). The in-capability directs the enclosing ambient to enter a sibling named N; this is illustrated in Figure 1 and will be explained in detail when we consider the semantics below. The out-capability directs the enclosing ambient to move out of its parent named N. The open-capability dissolves the border around a sibling ambient named N. Since capabilities can be communicated we also need variables ranging over them. Capabilities include the null capability as well as the sequential composition of capabilities describing a path to the desired destination. Namings are names but since they can be communicated we also need variables ranging over them.

Annotations. Let us now return to the two kinds of superscript annotations used in the syntax of Table 1. One class of annotations is composed of the *stable names*, ranged over by $\mu \in \mathbf{SNam}$, for names occurring in restrictions, and the *binders*, ranged over by $\beta^c \in \mathbf{Bnd}^c$ and $\beta^n \in \mathbf{Bnd}^n$, for variables occurring in input actions; we occasionally use $\beta \in \mathbf{Bnd} = \mathbf{Bnd}^c \cup \mathbf{Bnd}^n$. These annotations are necessary

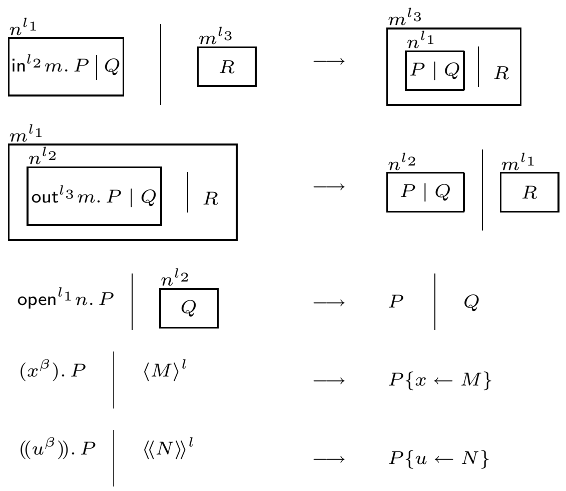

Fig. 1. Pictorial representation of the basic reduction rules.

for our analysis because the semantics of the ambient calculus borrows from the π-calculus in allowing α-conversion of bound names and variables. As an example, suppose we consider the ambient system

$$(\nu\, n)(\nu\, m)n[m[\mathbf{0}]]$$

and that our control flow estimate correctly says that m occurs inside n but not vice versa; unfortunately, this makes little sense since α-conversion allows us to change the above ambient system to

$$(\nu\, m)(\nu\, n)m[n[\mathbf{0}]]$$

and now the control flow estimate is no longer correct. To circumvent this problem we ensure that control flow estimates always refer to stable names and binders. Continuing the example, when we consider

$$(\nu\, n^{\mathsf{N}})(\nu\, m^{\mathsf{M}})n[m[\mathbf{0}]]$$

we say that M occurs inside N; this then remains correct for the α-converted system

$$(\nu\, m^{\mathsf{N}})(\nu\, n^{\mathsf{M}})m[n[\mathbf{0}]]$$

since stable names are never changed by α-conversion. Indeed, one way to understand the distinction between names and stable names is to regard the stable names as static representations of the names arising dynamically. The considerations for variables and binders are analogous.

The other class of annotations used in Table 1 are the labels. They assist in developing the control flow analysis by being able to precisely pin-point program points inside the ambient system; for this purpose it would be natural to let all labels be distinct since indeed all program points are distinct. As an example, in a system like

$$n[m^1[\mathsf{in}\, m] \mid m^2[\mathsf{out}\, n]]$$

this will allow us to distinguish the two occurrences of an ambient named m. As we shall see, labels allow us to control the complexity (and precision) of the analysis by treating one or more program points alike; for this purpose it may be appropriate only to use a few labels. Indeed, one way to understand the use of labels is to regard labels as amalgamations of a number of program points.

We use $l \in \mathbf{Lab}$ to range over the set of labels. More specifically, we view labels as being defined by

$$\mathbf{Lab} = \mathbf{Lab}^a \cup \mathbf{Lab}^i \cup \mathbf{Lab}^o \cup \mathbf{Lab}^p \cup \mathbf{Lab}^c \cup \mathbf{Lab}^n$$

and use $l^a \in \mathbf{Lab}^a$ to annotate ambients, $l^i \in \mathbf{Lab}^i$ to annotate in-capabilities, $l^o \in \mathbf{Lab}^o$ to annotate out-capabilities, $l^p \in \mathbf{Lab}^p$ to annotate open-capabilities, $l^c \in \mathbf{Lab}^c$ to annotate the output of capabilities, and $l^n \in \mathbf{Lab}^n$ to annotate the output of names.

We shall assume that the sets of labels, $\mathbf{Lab}$, stable names, $\mathbf{SNam}$, binders for the input of capabilities, $\mathbf{Bnd}^c$, and binders for the input of names, $\mathbf{Bnd}^n$, are pairwise disjoint. It may aid the intution to assume that the different sets of labels, listed above, are also pairwise disjoint. Finally, we assume that all the sets mentioned are non-empty; for any given program they can always be assumed to be finite since the semantics below does not create new labels, stable names or binders.

We write $\underline{\text{fn}}(P)$ for the set of *free names* of P and similarly for M and N. Similarly we write $\underline{\text{fv}}(P)$ for the set of *free variables* of P (and similarly for M and N); a process is *closed* whenever it has no free variables, i.e. $\underline{\text{fv}}(P) = \emptyset$, but may of course contain free names. Let $n_\star$ be a distinguished name and $l_\star$ a distinguished label; then the *programs* of interest are ambients of the form $n_\star^{l_\star}[P_\star]$ where $P_\star$ is closed, where $n_\star \notin \underline{\text{fn}}(P_\star)$ and where $l_\star$ does not occur in $P_\star$.

Example 1. Consider the following example from [7] for illustrating how an agent crosses a firewall using the prearranged passwords (or secret keys) $\mathtt{k}$, $\mathtt{k}'$ 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 labels of capabilities. □

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 is inductively defined by the axioms and rules of Table 2; apart from a few differences noted below it is a straightforward modification of a table in [7]. The axioms and rules in the left hand column ensure that the relation is an equivalence relation, that it is a congruence, that parallel composition is commutative and associative with the inactive process as a unit; they also describe the behaviour of replication.

The rules and axioms in the right hand column of Table 2 allow us to change the placement of restriction operators. In the axiom for $(\nu\, n^{\mu_n})(\nu\, m^{\mu_m})P$ we have added the side condition "$\underline{\text{if}}\ n \neq m$" to ensure that the association between names and stable names is not modified by the structural congruence. Next we have axioms for α-conversion; here we write $P\{n \leftarrow m\}$ for the process that is as P but with all free occurrences of n replaced by m (taking care to α-convert so as to avoid the capture of m by any restriction operator). The final two axioms control the expansion of capabilities.

The reduction relation is inductively defined by the axioms and rules of Table 3. It is as in [7] and a pictorial representation of the five basic axioms is given in Figure

$P \equiv P$	$(\nu\, n^{\mu_n})(\nu\, m^{\mu_m})P \equiv (\nu\, m^{\mu_m})(\nu\, n^{\mu_n})P$ $\underline{\text{if}}\ n \neq m$
$P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R$	
$P \equiv Q \Rightarrow Q \equiv P$	$(\nu\, n^{\mu})(P \mid Q) \equiv P \mid (\nu\, n^{\mu})Q$ $\underline{\text{if}}\ n \notin \underline{\text{fn}}(P)$
$P \equiv Q \Rightarrow (\nu\, n^{\mu})P \equiv (\nu\, n^{\mu})Q$	$(\nu\, n^{\mu})(m^{l}[P]) \equiv m^{l}[(\nu\, n^{\mu})P]$ $\underline{\text{if}}\ n \neq m$
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	
$P \equiv Q \Rightarrow !P \equiv !Q$	
$P \equiv Q \Rightarrow N^{l}[P] \equiv N^{l}[Q]$	$(\nu\, n^{\mu})P \equiv (\nu\, m^{\mu})(P\{n \leftarrow m\})$ $\underline{\text{if}}\ m \notin \underline{\text{fn}}(P)$ (α-renaming)
$P \equiv Q \Rightarrow M.\, P \equiv M.\, Q$	
$P \equiv Q \Rightarrow (x^{\beta^c}).\, P \equiv (x^{\beta^c}).\, Q$	$(x^{\beta^c}).\, P \equiv (x'^{\beta^c}).\, (P\{x \leftarrow x'\})$ $\underline{\text{if}}\ x' \notin \underline{\text{fv}}(P)$ (α-renaming)
$P \equiv Q \Rightarrow (\!(u^{\beta^n})\!).\, P \equiv (\!(u^{\beta^n})\!).\, Q$	$(\!(u^{\beta^n})\!).\, P \equiv (\!(u'^{\beta^n})\!).\, (P\{u \leftarrow u'\})$ $\underline{\text{if}}\ u' \notin \underline{\text{fv}}(P)$ (α-renaming)
$P \mid Q \equiv Q \mid P$	
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	
$P \mid \mathbf{0} \equiv P$	
$!P \equiv P \mid !P$	$(M.M').\, P \equiv M.\, M'.\, P$
$!\mathbf{0} \equiv \mathbf{0}$	$\epsilon.\, P \equiv P$
$(\nu\, n^{\mu})\mathbf{0} \equiv \mathbf{0}$	

Table 2. Structural congruence.

1. The remaining rules ensure that reduction can take place in the contexts of restrictions, ambients and parallel compositions and that the structural congruence can freely be used to rearrange ambient expressions. Note that no internal reduction can take place "inside" movement or input prefixes. Also note that in each reduction, exactly one of the basic axioms is used.

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

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

The transition sequence shows that the firewall (which has the private name w) sends out the pilot ambient named 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. □

Properties of the semantics. Recall that the programs of interest are ambients of the form $n_\star^{l_\star}[P_\star]$ where $P_\star$ is closed (and hence contains no free variables but may contain free names), $n_\star \notin \underline{\text{fn}}(P_\star)$ and $P_\star$ does not contain $l_\star$. It follows that only closed expressions are reduced and that no new names become free and that no new labels come into existence. Because of the structural congruence it is not the case that programs evolve into programs (in particular processes of the form $n_\star^{l_\star}[\cdots]$); to achieve this we could restrict the use of the congruence in the semantics. Instead, we shall use the following result showing that programs evolve into processes that are congruent to programs.

$P \to Q \Rightarrow (\nu\, n^{\mu})P \to (\nu\, n^{\mu})Q$	$n^{l_1}[\mathsf{in}^{l_2} m.\, P \mid Q] \mid m^{l_3}[R] \to m^{l_3}[n^{l_1}[P \mid Q] \mid R]$
$P \to Q \Rightarrow n^{l}[P] \to n^{l}[Q]$	$m^{l_1}[n^{l_2}[\mathsf{out}^{l_3} m.\, P \mid Q] \mid R] \to n^{l_2}[P \mid Q] \mid m^{l_1}[R]$
$P \to Q \Rightarrow P \mid R \to Q \mid R$	$\mathsf{open}^{l_1} n.\, P \mid n^{l_2}[Q] \to P \mid Q$
$\left.\begin{array}{l} P \equiv P' \\ P' \to Q' \\ Q' \equiv Q \end{array}\right\} \Rightarrow P \to Q$	$(x^{\beta}).\, P \mid \langle M \rangle^{l} \to P\{x \leftarrow M\}$ $(\!(u^{\beta})\!).\, P \mid \langle\!\langle N \rangle\!\rangle^{l} \to P\{u \leftarrow N\}$

Table 3. Reduction relation.

Fact 1. Programs evolve to programs (modulo the congruence):

> If P is congruent to a program and $P \to^{*} Q$ then also Q is congruent to a program.

Proof. We first investigate what it means for a process to be congruent to a program. For this we extend the syntax of Table 1 with

$$
\begin{array}{rll}
T ::= & (\nu\, n^{\mu})T & \\
\mid & \mathbf{0} & \\
\mid & T \mid T' & \\
\mid & !T & \\
\mid & L.\, T &
\end{array}
\qquad
\begin{array}{rll}
S ::= & (\nu\, n^{\mu})S & \text{if } n \neq n_{\star} \\
\mid & S \mid T & \\
\mid & T \mid S & \\
\mid & n_{\star}^{l_{\star}}[P] & \text{if } n_{\star}^{l_{\star}}[P] \text{ is a program} \\
\mid & L.\, S &
\end{array}
$$

$$L ::= \epsilon \mid L.L'$$

Let **Triv** be the set of processes described by T and let **Ser** be the set of processes described by S; the former clearly contains the inactive process $\mathbf{0}$ and the latter clearly contains all programs. We now show that **Triv**, respectively **Ser**, is the closure of the set $\{\mathbf{0}\}$, respectively the set of programs, under the congruence.

It is immediate to prove by induction in L that $L.\, T \equiv T$ and $L.\, S \equiv S$. It is also immediate to prove by induction in T that $T \equiv \mathbf{0}$; hence all processes in **Triv** are congruent to $\mathbf{0}$. Furthermore, by induction in S one can show that $S \equiv n_{\star}^{l_{\star}}[P]$ for some P such that $n_{\star}^{l_{\star}}[P]$ is a program; hence all processes in **Ser** are congruent to programs.

For the opposite inclusions we prove that if $P \equiv Q$ and $P \in$ **Triv** then $Q \in$ **Triv**, and similarly that if $P \equiv Q$ and $Q \in$ **Triv** then $P \in$ **Triv**, by induction in the inference; it follows that **Triv** is the set of processes that are congruent to $\mathbf{0}$. Next we prove that if $P \equiv Q$ and $P \in$ **Ser** then $Q \in$ **Ser**, and similarly that if $P \equiv Q$ and $Q \in$ **Ser** then $P \in$ **Ser**, by induction in the inference; hence **Ser** is the set of processes that are congruent to programs.

