Nesting Analysis of Mobile Ambients ⋆

Chiara Braghin [a] Agostino Cortesi [a] Riccardo Focardi [a]
Flaminia L. Luccio [b] Carla Piazza [a]

[a] *Dipartimento di Informatica, Università Ca' Foscari di Venezia,*
{braghin,cortesi,focardi,piazza}@dsi.unive.it

[b] *Dipartimento di Scienze Matematiche, Università di Trieste,*
luccio@dsm.univ.trieste.it

Abstract

A new algorithm is introduced for analyzing possible nestings in Mobile Ambient calculus. It improves both time and space complexities of the technique proposed by Nielson and Seidl. The improvements are achieved by enhancing the data structure representations, and by reducing the computation to the Control Flow Analysis constraints that are effectively necessary to get to the least solution. These theoretical results are also supported by experimental tests run on a Java-based tool that implements a suite of algorithms for nesting analysis of Mobile Ambients.

Key words:
Static Analysis, Ambient Calculus, Complexity, Tools.

1 Introduction

The calculus of Mobile Ambients has been introduced in [1,2] with the main aim of explicitly modeling mobility. In particular, ambients are arbitrarily nested entities which can move around through suitable capabilities. Recently, big efforts have been devoted to the study of Control Flow Analysis (CFA) of such a calculus [3,4]. In particular, some analyses have been applied to the verification of security properties [5–9]. The idea of [6,7,9] is to compute an over-approximation of ambient nestings that may occur during process

⋆ Partially supported by MIUR Project "Modelli formali per la sicurezza", the EU Contract IST-2001-32617 MyThs, and FIRB project "Interpretazione astratta e model checking per la verifica di sistemi embedded".

computation, thus detecting possible intrusions and unwanted information flows.

Time and space complexities are key-issues for evaluating scalability and practical impact of any static analysis proposal. They become even more important when code mobility is possible, as low complexities would allow the very useful task of performing on the fly analysis of untrusted code migrating into a system. The computation of ambient nesting analysis, like [6,3,4], requires considerably high complexities, thus the design of efficient techniques turns out to be very important. This is the main motivation behind [10], where Nielson and Seidl reduce the worst-case time complexity of [3] from $O(N^5)$ to $O(N^3)$ steps, with N being the size of the analyzed process.

The first contribution of this paper is to refine the complexity results of [10], by considering, for a given process, its number N_a of ambients, its number N_t of capabilities and the sum $N_L = N_a + N_t$. In particular, for the best algorithm proposed in [10], we find a time complexity of $O(N_a^2 \cdot N_L)$ steps and a space complexity of $O((N_a^2 \cdot N_L) \log N_L)$ bits. We also prove that this algorithm performs at least $2 \cdot N_a^2 \cdot N_L$ steps and uses at least $2 \cdot (N_a^2 \cdot N_L) \log N_L$ bits, even in the best case. As a matter of fact, the algorithm first performs a translation of the Control Flow Analysis constraints into Horn clauses. Then, these clauses are processed through satisfiability standard algorithms [11] in order to compute the least solution. As such algorithms always consider all the clauses corresponding to the CFA constraints, even in the best case, all the clauses need to be generated. It turns out that the number of clauses is exactly $2 \cdot N_a^2 \cdot N_L$. A similar analysis is also provided for the less efficient $O(N^4)$ algorithm of [10].

The second contribution of this paper is to propose two new algorithms that improve both time and space complexities of the ones proposed in [10].

The gist of our proposal is to face the problem by a direct operational approach (i.e., without passing through Horn formulas), and to limit the computation to the Control Flow Analysis constraints that are effectively necessary to determine the least solution. This is done in an *on the fly* (dynamic) fashion, by combining a careful choice of data representation (namely, a buffer suite) with a selection policy which identifies the constraints that are potentially activated by an element while adding such an element to the solution, so that no useless repetition occurs. We prove that our best algorithm has a worst-case time complexity of $O(N_a^2 \cdot N_L)$ steps and a space complexity of $O((N_a \cdot N_L) \log N_L)$ bits. Thus, it highly improves the space complexity of the best algorithm in [10]. More precisely, we also prove that time complexity depends on the size of the least solution and thus it may decrease down to $c \cdot N_a \cdot N_L$, for a constant c, when the solution is linear with respect to the dimension of the process. As $2 \cdot N_a^2 \cdot N_L$ steps are always performed by the best algorithm of [10], with our algorithm we obtain a significant reduction of the execution time for "small" solutions.

In order to get these complexity improvements, we first apply our new technique to the less efficient $O(N^4)$ algorithm of [10]. As such an algorithm works on a simpler analysis specification, we also obtain a simpler algorithm, easier to explain and understand. We then show that all the results scale up to the more efficient $O(N^3)$ solution.

The ideas behind our new proposals are quite general. Thus, this paper may be considered as an important step towards the definition of a technique that could be applicable to compute Control Flow Analyses in different settings.

Finally, we have implemented the new algorithms in the BANANA tool [12], a Java applet available at `http://www.dsi.unive.it/~focardi/BANANA/`, that allows us to provide some experimental results.

The rest of the paper is organized as follows. In Section 2 we introduce the basic terminology of Mobile Ambient calculus and we briefly report the Control Flow Analysis of [3]. In Section 3 we study in depth the complexity of the algorithms presented in [10]. Then, in Section 4, we present our algorithms and the complexity results. Section 5 reports some preliminary experimental results obtained through the BANANA tool. Section 6 concludes the paper with final remarks.

2 Background: Mobile Ambients

The Mobile Ambient calculus has been introduced in [1,2] with the main aim of explicitly modeling mobility. Ambients are arbitrarily nested boundaries which can move around through suitable capabilities. The syntax of processes is given in Figure 1, where $n \in \mathbf{Amb}$ denotes an ambient name.

Intuitively, the restriction $(\nu n)P$ introduces the new name n and limits its scope to P; process $\mathbf{0}$ does nothing; $P \mid Q$ is P and Q running in parallel; replication provides recursion and iteration as $!P$ represents any number of copies of P in parallel. By $n^{\ell^a}[\![\, P \,]\!]$ we denote the ambient named n with the process P running inside it. The capabilities $\mathbf{in}^{\ell^t}\, n$ and $\mathbf{out}^{\ell^t}\, n$ move their enclosing ambients in and out ambient n, respectively; the capability $\mathbf{open}^{\ell^t}\, n$ is used to dissolve the boundary of a sibling ambient n. The operational semantics of a process P is given through a suitable reduction relation $\rightarrow$. Intuitively, $P \rightarrow Q$ represents the possibility for P of reducing to Q through some computation.

Formally, the definition of $\rightarrow$ is given in terms of a structural congruence $\equiv$,

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

Fig. 1. Mobile Ambients Syntax

$$
\begin{array}{ll}
 & P \mid Q \equiv Q \mid P \\
P \equiv P & \\
P \equiv Q \Rightarrow Q \equiv P & (P \mid Q) \mid R \equiv P \mid (Q \mid R) \\
P \equiv Q, Q \equiv R \Rightarrow P \equiv R & !P \equiv P \mid !P \\
 & (\nu n)(\nu m)P \equiv (\nu m)(\nu n)P \\
P \equiv Q \Rightarrow (\nu n)P \equiv (\nu n)Q & (\nu n)(P \mid Q) \equiv P \mid (\nu n)Q \text{ if } n \notin fn(P) \\
P \equiv Q \Rightarrow P \mid R \equiv Q \mid R & (\nu n)\, m^{\ell^a}[\![\, P \,]\!] \equiv m^{\ell^a}[\![\, (\nu n)P \,]\!] \text{ if } n \neq m \\
P \equiv Q \Rightarrow !P \equiv !Q & \\
P \equiv Q \Rightarrow n^{\ell^a}[\![\, P \,]\!] \equiv n^{\ell^a}[\![\, Q \,]\!] & P \mid \mathbf{0} \equiv P \\
P \equiv Q \Rightarrow M.P \equiv M.Q & (\nu n)\mathbf{0} \equiv \mathbf{0} \\
 & !\mathbf{0} \equiv \mathbf{0}
\end{array}
$$

Fig. 2. Structural congruence

that equates terms up to trivial syntactic restructuring. Figure 2 reports the definition of $\equiv$, where M is a capability and $fn(P)$ denotes the set of free names of P, i.e., the names of P that are not bound by a restriction operator. Processes that only differ for renaming of bound names are implicitly equated. Reduction $\rightarrow$ is formally defined in Figure 3. We will use the standard notation $P \rightarrow^* Q$ to denote a reduction of process P to process Q performed in 0 or more steps.

Labels $\ell^a \in \mathsf{Lab}^a$ on ambients and labels $\ell^t \in \mathsf{Lab}^t$ on capabilities (transitions) are introduced as it is customary in static analysis to indicate "program points". They will be useful in the next sections when developing the analysis. We denote with Lab the set of all the labels $\mathsf{Lab}^a \cup \mathsf{Lab}^t$. We use the special label $env \in \mathsf{Lab}^a$ to denote the external environment, i.e., the environment containing the process under observation.

Given a process P, we also introduce the notation $\mathsf{Lab}^a(P)$ to denote the set

$$n_1^{\ell^a}[\,\mathbf{in}^{\ell^t}\ m\ .\ P \mid Q\,] \mid m_2^{\ell^a}[\,R\,] \to m_2^{\ell^a}[\ n_1^{\ell^a}[\,P \mid Q\,] \mid R\,]$$

$$m_2^{\ell^a}[\ n_1^{\ell^a}[\,\mathbf{out}^{\ell^t}\ m\ .\ P \mid Q\,] \mid R\,] \to n_1^{\ell^a}[\,P \mid Q\,] \mid m_2^{\ell^a}[\,R\,]$$

$$\mathbf{open}^{\ell^t}\ n\ .\ P \mid n^{\ell^a}[\,Q\,] \to P \mid Q$$

$$P' \equiv P, P \to Q, Q \equiv Q' \Rightarrow P' \to Q'$$

$$P \to Q \Rightarrow (\nu n)P \to (\nu n)Q$$

$$P \to Q \Rightarrow n^{\ell^a}[\,P\,] \to n^{\ell^a}[\,Q\,]$$

$$P \to Q \Rightarrow P \mid R \to Q \mid R$$

Fig. 3. Reduction relation

of ambient labels in P plus the special label env, $\mathsf{Lab}^t(P)$ to denote the set of capability labels in P, and $\mathsf{Lab}(P)$ to denote $\mathsf{Lab}^a(P) \cup \mathsf{Lab}^t(P)$. Moreover, $N_a = |\mathsf{Lab}^a(P)|$, $N_t = |\mathsf{Lab}^t(P)|$, and $N_L = |\mathsf{Lab}(P)| = N_a + N_t$. With N we denote the global number of operators occurring in P. Note that $N_L < N$, as there is at least one occurrence of $\mathbf{0}$ in every non-empty process.

