Complexity of Nesting Analysis in Mobile Ambients *

Chiara Braghin[1], Agostino Cortesi[1], Riccardo Focardi[1], Flaminia L. Luccio[2], and Carla Piazza[1]

[1] Dipartimento di Informatica, Università Ca' Foscari di Venezia, {braghin,cortesi,focardi,piazza}@dsi.unive.it
[2] Dipartimento di Scienze Matematiche, Università di Trieste, luccio@dsm.univ.trieste.it

Abstract. A new algorithm is introduced for analyzing possible nesting 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.

Keywords: Static Analysis, Ambient Calculus, Complexity.

1 Introduction

The calculus of Mobile Ambients has been introduced in [6] with the main aim of explicitly modeling mobility. In particular, ambients are arbitrarily nested boundaries 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 [11, 16]. In particular, some analyses have been applied to the verification of security properties [3–5, 10, 18]. The idea of [4, 5, 18] is to compute an over-approximation of ambient nestings that may occur during process 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 [4, 11, 16], requires considerably high complexities, thus the design of efficient techniques turns out to be very important. This is the main motivation behind [19], where Nielson and Seidl reduce the worst-case time complexity of [11] from $O(N^5)$ to $O(N^3)$ steps, with N being the size of the analyzed process.

* Partially supported by MIUR Projects "Interpretazione Astratta, Type Systems e Analisi Control-Flow", and "Modelli formali per la sicurezza", the EU Contract IST-2001-32617 MyThs, and project "Matematica per le scienze e la tecnologia", Università di Trieste.

The first contribution of this paper is to refine the complexity results of [19], 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 [19], 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. This is due to the fact that it is based on a translation of the Control Flow Analysis constraints into Horn clauses, which are processed through satisfiability standard algorithms in order to compute the least analysis 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 [19].

The second contribution of this paper is to propose two new algorithms that improve both time and space complexities of the ones proposed in [19].

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 [19]. 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 [19], with our algorithm we also obtain a significant reduction of the average execution time.

In order to get these complexity improvements, we first apply our new technique to the less efficient $O(N^4)$ algorithm of [19]. 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 a first step towards the definition of a more general technique that could be applicable to compute Control Flow Analyses in different settings, and more abstractly, to efficiently compute the least solution of a set of Horn clauses over a finite domain, as done in [19]. This is the object of our current research.

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 [11]. In Section 3 we study in depth the complexity of the algorithms presented in [19]. Then, in Section 4, we present our algorithms and complexity results. Section 5 concludes the paper with final remarks.

2 Background: Mobile Ambients

The Mobile Ambient calculus has been introduced in [6] 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.

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

Fig. 1. Mobile Ambients Syntax

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 (see [6] for more details).

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 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 operator occurrences in P. Note that $N_L < N$, as there is at least one occurrence of $\mathbf{0}$ in every non-empty process.

Example 1. Process P_1 models a *cab* driving a *client* from $site_1$ to $site_2$:

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

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:

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

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 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:

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

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

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 [11] 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 $\hat{I}$.

(res)	$\beta_\ell^{\mathrm{CF}}((\nu n)P)$	$= \beta_\ell^{\mathrm{CF}}(P)$
$(zero)$	$\beta_\ell^{\mathrm{CF}}(\mathbf{0})$	$= (\emptyset, \emptyset)$
(par)	$\beta_\ell^{\mathrm{CF}}(P \mid Q)$	$= \beta_\ell^{\mathrm{CF}}(P) \sqcup \beta_\ell^{\mathrm{CF}}(Q)$
$(repl)$	$\beta_\ell^{\mathrm{CF}}(!P)$	$= \beta_\ell^{\mathrm{CF}}(P)$
(amb)	$\beta_\ell^{\mathrm{CF}}(n^{\ell^a}[\,P\,])$	$= \beta_{\ell^a}^{\mathrm{CF}}(P) \sqcup (\{(\ell, \ell^a)\}, \{(\ell^a, n)\})$
(in)	$\beta_\ell^{\mathrm{CF}}(\mathbf{in}^{\ell^t}\ n\ .\ P)$	$= \beta_\ell^{\mathrm{CF}}(P) \sqcup (\{(\ell, \ell^t)\}, \emptyset)$
(out)	$\beta_\ell^{\mathrm{CF}}(\mathbf{out}^{\ell^t}\ n\ .\ P)$	$= \beta_\ell^{\mathrm{CF}}(P) \sqcup (\{(\ell, \ell^t)\}, \emptyset)$
$(open)$	$\beta_\ell^{\mathrm{CF}}(\mathbf{open}^{\ell^t}\ n\ .\ P)$	$= \beta_\ell^{\mathrm{CF}}(P) \sqcup (\{(\ell, \ell^t)\}, \emptyset)$