We now turn to the statement of the fact: if $P \in$ **Ser** and $P \to^{*} Q$ then $Q \in$ **Ser**. We proceed by induction in the length of the derivation. In the induction step, $P \to^{*} R \to Q$ with $R \in$ **Ser**, we consider the place where the basic axiom is used for establishing $R \to Q$. Since a process of the form T cannot contain any labels or binders the basic axiom used for $R \to Q$ cannot involve any subprocess of the form T. It follows that the basic axiom must take place inside the (necessarily unique) occurrence of $n_{\star}^{l_{\star}}[\cdots]$ in R. Inspection of the five basic axioms then immediately establish the result. □

It should be clear that the syntactic annotations in no way influence the semantics. We can make this precise as follows. Let $\mu_{\bullet}$ be a distinguished stable name, let $\beta_{\bullet}^{c}$

and $\beta^n_\bullet$ be distinguished binders, and let $l^a_\bullet$, $l^i_\bullet$, $l^o_\bullet$, $l^p_\bullet$, $l^c_\bullet$ and $l^n_\bullet$ be distinguished labels. Given a process P write $\lfloor P \rfloor$ for the process where all stable names, binders and labels are replaced by the appropriate distinguished stable names, binders and labels.

Fact 2. The semantics is invariant under annotations:

> If $P \to^* Q$ and $\lfloor P \rfloor = \lfloor P' \rfloor$ then there exists Q' such that $P' \to^* Q'$ and $\lfloor Q \rfloor = \lfloor Q' \rfloor$.

Proof. We first consider the similar statement for the congruence:

> If $P \equiv Q$ and $\lfloor P \rfloor = \lfloor P' \rfloor$ then there exists Q' such that $P' \equiv Q'$ and $\lfloor Q \rfloor = \lfloor Q' \rfloor$; similarly if $P \equiv Q$ and $\lfloor Q \rfloor = \lfloor Q' \rfloor$ then there exists P' such that $P' \equiv Q'$ and $\lfloor P \rfloor = \lfloor P' \rfloor$.

It is proved by induction in the inference tree for $P \equiv Q$.

We then prove the statement of the fact by induction in the length of the derivation $P \to^* Q$. For the induction step $P \to^* R \to Q$ we proceed by induction in the shape of the inference tree for $R \to Q$. $\square$

3 Control Flow Analysis

Immediate constituents of ambients. The main aim of the control flow analysis is to obtain the following information for each ambient: *(i)* which ambients *may* be immediately contained in it, *(ii)* which capabilities *may* it perform, *(iii)* which input actions *may* be performed immediately inside it, and *(iv)* which output actions *may* be performed immediately inside it. An ambient will be identified by its label $l^a \in \mathbf{Lab}^a$, an in-capability by its label $l^i \in \mathbf{Lab}^i$, an out-capability by its label $l^o \in \mathbf{Lab}^o$, an open-capability by its label $l^p \in \mathbf{Lab}^p$, an input of a capability by its binder[3] $\beta^c \in \mathbf{Bnd}^c$, an input of a name by its binder $\beta^n \in \mathbf{Bnd}^n$, the output of a capability by its label $l^c \in \mathbf{Lab}^c$, and the output of a name by its label $l^n \in \mathbf{Lab}^n$.

The analysis records this information in the following component:

$$I \in \mathbf{InAmb} = \mathbf{Lab}^a \to \mathcal{P}\begin{pmatrix} \mathbf{Lab}^a \cup \mathbf{Lab}^i \cup \mathbf{Lab}^o \cup \mathbf{Lab}^p \cup \\ \mathbf{Lab}^c \cup \mathbf{Lab}^n \cup \mathbf{Bnd}^c \cup \mathbf{Bnd}^n \end{pmatrix}$$

When specifying the analysis we shall also use the "inverse" mapping

$$I^{-1} : (\mathbf{Lab} \cup \mathbf{Bnd}) \to \mathcal{P}(\mathbf{Lab}^a)$$

that returns the set of ambients in which the given ambient, capability, input or output might occur; formally $z \in I(l^a)$ if and only if $l^a \in I^{-1}(z)$. Later we shall write $z \in I^+(l)$ to mean that there exists $l_1, \cdots, l_n$ (for $n \geq 1$) such that $l = l_1$, $z = l_n$, and $\forall i < n : l_{i+1} \in I(l_i)$.

[3] This actually confuses binders with labels; it would be notationally purer, but somewhat heavier, to demand input actions to be annotated not only with a binder but also with a label.

Stable names of ambients and capabilities. Ambients and capabilities have stable names associated with them and to keep track of this information the analysis also contains the following component:

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

As above we shall also use the "inverse mapping"

$$H^{-1} : \mathbf{SNam} \to \mathcal{P}(\mathbf{Lab}^a \cup \mathbf{Lab}^i \cup \mathbf{Lab}^o \cup \mathbf{Lab}^p)$$

that returns the set of ambients that might have the given stable name; formally $\mu \in H(l)$ if and only if $l \in H^{-1}(\mu)$. The information in H is needed to determine the ambients operated upon by the capabilities so as to accurately update the contents of I.

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

$$\begin{aligned} me \in \mathbf{MEnv} = (\mathbf{Nam} \cup \mathbf{Var}^c \cup \mathbf{Var}^n) &\to_{\text{fin}} (\mathbf{SNam} \cup \mathbf{Bnd}^c \cup \mathbf{Bnd}^n) \\ &\underline{\text{such}}\ \underline{\text{that}}\ me(n) \in \mathbf{SNam}, me(x) \in \mathbf{Bnd}^c, me(u) \in \mathbf{Bnd}^n \end{aligned}$$

Here we impose the condition that the marker environment "preserves the types" of names, variables ranging over capabilities and variables ranging over names.

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. Recall that for $n_\star^{l_\star}[P_\star]$ to be a program we demand that $P_\star$ is closed, that $n_\star \notin \underline{\text{fn}}(P_\star)$ and that $P_\star$ does not contain $l_\star$. For $(me_\star, n_\star^{l_\star}[P_\star])$ to constitute a program of interest we then demand that:

$n_\star^{l_\star}[P_\star]$ is a program,

$me_\star$ defines all the free names of $n_\star^{l_\star}[P_\star]$, i.e. $\underline{\text{fn}}(n_\star^{l_\star}[P_\star]) \subseteq dom(me_\star)$, and $me_\star$ does not define any variables, i.e. $dom(me_\star) \subseteq \mathbf{Nam}$, and

the stable name $\mu_\star$ is distinguished, does not occur in $P_\star$, and is only possessed by $n_\star$, i.e. $me_\star^{-1}(\mu_\star) = \{n_\star\}$.

(Here we write $me_\star^{-1}(\mu)$ for the set $\{n \mid me_\star(n) = \mu\}$ of names mapped to μ.)

Communication and stable capabilities. The *environment* R is responsible for collecting the values that can be bound to the variables as result of an *input action.* The variables are represented by their binders. In the case of input of names it is natural to represent the name by its stable name. In a similar way, in the case of input of capabilities, we shall represent the capability by its stable capability:

$$\begin{aligned} \tilde{m} &\in \mathbf{SCap} \\ \tilde{m} &::= \mathsf{in}^{l^i} \mid \mathsf{out}^{l^o} \mid \mathsf{open}^{l^p} \end{aligned}$$

There are no stable capabilities corresponding to null capabilities (ϵ) and paths ($M_1.M_2$); instead the analysis will break[4] them into the set of constituent in-, out-, and open-capabilities.

The *communication box* C is responsible for collecting the effects of the *output actions* taking place immediately inside the ambient: again the ambient is identified

[4] Clearly a more precise analysis can be devised; however, we choose the simpler approach so that the constraint solver of Subsection 3.3 only needs to operate in a finite universe and hence can compute solutions explicitly in polynomial time.

by its label and the value being communicated will be a stable capability or a stable name:

$$\begin{aligned} R = (R^c, R^n) &\in \ \ \mathbf{Env} \ \ = (\mathbf{Bnd}^c \to \mathcal{P}(\mathbf{SCap})) \times (\mathbf{Bnd}^n \to \mathcal{P}(\mathbf{SNam})) \\ C = (C^c, C^n) &\in \mathbf{Comm} = (\mathbf{Lab}^a \to \mathcal{P}(\mathbf{SCap})) \times (\mathbf{Lab}^a \to \mathcal{P}(\mathbf{SNam})) \end{aligned}$$

Also the information in R and C will be needed to accurately update the contents of I in the presence of communication.

Example 3. Consider the program $n_\star^{l_\star}[Firewall \mid Agent]$ of Example 1 and the following analysis estimate (where the initial naming environment $me_\star$ maps the names $n_\star$, $\mathtt{k}$, $\mathtt{k}'$ and $\mathtt{k}''$ to $\mu_\star$, k, k' and k'', respectively):

$$\begin{aligned} I(l_\star) &= \{\mathrm{A}, \mathrm{B}, \mathrm{C}\} \\ I(\mathrm{A}) &= \{1, 2, 3, 4, 5, 6, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\} \\ I(\mathrm{B}) &= \{1, 2, 3\} \\ I(\mathrm{C}) &= \{1, 2, 3, 6, \mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}\} \\ I(\mathrm{D}) &= \emptyset \end{aligned}$$

label	$l_\star$	A	B	C	D	1	2	3	4	5	6
H	$\{\mu_\star\}$	$\{w\}$	$\{k\}$	$\{k'\}$	$\{k''\}$	$\{w\}$	$\{k'\}$	$\{w\}$	$\{k'\}$	$\{k''\}$	$\{k\}$

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 indeed happens.

No communication is taking place and it is therefore safe to set $R^c(\beta^c) = \emptyset$ and $R^n(\beta^n) = \emptyset$ for all binders and to set $C^c(l^a) = \emptyset$ and $C^n(l^a) = \emptyset$ for all labels. $\square$

3.1 The acceptability relation

The acceptability of a control flow estimate is defined by the following four predicates (defined in Tables 4 and 5 and explained below):

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

Analysis of processes. Table 4 gives a simple syntax-directed definition of what it means for an analysis estimate (I, H, C, R) 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 or an input 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

$(I,H,C,R) \models^{l}_{me} (\nu\, n^{\mu})P$	iff	$(I,H,C,R) \models^{l}_{me[n\mapsto\mu]} P$
$(I,H,C,R) \models^{l}_{me} \mathbf{0}$	iff	true
$(I,H,C,R) \models^{l}_{me} P \mid P'$	iff	$(I,H,C,R) \models^{l}_{me} P \;\wedge$ $(I,H,C,R) \models^{l}_{me} P'$
$(I,H,C,R) \models^{l}_{me} !P$	iff	$(I,H,C,R) \models^{l}_{me} P$
$(I,H,C,R) \models^{l}_{me} N^{l^a}[P]$	iff	$(I,H,C,R) \models^{l^a}_{me} P \;\wedge\; l^a \in I(l) \;\wedge$ $(I,H,C,R) \Vert\!\equiv_{me} N : \tilde{N} \;\wedge\; \tilde{N} \subseteq H(l^a)$
$(I,H,C,R) \models^{l}_{me} M.\,P$	iff	$(I,H,C,R) \models^{l}_{me} P \;\wedge$ $(I,H,C,R) \rhd_{me} M : \tilde{M} \;\wedge$ $\forall \tilde{m} \in \tilde{M} : (I,H,C,R) \models^{l} \tilde{m}$
$(I,H,C,R) \models^{l}_{me} \langle M\rangle^{l^c}$	iff	$l^c \in I(l) \;\wedge\; (I,H,C,R) \rhd_{me} M : \tilde{M} \;\wedge$ $\forall l^a \in I^{-1}(l^c) : C^c(l^a) \supseteq \tilde{M}$
$(I,H,C,R) \models^{l}_{me} \langle\!\langle N\rangle\!\rangle^{l^n}$	iff	$l^n \in I(l) \;\wedge\; (I,H,C,R) \Vert\!\equiv_{me} N : \tilde{N} \;\wedge$ $\forall l^a \in I^{-1}(l^n) : C^n(l^a) \supseteq \tilde{N}$
$(I,H,C,R) \models^{l}_{me} (x^{\beta^c}).\,P$	iff	$(I,H,C,R) \models^{l}_{me[x\mapsto\beta^c]} P \;\wedge\; \beta^c \in I(l) \;\wedge$ $\forall l^a \in I^{-1}(\beta^c) : C^c(l^a) \subseteq R^c(\beta^c)$
$(I,H,C,R) \models^{l}_{me} (\!(u^{\beta^n})\!).\,P$	iff	$(I,H,C,R) \models^{l}_{me[u\mapsto\beta^n]} P \;\wedge\; \beta^n \in I(l) \;\wedge$ $\forall l^a \in I^{-1}(\beta^n) : C^n(l^a) \subseteq R^n(\beta^n)$

Table 4. Control flow analysis (for processes).

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 Table 5 explained below.

As in Prolog, all free identifiers (like $\tilde{N}$ and $\tilde{M}$) on the right-hand sides of clauses are implicitly assumed to be existentially quantified; this means that whenever a clause is applied we are free to supply suitable values for these identifiers.

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 $\tilde{M}$. Intuitively $\tilde{M}$ is the singleton $\{\mathsf{in}^{l^i}\}$ when M is $\mathsf{in}^{l^i} n$ and similarly for the other capabilities; this is made precise by Table 5 explained below (that also takes care of associating the stable name of n with the label l^i).

The two clauses for output actions first record that the action may take place inside the enclosing ambient. The next step is to translate the capability M (resp. the name N) into a set of stable capabilities $\tilde{M}$ (resp. a set of stable names $\tilde{N}$) thereby making it independent of the context. The communication box C is then updated to record that each ambient l^a (including l) that could contain the output action will in fact witness the output of $\tilde{M}$ (resp. $\tilde{N}$).