Example 2.1 Process P_1 models a *cab* driving a *client* from $site_1$ to $site_2$. The execution of P_1 is depicted in Figure 4 (where labels have been omitted for the sake of readability) and is described below:

$$\begin{array}{l} site_1^{\ell_1^a}\ [\ \ client^{\ell_2^a}[\,\mathbf{in}^{\ell_3^t}\, cab\ .\ call^{\ell_4^a}[\,\mathbf{out}^{\ell_5^t}\, client\ .\ \mathbf{out}^{\ell_6^t}\, site_1\ .\ \mathbf{in}^{\ell_7^t}\, site_2\ .\ \mathbf{0}\,]\,] \\ \qquad \mid\ cab^{\ell_8^a}[\,\mathbf{open}^{\ell_9^t}\, call\ .\ \mathbf{0}\,]\ ]\ \mid \\ site_2^{\ell_{10}^a}[\ \mathbf{0}\ ]. \end{array}$$

Initially, *cab* and *client* are in $site_1$, while $site_2$ is empty. The client enters the cab by applying its capability $\mathbf{in}^{\ell_3^t}\, cab$. Thus, process P_1 moves to:

$$\begin{array}{l} site_1^{\ell_1^a}\ [\ \ cab^{\ell_8^a}[\ \mathbf{open}^{\ell_9^t}\, call\ .\ \mathbf{0} \mid \\ \qquad\qquad client^{\ell_2^a}[\ call^{\ell_4^a}[\,\mathbf{out}^{\ell_5^t}\, client\ .\ \mathbf{out}^{\ell_6^t}\, site_1\ .\ \mathbf{in}^{\ell_7^t}\, site_2\ .\ \mathbf{0}\,]\,]\,]\,] \mid \\ site_2^{\ell_{10}^a}[\ \mathbf{0}\ ]. \end{array}$$

Now, the client tells the cab its destination by releasing ambient *call*, which consumes its $\mathbf{out}^{\ell_5^t}\, client$ capability.

$$\begin{array}{l} site_1^{\ell_1^a}[\ cab^{\ell_8^a}[\,\mathbf{open}^{\ell_9^t}\, call\ .\ \mathbf{0} \mid call^{\ell_4^a}[\,\mathbf{out}\ site_1\ .\ \mathbf{in}\ site_2\ .\ \mathbf{0}\,] \mid client^{\ell_2^a}[\,\mathbf{0}\,]\,]\,] \mid \\ site_2^{\ell_{10}^a}[\,\mathbf{0}\,]. \end{array}$$

Then, the client request satisfaction is modeled by opening (dissolving) the

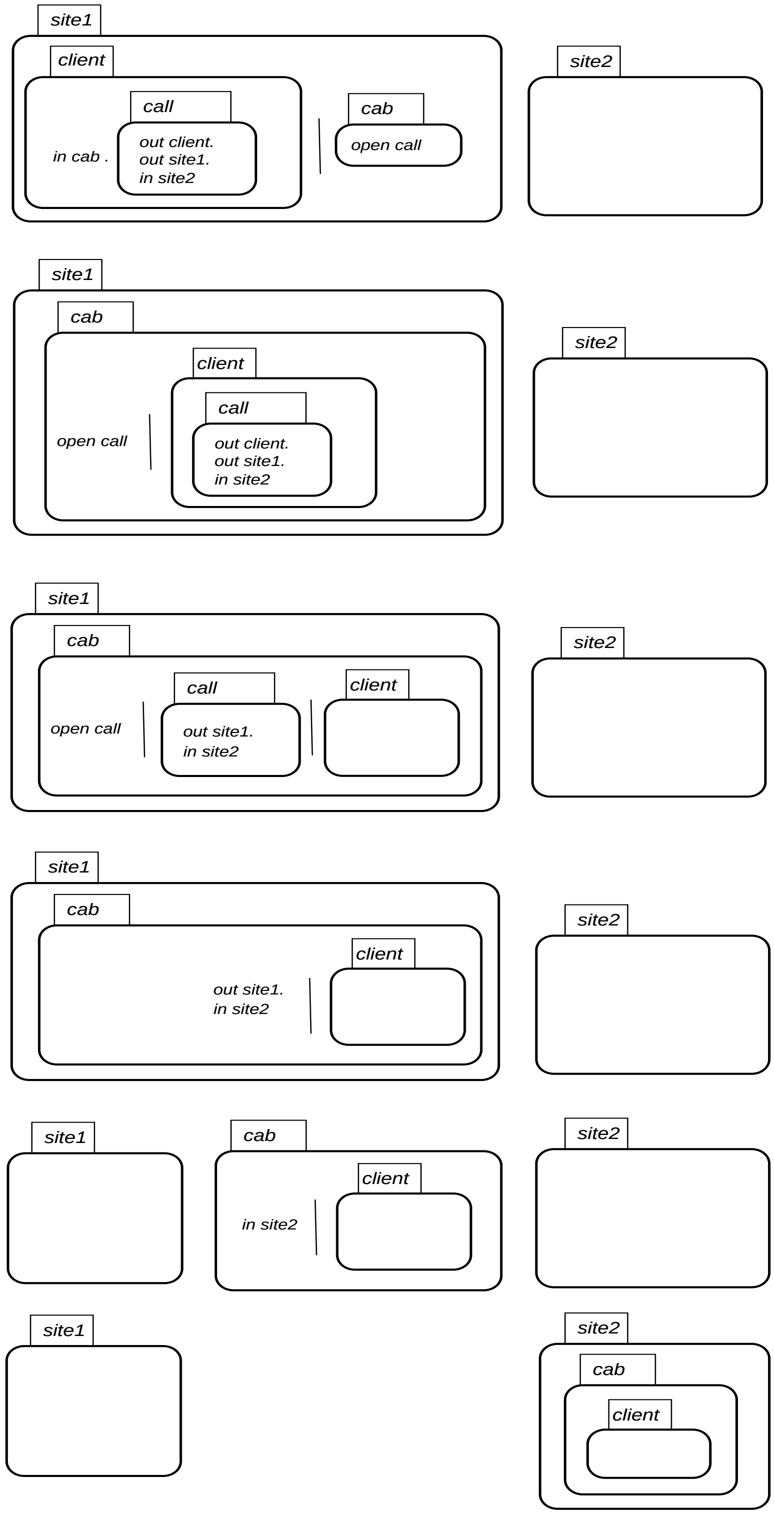

Fig. 4. The *cab* example

client call. At this point, process P_1 has reached the state:

$$site_1^{\ell_1^a}[\ cab^{\ell_9^a}[\,\mathbf{out}^{\ell_6^t}\ site_1\ .\ \mathbf{in}^{\ell_7^t}\ site_2.\mathbf{0} \mid client^{\ell_2^a}[\,\mathbf{0}\,]\ ]\ ] \mid site_2^{\ell_{11}^a}[\,\mathbf{0}\,]\,.$$

Then, the cab exits $site_1$ and it enters $site_2$, as expected by the client:

$$site_1^{\ell_1^a}[\,\mathbf{0}\,] \mid site_2^{\ell_{10}^a}[\,cab^{\ell_9^a}[\,client^{\ell_2^a}[\,\mathbf{0}\,]\,]\,].$$

Observe that for such a process P_1 the label sets are the following:

$$\begin{aligned}
\mathsf{Lab}^a(P_1) &= \{\ell_1^a, \ell_2^a, \ell_4^a, \ell_8^a, \ell_{10}^a\},\\
\mathsf{Lab}^t(P_1) &= \{\ell_3^t, \ell_5^t, \ell_6^t, \ell_7^t, \ell_9^t\},\\
\mathsf{Lab}(P_1) &= \{\ell_1^a, \ell_2^a, \ell_3^t, \ell_4^a, \ell_5^t, \ell_6^t, \ell_7^t, \ell_8^a, \ell_9^t, \ell_{10}^a\}.
\end{aligned}$$

Thus, $N_a = 5$, $N_t = 5$, $N_L = 10$, and $N = 15$ (N_L plus three $\mathbf{0}$ and two $\mid$).
$\square$

In the rest of the paper, we assume that the ambient and capability labels occurring in a process P are all distinct. Performing the Control Flow Analysis with all distinct labels produces a more precise result that can be later approximated by equating some labels.

2.1 Control Flow Analysis

The Control Flow Analysis of a process P described in [3] aims at modeling the possible ambient nestings occurring in the execution of P. It works on pairs $(\hat{I}, \hat{H})$, where:

- The first component $\hat{I}$ is an element of $\wp(\mathsf{Lab}^a(P) \times \mathsf{Lab}(P))$. If process P, during its execution, contains an ambient labeled ℓ^a having inside either a capability or an ambient labeled ℓ, then (ℓ^a, ℓ) is expected to belong to $\hat{I}$.
- The second component $\hat{H} \in \wp(\mathsf{Lab}^a(P) \times \mathbf{Amb})$ keeps track of the correspondence between names and labels. If process P contains an ambient labeled ℓ^a with name n, then (ℓ^a, n) is expected to belong to $\hat{H}$. [1]
- The pairs are component-wise partially ordered by set inclusion.

The analysis is defined as usual by a representation and a specification functions [13]. They are recalled in Figure 5 and Figure 6, respectively, where $\sqcup$ denotes the component-wise union of the elements of the pairs.

The representation function aims at mapping concrete values to their best abstract representation. It is given in terms of a function $\beta_\ell^{\mathrm{CF}}(P)$ which maps

[1] We are assuming that ambient names are *stable*, i.e., n is a representative for a class of α-convertible names, following the same approach of [4]. In [3,10], an alternative treatment of α-equivalence is used, where bound names are annotated with markers, and a marker environment me is associated to constraints.

$$
\begin{array}{lll}
(\mathit{res}) & \beta^{\mathrm{CF}}_{\ell}((\nu n)P) & = \beta^{\mathrm{CF}}_{\ell}(P) \\
(\mathit{zero}) & \beta^{\mathrm{CF}}_{\ell}(\mathbf{0}) & = (\emptyset, \emptyset) \\
(\mathit{par}) & \beta^{\mathrm{CF}}_{\ell}(P \mid Q) & = \beta^{\mathrm{CF}}_{\ell}(P) \sqcup \beta^{\mathrm{CF}}_{\ell}(Q) \\
(\mathit{repl}) & \beta^{\mathrm{CF}}_{\ell}(!P) & = \beta^{\mathrm{CF}}_{\ell}(P) \\
(\mathit{amb}) & \beta^{\mathrm{CF}}_{\ell}(n^{\ell^a}[\,P\,]) & = \beta^{\mathrm{CF}}_{\ell^a}(P) \sqcup (\{(\ell, \ell^a)\}, \{(\ell^a, n)\}) \\
(\mathit{in}) & \beta^{\mathrm{CF}}_{\ell}(\mathbf{in}^{\ell^t}\, n \,.\, P) & = \beta^{\mathrm{CF}}_{\ell}(P) \sqcup (\{(\ell, \ell^t)\}, \emptyset) \\
(\mathit{out}) & \beta^{\mathrm{CF}}_{\ell}(\mathbf{out}^{\ell^t}\, n \,.\, P) & = \beta^{\mathrm{CF}}_{\ell}(P) \sqcup (\{(\ell, \ell^t)\}, \emptyset) \\
(\mathit{open}) & \beta^{\mathrm{CF}}_{\ell}(\mathbf{open}^{\ell^t}\, n \,.\, P) & = \beta^{\mathrm{CF}}_{\ell}(P) \sqcup (\{(\ell, \ell^t)\}, \emptyset)
\end{array}
$$

Fig. 5. Representation Function for the Control Flow Analysis

$$
\begin{array}{lll}
(\mathit{res}) & (\hat{I}, \hat{H}) \models^{\mathrm{CF}} (\nu n)P & \text{iff } (\hat{I}, \hat{H}) \models^{\mathrm{CF}} P \\
(\mathit{zero}) & (\hat{I}, \hat{H}) \models^{\mathrm{CF}} \mathbf{0} & \text{always} \\
(\mathit{par}) & (\hat{I}, \hat{H}) \models^{\mathrm{CF}} P \mid Q & \text{iff } (\hat{I}, \hat{H}) \models^{\mathrm{CF}} P \;\wedge\; (\hat{I}, \hat{H}) \models^{\mathrm{CF}} Q \\
(\mathit{repl}) & (\hat{I}, \hat{H}) \models^{\mathrm{CF}} !P & \text{iff } (\hat{I}, \hat{H}) \models^{\mathrm{CF}} P \\
(\mathit{amb}) & (\hat{I}, \hat{H}) \models^{\mathrm{CF}} n^{\ell^a}[\,P\,] & \text{iff } (\hat{I}, \hat{H}) \models^{\mathrm{CF}} P \\
(\mathit{in}) & (\hat{I}, \hat{H}) \models^{\mathrm{CF}} \mathbf{in}^{\ell^t}\, n \,.\, P & \text{iff } (\hat{I}, \hat{H}) \models^{\mathrm{CF}} P \;\wedge \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathsf{Lab}^a(P) : ((\ell^a, \ell^t) \in \hat{I} \;\wedge\; (\ell^{a''}, \ell^a) \in \hat{I}} \\
& \multicolumn{2}{l}{\qquad \wedge\; (\ell^{a''}, \ell^{a'}) \in \hat{I} \;\wedge\; (\ell^{a'}, n) \in \hat{H}) \implies (\ell^{a'}, \ell^a) \in \hat{I}} \\
(\mathit{out}) & (\hat{I}, \hat{H}) \models^{\mathrm{CF}} \mathbf{out}^{\ell^t}\, n \,.\, P & \text{iff } (\hat{I}, \hat{H}) \models^{\mathrm{CF}} P \;\wedge \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathsf{Lab}^a(P) : ((\ell^a, \ell^t) \in \hat{I} \;\wedge\; (\ell^{a'}, \ell^a) \in \hat{I}} \\
& \multicolumn{2}{l}{\qquad \wedge\; (\ell^{a''}, \ell^{a'}) \in \hat{I} \;\wedge\; (\ell^{a'}, n) \in \hat{H}) \implies (\ell^{a''}, \ell^a) \in \hat{I}} \\
(\mathit{open}) & (\hat{I}, \hat{H}) \models^{\mathrm{CF}} \mathbf{open}^{\ell^t}\, n \,.\, P & \text{iff } (\hat{I}, \hat{H}) \models^{\mathrm{CF}} P \;\wedge \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'} \in \mathsf{Lab}^a(P), \forall \ell' \in \mathsf{Lab}(P) : ((\ell^a, \ell^t) \in \hat{I} \;\wedge\; (\ell^a, \ell^{a'}) \in \hat{I}} \\
& \multicolumn{2}{l}{\qquad \wedge\; (\ell^{a'}, n) \in \hat{H} \;\wedge\; (\ell^{a'}, \ell') \in \hat{I}) \implies (\ell^a, \ell') \in \hat{I}}
\end{array}
$$

Fig. 6. Specification of the Control Flow Analysis

process P into a pair $(\hat{I}\ ,\hat{H}\ )$ corresponding to the initial state of P, with respect to an enclosing ambient labeled with ℓ. The representation of a process P is defined as $\beta^{\mathrm{CF}}_{env}(P)$.