Finally, the two clauses for input action first update the environment me to record the binding of the variable before analysing the subprocess. It is then ensured that the component I contains the appropriate binder (β^c or β^n) representing the input action. To determine the possible value being communicated (and hence bound to the variable represented by the binder) we have to consult the communication box of the enclosing ambient. So for all ambients l^a (including l) that might contain the

$(I,H,C,R) \rhd_{me} \mathsf{in}^{l^i} N : \tilde{M}$	iff	$(I,H,C,R) \Vvdash_{me} N : \tilde{N} \ \wedge$ $\tilde{M} \supseteq \{\mathsf{in}^{l^i}\} \ \wedge \ H(l^i) \supseteq \tilde{N}$
$(I,H,C,R) \rhd_{me} \mathsf{out}^{l^o} N : \tilde{M}$	iff	$(I,H,C,R) \Vvdash_{me} N : \tilde{N} \ \wedge$ $\tilde{M} \supseteq \{\mathsf{out}^{l^o}\} \ \wedge \ H(l^o) \supseteq \tilde{N}$
$(I,H,C,R) \rhd_{me} \mathsf{open}^{l^p} N : \tilde{M}$	iff	$(I,H,C,R) \Vvdash_{me} N : \tilde{N} \ \wedge$ $\tilde{M} \supseteq \{\mathsf{open}^{l^p}\} \ \wedge \ H(l^p) \supseteq \tilde{N}$
$(I,H,C,R) \rhd_{me} x : \tilde{M}$	iff	$\tilde{M} \supseteq R^c(me(x))$
$(I,H,C,R) \rhd_{me} \epsilon : \tilde{M}$	iff	$\tilde{M} \supseteq \emptyset$
$(I,H,C,R) \rhd_{me} M_1.M_2 : \tilde{M}$	iff	$(I,H,C,R) \rhd_{me} M_1 : \tilde{M}_1 \ \wedge$ $(I,H,C,R) \rhd_{me} M_2 : \tilde{M}_2 \ \wedge$ $\tilde{M} \supseteq \tilde{M}_1 \cup \tilde{M}_2$
$(I,H,C,R) \Vvdash_{me} n : \tilde{N}$	iff	$\tilde{N} \supseteq \{me(n)\}$
$(I,H,C,R) \Vvdash_{me} u : \tilde{N}$	iff	$\tilde{N} \supseteq R^n(me(u))$
$(I,H,C,R) \models^l \mathsf{in}^{l^i}$	iff	$l^i \in I(l) \ \wedge$ $\forall l^a \in I^{-1}(l^i) : \forall \mu \in H(l^i) : \forall l^{a'} \in I^{-1}(l^a) :$ $\forall l^{a''} \in I(l^{a'}) \cap H^{-1}(\mu) \cap \mathbf{Lab}^a :$ $l^a \in I(l^{a''})$
$(I,H,C,R) \models^l \mathsf{out}^{l^o}$	iff	$l^o \in I(l) \ \wedge$ $\forall l^a \in I^{-1}(l^o) : \forall \mu \in H(l^o) :$ $\forall l^{a'} \in I^{-1}(l^a) \cap H^{-1}(\mu) :$ $\forall l^{a''} \in I^{-1}(l^{a'}) : l^a \in I(l^{a''})$
$(I,H,C,R) \models^l \mathsf{open}^{l^p}$	iff	$l^p \in I(l) \ \wedge$ $\forall l^a \in I^{-1}(l^p) : \forall \mu \in H(l^p) :$ $\forall l^{a'} \in I(l^a) \cap H^{-1}(\mu) \cap \mathbf{Lab}^a :$ $\forall l' \in I(l^{a'}) : l' \in I(l^a)$

Table 5. Control flow analysis (for capabilities and namings).

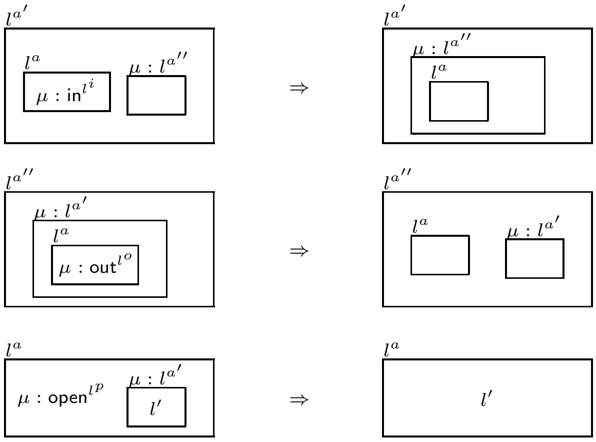

Fig. 2. Pictorial representation of the analysis of atomic capabilities.

input action we record that the contents of the communication box $C(l^a)$ can be a value of the binder and we use the environment R to capture this.

Translation of capabilities and namings. The first two parts of Table 5 translate capabilities and namings into a form where they are independent of the context. For capabilities we return (in $\tilde{M}$) the stable capability and in the H component the stable names relating to the capability. It would have been very natural to follow [18] in bypassing the H component for this purpose and to include the stable name in the stable capability; the reason for not doing so is to allow approximations that will yield faster analyses than possible using the approach of [18].

The entities recorded in R come into play in the clause for translating variables to stable capabilities and stable namings; as an example, for a capability variable x we consult me and R^c to determine the possible stable capabilities that x might stand for. The two clauses for null capabilities and paths show how capabilites are broken up into their atomic constituents; as mentioned earlier this is to facilitate the development of a simple constraint solver for implementing the analysis in polynomial time.

The form of the judgements for translating capabilities and namings combine the *verbose* and *succinct* forms of flow logic [16]. The verbose format, as used for the analysis of processes and capabilities, explicitly contains a record of the information as it pertains to all internal program points; this is part of the (I, H, C, R) component. The succinct format, to be specific the $\tilde{M}$ and $\tilde{N}$ components of the judgements for translation, directly expresses auxiliary information that is only of local interest. The use of succinct components frequently make specifications more readable and tend to give them the flavour of type systems. The relationship between verbose and succinct specifications is studied in [21].

Analysis of capabilities. The last part of Table 5 shows how to check stable capabilities against the control flow estimate (I, H, C, R). Figure 2 illustrates these clauses pictorially; the similarity between Figures 2 and 1 stresses the systematic way in which a control flow analysis may be developed from a formal semantics and we regard this as a strong point of our approach.

The clause for in^{l^i} 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 a stable name μ associated with in^{l^i}. This involves quantifying over all possible parent ambients $l^{a'}$ and using the component H to obtain the stable name of the ambient that $l^{a''}$ indicates.

The clause for out^{l^o} 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 properly recorded as being possible ambients in all the possible grandparents $l^{a''}$ provided that the parent $l^{a'}$ has a stable name μ associated with out^{l^o}.

For the stable capability open^{l^p} 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 again this includes l) and find all subambients $l^{a'}$ having a stable name μ associated with open^{l^p}; 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.

Example 4. Let us check the condition

$$(I,H,C,R) \models^{\mathrm{A}}_{me} \mathtt{k}^{\mathrm{B}}[\mathsf{out}^1\mathtt{w}.\,\mathsf{in}^2\mathtt{k}'.\,\mathsf{in}^3\mathtt{w}]$$

that arises when checking that the analysis estimate (I,H,C,R) of Example 3 correctly validates the program $n_\star^{l_\star}[\mathit{Firewall} \mid \mathit{Agent}]$ of Example 1; here the naming environment me maps $n_\star$, $\mathtt{k}$, $\mathtt{k}'$, $\mathtt{k}''$ and $\mathtt{w}$ to $\mu_\star$, k, k', k'' and w, respectively. First we decide to let $\tilde{N}$ be $\{k\}$. We then need to check that $(I,H,C,R) \Vdash_{me} \mathtt{k} : \{k\}$ (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 $(I,H,C,R) \models^{\mathrm{B}}_{me}$ $\mathsf{out}^1\mathtt{w}.\,\mathsf{in}^2\mathtt{k}'.\,\mathsf{in}^3\mathtt{w}$ (see below).

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

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

3.2 Properties of the analysis

In the terminology of data flow analysis [17] 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. We refer to [14, 19, 22] for more precise ways of analysing the communication-free fragment.

Semantic correctness. Having specified what it means for an analysis estimate (I,H,C,R) 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 3. The analysis enjoys the following monotonicity properties:

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

Proof. It is immediate to prove *(iii)* by inspection of the two cases for N.

We then prove *(ii)* by inspection of the six cases for M and all but one case is immediate. In the case for $M_1.M_2$ we have $(I,H,C,R) \rhd_{me} M_1 : \tilde{M}_1'$ and $(I,H,C,R) \rhd_{me} M_2 : \tilde{M}_2'$ for some $\tilde{M}_1'$ and $\tilde{M}_2'$ such that $\tilde{M}_1 \supseteq \tilde{M}_1' \cup \tilde{M}_2'$; it is then immediate that $\tilde{M}_2 \supseteq \tilde{M}_1' \cup \tilde{M}_2'$ as was to be shown.

Next we prove *(iv)* by inspection of the three cases for $\tilde{m}$. All cases are immediate since the label l_i (for $i = 1, 2$) is only used to establish a fact of the form $l' \in I(l_i)$.

Finally we prove *(i)* by structural induction in P. This is straightforward because the label l_i (for $i = 1, 2$) is only used in recursive calls, to establish a fact of the form $l' \in I(l_i)$ or $\beta' \in I(l_i)$, or to invoke *(iv)*. □

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 and variables of P; in a similar way we write $me_1 =_M me_2$ and $me_1 =_N me_2$ for capabilities M and namings N.

Fact 4. The analysis only depends on the free names and variables:

(i) If $me_1 =_P me_2$ and $(I, H, C, R) \models^l_{me_1} P$ then $(I, H, C, R) \models^l_{me_2} P$.
(ii) If $me_1 =_M me_2$ and $(I, H, C, R) \rhd_{me_1} M : \tilde{M}$ then $(I, H, C, R) \rhd_{me_2} M : \tilde{M}$.
(iii) If $me_1 =_N me_2$ and $(I, H, C, R) \Vert\!\!\equiv_{me_1} N : \tilde{N}$ then $(I, H, C, R) \Vert\!\!\equiv_{me_2} N : \tilde{N}$.

Proof. The proof of *(iii)* is immediate by inspection of the two cases for N. Next the proof of *(ii)* is straightforward by structural induction in M and using *(iii)*. Finally the proof *(i)* is by a straightforward structural induction using *(ii)* and *(iii)*; in particular, the induction hypothesis still applies in the cases of restriction, input of capabilities and input of names, where the naming environment is updated. □

Lemma 1. The analysis is invariant under the congruence:
If $P \equiv Q$ then $(I, H, C, R) \models^l_{me} P$ if and only if $(I, H, C, R) \models^l_{me} Q$.

Proof. The proof is by induction on the proof of $P \equiv Q$ and relies on Fact 4. Most cases are straightforward and we just illustrate one of the more interesting ones.

Consider the case $(\nu\, n^\mu)(P \mid Q) \equiv P \mid (\nu\, n^\mu)Q$ where $n \notin \underline{\mathrm{fn}}(P)$. It is immediate from the clauses of Table 4 that

$$(I, H, C, R) \models^l_{me} (\nu\, n^\mu)(P \mid Q)$$

is equivalent to $(I, H, C, R) \models^l_{me[n \mapsto \mu]} P \mid Q$ and hence to

$$(I, H, C, R) \models^l_{me[n \mapsto \mu]} P \quad \wedge \quad (I, H, C, R) \models^l_{me[n \mapsto \mu]} Q.$$

Since $n \notin \underline{\mathrm{fn}}(P)$ it follows from Fact 4 that this is equivalent to

$$(I, H, C, R) \models^l_{me} P \quad \wedge \quad (I, H, C, R) \models^l_{me[n \mapsto \mu]} Q$$

and using the clauses of Table 4 this is equivalent to

$$(I, H, C, R) \models^l_{me} P \mid (\nu\, n^\mu)Q$$

as desired. □

In order to establish the Subject Reduction Result we shall additionally need the following central result showing how substitutions work:

Lemma 2. The analysis enjoys the following substitution properties:

(i) If $(I, H, C, R) \rhd_{me} M : R^c(\beta^c)$ and $(I, H, C, R) \models^l_{me[x \mapsto \beta^c]} P$ then $(I, H, C, R) \models^l_{me} P[x \leftarrow M]$.

(ii) If $(I, H, C, R) \Vert\!\!\equiv_{me} N : R^n(\beta^n)$ and $(I, H, C, R) \models^l_{me[u \mapsto \beta^n]} P$ then $(I, H, C, R) \models^l_{me} P[u \leftarrow N]$.

Proof. As a preparation for proving *(ii)* we first prove

if $(I,H,C,R) \models\!\!\!\equiv_{me} N : R^n(\beta^n)$ and $(I,H,C,R) \models\!\!\!\equiv_{me[u\mapsto\beta^n]} N' : \tilde{N}'$
then $(I,H,C,R) \models\!\!\!\equiv_{me} N'[u \leftarrow N] : \tilde{N}'$

by inspection of the two cases for N'. Next we prove

if $(I,H,C,R) \models\!\!\!\equiv_{me} N : R^n(\beta^n)$ and $(I,H,C,R) \rhd_{me[u\mapsto\beta^n]} M : \tilde{M}$
then $(I,H,C,R) \rhd_{me} M[u \leftarrow N] : \tilde{M}$

by structural induction in M and using the result just established; the proof is straightforward. We are then ready to prove *(ii)* by structural induction in P and using the two results just established; also this proof is straightforward.

As a preparation for proving *(i)* we first prove

if $(I,H,C,R) \rhd_{me} M : R^c(\beta^c)$ and $(I,H,C,R) \rhd_{me[x\mapsto\beta^c]} M' : \tilde{M}'$ then $(I,H,C,R) \rhd_{me} M'[x \leftarrow M] : \tilde{M}'$

by structural induction in M'; the proof is straightforward. We are then ready to prove *(i)* by structural induction in P and using the result just established; also this proof is straightforward. □

Theorem 1. Subject Reduction:

If $(I,H,C,R) \models^{l}_{me} P$ and $P \rightarrow^* Q$ then $(I,H,C,R) \models^{l}_{me} Q$.

Proof. The proof is by induction in the length of the derivation. In the induction step, $P \rightarrow^* S \rightarrow Q$, we have $(I,H,C,R) \models^{l}_{me} S$ from the induction hypothesis and need to show $(I,H,C,R) \models^{l}_{me} Q$.

We proceed by induction in the transition $S \rightarrow Q$. We first consider the reduction rules in the left-hand side of Table 3. The proof for the first three are straightforward using the induction hypothesis and the specification of Table 4; the proof for the fourth reduction rule, the one involving the congruence, is a direct consequence of the induction hypothesis and Lemma 1. We next turn our attention to the basic axioms in the right-hand side of Table 3. The proof for the in- and out-capabilities are straightforward using the specifications of Tables 4 and 5. In the case of the open-capability we additionally make use of Fact 3. Finally, in the case of communication we make use of Lemma 2. This concludes the proof. □

As a consequence, if (I,H,C,R) is an acceptable analysis estimate for the program $(me_\star, n_\star^{l_\star}[P_\star])$ of interest then it will continue being so for all the derivatives of the program.

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

To obtain these results we shall show that the set of acceptable analysis estimates constitutes a *Moore family* (sometimes called 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

$$\mathbf{InAmb} \times \mathbf{HNam} \times \mathbf{Comm} \times \mathbf{Env}$$

of tuples of mappings (I, H, C, R) and the ordering is the pointwise extension of the subset ordering. It follows that greatest lower bounds are calculated in a pointwise manner:

$$\sqcap_{j\in J}(I_j, H_j, C_j, R_j) = (\sqcap_{j\in J}I_j, \sqcap_{j\in J}H_j, \sqcap_{j\in J}C_j, \sqcap_{j\in J}R_j)$$

Since $I \sqsubseteq I'$ holds if and only if $I^{-1} \sqsubseteq I'^{-1}$ we also have $(\sqcap_{j\in J}I_j)^{-1} = \sqcap_{j\in J}(I_j^{-1})$. Finally, recall that $\sqcap_{j\in J}\cdots$ produces the greatest element $\top$ when J is the empty set.

We can now prove that the set of acceptable control flow estimates constitute a Moore family and hence that there always is a least estimate:

Theorem 2. The set $\{(I, H, C, R) \mid (I, H, C, R) \models^l_{me} P\}$ is a Moore family for all l, me and P.

Proof. The proof is in four parts. The first parts amounts to proving that

$$\{(I, H, C, R, \tilde{N}) \mid (I, H, C, R) \Vvdash_{me} N : \tilde{N}\}$$

is a Moore family for all me and N. This is immediate by inspection of the two cases for N.

The second part establishes that

$$\{(I, H, C, R) \mid (I, H, C, R) \models^l \tilde{m}\}$$

is a Moore family for all l and $\tilde{m}$. We proceed by inspection of the three cases for $\tilde{m}$. In the case of in-capabilities we assume that

$$\forall j \in J : (I_j, H_j, C_j, R_j) \models^l \mathsf{in}^{l^i} \tag{1}$$

and note that then $l^i \in I_j(l)$ for all $j \in J$ and hence $l^i \in (\sqcap_{j\in J}I_j)(l)$. Next consider l^a, μ, $l^{a'}$ and $l^{a''}$ such that $l^a \in (\sqcap_{j\in J}I_j)^{-1}(l^i)$, $\mu \in (\sqcap_{j\in J}H_j)(l^i)$, $l^{a'} \in (\sqcap_{j\in J}I_j)^{-1}(l^a)$, $l^{a''} \in (\sqcap_{j\in J}I_j)(l^{a'}) \cap (\sqcap_{j\in J}H_j)^{-1}(\mu) \cap \mathbf{Lab}^a$. It is immediate that we then have $l^a \in I_j^{-1}(l^i)$, $\mu \in H_j(l^i)$, $l^{a'} \in I_j^{-1}(l^a)$, $l^{a''} \in I_j(l^{a'}) \cap H_j^{-1}(\mu) \cap \mathbf{Lab}^a$ for all $j \in J$; it then follows from (1) that $l^a \in I_j(l^{a''})$ for all $j \in J$ and hence that $l^a \in (\sqcap_{j\in J}I_j)(l^{a''})$. In conclusion we then have

$$(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \models^l \mathsf{in}^{l^i}$$

as desired. The cases of out- and open-capabilities are similar.

The third part shows that

$$\{(I, H, C, R, \tilde{M}) \mid (I, H, C, R) \rhd_{me} M : \tilde{M}\}$$

is a Moore family for all M and me. The proof is by structural induction in M and let us consider the case $M_1.M_2$. Here we assume that

$$\forall j \in J : (I_j, H_j, C_j, R_j) \rhd_{me} M_1.M_2 : \tilde{M}_j$$

Hence there exist families $(\tilde{M}_{j1})_{j\in J}$ and $(\tilde{M}_{j2})_{j\in J}$ such that

$$\begin{aligned}
&\forall j \in J : \ (I_j, H_j, C_j, R_j) \rhd_{me} M_1 : \tilde{M}_{j1} \\
&\forall j \in J : \ (I_j, H_j, C_j, R_j) \rhd_{me} M_2 : \tilde{M}_{j2} \\
&\forall j \in J : \ \tilde{M}_j \supseteq \tilde{M}_{j1} \cup \tilde{M}_{j2}
\end{aligned}$$

By the induction hypothesis it follows that

$$\begin{aligned}
&(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \rhd_{me} M_1 : \cap_{j\in J}\tilde{M}_{j1} \\
&(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \rhd_{me} M_2 : \cap_{j\in J}\tilde{M}_{j2} \\
&(\cap_{j\in J}\tilde{M}_j) \supseteq (\cap_{j\in J}\tilde{M}_{j1}) \cup (\cap_{j\in J}\tilde{M}_{j2})
\end{aligned}$$

so that

$$(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \rhd_{me} M_1.M_2 : \cap_{j\in J}\tilde{M}_j$$

as desired. The remaining cases are similar.

Finally, the fourth part of the proof establishes the statement of the theorem. It is proved by structural induction in P and involves no new methods of reasoning beyond those already illustrated; we therefore only illustrate the case $N^{l^a}[P]$ of ambients. Here we assume that

$$\forall j \in J : (I_j, H_j, C_j, R_j) \models^{l}_{me} N^{l^a}[P]$$

It follows that there exists a family $(\tilde{N}_j)_{j\in J}$ such that

$$\begin{aligned}
&\forall j \in J : \ (I_j, H_j, C_j, R_j) \models^{l^a}_{me} P \\
&\forall j \in J : \ l^a \in I_j(l) \\
&\forall j \in J : \ (I_j, H_j, C_j, R_j) \Vert\!\equiv_{me} N : \tilde{N}_j \\
&\forall j \in J : \ \tilde{N}_j \subseteq H_j(l^a)
\end{aligned}$$

Using the induction hypothesis and the results established above we then have

$$\begin{aligned}
&(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \models^{l^a}_{me} P \\
&l^a \in (\sqcap_{j\in J} I_j)(l) \\
&(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \Vert\!\equiv_{me} N : \cap_{j\in J}\tilde{N}_j \\
&\cap_{j\in J}\tilde{N}_j \subseteq (\sqcap_{j\in J} H_j)(l^a)
\end{aligned}$$

and the desired

$$(\sqcap_{j\in J}(I_j, H_j, C_j, R_j)) \models^{l}_{me} N^{l^a}[P]$$

then follows. This concludes the proof. □

3.3 Algorithmic properties

We now show how to compute in polynomial time the least solutions guaranteed by Theorem 2. We shall proceed in three stages. In the first we generate a set of master constraints. In the second we expand the master constraints to a larger set of conditional constraints. Finally, we solve the conditional constraints in polynomial time. As mentioned earlier we may without loss of generality assume that **Lab** $\cup$ **Bnd** $\cup$ **SNam** is finite.

Master constraints. To generate the master constraints we use four functions. The auxiliary functions **VM** and **VN** of Table 7 extract the "succinct results" produced by the analysis of capabilities and namings; these are the analogues of the sets $\tilde{M}$ and $\tilde{N}$ and here take the forms $\{\overline{m}_1, \cdots, \overline{m}_k\}$ and $\{\overline{n}\}$, where

$$\begin{array}{l} \overline{m} ::= \{\!\!\{\mathsf{in}^{l^i}\}\!\!\} \mid \{\!\!\{\mathsf{out}^{l^o}\}\!\!\} \mid \{\!\!\{\mathsf{open}^{l^p}\}\!\!\} \mid \mathtt{Rc}(\beta^c) \\ \overline{n} ::= \{\!\!\{\mu\}\!\!\} \mid \mathtt{Rn}(\beta^n) \end{array}$$

and $\beta^c \in \mathbf{Bnd}^c$, $\mu \in \mathbf{SNam}$ and $\beta^n \in \mathbf{Bnd}^n$. To make it clear that we are now talking about *syntax* we write $\{\!\!\{\cdots\}\!\!\}$ instead of $\{\cdots\}$ and $\mathtt{Rn}$ instead of R^n etc. Intuitively, the minimal $\tilde{M}$ such that $(I, H, C, R) \rhd_{me} M : \tilde{M}$ is given by $\bigcup\{\overline{m} \mid \overline{m} \in \mathbf{VM}[\![M]\!]_{me}\}$ and the minimal $\tilde{N}$ such that $(I, H, C, R) \Vdash_{me} N : \tilde{N}$ is given by $\bigcup\{\overline{n} \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\}$; we shall be slightly more precise in Lemma 3 below. (There is no need for an analogous function **VP** for processes since they do not have a "succinct" component.)

The constraint generation functions **CP** and **CM** of Tables 6 and 7 generate master constraints for processes and capabilities, respectively. Master constraints take the following rather permissive forms:

$$\begin{array}{ll} \{\!\!\{l\}\!\!\} \subseteq \mathtt{I}(l^a) & \\ \{\!\!\{\beta\}\!\!\} \subseteq \mathtt{I}(l^a) & \\ \overline{n} \subseteq \mathtt{H}(l) & \\ \models^\bullet \mathsf{in}^l & \text{and similarly for } \mathsf{out}^l \text{ and } \mathsf{open}^l \\ \forall l \in \mathbf{Lab}^i : \{\!\!\{\mathsf{in}^l\}\!\!\} \subseteq \mathtt{Rc}(\beta^c) \Rightarrow \{\!\!\{l\}\!\!\} \subseteq \mathtt{I}(l^a) & \text{and similarly for } \mathsf{out}^l \text{ and } \mathsf{open}^l \\ \forall l \in \mathbf{Lab}^i : \{\!\!\{\mathsf{in}^l\}\!\!\} \subseteq \mathtt{Rc}(\beta^c) \Rightarrow \models^\bullet \mathsf{in}^l & \text{and similarly for } \mathsf{out}^l \text{ and } \mathsf{open}^l \\ \forall l \in \mathtt{I}^{-1}(l^c) : \overline{m} \subseteq \mathtt{Cc}(l) & \text{and similarly for } l^n, \overline{n} \text{ and } \mathtt{Cn} \\ \forall l \in \mathtt{I}^{-1}(\beta^c) : \mathtt{Cc}(l) \subseteq \mathtt{Rc}(\beta^c) & \text{and similarly for } \beta^n, \mathtt{Cn} \text{ and } \mathtt{Rn} \end{array}$$

The constraint $\models^\bullet \mathsf{in}^{l^i}$ intuitively stands for $\models^l \mathsf{in}^{l^i}$ of Table 5 but without the condition $l^i \in I(l)$, i.e.

$$\begin{array}{c} \forall l^a \in I^{-1}(l^i) : \forall \mu \in H(l^i) : \forall l^{a'} \in I^{-1}(l^a) : \\ \forall l^{a''} \in I(l^{a'}) \cap H^{-1}(\mu) \cap \mathbf{Lab}^a : \\ l^a \in I(l^{a''}) \end{array}$$

and similarly for $\models^\bullet \mathsf{out}^{l^o}$ and $\models^\bullet \mathsf{open}^{l^p}$. (There is no need for a constraint generation function **CN** for namings.)

We shall dispense with formally defining what it means for a control flow estimate (I, H, C, R) to satisfy a set of master constraints since the idea is clear: each constraint must be fulfilled when interpreting the formula with $\{\cdots\}$ for $\{\!\!\{\cdots\}\!\!\}$ and $R^n(\cdots)$ for $\mathtt{Rn}(\cdots)$ etc. We shall write $(I, H, C, R) \models \mathcal{C}$ when (I, H, C, R) satisfies a set $\mathcal{C}$ of constraints and $[\![\overline{m}]\!]_{(I,H,C,R)}$ for the value of $\overline{m}$ under the interpretation (I, H, C, R) and similarly for $[\![\overline{n}]\!]_{(I,H,C,R)}$. It is then straightforward to prove that the master constraints generated by Tables 6 and 7 faithfully model the acceptability relation of Tables 4 and 5.

Lemma 3. $(I, H, C, R) \models^l_{me} P$ if and only if $(I, H, C, R) \models \mathbf{CP}[\![P]\!]^l_{me}$.