Example 2.2 Let P_2 be the process $n^{\ell^a_1}[\, m^{\ell^a_2}[\,\mathbf{out}^{\ell^t}\, n.\mathbf{0}\,]\,]$. The representation function of P_2 is $\beta^{\mathrm{CF}}_{env}(P_2) = (\{(env, \ell^a_1), (\ell^a_1, \ell^a_2), (\ell^a_2, \ell^t)\}, \{(\ell^a_1, n), (\ell^a_2, m)\})$.

Notice that all ambient nestings are captured by the first component $\{(env, \ell_1^a), (\ell_1^a, \ell_2^a), (\ell_2^a, \ell^t)\}$, while all the correspondences between ambients and labels of P_2 are kept by the second one, i.e., $\{(\ell_1^a, n), (\ell_2^a, m)\}$. $\square$

The specification states a closure condition of a pair $(\hat{I}, \hat{H})$ with respect to all the possible moves executable on a process P. It mostly relies on recursive calls on subprocesses except for the three capabilities *open*, *in*, and *out*. For instance, the rule for *open*-capability states that if some ambient labeled ℓ^a has an *open*-capability ℓ^t on an ambient n, that may apply due to the presence of a sibling ambient labeled $\ell^{a'}$ whose name is n, then the result of performing that capability should also be recorded in $\hat{I}$, i.e., all the ambients/capabilities nested in $\ell^{a'}$ have to be nested also in ℓ^a.

Proposition 2.3 *[3] Let P be a process. If $(\hat{I}, \hat{H}) \models^{\mathrm{CF}} P$ and $\beta^{\mathrm{CF}}_{env}(P) \subseteq (\hat{I}, \hat{H})$ and $P \rightarrow^* P'$, then $\beta^{\mathrm{CF}}_{env}(P') \subseteq (\hat{I}, \hat{H})$.*

Example 2.4 Consider again process P_2 of Example 2.2. Note that it may evolve to $n^{\ell_1^a}[\mathbf{0}] \mid m^{\ell_2^a}[\mathbf{0}]$. It is easy to prove that the least solution for P_2 is $(\hat{I},\hat{H})$, where $\hat{I}= \{(env, \ell_1^a), (env, \ell_2^a), (\ell_1^a, \ell_2^a), (\ell_2^a, \ell^t)\}$, $\hat{H} = \{(\ell_1^a, n), (\ell_2^a, m)\}$. Notice that the analysis correctly captures through the pair (env, ℓ_2^a) the possibility for m to exit from n. $\square$

3 Refining the Complexity Analysis for Nielson and Seidl Algorithms

In this section, we refine the worst case complexity results for the algorithms presented in [10] by recalculating them as functions of N_a, N_t, and N_L, instead of N. We also calculate the minimum number of steps performed by the algorithms even in the best case. The results of this section will be useful to compare the techniques of [10] with our new algorithms that will be presented in Section 4.

3.1 The first Algorithm of Nielson and Seidl – NS1

In the following, we will use NS1 to refer to the $O(N^4)$ algorithm for the Control Flow Analysis of Mobile Ambients presented in [10]. NS1 is based on a formulation of the analysis which is equivalent to the one presented in the previous section. The constraints in Figure 6 are rewritten as ground Horn clauses by instantiating the universally quantified variables in all possible ways. To estimate the number of these ground Horn clauses, notice that:

- the number of capabilities is obviously $O(N_t)$, since N_t is the cardinality of $\textsf{Lab}^t(P)$;
- a constraint for an *open*-capability involves two universal quantifications that range over $\textsf{Lab}^a(P)$, whose cardinality is N_a, plus another universal quantification that ranges over $\textsf{Lab}(P)$, whose cardinality is N_L. Constraints for *in* and *out*-capabilities have three universal quantifications ranging over $\textsf{Lab}^a(P)$.

Since $\textsf{Lab}^a(P) \subseteq \textsf{Lab}(P)$, we have that the greatest number of ground Horn clauses is generated by the algorithm when all the capabilities are *open* ones. Namely, the number of generated clauses is $O(N_t \cdot N_a^2 \cdot N_L)$. Moreover, they require $O((N_t \cdot N_a^2 \cdot N_L) \log N_L)$ bits to be represented.

The next step of NS1 is to apply the algorithm presented in [11] (which represents a set of ground Horn clauses as a graph, and solves a pebbling problem on that graph) to this set, in order to find the least solution. As such algorithm uses $O(n)$ steps and $O(n \log n)$ space, where n is the size of the set of ground Horn clauses, we obtain the following (considering that the parsing of the process has already been done):

Proposition 3.1 *The complexity of NS1 is $O(N_t \cdot N_a^2 \cdot N_L)$ steps and $O((N_t \cdot N_a^2 \cdot N_L) \log N_L)$ bits.*

Example 3.2 Let P_3 be the process $n^{\ell_1^a}[\ m^{\ell_2^a}[\mathbf{open}^{\ell_1^t}\ n.\mathbf{0}\,]\ ]$. The constraint for the *open*-capability is

$$\forall \ell^a, \ell^{a'} \in \textsf{Lab}^a(P), \forall \ell' \in \textsf{Lab}(P):$$
$$((\ell^a, \ell^t) \in \hat{I}\ \wedge\ (\ell^a, \ell^{a'}) \in \hat{I}\ \wedge\ (\ell^{a'}, n) \in \hat{H}\ \wedge\ (\ell^{a'}, \ell') \in \hat{I}) \implies (\ell^a, \ell') \in \hat{I}.$$

In order to generate the Horn clauses, ℓ^a and $\ell^{a'}$ have to be instantiated in all the possible ways in the set $\textsf{Lab}^a(P) = \{\ell_1^a, \ell_2^a\}$, whose cardinality is $N_a = 2$, and ℓ' ranges over $\textsf{Lab}(P) = \{\ell_1^a, \ell_2^a, \ell_1^t\}$, whose cardinality is $N_L = 3$. This introduces $N_a^2 \cdot N_L = 12$ ground Horn clauses. For instance, one of them is the one obtained by instantiating ℓ^a to ℓ_1^a, $\ell^{a'}$ to ℓ_2^a and ℓ' to ℓ_1^t, i.e.,

$$((\ell_1^a, \ell_1^t) \in \hat{I}\ \wedge\ (\ell_1^a, \ell_2^a) \in \hat{I}\ \wedge\ (\ell_2^a, n) \in \hat{H}\ \wedge\ (\ell_2^a, \ell_1^t) \in \hat{I}) \implies (\ell_1^a, \ell_1^t) \in \hat{I}.$$

In P there are no other capabilities, hence we obtain only these 12 ground Horn clauses. Therefore, in this case $N_t \cdot N_a^2 \cdot N_L = 12$. □

Observe that, even in the best case (i.e., no *open* capabilities) at least $N_t \cdot N_a^3$ steps are performed to generate all the ground clauses. We then obtain the following:

Corollary 3.3 *Algorithm NS1 performs at least $N_t \cdot N_a^3$ steps and uses at least $N_t \cdot N_a^3 \cdot \log N_L$ bits.*

We now consider NS2, the cubic-time algorithm presented in [10]. It is based on an optimization of the analysis depicted in Figure 6 which we report in Figure 7. [2] The equivalence between the analysis of Figures 6 and 7 follows from [10]. The main idea behind the optimized analysis is to reduce the number of universal quantifications in each analysis constraint. This is achieved by adding some new components that keep further information on the nestings, and that may be globally computed.

As an example, consider the *in* constraint of Figure 6. It requires to find three labels ℓ^a, $\ell^{a'}$, $\ell^{a''} \in \mathbf{Lab}^a(P)$ such that $(\ell^a, \ell^t) \in \hat{I} \ \wedge \ (\ell^{a''}, \ell^a) \in \hat{I} \ \wedge \ (\ell^{a''}, \ell^{a'}) \in \hat{I}$. Notice that $\ell^{a''}$ is only used to check if ℓ^a and $\ell^{a'}$ are *siblings*. Thus, having a set $\hat{S} \in \wp(\mathbf{Lab}^a(P) \times \mathbf{Lab}^a(P))$ containing all the pairs of labels corresponding to sibling ambients, allows to limit the quantification on two labels only. In particular, it is sufficient to find two labels $\ell^a, \ell^{a'}$, such that $(\ell^a, \ell^t) \in \hat{I} \wedge (\ell^a, \ell^{a'}) \in \hat{S}$. In order to calculate set $\hat{S}$, a new global constraint is now required (*global*, in Figure 7): $((\ell^{a''}, \ell^a) \in \hat{I} \ \wedge \ (\ell^{a''}, \ell^{a'}) \in \hat{I}) \implies (\ell^a, \ell^{a'}) \in \hat{S}$. Similar optimizations are applied to the other constraints, by introducing the components $\hat{O}, \hat{P} \in \wp(\mathbf{Lab}^a(P) \times \mathbf{Lab}^a(P))$, where $(\ell^{a'}, \ell^a) \in \hat{O}$ represents the fact that ℓ^a may move out of $\ell^{a'}$, and $(\ell^a, \ell^{a'}) \in \hat{P}$ indicates that $\ell^{a'}$ may be opened inside ℓ^a. Note that the rule (*global*) is applied only once during the analysis.

As for NS1, the NS2 algorithm is based on a translation of constraints into a set of ground Horn clauses, on which the algorithm in [11] is applied to compute the least solution. To estimate the size of the set of ground Horn clauses obtained by instantiating the variables in all the possible ways, notice that:

- there are N_t capabilities and all their constraints involve two universal quantifications over $\mathbf{Lab}^a(P)$, whose size is N_a;
- the first two constraints in the (*global*) rule involve three universal quantifications over $\mathbf{Lab}^a(P)$;
- the third constraint in the (*global*) rule involves two universal quantifications over $\mathbf{Lab}^a(P)$, and one over $\mathbf{Lab}(P)$, whose cardinality is N_L.