Proof. The proof is by structural induction establishing also the following results:

$$(I, H, C, R) \Vdash_{me} N : \tilde{N} \ \text{ iff } \ \forall \overline{n} \in \mathbf{VN}[\![N]\!]_{me} : [\![\overline{n}]\!]_{(I,H,C,R)} \subseteq \tilde{N}$$

$$(I, H, C, R) \rhd_{me} M : \tilde{M} \ \text{ iff } \ \begin{array}{l} \forall \overline{m} \in \mathbf{VM}[\![M]\!]_{me} : [\![\overline{m}]\!]_{(I,H,C,R)} \subseteq \tilde{M} \wedge \\ (I, H, C, R) \models \mathbf{CM}[\![M]\!]_{me} \end{array}$$

We dispense with the details.

$$
\begin{aligned}
&\mathbf{CP}[\![(\nu\, n^{\mu})P]\!]^{l}_{me} &&= \mathbf{CP}[\![P]\!]^{l}_{me[n\mapsto\mu]}\\
&\mathbf{CP}[\![\mathbf{0}]\!]^{l}_{me} &&= \emptyset\\
&\mathbf{CP}[\![P \mid P']\!]^{l}_{me} &&= \mathbf{CP}[\![P]\!]^{l}_{me} \;\cup\; \mathbf{CP}[\![P']\!]^{l}_{me}\\
&\mathbf{CP}[\![!P]\!]^{l}_{me} &&= \mathbf{CP}[\![P]\!]^{l}_{me}\\
&\mathbf{CP}[\![N^{l^a}[P]]\!]^{l}_{me} &&= \mathbf{CP}[\![P]\!]^{l^a}_{me} \;\cup\; \{\{\!\!\{l^a\}\!\!\} \subseteq \mathtt{I}(l)\} \;\cup\\
& &&\quad \{\overline{n} \subseteq \mathtt{H}(l^a) \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\}\\
&\mathbf{CP}[\![M.\, P]\!]^{l}_{me} &&= \mathbf{CP}[\![P]\!]^{l}_{me} \;\cup\; \mathbf{CM}[\![M]\!]_{me} \;\cup\\
& &&\bigcup_{\overline{m}\in\mathbf{VM}[\![M]\!]_{me}} \left(\begin{array}{l}
\text{if } \overline{m} \text{ is } \{\!\!\{\mathsf{in}^{l^i}\}\!\!\} \text{ then } \{\models^{\bullet} \mathsf{in}^{l^i}, \{\!\!\{l^i\}\!\!\} \subseteq \mathtt{I}(l)\} \text{ else}\\
\text{if } \overline{m} \text{ is } \{\!\!\{\mathsf{out}^{l^o}\}\!\!\} \text{ then } \{\models^{\bullet} \mathsf{out}^{l^o}, \{\!\!\{l^o\}\!\!\} \subseteq \mathtt{I}(l)\} \text{ else}\\
\text{if } \overline{m} \text{ is } \{\!\!\{\mathsf{open}^{l^p}\}\!\!\} \text{ then } \{\models^{\bullet} \mathsf{open}^{l^p}, \{\!\!\{l^p\}\!\!\} \subseteq \mathtt{I}(l)\} \text{ else}\\
\left\{\begin{array}{l}
\forall l^i \in \mathbf{Lab}^i : \{\!\!\{\mathsf{in}^{l^i}\}\!\!\} \subseteq \overline{m} \Rightarrow \models^{\bullet} \mathsf{in}^{l^i},\\
\forall l^i \in \mathbf{Lab}^i : \{\!\!\{\mathsf{in}^{l^i}\}\!\!\} \subseteq \overline{m} \Rightarrow \{\!\!\{l^i\}\!\!\} \subseteq \mathtt{I}(l),\\
\forall l^o \in \mathbf{Lab}^o : \{\!\!\{\mathsf{out}^{l^o}\}\!\!\} \subseteq \overline{m} \Rightarrow \models^{\bullet} \mathsf{out}^{l^o},\\
\forall l^o \in \mathbf{Lab}^o : \{\!\!\{\mathsf{out}^{l^o}\}\!\!\} \subseteq \overline{m} \Rightarrow \{\!\!\{l^o\}\!\!\} \subseteq \mathtt{I}(l),\\
\forall l^p \in \mathbf{Lab}^p : \{\!\!\{\mathsf{open}^{l^p}\}\!\!\} \subseteq \overline{m} \Rightarrow \models^{\bullet} \mathsf{open}^{l^p},\\
\forall l^p \in \mathbf{Lab}^p : \{\!\!\{\mathsf{open}^{l^p}\}\!\!\} \subseteq \overline{m} \Rightarrow \{\!\!\{l^p\}\!\!\} \subseteq \mathtt{I}(l)
\end{array}\right\}
\end{array}\right)\\
&\mathbf{CP}[\![\langle M\rangle^{l^c}]\!]^{l}_{me} &&= \{\{\!\!\{l^c\}\!\!\} \subseteq \mathtt{I}(l)\} \;\cup\; \mathbf{CM}[\![M]\!]_{me} \;\cup\\
& &&\quad \{\forall l^a \in \mathtt{I}^{-1}(l^c) : \overline{m} \subseteq \mathtt{Cc}(l^a) \mid \overline{m} \in \mathbf{VM}[\![M]\!]_{me}\}\\
&\mathbf{CP}[\![\langle\!\langle N\rangle\!\rangle^{l^n}]\!]^{l}_{me} &&= \{\{\!\!\{l^n\}\!\!\} \subseteq \mathtt{I}(l)\} \;\cup\\
& &&\quad \{\forall l^a \in \mathtt{I}^{-1}(l^n) : \overline{n} \subseteq \mathtt{Cn}(l^a) \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\}\\
&\mathbf{CP}[\![(x^{\beta^c}).\, P]\!]^{l}_{me} &&= \mathbf{CP}[\![P]\!]^{l}_{me[x\mapsto\beta^c]} \;\cup\; \{\{\!\!\{\beta^c\}\!\!\} \subseteq \mathtt{I}(l)\} \;\cup\\
& &&\quad \{\forall l^a \in \mathtt{I}^{-1}(\beta^c) : \mathtt{Cc}(l^a) \subseteq \mathtt{Rc}(\beta^c)\}\\
&\mathbf{CP}[\![(\!(u^{\beta^n})\!).\, P]\!]^{l}_{me} &&= \mathbf{CP}[\![P]\!]^{l}_{me[u\mapsto\beta^n]} \;\cup\; \{\{\!\!\{\beta^n\}\!\!\} \subseteq \mathtt{I}(l)\} \;\cup\\
& &&\quad \{\forall l^a \in \mathtt{I}^{-1}(\beta^n) : \mathtt{Cn}(l^a) \subseteq \mathtt{Rn}(\beta^n)\}
\end{aligned}
$$

Table 6. Constraint generation (for processes).

Example 5. The set $\mathbf{CP}[\![\mathit{Firewall} \mid \mathit{Agent}]\!]^{l_\star}_{me_\star}$ of master contraints for the program $n_\star^{l_\star}[\mathit{Firewall} \mid \mathit{Agent}]$ of Example 1 consists of the following master constraints (assuming that $me_\star$ is an in Example 3 and ignoring the subprocesses P and Q):

$$
\begin{array}{lll}
\{\!\!\{\mathrm{A}\}\!\!\} \subseteq \mathtt{I}(l_\star) & & \{\!\!\{w\}\!\!\} \subseteq \mathtt{H}(\mathrm{A})\\
\{\!\!\{\mathrm{B}\}\!\!\} \subseteq \mathtt{I}(\mathrm{A}) & & \{\!\!\{k\}\!\!\} \subseteq \mathtt{H}(\mathrm{B})\\[4pt]
\{\!\!\{1\}\!\!\} \subseteq \mathtt{I}(\mathrm{B}) & \models^{\bullet} \mathsf{out}^1 & \{\!\!\{w\}\!\!\} \subseteq \mathtt{H}(1)\\
\{\!\!\{2\}\!\!\} \subseteq \mathtt{I}(\mathrm{B}) & \models^{\bullet} \mathsf{in}^2 & \{\!\!\{k'\}\!\!\} \subseteq \mathtt{H}(2)\\
\{\!\!\{3\}\!\!\} \subseteq \mathtt{I}(\mathrm{B}) & \models^{\bullet} \mathsf{in}^3 & \{\!\!\{w\}\!\!\} \subseteq \mathtt{H}(3)\\[4pt]
\{\!\!\{4\}\!\!\} \subseteq \mathtt{I}(\mathrm{A}) & \models^{\bullet} \mathsf{open}^4 & \{\!\!\{k'\}\!\!\} \subseteq \mathtt{H}(4)\\
\{\!\!\{5\}\!\!\} \subseteq \mathtt{I}(\mathrm{A}) & \models^{\bullet} \mathsf{open}^5 & \{\!\!\{k''\}\!\!\} \subseteq \mathtt{H}(5)\\[4pt]
\{\!\!\{\mathrm{C}\}\!\!\} \subseteq \mathtt{I}(l_\star) & & \{\!\!\{k'\}\!\!\} \subseteq \mathtt{H}(\mathrm{C})\\
\{\!\!\{6\}\!\!\} \subseteq \mathtt{I}(\mathrm{C}) & \models^{\bullet} \mathsf{open}^6 & \{\!\!\{k\}\!\!\} \subseteq \mathtt{H}(6)\\
\{\!\!\{\mathrm{D}\}\!\!\} \subseteq \mathtt{I}(\mathrm{C}) & & \{\!\!\{k''\}\!\!\} \subseteq \mathtt{H}(\mathrm{D})
\end{array}
$$

$$
\begin{array}{ll}
\mathbf{CM}[\![\mathsf{in}^{l^i} N]\!]_{me} & = \{\overline{n} \subseteq \mathtt{H}(l^i) \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\} \\
\mathbf{CM}[\![\mathsf{out}^{l^o} N]\!]_{me} & = \{\overline{n} \subseteq \mathtt{H}(l^o) \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\} \\
\mathbf{CM}[\![\mathsf{open}^{l^p} N]\!]_{me} & = \{\overline{n} \subseteq \mathtt{H}(l^p) \mid \overline{n} \in \mathbf{VN}[\![N]\!]_{me}\} \\
\mathbf{CM}[\![x]\!]_{me} & = \emptyset \\
\mathbf{CM}[\![\epsilon]\!]_{me} & = \emptyset \\
\mathbf{CM}[\![M_1.M_2]\!]_{me} & = \mathbf{CM}[\![M_1]\!]_{me} \ \cup \ \mathbf{CM}[\![M_2]\!]_{me} \\
\hline
\mathbf{VM}[\![\mathsf{in}^{l^i} N]\!]_{me} & = \{\{\!\!\{\mathsf{in}^{l^i}\}\!\!\}\} \\
\mathbf{VM}[\![\mathsf{out}^{l^o} N]\!]_{me} & = \{\{\!\!\{\mathsf{out}^{l^o}\}\!\!\}\} \\
\mathbf{VM}[\![\mathsf{open}^{l^p} N]\!]_{me} & = \{\{\!\!\{\mathsf{open}^{l^p}\}\!\!\}\} \\
\mathbf{VM}[\![x]\!]_{me} & = \{\mathtt{Rc}(me(x))\} \\
\mathbf{VM}[\![\epsilon]\!]_{me} & = \emptyset \\
\mathbf{VM}[\![M_1.M_2]\!]_{me} & = \mathbf{VM}[\![M_1]\!]_{me} \ \cup \ \mathbf{VM}[\![M_2]\!]_{me} \\
\hline
\mathbf{VN}[\![n]\!]_{me} & = \{\{\!\!\{me(n)\}\!\!\}\} \\
\mathbf{VN}[\![u]\!]_{me} & = \{\mathtt{Rn}(me(u))\}
\end{array}
$$

Table 7. Constraint generation (for capabilities and namings).

It is straightforward to check that the analysis estimate (I, H, C, R) of Example 3 indeed satisfies these master constraints. □

Suppose that the program is of size $p \geq 1$ and that the size of the finite set $\mathbf{Lab} \cup \mathbf{Bnd} \cup \mathbf{SNam}$ is $q \geq 1$. It is natural to assume that $q = O(p)$ as when $\mathbf{Lab} \cup \mathbf{Bnd} \cup \mathbf{SNam}$ is a finite set consisting only of the entities used in the program; however, we shall allow to let q be less than p so as to trade precision for efficiency.

Note that $\mathbf{VN}[\![N]\!]_{me}$ is always a singleton and that $\mathbf{CM}[\![M]\!]_{me}$ and $\mathbf{VM}[\![M]\!]_{me}$ contain no more elements than corresponds to the size of M. It is then immediate that constraint generation operates in time $O(p)$ and that it produces $O(p)$ master constraints each of size $O(1)$. — Since q may be much smaller than p it will be useful to observe that the number of constraints can also be given as $O(q^2)$; to see this simply note that each master constraint contains at most two "free" symbols from $\mathbf{Lab} \cup \mathbf{Bnd} \cup \mathbf{SNam}$. In this case constraint generation time should be estimated as $O(p + q^2)$.

Conditional constraints. The next stage is to expand the master constraints into sets of conditional constraints that do not involve quantifiers and that do not rely on the $\models^{\bullet} \cdots$ abbreviations adapted from Table 5. The general syntax of conditional constraints is based on constants (denoted *set*), variables (denoted *var*), conditions (denoted *cond*) and constraints (denoted *constr*):

$$
\begin{aligned}
\mathit{set} &::= \{\!\!\{l\}\!\!\} \mid \{\!\!\{\beta\}\!\!\} \mid \{\!\!\{\mu\}\!\!\} \mid \{\!\!\{\mathsf{in}^l\}\!\!\} \mid \{\!\!\{\mathsf{out}^l\}\!\!\} \mid \{\!\!\{\mathsf{open}^l\}\!\!\} \\
\mathit{var} &::= \mathtt{I}(l) \mid \mathtt{H}(l) \mid \mathtt{Cc}(l) \mid \mathtt{Cn}(l) \mid \mathtt{Rc}(\beta) \mid \mathtt{Rn}(\beta) \mid \cdots \\
\mathit{cond} &::= \mathit{set} \subseteq \mathit{var} \\
\mathit{constr} &::= \mathit{cond} \mid \mathit{cond} \Rightarrow \mathit{constr}
\end{aligned}
$$

To transform master constraints into conditional constraints we perform the following operations:

- expand $var_1 \subseteq var_2$ into $\forall set : set \subseteq var_1 \Rightarrow set \subseteq var_2$;
- unfold the definition of $\models^{\bullet} \cdots$;
- move quantifiers outermost;
- eliminate quantifiers by instantiating the bodies with all possible labels;
- eliminate all "-1" operations by changing $\{\!\{x\}\!\} \subseteq I^{-1}(y)$ to $\{\!\{y\}\!\} \subseteq I(x)$.

We illustrate the development for the following master constraint:

$$\forall l^i \in \mathbf{Lab}^i : \{\!\{\mathsf{in}^{l^i}\}\!\} \subseteq \mathtt{Rc}(\beta^c) \Rightarrow \models^{\bullet} \mathsf{in}^{l^i} \tag{2}$$

Straightforward application of the above operations gives rise to the following set of conditional constraints:

$$\left\{ \begin{array}{c|c} (\{\!\{\mathsf{in}^{l^i}\}\!\} \subseteq \mathtt{Rc}(\beta^c)) \Rightarrow & \\ (\{\!\{l^i\}\!\} \subseteq \mathtt{I}(l^a)) \quad \Rightarrow & l^i \in \mathbf{Lab}^i, \\ (\{\!\{\mu\}\!\} \subseteq \mathtt{H}(l^i)) \quad \Rightarrow & l^a \in \mathbf{Lab}^a, \\ (\{\!\{l^a\}\!\} \subseteq \mathtt{I}(l^{a'})) \quad \Rightarrow & \mu \in \mathbf{SNam}, \\ (\{\!\{l^{a''}\}\!\} \subseteq \mathtt{I}(l^{a'})) \quad \Rightarrow & l^{a'} \in \mathbf{Lab}^a, \\ (\{\!\{\mu\}\!\} \subseteq \mathtt{H}(l^{a''})) \Rightarrow & l^{a''} \in \mathbf{Lab}^a \\ \{\!\{l^a\}\!\} \subseteq \mathtt{I}(l^{a''}) & \end{array} \right\}$$

It turns out that this suffices for obtaining a polynomial time algorithm for computing the least solution; however, in the interest of obtaining a qubic time algorithm we shall "tile" the conditional constraints in the manner of [20]. To do so we introduce three auxiliary relations:

- $\{\!\{l^i\}\!\} \subseteq \mathit{IN}(\diamond)$ meaning that in^{l^i} has been found to be "active";
- $\{\!\{l^{a''}\}\!\} \subseteq \mathit{NAM}(l^i)$ meaning that l^i and $l^{a''}$ have the same name;
- $\{\!\{l^a\}\!\} \subseteq \mathit{SIB}(l^{a''})$ meaning that l^a and $l^{a''}$ are siblings.

(As will be shown below, their values are all derived from the estimate (I, H, C, R) under consideration.)

The master constraint (2) then gives rise to a set of "local" constraints and three sets of "global" constraints that are shared for all values of β^c occurring in (2). The set of "local" constraints, generated for each occurrence of (2), computes the IN relation from (I, H, C, R):

$$\left\{ (\{\!\{\mathsf{in}^{l^i}\}\!\} \subseteq \mathtt{Rc}(\beta^c)) \Rightarrow (\{\!\{l^i\}\!\} \subseteq \mathit{IN}(\diamond)) \,\middle|\, l^i \in \mathbf{Lab}^i \right\}$$

One of the sets of "global" constraints, shared for all occurrences of (2), computes the NAM relation from (I, H, C, R):

$$\left\{ \begin{array}{c|c} (\{\!\{\mu\}\!\} \subseteq \mathtt{H}(l^i)) \quad \Rightarrow & l^i \in \mathbf{Lab}^i, \\ (\{\!\{\mu\}\!\} \subseteq \mathtt{H}(l^{a''})) \quad \Rightarrow & \mu \in \mathbf{SNam}, \\ \{\!\{l^{a''}\}\!\} \subseteq \mathit{NAM}(l^i) & l^{a''} \in \mathbf{Lab}^a \end{array} \right\}$$

The other set of global constraints, shared for all occurrences of (2), computes the SIB relation from (I, H, C, R):

$$\left\{ \begin{array}{c|c} (\{\!\{l^a\}\!\} \subseteq \mathtt{I}(l^{a'})) \quad \Rightarrow & l^a \in \mathbf{Lab}^a, \\ (\{\!\{l^{a''}\}\!\} \subseteq \mathtt{I}(l^{a'})) \quad \Rightarrow & l^{a'} \in \mathbf{Lab}^a, \\ \{\!\{l^a\}\!\} \subseteq \mathit{SIB}(l^{a''}) & l^{a''} \in \mathbf{Lab}^a \end{array} \right\}$$

Type of Master Constraint	#Instances	#"Local"	#"Global"
$\{\!\{l\}\!\} \subseteq \mathtt{I}(l^a)$	$O(p)\,\&\,O(q^2)$	$O(1)$	$O(1)$
$\{\!\{\beta\}\!\} \subseteq \mathtt{I}(l^a)$	$O(p)\,\&\,O(q^2)$	$O(1)$	$O(1)$
$\overline{n} \subseteq \mathtt{H}(l)$	$O(p)\,\&\,O(q^2)$	$O(q)$	$O(1)$
$\models^\bullet \mathsf{in}^l$ etc.	$O(p)\,\&\,O(q)$	$O(1)$	$O(q^3)$
$\forall l \in \mathbf{Lab}^i : \{\!\{\mathsf{in}^l\}\!\} \subseteq \mathtt{Rc}(\beta^c) \Rightarrow \{\!\{l\}\!\} \subseteq \mathtt{I}(l^a)$ etc.	$O(p)\,\&\,O(q^2)$	$O(q)$	$O(1)$
$\forall l \in \mathbf{Lab}^i : \{\!\{\mathsf{in}^l\}\!\} \subseteq \mathtt{Rc}(\beta^c) \Rightarrow \models^\bullet \mathsf{in}^l$ etc.	$O(p)\,\&\,O(q)$	$O(q)$	$O(q^3)$
$\forall l \in \mathtt{I}^{-1}(l^c) : \overline{m} \subseteq \mathtt{Cc}(l)$ etc.	$O(p)\,\&\,O(q^2)$	$O(q)$	$O(q^3)$
$\forall l \in \mathtt{I}^{-1}(\beta^c) : \mathtt{Cc}(l) \subseteq \mathtt{Rc}(\beta^c)$ etc.	$O(p)\,\&\,O(q)$	$O(q^2)$	$O(1)$

Table 8. From master constraints to conditional constraints.

The final set of global constraints, shared for all occurrences of (2), performs the update of I:

$$\left\{ \begin{array}{rl|c} (\{\!\{l^i\}\!\} \subseteq \mathit{IN}(\diamond)) & \Rightarrow & \\ (\{\!\{l^i\}\!\} \subseteq \mathtt{I}(l^a)) & \Rightarrow & l^i \in \mathbf{Lab}^i, \\ (\{\!\{l^a\}\!\} \subseteq \mathit{SIB}(l^{a''})) & \Rightarrow & l^a \in \mathbf{Lab}^a, \\ (\{\!\{l^{a''}\}\!\} \subseteq \mathit{NAM}(l^i)) & \Rightarrow & l^{a''} \in \mathbf{Lab}^a \\ \{\!\{l^a\}\!\} \subseteq \mathtt{I}(l^{a''}) & & \end{array} \right\}$$

We proceed in a similar way for the other master constraints and write $\mathbf{C}[\![P]\!]^l_{me}$ for the set of conditional constraints obtained by expanding $\mathbf{CP}[\![P]\!]^l_{me}$. Once more we dispense with formally defining what it means for a control flow estimate (I, H, C, R) to satisfy a set of conditional constraints since the idea is clear: the auxiliary relations (IN, NAM, etc.) should be given as the least relations satisfying their defining constraints. In analogy with Lemma 3 we then have the following result:

Lemma 4. $(I, H, C, R) \models \mathbf{C}[\![P]\!]^l_{me}$ if and only if $(I, H, C, R) \models^l_{me} P$.

Table 8 shows for each form of master constraint how may instances are generated, how many "local" conditional constraints are generated by expanding each occurrence of a master constraint, and how many "global" conditional constraints are shared for all master constraints of the form considered. The number of master constraints is generally written as $O(p)\&O(q^2)$ to indicate that both bounds $O(p)$ and $O(q^2)$ apply as discussed previously; also note that only $O(p)\&O(q)$ master constraints involve $\models^\bullet \cdots$ since the relevant master constraints only contain one "free" symbol from $\mathbf{Lab} \cup \mathbf{Bnd} \cup \mathbf{SNam}$ and similarly for two other types of master constraints. In the example treated in detail, $O(q)$ "local" conditional constraints are produced for each occurrence of a master constraint and $O(q^3)$ "global" conditional constraints are shared; without the use of "tiling" [20] we would have generated $O(q^5)$ constraints for each occurrence of a master constraint.

Since $q = O(p)$ it follows that constraint generation operates in time $O(p^3)$ producing $O(p^3)$ conditional constraints of size $O(1)$. — It is useful to observe that constraint generation can also be estimated as $O(p + q^3)$ steps for generating $O(q^3)$ conditional constraints of size $O(1)$.

Constraint solving. We now consider how to turn $\mathbf{C}[\![P]\!]^l_{me}$ into the smallest acceptable analysis estimate (I, H, C, R). Perhaps the simplest approach is to use a Round Robin algorithm (see e.g. [17, Chapter 6]). A more efficient approach is based on worklist algorithms (see e.g. [17, Chapter 6]) and makes sure to consider constraints only when they are likely to have been enabled due to recent changes. This is the key idea behind the approach of [11] and allows us to solve the conditional constraints in $\mathbf{C}[\![P]\!]^l_{me}$ in time proportional to their size.

Theorem 3 (Theorem 2 of [20]). The least control flow estimate,

$$\sqcap\{(I,H,C,R) \mid (I,H,C,R) \models^l_{me} P\},$$

can be computed in cubic time.

Proof. Thanks to Lemmas 3 and 4 we have

$$\sqcap\{(I,H,C,R) \mid (I,H,C,R) \models^l_{me} P\} = \sqcap\{(I,H,C,R) \mid (I,H,C,R) \models \mathbf{C}[\![P]\!]^l_{me}\}$$

and we already established that $\mathbf{C}[\![P]\!]^l_{me}$ has size $O(p^3)\&O(q^3)$ and can be generated in time $O(p^3)\&O(p+q^3)$. Thanks to the techniques of [11] the least solution can be found in time proportional to the size of the constraint system.

A constructive algorithm for computing the least solution can be found in Appendix A. Theorem 3 is attributed to [20] that developed "tiling" to establish a similar result for the communication-free fragment of the ambient calculus; no new complications were encountered when dealing with communication. □

The cubic time bound is satisfying since this is the complexity of control flow analyses also for functional and object oriented languages where control flow analysis is used with success also on "medium-sized" programs (between 10K and 100K lines of code). Furthermore, since q can be chosen arbitrarily small (although at the cost of precision) this gives confidence in exploring our current technology on fully realistic internet programs.

4 Validating Firewalls

In the examples we have studied a notion of firewall given by the passwords $\mathtt{k}$, $\mathtt{k}'$ and $\mathtt{k}''$ 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: $\mathit{Firewall} \mid \mathit{Agent} \to^* (\nu\,\mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[P \mid Q]$ (assuming that $\mathtt{w} \notin \underline{\mathrm{fn}}(Q)$). It is shown in [7] that 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 $\underline{\mathrm{fn}}(P) \cap \{\mathtt{k},\mathtt{k}',\mathtt{k}''\} = \emptyset = \underline{\mathrm{fn}}(Q) \cap \{\mathtt{w},\mathtt{k},\mathtt{k}',\mathtt{k}''\}$).

Another aspect of being a firewall, not dealt with in [7], 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 6 below).

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

Example 6. 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

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

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

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

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

In the development below we focus on one particular interface for firewalls, as given by the three passwords and formats shown above, but the development can be adapted to other interfaces as well. With respect to the chosen interface we then aim at developing a safe test for when a proposed realisation, e.g. *Firewall*$'$ or *Firewall*, can be proved to be protective (i.e. to live up to the expectations).

We proceed by devising a test based on the control flow analysis; to facilitate this we need to formalise the notions of "proposed firewall" and "unaware process" (or unaware attacker) in the terminology of the control flow analysis. In essense this amounts to shifting our attention from names to stable names.

A *proposed firewall* is specified by a tuple of the form

$$(me_\star, ((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]), k, k', k'')$$

such that $(me_\star, n_\star^{l_\star}[(\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]])$ is a program. As in the examples we assume that there are three passwords but the development can easily be adapted to an arbitrary selection of passwords.

To enable the proposed firewall to pass the test to be developed, it will be helpful to arrange that the naming environment respects the privacy of the name of the firewall, i.e. $me_\star^{-1}(w) = \emptyset$, that the naming environment respects the uniqueness of the passwords, i.e. $me_\star^{-1}(k) = \{\mathtt{k}\}$, $me_\star^{-1}(k') = \{\mathtt{k}'\}$ and $me_\star^{-1}(k'') = \{\mathtt{k}''\}$, and that the process F does not contain any of the stable names w, k, k' or k'', although this is not formally required.

For technical reasons we shall arrange that the the proposed firewall does not contain any distinguished symbols (in particular those marked "$\bullet$") and that the naming environment does not map any names to the distinguished stable name $\mu_\bullet$; this can always be achieved by adjusting the choice of distinguished symbols and by Fact 2 this has no semantic consequences.

An *unaware attacker* (relative to the form of proposed firewall considered here) is a process U such that

$(me_\star, n_\star^{l_\star}[U])$ is a program, and

no free name in U is mapped to any of the "private" stable names, i.e. $\{me_\star(n) \mid n \in \underline{\mathrm{fn}}(U)\} \cap \{w, k, k', k''\} = \emptyset$.

When the above recommendations are adhered to, the second condition follows from the assumption that none of the passwords $\mathtt{k}$, $\mathtt{k}'$ or $\mathtt{k}''$ are free in U. Note that the first and second condition together establish the following stronger version of the second condition: $\{me_\star(n) \mid n \in \underline{\mathrm{fn}}(U)\} \cap \{\mu_\star, w, k, k', k''\} = \emptyset$; we shall exploit this fact in the proof of Proposition 1.

INPUT:	a proposed firewall $(me_\star, ((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]), k, k', k'')$ without distinguished symbols in $\mathtt{w}^{\mathrm{A}}[F]$
OUTPUT:	"accept" or "reject"
METHOD:	let $\{n_1, \cdots, n_m\} = \{n \in dom(me_\star) \mid me_\star(n) \notin \{\mu_\star, w, k, k', k''\}\}$
	let $n \notin dom(me_\star)$ and $x \in \mathbf{Var}^c$ and $u \in \mathbf{Var}^n$
	construct $U_\star^\bullet = \begin{pmatrix} \langle \mathsf{in}^{l^i_\bullet} n\rangle^{l^c_\bullet} \mid \langle \mathsf{out}^{l^o_\bullet} n\rangle^{l^c_\bullet} \mid \langle \mathsf{open}^{l^p_\bullet} n\rangle^{l^c_\bullet} \mid \\ (x^{\beta^c_\bullet}).\,(\langle x\rangle^{l^c_\bullet} \mid x.\,\mathbf{0}) \mid \\ \langle\!\langle n\rangle\!\rangle^{l^n_\bullet} \mid \langle\!\langle n_1\rangle\!\rangle^{l^n_\bullet} \mid \cdots \mid \langle\!\langle n_m\rangle\!\rangle^{l^n_\bullet} \mid \\ (\!(u^{\beta^n_\bullet})\!).\,(\langle\!\langle u\rangle\!\rangle^{l^n_\bullet} \mid u^{l^a_\bullet}[\mathbf{0}] \mid \mathsf{in}^{l^i_\bullet} u \mid \mathsf{out}^{l^o_\bullet} u \mid \mathsf{open}^{l^p_\bullet} u) \end{pmatrix}$
	construct $U_\star = !(\nu n^{\mu_\bullet})(U_\star^\bullet \mid n^{l^a_\bullet}[U_\star^\bullet])$
	compute the least (I, H, C, R) such that
	$(I, H, C, R) \models^{l_\star}_{me_\star} ((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid U_\star$
	if $\exists\, l^a \in \mathbf{Lab}^a : l^a_\bullet \in I^+(l^a) \wedge w \in H(l^a)$
	then "reject"
	else "accept"

Table 9. Testing for protectiveness.