We obtain that the number of ground clauses is

$$N_t \cdot N_a^2 + N_a^3 + N_a^2 \cdot N_L = (N_t + N_a)N_a^2 + N_a^2 \cdot N_L = 2 \cdot N_a^2 \cdot N_L.$$

Proposition 3.4 *The complexity of the NS2 algorithm is* $O(N_a^2 \cdot N_L)$ *steps and* $O((N_a^2 \cdot N_L) \log N_L)$ *bits.*

[2] In [10] the optimized Analysis is presented using a slightly different formalism.

$$
\begin{array}{lll}
(global) & (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{CFOpt}} P & \text{iff } (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} P \wedge \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathsf{Lab}^a(P) : ((\ell^{a''}, \ell^a) \in \hat{I} \ \wedge \ (\ell^{a''}, \ell^{a'}) \in \hat{I})} \\
& \multicolumn{2}{l}{\qquad \Longrightarrow (\ell^a, \ell^{a'}) \in \hat{S} \wedge} \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathsf{Lab}^a(P) : ((\ell^{a'}, \ell^a) \in \hat{O} \ \wedge \ (\ell^{a''}, \ell^{a'}) \in \hat{I})} \\
& \multicolumn{2}{l}{\qquad \Longrightarrow (\ell^{a''}, \ell^a) \in \hat{I} \wedge} \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'} \in \mathsf{Lab}^a(P), \ell' \in \mathsf{Lab}(P) : ((\ell^a, \ell^{a'}) \in \hat{P} \ \wedge \ (\ell^{a'}, \ell') \in \hat{I})} \\
& \multicolumn{2}{l}{\qquad \Longrightarrow (\ell^a, \ell') \in \hat{I}} \\
(res) & (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} (\nu n)P & \text{iff } (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} P \\
(zero) & (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} \mathbf{0} & \text{always} \\
(par) & (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} P \mid Q & \text{iff } (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} P \wedge \\
& & \qquad (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} Q \\
(repl) & (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} !P & \text{iff } (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} P \\
(amb) & (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} n^{\ell^a}[\, P\,] & \text{iff } (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} P \\
(in) & (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} \mathbf{in}^{\ell^t}\, n\, .\, P & \text{iff } (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} P \wedge \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'} \in \mathsf{Lab}^a(P) : ((\ell^a, \ell^t) \in \hat{I} \ \wedge \ (\ell^a, \ell^{a'}) \in \hat{S} \ \wedge \ (\ell^{a'}, n) \in \hat{H})} \\
& \multicolumn{2}{l}{\qquad \Longrightarrow (\ell^{a'}, \ell^a) \in \hat{I}} \\
(out) & (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} \mathbf{out}^{\ell^t}\, n\, .\, P & \text{iff } (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} P \wedge \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'} \in \mathsf{Lab}^a(P) : ((\ell^a, \ell^t) \in \hat{I} \ \wedge \ (\ell^{a'}, \ell^a) \in \hat{I} \ \wedge \ (\ell^{a'}, n) \in \hat{H})} \\
& \multicolumn{2}{l}{\qquad \Longrightarrow (\ell^{a'}, \ell^a) \in \hat{O}} \\
(open) & (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} \mathbf{open}^{\ell^t}\, n\, .\, P & \text{iff } (\hat{I},\hat{H},\hat{S},\hat{O},\hat{P}) \models^{\mathrm{Opt}} P \wedge \\
& \multicolumn{2}{l}{\quad \forall \ell^a, \ell^{a'} \in \mathsf{Lab}^a(P) : ((\ell^a, \ell^t) \in \hat{I} \ \wedge \ (\ell^a, \ell^{a'}) \in \hat{I} \ \wedge \ (\ell^{a'}, n) \in \hat{H})} \\
& \multicolumn{2}{l}{\qquad \Longrightarrow (\ell^a, \ell^{a'}) \in \hat{P}}
\end{array}
$$

Fig. 7. The Optimized Control Flow Analysis

By following the same reasoning as above, it is also easy to see that:

Corollary 3.5 *Algorithm NS2 performs at least* $2 \cdot N_a^2 \cdot N_L$ *steps and uses at least* $2 \cdot (N_a^2 \cdot N_L) \log N_L$ *bits.*

4 The new algorithms

In this section, we present our new algorithms for nesting analysis of Figures 6 and 7, and we compare them with the NS1 and NS2 algorithms above.

As highlighted in Section 3, the main idea behind NS1 and NS2 is to instantiate the analysis constraints with respect to all the possible labels in order to obtain a set of ground Horn clauses. Unfortunately, instantiating all the constraints causes that much space is used and, even in the best case, $N_t \cdot N_a^3$ and $2 \cdot N_L \cdot N_a^2$ steps are performed by NS1 and NS2, respectively (see Corollaries 3.3 and 3.5).

In order to avoid these problems, our algorithms only consider the constraints that are effectively necessary for the computation of the analysis. The algorithms take a more direct approach, in a sense that they do neither translate constraints into Horn clauses, nor apply the algorithm of [11]. The algorithms start with an empty analysis $\hat{I}$ and with a buffer containing all the pairs corresponding to the initial process representation. [3] Recall that, for the correctness of the analysis, these pairs should be contained in the final $\hat{I}$. At each round, one pair is extracted from the buffer and it is added to the solution $\hat{I}$. Only the constraints that are potentially "activated" by the extracted pair are then considered, i.e., only the constraints that have such an element in the premise. All the pairs required by such constraints are then inserted into the buffer, so that they will be eventually added to the solution. This is repeated until a fix-point is reached, i.e., until all the elements required by the constraints are in the solution. The most important ingredient in this on-the-fly generation is the use of a buffer together with a matrix which allow to use each pair of labels in the buffer *exactly once* to generate new pairs.

We show that our first algorithm has a space complexity of $O((N_a \cdot N_L) \log N_L)$ bits and a time complexity of $O(S_I^a \cdot N_t \cdot N_L + S_I^t \cdot N_a \cdot N_L)$ steps, where S_I^a (S_I^t) is the number of pairs of the form $(\ell^a, \ell^{a'})$ ((ℓ^a, ℓ^t), respectively) in the least solution. First, note that $O((N_a \cdot N_L) \log N_L)$ bits highly decreases the $O((N_t \cdot N_a^2 \cdot N_L) \log N_L)$ space complexity of NS1. Note also that the maximum size of the solution is $S_I^a = N_a^2$ and $S_I^t = N_a \cdot N_t$. Thus, only in the worst-case, our algorithm has a time complexity equal to the one of NS1. The best case arises instead when the solution is linear with respect to the process dimension, i.e., $S_I^a = N_a$, $S_I^t = N_t$, thus reducing time-complexity to $c \cdot N_a \cdot N_t \cdot N_L$, where c is a suitable constant. [4] The solution cannot be less than linear as it immediately follows from the definition of the representation function.

Our second algorithm, decreases with respect to NS2, space complexity to $O((N_a \cdot N_L) \log N_L)$ bits and time complexity to $O(S_I^a \cdot N_L + S_I^t \cdot N_a + S_S \cdot N_t + S_P \cdot N_L + S_O \cdot N_a)$ steps, where S_I^a and S_I^t are defined as above, and S_S, S_O, S_P are the final dimensions of $\hat{S}$, $\hat{O}$, $\hat{P}$, respectively. First, note that our space complexity $O((N_a \cdot N_L) \log N_L)$ greatly improves the $O((N_a^2 \cdot N_L) \log N_L)$ space complexity of NS2. Moreover, the maximum size of the solution is $S_I^a = S_S =$

---

[3] Indeed, our second algorithm uses a set of buffers, but this does not change the underlying ideas of the algorithm.

[4] By exploiting the same argument we used for calculating the best case of NS1, we could lower this complexity down to $c \cdot N_a^2 \cdot N_t$, for a constant c, which is strongly better (up to multiplicative constants) than $N_t \cdot N_a^3$, i.e., the best case for NS1.

$S_O = S_P = N_a^2$ and $S_I^t = N_a \cdot N_t$, thus, in the worst case, time complexity becomes equal to the one of NS2. The best case is instead when the solution is linear with respect to the process dimension, thus reducing time-complexity to $c \cdot N_a \cdot N_L$ for a constant $c \leq 5$ (see Corollary 4.4) which is strictly better than $2 \cdot N_a^2 \cdot N_L$, i.e., the best case of NS2.

Note that the cases in which the solutions are maximal, i.e., when our algorithms have the same time complexity of NS1 and NS2, correspond to analysis solutions that contain all the possible nestings. Such cases are either related to quite rare processes showing all possible nestings at run-time, or to excessive approximations of more common processes.

We now present the two algorithms in detail.

4.1 Improving space: Algorithm 1

Our first algorithm, called Algorithm 1, is depicted in Figure 8. We assume that the parsing of the process has already been done, producing an array `cap` of length N_t containing all the capabilities of the input process. For instance, `cap`[i] may contain "$\mathbf{in}^{\ell^t}$ n", representing an *in* capability labeled with ℓ^t and with n as target. [5] During the parsing, the representation $\beta^{\mathrm{CF}}_{env}(P)$ is computed giving two initial sets $\hat{I}_0$ and $\hat{H}_0$ that are stored into an $N_a \times N_L$ bit matrix $\mathtt{B}_{\hat{I}}$, and into an $N_a \times N_a$ bit matrix $\mathtt{M}_{\hat{H}}$, respectively. By parsing P twice, we can build $\mathtt{B}_{\hat{I}}$ in such a way that columns from 1 to N_a are indexed by ambient labels, while all the other columns by capability ones. All the pairs in $\hat{I}_0$ are also stored in a stack $\mathtt{buf}_{\hat{I}}$, on which the usual operations $\mathtt{push}_{\hat{I}}$ (l,l') and $\mathtt{pop}_{\hat{I}}$ () apply. Matrix $\mathtt{B}_{\hat{I}}$ is used to efficiently check whether an element has ever been inserted into $\mathtt{buf}_{\hat{I}}$, thus ensuring that a pair is inserted in $\mathtt{buf}_{\hat{I}}$ at most once. In particular, the new command $\mathtt{push\_c}_{\hat{I}}$ (l,l') applies if $\mathtt{B}_{\hat{I}}$ [l,l']=`false`, and it both executes $\mathtt{push}_{\hat{I}}$ (l,l') and sets $\mathtt{B}_{\hat{I}}$ [l,l'] to `true`. Finally, we initialize to `false` another bit matrix $\mathtt{M}_{\hat{I}}$ of size $N_a \times N_L$ that will contain the final result of the analysis. Also in $\mathtt{M}_{\hat{I}}$ the columns from 1 to N_a are indexed by ambient labels and the ones from $N_a + 1$ to N_L by capability labels. This initialization phase requires only $O(N)$ steps, since two parsings of P are sufficient.

Example 4.1 Let P be the Firewall Access process of [1,2], where an agent crosses a firewall by means of previously arranged passwords k, k' and k''. Figure 9 shows the execution of P: by only knowing the three passwords it is possible to enter the firewall w (see [9] for a detailed analysis of the security

[5] n here represents an integer $1 \leq n \leq N_a$ corresponding to the n-th ambient name. The correspondence between names and integers is kept in the symbol-table produced at the parsing-time.

```
while buf_Î != NIL do
  (l,l') := pop_Î (); M_Î [l,l'] := true;
  for i := 1 to N_t do
    case cap[i] of:
      in^ℓt n: if (l' ∈ Lab^t(P) and l' = ℓ^t)
              then for j := 1 to N_a do
                    for k := 1 to N_a do
                       if (M_Î [k,l] and M_Î [k,j] and M_Ĥ [j,n]) then push_c_Î (j,l)
              else if (l' ∈ Lab^a(P) and M_Î [l',ℓ^t]) then for j:= 1 to N_a do
                         if (M_Î [l,j] and M_Ĥ [j,n]) then push_c_Î (j,l');
                    if (l' ∈ Lab^a(P) and M_Ĥ [l',n]) then for j:= 1 to N_a do
                         if (M_Î [j,ℓ^t] and M_Î [l,j]) then push_c_Î (l',j);
      out^ℓt n: if (l' ∈ Lab^t(P) and l' = ℓ^t)
              then for j := 1 to N_a do
                    for k := 1 to N_a do
                       if (M_Î [j,l] and M_Î [k,j] and M_Ĥ [j,n]) then push_c_Î (k,l)
              else if (l' ∈ Lab^a(P) and M_Î [l',ℓ^t] and M_Ĥ [l,n]) then for j:= 1 to N_a do
                         if M_Î [j,l] then push_c_Î (j,l');
                    if (l' ∈ Lab^a(P) and M_Ĥ [l',n]) then for j:= 1 to N_a do
                         if (M_Î [j,ℓ^t] and M_Î [l',j]) then push_c_Î (l,j);
      open^ℓt n: if (l' ∈ Lab^t(P) and l' = ℓ^t)
              then for j := 1 to N_a do
                    for k := 1 to N_L do
                       if (M_Î [l,j] and M_Î [j,k] and M_Ĥ [j,n]) then push_c_Î (l,k)
              else if (l' ∈ Lab^a(P) and M_Î [l,ℓ^t] and M_Ĥ [l',n]) then for j:= 1 to N_L do
                         if M_Î [l',j] then push_c_Î (l,j);
                    if (l' ∈ Lab^a(P) and M_Ĥ [l,n]) then for j:= 1 to N_a do
                         if (M_Î [j,ℓ^t] and M_Î [j,l]) then push_c_Î (j,l');
```

Fig. 8. Algorithm 1

issues related to this example).

$$P = (\nu w)\, w^{a1}[\; k^{a2}[\,\mathbf{out}^{t1}\, w.\mathbf{in}^{t2}\, k'.\mathbf{in}^{t3}\, w.\mathbf{0}\,] \;|\; \mathbf{open}^{t4}\, k' \,.\, \mathbf{open}^{t5}\, k'' \,.\, \mathbf{0}\,] \;|$$
$$k'^{a3}[\,\mathbf{open}^{t6}\, k \,.\; k''^{a4}[\,\mathbf{0}\,]\;]$$

The least solution of P, as computed using the specification of the Control Flow Analysis depicted in Figure 6, is the pair $(\hat{I}, \hat{H})$, where:

$\hat{I} = \{(env, a1), (env, a2), (env, a3), (a1, a1), (a1, a2), (a1, a3), (a1, a4),$
$(a1, t1), (a1, t2), (a1, t3), (a1, t4), (a1, t5), (a1, t6), (a2, t1), (a2, t2), (a2, t3),$
$(a3, a1), (a3, a2), (a3, a3), (a3, a4), (a3, t1), (a3, t2), (a3, t3), (a3, t6)\},$
$\hat{H} = \{(a1, w), (a2, k), (a3, k'), (a4, k'')\}.$

Let us see how Algorithm 1 applies to process P. In this case, $N_a = 5$ and $N_t =$

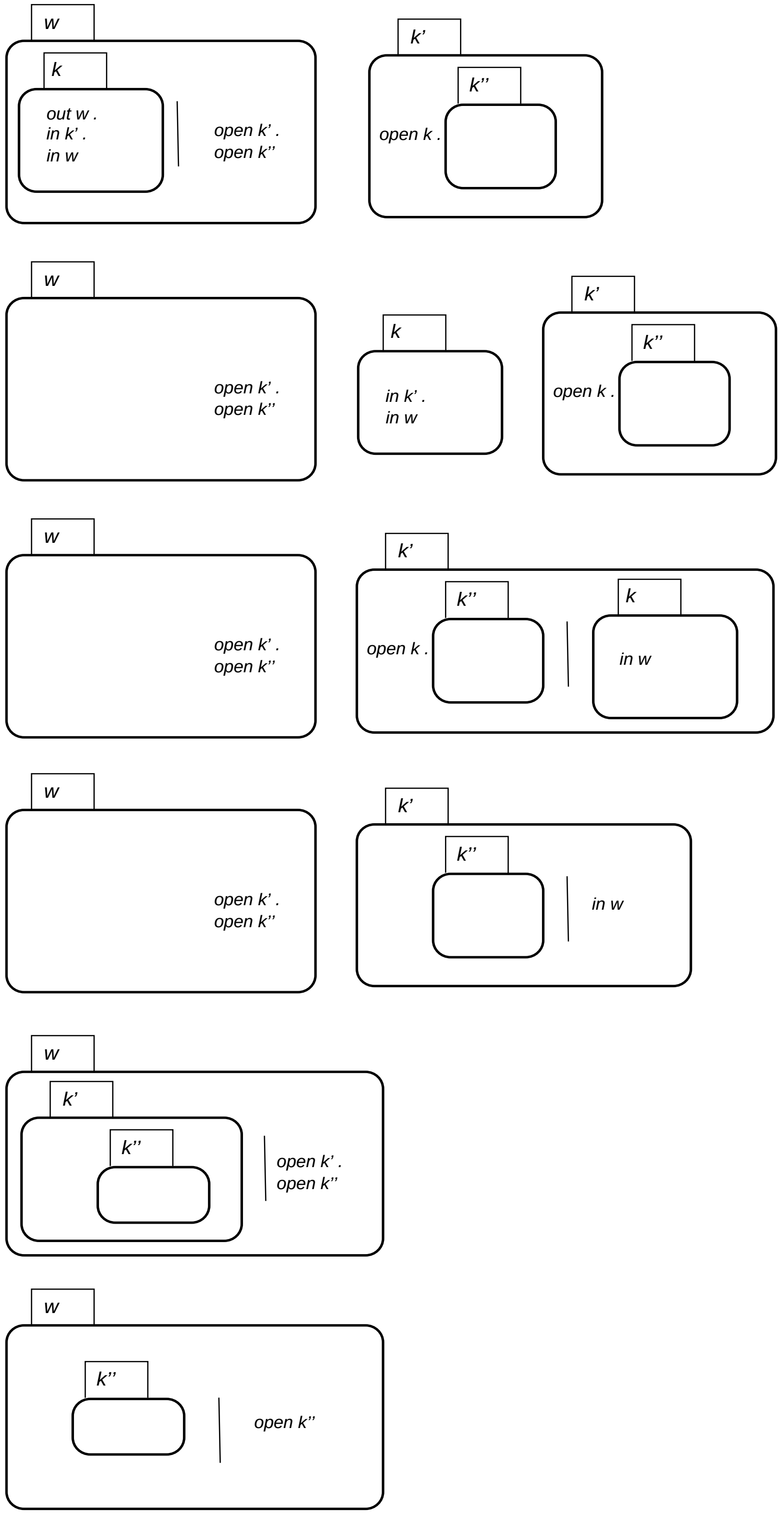

Fig. 9. The firewall example: the process that initially is inside ambient k'' (in this example $\mathbf{0}$), at the end is executed inside the firewall w.

6, thus $\mathtt{B}_{\hat{I}}$ and $\mathtt{M}_{\hat{I}}$ are 5×11 bit matrices, $\mathtt{M}_{\hat{H}}$ is a 5×5 bit matrix, and $\mathtt{cap}$ an array of length 6, initialized as $\langle \mathbf{out}^{t1}\, w, \mathbf{in}^{t2}\, k', \mathbf{in}^{t3}\, w, \mathbf{open}^{t4}\, k', \mathbf{open}^{t5}\, k'',$

open$^{t6}\, k\rangle$. After the initial parsing, the only pairs in $\mathtt{M}_{\hat{H}}$ which are set to `true` are $\{(a1, w),\ (a2, k),\ (a3, k'), (a4, k'')\}$, while $\mathtt{buf}_{\hat{I}}$ and $\mathtt{B}_{\hat{I}}$ contain the pairs $\langle (env, a1),\ (env, a3),\ (a1, a2),\ (a3, a4), (a1, t4), (a1, t5), (a2, t1), (a2, t2), (a2, t3), (a3, t6)\rangle$.

Let the pair $(env, a1)$ be the top element of $\mathtt{buf}_{\hat{I}}$. The first 6 rounds of the `while`-loop just move pairs from $\mathtt{buf}_{\hat{I}}$ to $\mathtt{M}_{\hat{I}}$ (no push is performed). Then, at round 7:

- $\mathtt{buf}_{\hat{I}} = \langle (a2, t1), (a2, t2), (a2, t3), (a3, t6)\rangle$
- $\mathtt{M}_{\hat{I}} = \langle (env, a1), (env, a3), (a1, a2), (a3, a4), (a1, t4), (a1, t5)\rangle$
- $\mathtt{B}_{\hat{I}} = \langle (env, a1), (env, a3), (a1, a2), (a3, a4), (a1, t4), (a1, t5), (a2, t1), (a2, t2), (a2, t3), (a3, t6)\rangle$.

We extract the top element $(a2, t1)$ of $\mathtt{buf}_{\hat{I}}$, thus l $:= a2$, and l' $:= t1$. We show the first iteration, $i = 1$, where $\mathtt{cap}[1]$ is "**out**$^{t1}\, w$". Thus, we have l' $= \ell^t$ and $n = w$. Since l' $\in \mathsf{Lab}^t(P)$ and l' $= \ell^t$, we are in the "then" branch. The only case that makes true the if condition is when j $= a1$ and k $= env$. Since $(a1, w) \in \mathtt{M}_{\hat{H}}$ and both $(a1, a2)$ and $(env, a1)$ are in $\mathtt{M}_{\hat{I}}$, the pair $(env, a2)$ is pushed in $\mathtt{buf}_{\hat{I}}$ (note that it is not already in $\mathtt{B}_{\hat{I}}$). The algorithm ends after the 24^{th} round, when $\mathtt{buf}_{\hat{I}}$ is empty. $\square$

We can prove the following result:

Theorem 4.2 *Algorithm 1 is correct. It has a time complexity of* $O(S^a_I \cdot N_t \cdot N_L + S^t_I \cdot N_a \cdot N_L)$ *steps where*

$$S^a_I = |\{(\ell^a, \ell^{a'}) \mid \ell^a, \ell^{a'} \in \mathsf{Lab}^a(P), (\ell^a, \ell^{a'}) \in \hat{I} \text{ at the end of the computation}\}|,$$
$$S^t_I = |\{(\ell^a, \ell^t) \mid \ell^a \in \mathsf{Lab}^a(P), \ell^t \in \mathsf{Lab}^t(P), (\ell^a, \ell^t) \in \hat{I} \text{ at the end of the computation}\}|.$$

It also has a space complexity of $O((N_a \cdot N_L)\log N_L)$ *bits.*

Proof. The proof follows mainly two steps: first, we show an invariant on the outermost `while`−loop of Algorithm 1, and we use such a condition to derive minimality of the solution; then, we prove its time and space complexities. Notice that, by construction, $\mathtt{B}_{\hat{I}}$ contains the information of $\mathtt{buf}_{\hat{I}}$ and $\mathtt{M}_{\hat{I}}$. Initially, $\mathtt{B}_{\hat{I}}$ contains exactly the same information of $\mathtt{buf}_{\hat{I}}$ while $\mathtt{M}_{\hat{I}}$ is empty (contains all false). Moreover, all the elements inserted in $\mathtt{buf}_{\hat{I}}$ are also set to true in $\mathtt{B}_{\hat{I}}$ and, when an element of $\mathtt{buf}_{\hat{I}}$ is moved to $\mathtt{M}_{\hat{I}}$ it remains included in $\mathtt{B}_{\hat{I}}$. The algorithm ends when $\mathtt{buf}_{\hat{I}}$ is empty, therefore when $\mathtt{B}_{\hat{I}}$=$\mathtt{M}_{\hat{I}}$.

- **Correctness.** We have to show that the algorithm verifies the specification of the Control Flow Analysis depicted in Figure 6, and that it computes the least solution. First, we prove the following condition:

Let a round be one iteration of the outermost `while`*-loop. At a generic round k: if we apply the Control Flow Analysis by considering the set $\hat{I}$ corresponding to matrix* $\mathtt{M}_{\hat{I}}$*, then the set of pairs (l,l') for which the analysis fails (i.e., such that* $\mathtt{M}_{\hat{I}}$ *[l,l']=*`false` *and (l,l') is in the rightmost part of an applicable capability rule) are in* $\mathtt{B}_{\hat{I}}$*.*

We prove it by induction on k. At step $k = 0$, $\mathtt{M}_{\hat{I}}$ contains all false, therefore the hypotheses of all constraints are false and we are done, since the analysis is always satisfied. Let us now assume, by induction, that the property above holds up to step i. At step $i+1$, we have a new matrix $\mathtt{M}'_{\hat{I}}$ that is equal to matrix $\mathtt{M}_{\hat{I}}$ computed at step i, plus $\mathtt{M}_{\hat{I}}$ [l,l']:=`true`, i.e., the pair (l,l') is processed. Moreover, $\mathtt{B}_{\hat{I}}$ is increased to $\mathtt{B}'_{\hat{I}}$. We have to prove that, if we apply the Control Flow Analysis to matrix $\mathtt{M}'_{\hat{I}}$, then the set of pairs (l,l') for which the analysis fails are in $\mathtt{B}'_{\hat{I}}$. Let us now assume, by contradiction, that there exists a constraint in the analysis that requires a new pair (x, y) that does not belong to $\mathtt{B}'_{\hat{I}}$. Since $\mathtt{B}'_{\hat{I}} \supseteq \mathtt{B}_{\hat{I}}$, we also have that (x, y) does not belong to $\mathtt{B}_{\hat{I}}$. By induction hypothesis, we know that the constraint above requiring (x, y) was not applicable to matrix $\mathtt{M}_{\hat{I}}$ (otherwise the pair (x, y) would necessarily be in $\mathtt{B}_{\hat{I}}$). This means that such a constraint has in the hypothesis the fact that (l,l') belongs to $\hat{I}$, and so that $\mathtt{M}_{\hat{I}}$ [l,l']=`true`. Now, it is sufficient to observe that one round of the algorithm exactly adds to $\mathtt{B}_{\hat{I}}$ and $\mathtt{buf}_{\hat{I}}$ every pair that is required by all the constraints containing "(l,l') $\in \hat{I}$" in the hypothesis (this may be verified by considering all the instances of the constraints in which (l,l') is placed in every possible position of the hypothesis). Thus (x, y) is in $\mathtt{B}'_{\hat{I}}$, leading to a contradiction.

When the algorithm terminates, we have $\mathtt{B}_{\hat{I}}=\mathtt{M}_{\hat{I}}$. This proves that $\mathtt{M}_{\hat{I}}$ is a solution of the analysis. In fact, the property above states that all the pairs required by $\mathtt{M}_{\hat{I}}$ are at least in $\mathtt{B}_{\hat{I}}$. Since $\mathtt{B}_{\hat{I}}=\mathtt{M}_{\hat{I}}$, we obtain that $\mathtt{M}_{\hat{I}}$ satisfies all the analysis constraints. Note also that $\mathtt{buf}_{\hat{I}}$ initially contains the representation of the process $\beta^{\mathrm{CF}}_{env}(P)$, thus proving that $\mathtt{M}_{\hat{I}}$ is indeed a correct solution for P. Since the procedure is incremental, it is trivial to prove that the analysis is the least one.