To develop a sound test for validating the protectiveness of a proposed firewall (see Table 9) we proceed in two stages. First recall that we defined a proposed firewall to be protective whenever the semantics prevents any unaware process from entering. The first stage of the development then is to define a related notion where the control flow analysis plays the role of the semantics: a firewall is *strongly protective* whenever the control flow analysis is able to demonstrate that no unaware process can enter the firewall. It follows from the correctness of the control flow analysis (Theorem 1) that a strongly protective firewall is also protective. Since the control flow analysis is approximate it would be unlikely for the converse result to hold; however, we shall see that *Firewall* is both protective and strongly protective whereas *Firewall*$'$ is neither.

This then leads to the following interim suggestion for testing the strong protectiveness of a proposed firewall $(me_\star, ((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]), k, k', k'')$:

if there exists an unaware attacker U such that
    for the least (I, H, C, R) satisfying $(I, H, C, R) \models^{l_\star}_{me_\star} ((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid U$
    there exists $l^a_U \in \mathbf{Lab}^a$ labelling an entity in U and $l^a \in \mathbf{Lab}^a$
        such that $l^a_U \in I^+(l^a) \wedge w \in H(l^a)$
then "reject" else "accept"

While this test is sound it involves a search over an infinity of unaware attackers and so is not readily implementable. The second stage of the development therefore is to restrict the search to a finite set of unaware attackers that are as hard to protect against as any other unaware attackers; in our case a single unaware attacker will do and it is called the *hardest attacker*. It amounts to the process $U_\star$ of Table 9. Clearly the hardest attacker should have access to all stable names except those of $\mu_\star, w, k, k', k''$; indeed, we ensure that only the stable name $\mu_\bullet$ is introduced internally. In a similar way we ensure that only the binders $\beta^c_\bullet$ and $\beta^n_\bullet$ and only the distinguished labels $l^a_\bullet$, $l^i_\bullet$, $l^o_\bullet$, $l^p_\bullet$, $l^c_\bullet$ and $l^n_\bullet$ are used internally. Over this universe of names, binders and labels, the hardest attacker must be able to perform all outputs of names, to create ambients and capabilities with the required names, to output and input all kinds of capabilities and to enact all relevant capabilities.

It is hardly immediate that $U_\star$ of Table 9 lives up to these expectations. However, we are developing a sound test for strong protectiveness and so can restrict our attention to the level of granularity embodied in the control flow analysis. This allows us to prove in Proposition 1 below that the definition of $U_\star$ is sufficiently general to capture the behaviour of all unaware attackers — as far as the control flow analysis is concerned.

Example 7. To test *Firewall* from Example 1 and *Firewall′* from Example 6 we need to be more precise about the subprocess F; 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 $l^a_\bullet \notin I^+(\mathrm{A})$ but *Firewall′* fails the test because $H^{-1}(w) = \{\mathrm{A}\}$ and $l^a_\bullet \in I^+(\mathrm{A})$. □

As explained above 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 against the process $U_\star$ of Table 9 as it is to protect it against any other unaware process U. The formulation uses the operation $\lfloor U \rfloor$ also used to express Fact 2: all stable names, binders and labels are replaced by the appropriate distinguished stable names, binders and labels.

Proposition 1. Let $(me_\star, ((\nu\,\mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]), k, k', k'')$ be a proposed firewall as demanded in Table 9 and let (I, H, C, R) be as in Table 9. Then

$$(I, H, C, R) \models^{l_\star}_{me_\star} ((\nu\,\mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid \lfloor U \rfloor$$

whenever U is an unaware attacker.

Proof. The proof is in two parts. The first draws a number of consequences from the fact that $(I, H, C, R) \models^{l_\star}_{me_\star} U_\star$ and the second proves a number of "general" results from which $(I, H, C, R) \models^{l_\star}_{me_\star} \lfloor U \rfloor$ immediately follows.

Part 1. For the first part we unfold the formula defining $(I, H, C, R) \models^{l_\star}_{me_\star} U_\star$. Since $U^\bullet_\star$ appears outermost as well as inside $n^{l^a_\bullet}[\cdots]$ we get:

$$\begin{aligned} I(l_\star) &\supseteq \{l^a_\bullet, l^i_\bullet, l^o_\bullet, l^p_\bullet, l^c_\bullet, l^n_\bullet, \beta^c_\bullet, \beta^n_\bullet\} \\ I(l^a_\bullet) &\supseteq \{l^a_\bullet, l^i_\bullet, l^o_\bullet, l^p_\bullet, l^c_\bullet, l^n_\bullet, \beta^c_\bullet, \beta^n_\bullet\} \end{aligned} \tag{3}$$

In the first two lines of $U^\bullet_\star$ we output all three kinds of capabilities and then input them again; similarly in the last two lines we output all the stable names $\mu_\bullet, me_\star(n_1), \cdots, me_\star(n_m)$ accessible to $U^\bullet_\star$ and then input them again; this yields:

$$\begin{aligned} R^c(\beta^c_\bullet) &\supseteq \{\mathsf{in}^{l^i_\bullet}, \mathsf{out}^{l^o_\bullet}, \mathsf{open}^{l^p_\bullet}\} \\ R^n(\beta^n_\bullet) &\supseteq \{\mu_\bullet, me_\star(n_1), \cdots, me_\star(n_m)\} \end{aligned} \tag{4}$$

More generally, the fact that we input what has been output, ensures that the clauses for input of Table 4 establish the following containments:

$$\begin{aligned} \forall l^a \in I^{-1}(\beta^c_\bullet) &: C^c(l^a) \subseteq R^c(\beta^c_\bullet) \\ \forall l^a \in I^{-1}(\beta^n_\bullet) &: C^n(l^a) \subseteq R^n(\beta^n_\bullet) \end{aligned} \tag{5}$$

In a similar way we also make sure to output what has just been input and the clauses for output of Table 4 then establish the “dual” containments:

$$\begin{array}{l} \forall l^a \in I^{-1}(l^c_\bullet) : C^c(l^a) \supseteq R^c(\beta^c_\bullet) \\ \forall l^a \in I^{-1}(l^n_\bullet) : C^n(l^a) \supseteq R^n(\beta^n_\bullet) \end{array} \tag{6}$$

In the second line of $U^\bullet_\star$ we perform the capability just input; since this takes place both outermost and inside $n^{l^a_\bullet}[\cdots]$, the clause for capabilities gives:

$$\forall \tilde{m} \in R^c(\beta^c_\bullet) : \forall l \in \{l_\star, l^a_\bullet\} : (I, H, C, R) \models^l \tilde{m} \tag{7}$$

since the $\tilde{M}$ used for validating $x.\,\mathbf{0}$ must satisfy $\tilde{M} \supseteq R^c(\beta^c_\bullet)$. Similarly, in the fourth line of $U^\bullet_\star$ we construct ambients and capabilities with the names just input; this yields:

$$\begin{array}{l} R^n(\beta^n_\bullet) \subseteq H(l^a_\bullet) \\ R^n(\beta^n_\bullet) \subseteq H(l^i_\bullet) \\ R^n(\beta^n_\bullet) \subseteq H(l^o_\bullet) \\ R^n(\beta^n_\bullet) \subseteq H(l^p_\bullet) \end{array} \tag{8}$$

since the $\tilde{N}$ used for validating $u^{l^a_\bullet}[\mathbf{0}]$ must satisfy $\tilde{N} \supseteq R^n(\beta^n_\bullet)$ and similarly for the three capabilities.

Part 2. We now consider the second part of the proof. We shall say that a naming environment me is acceptable for the process P whenever it defines all names and variables in P, i.e. $\underline{\mathrm{fv}}(P) \cup \underline{\mathrm{fn}}(P) \subseteq \mathit{dom}(me)$, and whenever it only maps variables and names to “acceptable” symbols:

$$\mathit{range}(me) \subseteq \{\mu_\bullet, me_\star(n_1), \cdots, me_\star(n_m), \beta^c_\bullet, \beta^n_\bullet\}.$$

In a similar way we define the acceptability of me for a capability M and a naming N.

First we prove for all me and N that

$$\begin{array}{l} (I, H, C, R) \Vdash_{me} N : R^n(\beta^n_\bullet) \\ \text{provided } me \text{ is acceptable for } N \end{array} \tag{9}$$

where (I, H, C, R) is as above. The proof is by inspection of the two cases for N. The case n is immediate given (4) and the acceptability of me. The case u is trivial.

Next we prove for all me and M that

$$\begin{array}{l} (I, H, C, R) \rhd_{me} \lfloor M \rfloor : R^c(\beta^c_\bullet) \\ \text{provided } me \text{ is acceptable for } M \end{array} \tag{10}$$

where again (I, H, C, R) is as above. The proof is by structural induction in $\lfloor M \rfloor$. In the cases $\mathsf{in}^{l^i_\bullet} N$, $\mathsf{out}^{l^o_\bullet} N$ and $\mathsf{open}^{l^p_\bullet} N$ we take $\tilde{N} = R^n(\beta^n_\bullet)$; the result is then immediate from (9), (4) and (8). The cases x and ϵ are trivial and the case $M_1.M_2$ is immediate from the induction hypothesis.

Finally we prove for all me and P that

$$\begin{array}{l} \forall l \in \{l_\star, l^a_\bullet\} : (I, H, C, R) \models^l_{me} \lfloor P \rfloor \\ \text{whenever } me \text{ is acceptable for } P \end{array} \tag{11}$$

and where (I, H, C, R) once more is as above. The proof is by structural induction in $\lfloor P \rfloor$. The case $(\nu\, n^{\mu_\bullet})\lfloor P \rfloor$ follows from the induction hypothesis (since $me[n \mapsto \mu_\bullet]$

is acceptable for $\lfloor P \rfloor$). The cases $\mathbf{0}$, $\lfloor P \rfloor \mid \lfloor P' \rfloor$ and $!\lfloor P \rfloor$ are immediate using the induction hypothesis. In the case $N^{l^a_\bullet}[\lfloor P \rfloor]$ we choose $\tilde{N} = R^n(\beta^n_\bullet)$; the result then is a consequence of the induction hypothesis, (3), (9) and (8). In the case $\lfloor M \rfloor.\,\lfloor P \rfloor$ we take $\tilde{M} = R^c(\beta^c_\bullet)$; the result then follows from the induction hypothesis, (10) and (7). In the case for $\langle M \rangle^{l^c_\bullet}$ we take $\tilde{M} = R^c(\beta^c_\bullet)$; the result then is a consequence of (3), (10) and (6). In the case for $\langle\!\langle N \rangle\!\rangle^{l^n_\bullet}$ we take $\tilde{N} = R^n(\beta^n_\bullet)$; the result then follows from (3), (9) and (6). The case $(x^{\beta^c_\bullet}).\,\lfloor P \rfloor$ is a consequence of the induction hypothesis (since $me[x \mapsto \beta^c_\bullet]$ is acceptable), (3) and (5). The case $(\!(u^{\beta^n_\bullet})\!).\,\lfloor P \rfloor$ follows from the induction hypothesis (since $me[u \mapsto \beta^n_\bullet]$ is acceptable), (3) and (5).

Returning to the unaware process U we observe that

$$\{me_\star(n) \mid n \in \underline{\mathrm{fn}}(U)\} \subseteq \{me_\star(n_1), \cdots, me_\star(n_m)\}$$

and it then follows from (11) and Fact 4 that $me_\star$ can be restricted so as to be acceptable for U; this gives $(I, H, C, R) \models^{l_\star}_{me_\star} \lfloor U \rfloor$ as desired. □

When $(\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]$ passes the test and U is an unaware process we want to show that no subambient of U ever passes inside $\mathtt{w}$. Informally, this will take the form of assuming that $((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid U \to^* S$ and guaranteeing that S contains no subambient $\mathtt{w}^{l^a_1}[\cdots \mathtt{u}^{l^a_2}[\cdots]\cdots]$ where $\mathtt{u}$ comes from U. To formalise this we shall avail ourselves of Fact 2 that allows us to arrange the labelling to suit our needs. Indeed, if $((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid U \to^* S$ then $((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid \lfloor U \rfloor \to^* S'$ for some S' such that $\lfloor S \rfloor = \lfloor S' \rfloor$.

Theorem 4. Suppose that $(\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]$ passes the test of Table 9 and that U is an unaware process; if $((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid \lfloor U \rfloor \to^* S$ then S 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 $l^a_\bullet$.

Proof. Letting (I, H, C, R) be as in Table 9, it follows from Proposition 1 that $(I, H, C, R) \models^{l_\star}_{me_\star} ((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid \lfloor U \rfloor$. Then $(I, H, C, R) \models^{l_\star}_{me_\star} S$ follows from Theorem 1. Suppose next, for the sake of contradiction, that S does contain a subprocess $n_1^{l^a_1}[\cdots n_2^{l^a_2}[\cdots]\cdots]$ where n_1 has stable name w and l^a_2 is $l^a_\bullet$. Then it follows from $(I, H, C, R) \models^{l_\star}_{me_\star} S$ that $l^a_\bullet \in I^+(l^a_1) \wedge w \in H(l^a_1)$ showing that the test could not have been passed. □

Theorem 5. The test in Table 9 operates in cubic time.

Proof. Given a firewall of size $O(p)$ we may without loss of generality assume that the set $\{n_1, \cdots, n_m\}$ of variables constructed in Table 9 does not contain any variables not in the firewall; this ensures that also $((\nu\, \mathtt{w}^w)\mathtt{w}^{\mathrm{A}}[F]) \mid U_\star$ will have size $O(p)$. It follows from Theorem 3 that the least solution can be found in time $O(p + q^3)$ where, as before, q is the number of elements of $\mathbf{Lab} \cup \mathbf{Bnd} \cup \mathbf{SNam}$. This dominates the $O(q^3)$ operations needed to calculate the transitive closure I^+ of the relation I. Hence the overall operation of Table 9 is $O(p^3)\&O(p + q^3)$ where p is the size of the firewall and $q = O(p)$ can be chosen "arbitrarily" small. □

In summary, we have succeeded in using the control flow analysis to devise a cubic time algorithm for correctly validating that a proposed firewall is indeed protective; unlike [18] this development applies to the full ambient calculus. By judicious choice of the sets of labels, binders and markers the test can be performed in near-linear time (although at the cost of precision); this gives confidence in exploring our current approach on fully realistic internet programs.

5 Conclusion

Static analysis provides a summary of the behaviour of programs; we have shown that classical control flow analysis techniques can be adapted to tackle the much more dynamic setting of the ambient calculus. Our flow logic approach facilitates fully automatic validation of an analysis estimate as well as fully automatic computation of the best estimate; in spirit this is rather similar to the approaches of type inference. Type systems have already been extensively used to study the properties of web-based languages and related calculi (e.g. [1, 8]); the use of flow logic offers a flexible approach to adapting the vast amount of more "traditional" approaches to static analysis [17].

In this paper we developed a control flow analysis for the ambient calculus building on recent developments for the π-calculus [2–4] and the ambient calculus [14, 18, 19, 22]. In fact a notion of cryptography is implicitly part of the ambient calculus as presented here. As in the spi-calculus it is the restriction operator that is used to model "secrets" that cannot be guessed by brute force attack. Thus a message M, encrypted under the key K, is represented simply as the ambient $K[M]$ whereas an attempt at decrypting such a message is represented by the ambient $\mathsf{open}\,K$. If K is a secret only known to the principals $P_1, \cdots, P_n$ the entire system is represented as $(\nu\, K)(P_1 \mid \cdots \mid P_n)$ where each P_i may contain $K[M]$ as well as $\mathsf{open}\,K$.

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 unaware attackers based on its operation in conjunction with the hardest attacker; if it successfully protects against the hard attacker it will also protect against all other attackers not knowing the required passwords. We believe this to be typical of applications where software developed by subcontractors is *validated* before being embedded in the software system under construction. To make this practical we ensured that the test could be performed in cubic time.

It is important to stress that we circumvent the undecidability of dealing with all possible execution contexts, in particular all attackers, by coarsening the "level of granularity" of our observations to coincide with those of the static analysis. We maintain soundness because we proved the static analysis to be sound (but of course not complete) with respect to the dynamic semantics.

The analysis leading to the hardest attacker can be compared with the approaches of the "Dolev-Yao tradition" [10], including [5, 23, 12]. Indeed, the analyses performed in these studies amount to an "informal" analysis of the capabilities of the attacker, leading to an inductively defined behaviour. Such an analysis is not straightforward when the computational capabilities are modified, such as when adding mobility. The advantage of our approach therefore is two-fold. On the one hand, it allows us to make the analysis in a "formal" manner, because we can prove theorems with respect to the dynamic semantics (as mitigated by the analysis). On the other hand, the techniques used to implement the analysis provide insights on how to define the capabilities of the hardest attacker.

In summary, we have illustrated a novel approach to the validation of safety and security properties of software systems. We are hopeful that it will scale up to other calculi and web-based languages with explicit cryptographic primitives. By considering flow logics that express more powerful analyses it is likely that one can capture attackers in a more precise manner; perhaps one can show that the firewall protects againsts attackers knowing only some of the secret passwords or even against attackers knowing all the secret passwords but unable to use them in the appropriate manner (as might be the case for an agent representing a "courier" service).

Acknowledgements. We would like to thank Jacob Grydholt Jensen for working with us on [18] and Helmut Seidl for cooperating on "tiling" [20]. We would also like to thank the referees for useful feed-back (leading to an improved complexity result).

References

1. M. Abadi. Secrecy by typing in security protocols. In *Proceedings of Theoretical Aspects of Computer Software 1997, LNCS 1281*, pages 611–638. Springer, 1997.
2. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Control flow analysis for the π-calculus. In *Proceedings of CONCUR'98, LNCS 1466*, pages 84–98. Springer, 1998.
3. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis of processes for no read-up and no write-down. In *Proceedings of* 2^{nd} *FoSSaCS'99, LNCS 1578*, pages 120–134. Springer, 1999.
4. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static analysis for the π-calculus with applications to security. *Information and Computation*, (to appear), 2000.
5. D. Bolignano. An approach to the formal verification of cryptographic protocols. In *Proc. 3rd ACM Conf. on Computer and Communications Security*, pages 106–118. ACM Press, 1996.
6. L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility types for mobile ambients. In *Proceedings of ICALP'99, LNCS 1644*, pages 230–239. Springer, 1999.
7. L. Cardelli and A. D. Gordon. Mobile ambients. In *Proceedings of FoSSaCS'98, LNCS 1378*, pages 140–155. Springer, 1998.
8. L. Cardelli and A. D. Gordon. Types for mobile ambients. In *Proceedings of POPL'99*, pages 79–92. ACM Press, 1999.
9. L. Cardelli and A. D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proceedings of POPL'00*, pages 365–377. ACM Press, 2000.
10. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(12):198–208, 1983.
11. W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional Horn formulae. *Journal of Logic Programming*, 3:267–284, 1984.
12. F. J. T. Fábrega, J. C. Herzog, and J. D. Guttman. Strand Spaces: Why is a security protocol correct? In *Proceedings of 1998 IEEE Symposium on Security and Privacy*, pages 160–171. IEEE Press, 1998.
13. Li Gong. *Inside Java 2 Platform Security*. Addison-Wesley, 1999.
14. R. R. Hansen, J. G. Jensen, F. Nielson, and H. Riis Nielson. Abstract interpretation of mobile ambients. In *Proceedings of SAS'99, LNCS 1694*, pages 135–148, 1999.
15. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes (I and II). *Information and Computation*, 100(1):1–77, 1992.
16. F. Nielson and H. Riis Nielson. Flow logics and operational semantics. *Electronic Notes of Theoretical Computer Science*, 10, 1998.
17. F. Nielson, H. Riis Nielson, and C. Hankin. *Principles of Program Analysis*. Springer, 1999.
18. F. Nielson, H. Riis Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in mobile ambients. In *Proceedings of Concur'99, LNCS 1664*, pages 463–477. Springer, 1999.
19. F. Nielson, H. Riis Nielson, and M. Sagiv. A Kleene analysis of mobile ambients. In *Proceedings of ESOP'00, LNCS 1782*, pages 305–319. Springer, 2000.
20. F. Nielson and H. Seidl. Control flow analysis in cubic time. Submitted for publication, 2000.
21. H. Riis Nielson and F. Nielson. Flow logics for constraint based analysis. In *Proc. CC'98, LNCS 1383*, pages 109–127. Springer, 1998.

GIVEN:	a bounded number of binary relations $R_1, \cdots, R_m$, and a bound on the size of each conditional constraint.
INPUT:	a set $\mathcal{C}$ of conditional constraints over the binary relations.
OUTPUT:	the least solution $(\mathtt{R}_1, \cdots, \mathtt{R}_m)$ such that $(\mathtt{R}_1, \cdots, \mathtt{R}_m) \models \mathcal{C}$.

INITIALISE:
```
for i := 1 to m do
  for each argument arg occurring in C do
    Ri(arg) := ∅;

for each set ⊆ var occurring in C do
  INFL[set,var] := NIL;

cno := 0; stack := NIL;
for each set_k ⊆ var_k ⇒ ··· set_1 ⊆ var_1 ⇒ set_0 ⊆ var_0 in C do
  cno:= cno + 1;
  CT[cno] := (set_k ⊆ var_k ⇒ ··· set_1 ⊆ var_1 ⇒ set_0 ⊆ var_0);
  if k = 0 then push((set_0, var_0)) else
    for i := 1 to k do
      INFL[set_i,var_i] := CONS(cno, INFL[set_i,var_i]);
```

ITERATE:
```
while stack ≠ NIL do
  let (set, var) = pop() in
    for cno in INFL[set,var] do
      let (set_k ⊆ var_k ⇒ ··· set_1 ⊆ var_1 ⇒ set_0 ⊆ var_0) = CT[cno] in
        if ⋀_{i=1}^{k} [[set_i]] ⊆ [[var_i]] then push((set_0, var_0)) else skip;
```

USING:
```
procedure push((set, var)) is
if [[set]] ⊆ [[var]] then skip else
  [[var]] := [[var]] ∪ [[set]]; stack := CONS((set, var),stack);

function pop() is
let (set, var) = HEAD(stack) in
  stack := TAIL(stack); return((set, var));
```

Table 10. Worklist solution of constraints.

22. H. Riis Nielson and F. Nielson. Shape analysis for mobile ambients. In *Proceedings of POPL'00*, pages 142–154. ACM Press, 2000.
23. L. C. Paulson. Proving properties of security protocols by induction. In *Proc. 10th Computer Security Foundations Workshop*, pages 70–83. IEEE, 1997.

A Constraint Solving

We now consider how to turn a set $\mathcal{C}$ of conditional constraints of size $|\mathcal{C}|$ into the smallest acceptable analysis estimate. To this end we develop a worklist algorithm (see e.g. [17, Chapter 6]). It operates on data structures $\mathtt{R}_1, \cdots, \mathtt{R}_m$ corresponding to a constant number of binary relations. Additionally, it makes use of a list `stack` of "bit positions" that have just been set to 1 and whose consequences remain to be explored, a table `CT` that maps constraint numbers to the constraint in question, and a table `INFL` that for each "bit position" gives a list of constraint numbers influenced by that "bit position". We operate on lists using the constructors `CONS` and `NIL` and the destructors `HEAD` and `TAIL`. To operate on `stack` we make use of the function `push` for setting a "bit position" to 1 and for placing the "bit position" on `stack`, and of the function `pop` for returning the topmost "bit position" on `stack` and at the same time removing it from `stack`.

The worklist algorithm is displayed in Table 10. The initialisation of $\mathtt{R}_1, \cdots, \mathtt{R}_m$ amounts to setting $(\mathtt{R}_1, \cdots, \mathtt{R}_m)$ equal to the least element $(\bot, \cdots, \bot)$ of the appropriate lattice of values. The initialisation next sets `stack` and all `INFL[`*set*`,`*var*`]` to `NIL` and then computes the correct contents of the structures `stack`, `CT` and `INFL` by iterating through all conditional constraints in $\mathcal{C}$. The iteration phase amounts to propagating "bit positions" as long as there are new "bit positions" recently set to 1 whose consequences need to be explored. We write $[\![set]\!]$ and $[\![variable]\!]$ whenever we interpret the entities *set* and *var* as denoting values and variables, i.e. whenever they are used for comparisons or for updating the data structures. The algorithm clearly terminates because no "bit position" is placed twice on `stack` thanks to the test performed by the `push` operation.

We can now state the correctness of the worklist algorithm:

Lemma 5. Table 10 computes $\sqcap\{(R_1, \cdots, R_m) \mid (R_1, \cdots, R_m) \models \mathcal{C}\}$.

Proof. Write $(\bot, \cdots, \bot)$ for the least element and write

$$(R_1^\bullet, \cdots, R_m^\bullet) = \sqcap\{(R_1, \cdots, R_m) \mid (R_1, \cdots, R_m) \models \mathcal{C}\}$$

for the least solution (guaranteed by the obvious extension of Theorem 2). It follows from the "Horn clause" format of the conditional constraints that the invariant

$$(\bot, \cdots, \bot) \sqsubseteq (\mathtt{R}_1, \cdots, \mathtt{R}_m) \sqsubseteq (R_1^\bullet, \cdots, R_m^\bullet)$$

remains fulfilled throughout the iteration. Next write $(\mathtt{R}_1, \cdots, \mathtt{R}_m)$ for the resulting value. It follows from the above that

$$(\mathtt{R}_1, \cdots, \mathtt{R}_m) \sqsubseteq (R_1^\bullet, \cdots, R_m^\bullet)$$

and since it is clear that all constraints in $\mathcal{C}$ are fulfilled upon termination it follows that

$$(\mathtt{R}_1, \cdots, \mathtt{R}_m) \models \mathcal{C}.$$

Using the definition of $(R_1^\bullet, \cdots, R_m^\bullet)$ we then have

$$(\mathtt{R}_1, \cdots, \mathtt{R}_m) = \sqcap\{(R_1, \cdots, R_m) \mid (R_1, \cdots, R_m) \models \mathcal{C}\}$$

showing that the algorithm computes the least solution. □

To state the complexity of the worklist algorithm we assume that there is a constant bound on the size of the conditional constraints occurring in $\mathcal{C}$:

Lemma 6 (Special case of [11]). Table 10 operates in time $O(|\mathcal{C}|)$.

Proof. The initialisation of $\mathtt{R}_1, \cdots, \mathtt{R}_m$ takes time $O(|\mathcal{C}|)$. Setting `INFL[`*set*`,`*var*`]` to `NIL` also takes time $O(|\mathcal{C}|)$. The remaining initialisation performs $O(1)$ steps for each of the $O(|\mathcal{C}|)$ conditional constraints (recalling that $k = O(1)$, i.e. the length of conditions is bounded by a constant). For the iteration at most $O(|\mathcal{C}|)$ "bit positions" are placed on the `stack` thanks to the test performed by the `push` operation and to the fact that a "bit position" set to 1 is never reset to 0 given the "Horn clause" format of the conditional constraints. Ignoring the inspection of `INFL[`*set*`,`*var*`]` we therefore use time $O(|\mathcal{C}|)$. The inspection of `INFL[`*set*`,`*var*`]` can be amortised over all iterations since each constraint is only considered $O(1)$ times (again because $k = O(1)$) and thus takes overall time $O(|\mathcal{C}|)$. □