- **Complexity.** To prove the time complexity of Algorithm 1, we first recall that we are assuming all capability labels are distinct. Notice that at the beginning $\mathtt{buf}_{\hat{I}}$ contains only pairs of the form (l,l') with l$\in \mathsf{Lab}^a(P)$ and l'$\in \mathsf{Lab}(P)$. Moreover, all the elements which are added to $\mathtt{buf}_{\hat{I}}$ are of such a form. Each pair which is inserted in $\mathtt{buf}_{\hat{I}}$ is also in the final solution and it is extracted from $\mathtt{buf}_{\hat{I}}$ only once, since an initial check is made on matrix $\mathtt{B}_{\hat{I}}$ before inserting new elements in $\mathtt{buf}_{\hat{I}}$. Now we compute the cost of each extraction of a pair from $\mathtt{buf}_{\hat{I}}$ distinguishing the case of a pair of the form (l,l')$\in \mathsf{Lab}^a(P) \times \mathsf{Lab}^a(P)$ from that of a pair of form (l,l')$\in \mathsf{Lab}^a(P) \times \mathsf{Lab}^t(P)$. When a pair of the form (l,l')$\in \mathsf{Lab}^a(P) \times \mathsf{Lab}^a(P)$ is extracted, the external `for`$-$cycle is executed N_t times and each time the `else`$-$branch is chosen. Hence, in the worst-case (for the *open*-capabilities) during each iteration at most N_L steps are required. Therefore for each pair of the form (l,l')$\in$ $\mathbf{Lab}^a(P) \times \mathbf{Lab}^a(P)$ at most $N_t \cdot N_L$ steps are performed. When a pair of the form (l,l')$\in$ $\mathbf{Lab}^a(P) \times \mathbf{Lab}^t(P)$ is extracted the external `for`-cycle is executed N_t times but only one of the `if`-cases may apply. In the worst-case (for the *open*-capabilities) this execution costs $N_a \cdot N_L + N_t$ steps, i.e., $O(N_a \cdot N_L)$ steps. We can conclude that globally the steps performed are $O(S_I^a \cdot N_t \cdot N_L + S_I^t \cdot N_a \cdot N_L)$, and each step involves only constant time operations.

Space complexity is computed as follows: the data structures used are two $N_a \times N_L$ bit matrices ($\mathtt{B}_{\hat{I}}$ and $\mathtt{M}_{\hat{I}}$), one $N_a \times N_a$ bit matrix ($\mathtt{M}_{\hat{H}}$), and one buffer `cap` which contains at most N_t elements of at most $O(\log N_L)$ bits. Finally, $\mathtt{buf}_{\hat{I}}$ may contain at most $N_a \cdot N_L$ pairs of at most $\log N_L$ bits for a total of at most $(N_a \cdot N_L) \log N_L$ bits, in the worst-case. $\square$

Note that the worst-case time complexity of Algorithm 1 is $O(N_t \cdot N_a^2 \cdot N_L)$, since in the worst-case $S_I^a = N_a^2$ and $S_t^I = N_a \cdot N_t$.

4.2 Improving time: Algorithm 2

Time complexity of Algorithm 1 can be reduced by applying buffering techniques also to the optimized analysis of Figure 7. This leads to our second algorithm, called Algorithm 2 and depicted in Figure 10. Also in this case, we assume that the parsing of the process has already been done twice. As a result, the same data structures as in Algorithm 1 (i.e., `cap`, $\mathtt{buf}_{\hat{I}}$, $\mathtt{B}_{\hat{I}}$, $\mathtt{M}_{\hat{I}}$ and $\mathtt{M}_{\hat{H}}$), are initialized. In addition, we consider the additional buffers $\mathtt{buf}_{\hat{S}}$, $\mathtt{buf}_{\hat{O}}$ and $\mathtt{buf}_{\hat{P}}$, and three $N_a \times N_a$ bit matrices $\mathtt{B}_{\hat{S}}$, $\mathtt{B}_{\hat{O}}$, and $\mathtt{B}_{\hat{P}}$ set to `false`. These matrices have the same rôle of matrix $\mathtt{B}_{\hat{I}}$, i.e., they avoid that a pair is put twice in one of the buffers $\mathtt{buf}_{\hat{S}}$, $\mathtt{buf}_{\hat{O}}$, and $\mathtt{buf}_{\hat{P}}$. We also initialize to `false` the $N_a \times N_a$ bit matrices $\mathtt{M}_{\hat{S}}$, $\mathtt{M}_{\hat{O}}$, and $\mathtt{M}_{\hat{P}}$ that will contain the final result of the analysis concerning the sets $\hat{S}$, $\hat{O}$, and $\hat{P}$, respectively. As for Algorithm 1, we assume that in $\mathtt{B}_{\hat{I}}$ and in $\mathtt{M}_{\hat{I}}$ columns from 1 to N_a are assigned to ambient labels and the ones from $N_a + 1$ to N_L to capability labels.

The main difference between Algorithm 1 and Algorithm 2 is the use of the data structures related to $\hat{S}$, $\hat{O}$, and $\hat{P}$, thus merging the ideas of NS2 with our on-the-fly approach. Observe that the last block of `if`-statements in Algorithm 2 corresponds to the *global* constraints in Figure 7.

We can now prove the following theorem:

Theorem 4.3 *Algorithm 2 is correct. It has a time complexity of* $O(S_I^a \cdot N_L +$

```
while (buf_Î != NIL or buf_Ŝ != NIL or buf_Ô != NIL or buf_P̂ != NIL) do
  if buf_Î != NIL then
    (l,l') := pop_Î (); M_Î [l,l'] := true; b:= "I"
  else if buf_Ŝ != NIL then
    (l,l') := pop_Ŝ (); M_Ŝ [l,l'] := true; b:= "S"
  else if buf_Ô != NIL then
    (l,l') := pop_Ô (); M_Ô [l,l'] := true; b:= "O"
  else if buf_P̂ != NIL then
    (l,l') := pop_P̂ (); M_P̂ [l,l'] := true; b:= "P";
  if ((b="I" and l' ∈ Lab^a(P)) or b="S") then for i := 1 to N_t do
    case cap[i] of:
      in^ℓt n: if (b="S" and M_Î [l,ℓt] and M_Ĥ [l',n]) then push_c_Î (l',l);
      out^ℓt n: if (b="I"  and M_Î [l',ℓt] and M_Ĥ [l,n]) then push_c_Ô (l,l');
      open^ℓt n: if (b="I"  and M_Î [l,ℓt] and M_Ĥ [l',n]) then push_c_P̂ (l,l');
  if (b="I" and l' ∈ Lab^t(P))
    case cap[l'] of:
      in^l' n: for j := 1 to N_a do
             if (M_Ŝ [l,j] and M_Ĥ [j,n]) then push_c_Î (j,l)
      out^l' n: for j := 1 to N_a do
             if (M_Î [j,l] and M_Ĥ [j,n]) then push_c_Ô (j,l)
      open^l' n: for j := 1 to N_a do
             if (M_Î [l,j] and M_Ĥ [j,n]) then push_c_P̂ (l,j)
  if (b="I" and l' ∈ Lab^a(P)) then for j:= 1 to N_a do
    if M_Î [l,j] then push_c_Ŝ (l',j); push_c_Ŝ (j,l');
    if M_Ô [l',j] then push_c_Î (l,j);
    if M_P̂ [j,l] then push_c_Î (j,l');
  if b="O" then for j:= 1 to N_a do
    if M_Î [j,l] then push_c_Î (j,l');
  if b="P" then for j:= 1 to N_L do
    if M_Î [l',j] then push_c_Î (l,j);
```

Fig. 10. Algorithm 2

$S_I^t \cdot N_a + S_S \cdot N_t + S_O \cdot N_a + S_P \cdot N_L)$ *steps, where:*

$$S_I^a = |\{(\ell^a, \ell^{a'}) \mid \ell^a, \ell^{a'} \in \mathsf{Lab}^a(P), (\ell^a, \ell^{a'}) \in \hat{I} \textit{ at the end of the computation}\}|,$$

$$S_I^t = |\{(\ell^a, \ell^t) \mid \ell^a \in \mathsf{Lab}^a(P), \ell^t \in \mathsf{Lab}^t(P), (\ell^a, \ell^t) \in \hat{I} \textit{ at the end of the computation}\}|,$$

and S_S, S_O, S_P are the cardinality of $\hat{S}$, $\hat{O}$, $\hat{P}$, respectively, at the end of the execution. It also has a worst-case space complexity of $O((N_a \cdot N_L)\log N_L)$ bits.

Proof.

To prove the correctness of Algorithm 2, we have to show that it verifies the specification of the Optimized Control Flow Analysis depicted in Figure 7, and that it computes a least solution. The rest of the proof follows the lines of the proof of Theorem 4.2.

Similarly to what we have done in the case of Algorithm 1, we first prove that $\mathtt{B}_*$ contains the information of $\mathtt{buf}_*$ and $\mathtt{M}_*$, with $* \in \{\hat{I}, \hat{S}, \hat{O}, \hat{P}\}$. For all $* \in \{\hat{I}, \hat{S}, \hat{O}, \hat{P}\}$, $\mathtt{B}_*$ initially contains the same information of $\mathtt{buf}_*$, while $\mathtt{M}_*$ is empty (contains all false). Moreover all the elements inserted in $\mathtt{buf}_*$ are also inserted in $\mathtt{B}_*$, and when an element of $\mathtt{buf}_*$ is moved to $\mathtt{M}_*$, it is still included in $\mathtt{B}_*$. Hence, as the algorithm terminates when $\mathtt{buf}_*$ is empty for all $* \in \{\hat{I}, \hat{S}, \hat{O}, \hat{P}\}$, we have that at the end of the execution $\mathtt{B}_* = \mathtt{M}_*$ for all $* \in \{\hat{I}, \hat{S}, \hat{O}, \hat{P}\}$ holds.

- **Correctness.** In order to prove that Algorithm 2 really finds the least solution to the analysis, we first prove the following result:

  Let a round be one iteration of the outermost `while`*−loop. At a generic round k: If we apply the Control Flow Analysis by considering the set * corresponding to matrix* $\mathtt{M}_*$*, for all* $* \in \{\hat{I}, \hat{S}, \hat{O}, \hat{P}\}$*, then all the pairs (l,l') such that (l,l') is required to be in* $*$ *and (l,l') is not in* $\mathtt{M}_*$*, are in* $\mathtt{B}_*$*.*

  The proof proceeds by induction on k. At step $k = 0$ the matrices $\mathtt{M}_*$, with $* \in \{\hat{I}, \hat{S}, \hat{O}, \hat{P}\}$, are empty, hence we immediately have the thesis. At step $i + 1$ we have that an element has been added to $\mathtt{M}_{\hat{I}}$ or to $\mathtt{M}_{\hat{S}}$ or to $\mathtt{M}_{\hat{O}}$ or to $\mathtt{M}_{\hat{P}}$. For sake of simplicity, we prove the thesis in the first of the four cases, since the other ones are similar. Consider a new matrix $\mathtt{M}'_{\hat{I}}$ that is equal to matrix $\mathtt{M}_{\hat{I}}$ computed at step i, plus $\mathtt{M}_{\hat{I}}$ [l,l']:=`true`; i.e., the pair (l,l') is processed. Moreover, $\mathtt{B}_{\hat{I}}$ is increased to $\mathtt{B}'_{\hat{I}}$. We have to prove that, if we apply the Control Flow Analysis to the matrices $\mathtt{M}'_{\hat{I}}$, $\mathtt{M}_{\hat{S}}$, $\mathtt{M}_{\hat{O}}$, and $\mathtt{M}_{\hat{P}}$ then the set of pairs (l,l') for which the analysis fails are in $\mathtt{B}'_{\hat{I}}$, $\mathtt{B}'_{\hat{S}}$, $\mathtt{B}'_{\hat{O}}$, and $\mathtt{B}'_{\hat{P}}$, respectively. Let us now assume, by contradiction, that there exists a constraint in the analysis that requires a new pair (x, y) that does not belong to $\mathtt{B}'_{\hat{S}}$. Since $\mathtt{B}'_{\hat{S}} \supseteq \mathtt{B}_{\hat{S}}$, we also have that (x, y) does not belong to $\mathtt{B}_{\hat{S}}$. By induction hypothesis, we know that the constraint above requiring (x, y) was not applicable to matrices $\mathtt{M}_{\hat{I}}$, $\mathtt{M}_{\hat{S}}$, $\mathtt{M}_{\hat{O}}$, $\mathtt{M}_{\hat{P}}$ (otherwise the pair (x, y) would necessarily be in $\mathtt{B}_{\hat{S}}$). This means that such a constraint has in the hypothesis the fact that (l,l') belongs to $\hat{I}$ and so that $\mathtt{M}_{\hat{I}}$ [l,l']=`true`. Now, it is sufficient to observe that one round of the algorithm exactly adds to $\mathtt{B}_{\hat{I}}$, $\mathtt{B}_{\hat{S}}$, $\mathtt{B}_{\hat{O}}$, $\mathtt{B}_{\hat{P}}$, and $\mathtt{buf}_{\hat{I}}$, $\mathtt{buf}_{\hat{S}}$, $\mathtt{buf}_{\hat{O}}$, $\mathtt{buf}_{\hat{P}}$ every pair that is required by all the constraints containing "(l,l') $\in \hat{I}$,..." in the hypothesis (this may be verified by considering all the instances of the constraints in which (l,l') is placed in every possible position of the hypothesis). Thus, (x, y) must necessarily be in $\mathtt{B}'_{\hat{S}}$, giving a contradiction. Similarly we would obtain a contradiction by assuming that there the analysis requires a new pair (x, y) that does not belong to $\mathtt{B}'_{\hat{I}}$ (or to $\mathtt{B}'_{\hat{O}}$, or to $\mathtt{B}'_{\hat{P}}$).

  When the algorithm terminates, $\mathtt{B}_{\hat{I}} = \mathtt{M}_{\hat{I}}$, $\mathtt{B}_{\hat{S}} = \mathtt{M}_{\hat{S}}$, $\mathtt{B}_{\hat{O}} = \mathtt{M}_{\hat{O}}$, and $\mathtt{B}_{\hat{P}} = \mathtt{M}_{\hat{P}}$, and the sets $\mathtt{M}_{\hat{I}}$, $\mathtt{M}_{\hat{S}}$, $\mathtt{M}_{\hat{O}}$, and $\mathtt{M}_{\hat{P}}$ constitute a solution of the analysis. Since the procedure is incremental, the analysis produces the least solution.

- **Complexity.** We first recall that all the capability labels are distinct. The external `while`-cycle is executed $S_I^a + S_I^t + S_S + S_O + S_P$ times, since from the definition of the operations $\texttt{push\_c}_{\hat{I}}$, $\texttt{push\_c}_{\hat{S}}$, $\texttt{push\_c}_{\hat{O}}$, and $\texttt{push\_c}_{\hat{P}}$ we have that it is never the case that a pair is inserted twice in one of the $\texttt{buf}_{\hat{I}}$, $\texttt{buf}_{\hat{S}}$, $\texttt{buf}_{\hat{O}}$, and $\texttt{buf}_{\hat{P}}$. Consider all the possible cases of extraction of a pair from $\texttt{buf}_{\hat{I}}$, $\texttt{buf}_{\hat{S}}$, $\texttt{buf}_{\hat{O}}$, and $\texttt{buf}_{\hat{P}}$. Recall that the pairs in $\texttt{buf}_{\hat{I}}$ are either of the form (l,l')$\in \mathsf{Lab}^a(P) \times \mathsf{Lab}^a(P)$ or of the form (l,l')$\in \mathsf{Lab}^a(P) \times \mathsf{Lab}^t(P)$.

  If an element is taken out from $\texttt{buf}_{\hat{I}}$, and it is of the form (l,l')$\in \mathsf{Lab}^a(P) \times \mathsf{Lab}^a(P)$ then in the first `if`-condition the `for`-loop is executed exactly N_t times and each of these iterations has a constant cost. The third `if`-condition requires N_a steps of constant cost. Hence for each pair of the form (l,l')$\in \mathsf{Lab}^a(P) \times \mathsf{Lab}^a(P)$ extracted from $\texttt{buf}_{\hat{I}}$ we have a cost of $N_t + N_a = N_L$ constant steps, i.e., globally $O(S_I^a \cdot N_L)$ steps.

  If an element is taken out from $\texttt{buf}_{\hat{I}}$, and it is of the form (l,l')$\in \mathsf{Lab}^a(P) \times \mathsf{Lab}^t(P)$ (second `if`-condition), then the the unique applicable case is repeated N_a times with constant cost. Hence the complexity is $O(S_I^t \cdot N_a)$ steps. [6]

  If an element is taken out from $\texttt{buf}_{\hat{S}}$, then in the first `if`-condition each iteration of the `for`-loop requires a constant number of steps. Hence, for each pair N_t steps are performed, i.e., globally $O(S_S \cdot N_t)$.

  If an element is taken out from $\texttt{buf}_{\hat{O}}$, then only the fourth external `if`-condition is satisfied and it requires N_a steps of constant cost, i.e., globally $O(S_O \cdot N_a)$.

  If an element is taken out from $\texttt{buf}_{\hat{P}}$, then only the fifth external `if`-condition is satisfied and it requires N_L steps of constant cost, i.e., globally $O(S_P \cdot N_L)$.

  From the above considerations, we obtain the desired time complexity result.

  Space complexity is computed as follows: the data structures used are two $N_a \times N_L$ bit matrices ($\texttt{B}_{\hat{I}}$, $\texttt{M}_{\hat{I}}$), seven $N_L \times N_L$ bit matrices ($\texttt{M}_{\hat{H}}$, $\texttt{B}_{\hat{S}}$, $\texttt{B}_{\hat{O}}$, $\texttt{B}_{\hat{P}}$, $\texttt{M}_{\hat{S}}$, $\texttt{M}_{\hat{O}}$, and $\texttt{M}_{\hat{P}}$), one buffer `cap` containing at most N_t elements of at most $O(\log N_L)$ bits. Finally, $\texttt{buf}_{\hat{I}}$, may contain at most $N_a \cdot N_L$ pairs of $\log N_L$ bits, while $\texttt{buf}_{\hat{S}}$, $\texttt{buf}_{\hat{O}}$, $\texttt{buf}_{\hat{P}}$ may contain at most $3N_a^2$ pairs of $\log N_L$ bits for a total of $O((N_a \cdot N_L)\log N_L)$ bits. □

Corollary 4.4 *Algorithm 2 has time complexity smaller than* $5 \cdot N_a^2 \cdot N_t + 3 \cdot N_a^3$ *steps and it requires* $N_a \cdot N_L \cdot \log N_L + 2 \cdot N_a \cdot N_L + 3 \cdot N_a^2 \cdot \log N_L + 7 \cdot N_a^2$ *bits for space complexity.*

Proof. As far as the time complexity is concerned, it is sufficient to count exactly the number of executions of the loops in the worst-case. While the

[6] Notice that we are implicitly assuming that the `cap[i]` array is indexed by the ℓ^t labels. This can be trivially achieved since such labels are all different.

constants in the space complexity follow from the fact that we have one buffer ($\mathtt{buf}_{\hat{I}}$) and two matrices ($\mathtt{B}_{\hat{I}}$ and $\mathtt{M}_{\hat{I}}$) of dimension $N_a \cdot N_L$, three buffers ($\mathtt{buf}_{\hat{S}}$, $\mathtt{buf}_{\hat{O}}$, and $\mathtt{buf}_{\hat{P}}$), and seven matrices ($\mathtt{B}_{\hat{S}}$, $\mathtt{B}_{\hat{O}}$, $\mathtt{B}_{\hat{P}}$, $\mathtt{M}_{\hat{S}}$, $\mathtt{M}_{\hat{O}}$, $\mathtt{M}_{\hat{P}}$, $\mathtt{M}_{\hat{H}}$) of dimension N_a^2. □

Observe that these space and time complexities may boil down to quadratic and even linear size in the practice, e.g., when few nestings are actually present in the process, or when capabilities belong to few ambients.

The worst-case time complexity of Algorithm 2 is $O(N_a^2 \cdot N_L)$, since $S_I^t \leq N_a \cdot N_t$, while S_I^a, S_S, S_O, $S_P \leq N_a^2$, and $N_L = N_a + N_t$.

5 The BANANA tool: experimental results

The control flow analysis algorithms described in the previous sections have been implemented in the BANANA (Boundary Ambient Nesting ANAlysis) tool, a Java applet available at `http://www.dsi.unive.it/~focardi/BANANA/`.

The main components of BANANA can be summarized as follows:

- A textual and graphical editor for Mobile Ambients, to specify and modify the process by setting ambient nesting capabilities and security attributes in a very user-friendly fashion.
- A parser which checks for syntax errors and builds the syntax tree out of the Mobile Ambient process.
- An analyzer which computes an over approximation of all possible nestings occurring at run-time. The tool supports three different control flow analyses, namely the one of Nielson et al. in [3], the one by Braghin et al. in [5] (called Focardi Cortesi Braghin in the tool), and the one by Braghin et al. in [7] (FCB Boundary Inference). Five different implementations of the analysis described in [3] are available in the tool. They correspond to:
  - a fix-point computation of the least solution of the constraints in Figure 6 (called Nielson in the tool) [7];
  - a fix-point computation of the least solution of the constraints in Figure 7 (Nielson Optimized);
  - Algorithm 1 of Figure 8 (Buffered Boundary Analysis, B.B.A., v1);
  - Algorithm 2 of Figure 10 (B.B.A. v2);
  - Algorithm 2 of Figure 10 with some code optimizations (B.B.A. v2 Optimized).

[7] This implementation does not use the algorithm in [11] and it has a $O(N^5)$ worst-case time complexity.

- A post-processing module, that interprets the results of the analysis in terms of the boundary-based information-flow model proposed in [5], where information flows correspond to leakages of high-level (i.e., secret) ambients out of protective (i.e., boundary) ambients, toward the low-level (i.e., untrusted) environment.
- A detailed output window reporting both the analysis and the security results obtained by the post-processing module, and some statistics about the computational costs of the performed analysis.

Figure 11 gives an overview of the architecture of the tool. BANANA is implemented in Java and strongly exploits the modularity of object-oriented technology, thus allowing scalability to other analyses and extensions of the target language (e.g., [14]). Moreover, BANANA is conceived as an applet based on AWT and thus compatible with the majority of current web browsers supporting Java.

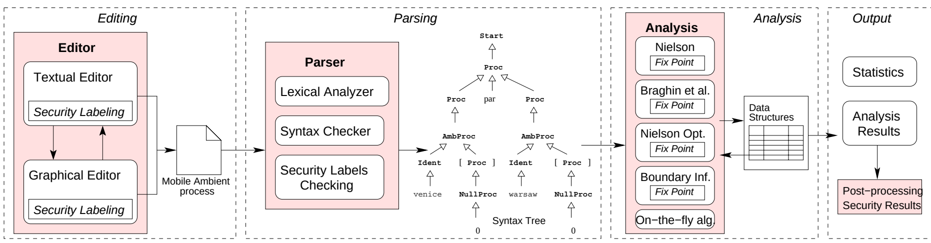

Fig. 11. Overview of the BANANA tool.

A screen-shot of the BANANA tool is shown in Figure 12. A user can edit

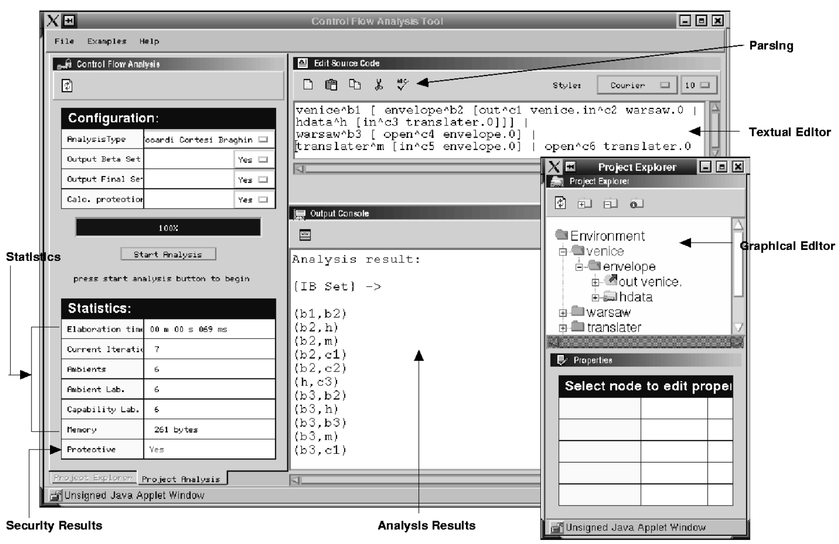

Fig. 12. Screen-shot of the BANANA tool.

the process to be analyzed by using either the Textual or the Graphical Editor. The security labelling (i.e., the labels denoting untrusted, confidential, and boundary ambients) can be inserted directly by the user, or automatically derived by the tool during the parsing phase. In the latter case, ambients starting with letter 'b' are labelled boundaries, while ambients starting with 'h' are labelled high. By selecting an item in the Project Explorer window, the user can check/modify the properties of the ambient/capability. The syntax correctness of the process can be verified by selecting the Parsing button.

The user can then choose to launch one of the algorithms which implement the analysis described in [3], [5], and [7]. Once the analysis has started, the tool parses the process, builds a syntax tree, and computes the algorithm yielding to an over-approximation of all possible ambient nestings. The result of the analysis is reported in the Output Console as a list of pairs of labels.

By post-processing the analysis results, BANANA reports in the filed Protective the sure absence of information leakages.

The BANANA tool has been tested using a suite of use cases consisting of processes differing in the size and number of capabilities. In Table 1 we show some preliminary results obtained by testing the tool on a PII, 300 MHz PC, 192 Mb RAM, OS Windows 98 SE, web browser Microsoft Internet Explorer, Java 1.4.1. As far as the space complexity is concerned, in Table 1 we omit B.B.A. v2 Opt since it uses exactly the same space as B.B.A. v2. In Table 2 we also report the time and space complexity of Nielson and Nielson Opt algorithms corresponding to *direct* fix-point implementations of the control flow analyses of Figure 6 and 7, respectively. Notice that these do not correspond to the algorithms presented in [10] that have not been implemented (yet) in BANANA. Notice also that the space used by these direct fix-point implementations is less than the space used by B.B.A., by a constant factor. This is due to the simpler data structures needed by the two algorithms.

The processes used in the tests are also available in the tool (enabling the reader to exercise them on other machines and operating systems). Let us briefly describe them.

- The process Par. Prot. is the parallel composition of 40 processes of the form

  $$bsite1[\, bmsg[\, hcc[\mathbf{0}] \mid \mathbf{out}\, bsite1 \,.\, \mathbf{in}\, bsite2 \,.\, \mathbf{0}]\,] \mid bsite2[\mathbf{open}\, bmsg \,.\, \mathbf{0}]$$

  without labels. The tool automatically assigns a different label to each ambient. We have that $bmsg$ is opened inside $bside2$, which is boundary since its name starts with b. Hence, the process is secure.
- The process Par. Unprot. is the parallel composition of 40 processes of the form

  $$site1^{ba_i}[\, msg^{bb_i}[\, ccn^{hc_i}[\mathbf{0}] \mid \mathbf{out}^{ta_i}\, site1 \,.\, \mathbf{in}^{tb_i}\, site2 \,.\, \mathbf{0}]\,] \mid$$
  $$site2^{bd_i}[\mathbf{open}^{tc_i}\, msg \,.\, \mathbf{0}],$$

Table 1
Time and Space results for the new algorithms

Process	Time			Space in Bytes	
	B.B.A. v1	B.B.A. v2	B.B.A. v2 Opt	B.B.A. v1	B.B.A. v2
Par. Prot.	08s 070ms	01s 160ms	01s 100ms	173.085	382.588
Par. Unprot.	06s 150ms	00s 930ms	00s 880ms	149.970	331.417
Nested IN	12s 200ms	00s 390ms	00s 330ms	1.771.059	3.542.122

Table 2
Time results for the fix-point implementations

Process	Time		Space in Bytes	
	Nielson	Nielson Opt	Nielson	Nielson Opt
Par. Prot.	02m 43ms 020ms	04s 560ms	85.212	174.999
Par. Unprot.	01m 54s 630ms	03s 570ms	74.170	151.933
Nested IN	26s 970ms	01m 44s 030ms	1.021.751	1.780.778

for $i = 0, \ldots, 39$, and one process of the form

$$site1^{ba}[\ msg^{bb}[\ ccn^{hc}[\mathbf{0}]\ \mid \mathbf{out}^{ta}\ site1\ .\ \mathbf{in}^{tb}\ site2\ .\ \mathbf{0}]\ ]\ \mid$$
$$site2^{low}[\mathbf{open}^{tc}\ msg\ \ \mathbf{0}]\,.$$

The process evolves exactly as Par. Prot., but since msg is opened also inside $site2^{low}$, which is not boundary (its label does not start with b), the process is not secure.

- The process Nested In is of the form

$$amb[\mathbf{in}\ amb_0\ .\ \mathbf{in}\ amb_1\ .\ \mathbf{in}\ amb_2 \ldots \mathbf{in}\ amb_{500}\ .\ \mathbf{0}]\ \mid$$
$$amb_0[\ amb_1[\ amb_2[\ \ldots\ amb_{498}[\ amb_{499}[\ amb_{500}[\mathbf{0}]\ ]\ ]\ \ldots\ ]\ ]\ ]\,.$$

Hence, amb enters in all the amb_i, for $i = 0, \ldots, 500$.

As expected, Algorithm 2 dramatically improves time complexity with respect to Algorithm 1, though a price has to be paid for such an improvement in terms of memory resources.

6 Related Works and Conclusions

Complexity of static analysis is an issue that has attracted many researchers, since seminal papers like [15]. Decidability of analysis has been considered in [16], while the question why certain dataflow analysis problems can be solved efficiently, but not others, is treated in [17]. Focusing on flow-sensitive analyses, the last paper shows that analysis that requires the use of relational attributes for precision must be PSPACE-hard in general, and as soon as the language constructs are slightly strengthened to allow a computation to maintain a very limited summary of what happens along an execution path, inter-procedural analysis becomes EXPTIME-hard. On different perspectives, [18] investigates bottom-up logic programming as a formalism for expressing and analyzing static analysis, while [19,20] investigate the complexity of model checking Mobile Ambients.

As we mentioned in the introduction, [10] is the first contribution facing the issue of estimating the complexity of Control Flow Analysis for Mobile Ambients [1,2], by combining a new optimization technique (sharing and tiling) with previous results on Horn clauses [11]. In [21], Nielson et al. improve on [10] by using a sparsity analysis that results in $O(N \cdot s^3)$ time complexity, where s depends on the solution size. But no improvement in space complexity is achieved. Observe that in our approach, there is no need to translate the problem into Horn clauses, neither of performing asymptotic sparsity analysis. The simplicity of our direct approach allowed us to develop very easy and efficient implementations of the algorithm, now included in BANANA.

We are currently investigating how the method scales up to a class of Control Flow Analyses with particular rule formats. Our claim is that, in this case, the complexity depends both on the size of the solution and on the number of nested quantifiers. This generalization of the method would allow us to obtain algorithms for Control Flow Analyses in different settings and for refinements of the analyses presented in this paper. In particular, it would be interesting to study how the complexity is affected when communication primitives are taken into account and when the analysis is made more precise.

References

[1] L. Cardelli, A. D. Gordon, Mobile Ambients, in: M. Nivat (Ed.), Proc. of Int. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS'98), Vol. 1378 of Lecture Notes in Computer Science, Springer–Verlag, Berlin, 1998, pp. 140–155.

[2] L. Cardelli, A. D. Gordon, Mobile Ambients, Theoretical Computer Science 240 (1) (2000) 177–213.

[3] R. R. Hansen, J. G. Jensen, F. Nielson, H. R. Nielson, Abstract Interpretation of Mobile Ambients, in: A. Cortesi, G. File' (Eds.), Proc. of Static Analysis Symposium (SAS'99), no. 1694 in Lecture Notes in Computer Science, Springer-Verlag, 1999, pp. 134–148.

[4] F. Nielson, R. R. Hansen, H. R. Nielson, Abstract Interpretation of Mobile Ambients, Science of Computer Programming 47 (2003) 145–175, special Issue on Static Analysis.

[5] C. Braghin, A. Cortesi, R. Focardi, Security Boundaries in Mobile Ambients, Computer Languages 28 (1) (2002) 101–127, revised and extended version of [6].

[6] C. Braghin, A. Cortesi, R. Focardi, Control Flow Analysis of Mobile Ambients with Security Boundaries, in: B. Jacobs, A. Rensink (Eds.), Proc. of the 5th IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS'02), Kluwer Academy Publishers, Dordrecht, NL, 2002, pp. 197–212.

[7] C. Braghin, A. Cortesi, R. Focardi, S. van Bakel, Boundary Inference for Enforcing Security Policies in Mobile Ambients, in: R. Baeza-Yates, U. Montanari, N. Santoro (Eds.), Proc. of the 2nd IFIP Int. Conf. on Theoretical Computer Science (IFIP TCS'02), Kluwer Academy Publishers, Dordrecht, NL, 2002, pp. 383–395.

[8] P. Degano, F. Levi, C. Bodei, Safe Ambients: Control Flow Analysis and Security, in: J. He, M. Sato (Eds.), Proc. of the 6th Asian Computing Science Conference (ASIAN'00), Vol. 1961 of Lecture Notes in Computer Science, Springer–Verlag, Berlin, 2000, pp. 199–214.

[9] F. Nielson, H. R. Nielson, R. R. Hansen, J. G. Jensen, Validating Firewalls in Mobile Ambients, in: Proc. of the 10th Int. Conf. on Concurrency Theory (CONCUR'99), no. 1664 in Lecture Notes in Computer Science, Springer–Verlag, Berlin, 1999, pp. 463–477.

[10] F. Nielson, H. Seidl, Control-Flow Analysis in Cubic Time, in: D. Sands (Ed.), Proc. of the 10th European Symposium On Programming (ESOP'01), Vol. 2028 of Lecture Notes in Computer Science, Springer–Verlag, Berlin, 2001, pp. 252–268.

[11] W. F. Dowling, J. H. Gallier, Linear–Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae, Journal of Logic Programming 3 (1984) 267–284.

[12] C. Braghin, A. Cortesi, S. Filippone, R. Focardi, F. L. Luccio, C. Piazza, Banana: A Tool for Boundary Ambients Nesting ANAlysis, in: Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03), Vol. 2619 of Lecture Notes in Computer Science, Springer–Verlag, Berlin, 2003, pp. 437–441.

[13] F. Nielson, H. R. Nielson, C. Hankin, Principles of Program Analysis, Springer–Verlag, Berlin, 1999.

[14] F. Levi, D. Sangiorgi, Controlling Interference in Ambients, in: Proc. of the 28th ACM Symposium on Principles of Programming Languages (POPL'00), ACM Press, New York, 2000, pp. 352–364.

[15] N. Jones, S. Muchnick, Complexity of flow analysis, inductive assertion synthesis, and a language due to Dijkstra, Vol. Program Flow Analysis: Theory and Applications, Prentice-Hall, 1981, Ch. 12, pp. 380–393.

[16] W. Landi, Undecidability of static analysis, ACM Letters on Programming Languages and Systems 1 (4) (1992) 323–337.

[17] R. Muth, S. K. Debray, On the Complexity of Flow-Sensitive Dataflow Analyses, in: Proc. of the 28th ACM Symposium on Principles of Programming Languages (POPL'00), ACM Press, New York, 2000, pp. 67–80.

[18] D. McAllester, On the Complexity Analysis of Static Analyses, Journal of the ACM 49 (4) (2002) 512–537.

[19] W. Charatonik, J.-M. Talbot, The Decidability of Model Checking Mobile Ambients, in: L. Fribourg (Ed.), Proc. of Annual Conference of the European Association for Computer Science Logic (CSL'01), Vol. 2142 of Lecture Notes in Computer Science, Springer–Verlag, Berlin, 2001, pp. 339–354.

[20] W. Charatonik, S. Dal Zilio, A. D. Gordon, S. Mukhopadhyay, J.-M. Talbot, The Complexity of Model Checking Mobile Ambients, in: Proc. of Int. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS'01), Vol. 2030 of Lecture Notes in Computer Science, Springer–Verlag, Berlin, 2001, pp. 152–167.

[21] F. Nielson, H. R. Nielson, H. Seidl, Automatic Complexity Analysis, in: D. L. Metayer (Ed.), Proc. of the 10th European Symposium On Programming (ESOP'02), Vol. 2305 of Lecture Notes in Computer Science, Springer–Verlag, Berlin, 2002, pp. 243–261.