Fig. 2. Representation Function for the Control Flow Analysis

- 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 $\hat{H}$. [3]
- The pairs are component-wise partially ordered by set inclusion.

The analysis is defined as usual by a representation function and a specification [17]. They are recalled in Figure 2 and Figure 3, 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 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 $\beta_{env}^{\mathrm{CF}}(P)$.

Example 2. Let P_2 be the process $n^{\ell_1^a}[\ m^{\ell_2^a}[\,\mathbf{out}^{\ell^t}\ n.\mathbf{0}\,]\ ]$. The representation function of P_2 is : $\beta_{env}^{\mathrm{CF}}(P_2) = (\{(env, \ell_1^a), (\ell_1^a, \ell_2^a), (\ell_2^a, \ell^t)\}, \{(\ell_1^a, n), (\ell_2^a, m)\})$. 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., $\{(\ell_1^a, n), (\ell_2^a, m)\}$. □

The specification depicts how the process transforms one abstract representation into another one, and 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.

[3] 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 [16]. In [11, 19], an alternative treatment of α-equivalence is used, where bound names are annotated with markers, and a marker environment *me* is associated to constraints.

(res)	$(\hat{I},\hat{H}) \models^{\mathrm{CF}} (\nu n)P$	iff $(\hat{I},\hat{H}) \models^{\mathrm{CF}} P$
$(zero)$	$(\hat{I},\hat{H}) \models^{\mathrm{CF}} \mathbf{0}$	always
(par)	$(\hat{I},\hat{H}) \models^{\mathrm{CF}} P \mid Q$	iff $(\hat{I},\hat{H}) \models^{\mathrm{CF}} P \ \wedge\ (\hat{I},\hat{H}) \models^{\mathrm{CF}} Q$
$(repl)$	$(\hat{I},\hat{H}) \models^{\mathrm{CF}} !P$	iff $(\hat{I},\hat{H}) \models^{\mathrm{CF}} P$
(amb)	$(\hat{I},\hat{H}) \models^{\mathrm{CF}} n^{\ell^a}[\,P\,]$	iff $(\hat{I},\hat{H}) \models^{\mathrm{CF}} P$
(in)	$(\hat{I},\hat{H}) \models^{\mathrm{CF}} \mathbf{in}^{\ell^t}\ n\ .\ P$	iff $(\hat{I},\hat{H}) \models^{\mathrm{CF}} P\ \wedge$ $\forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathsf{Lab}^a(P) : ((\ell^a,\ell^t) \in \hat{I} \ \wedge\ (\ell^{a''},\ell^a) \in \hat{I} \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}$ $\wedge\ (\ell^{a'},n) \in \hat{H}) \implies (\ell^{a'},\ell^a) \in \hat{I}$
(out)	$(\hat{I},\hat{H}) \models^{\mathrm{CF}} \mathbf{out}^{\ell^t}\ n\ .\ P$	iff $(\hat{I},\hat{H}) \models^{\mathrm{CF}} P\ \wedge$ $\forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathsf{Lab}^a(P) : ((\ell^a,\ell^t) \in \hat{I} \ \wedge\ (\ell^{a'},\ell^a) \in \hat{I} \ \wedge\ (\ell^{a''},\ell^{a'}) \in \hat{I}$ $\wedge\ (\ell^{a'},n) \in \hat{H}) \implies (\ell^{a''},\ell^a) \in \hat{I}$
$(open)$	$(\hat{I},\hat{H}) \models^{\mathrm{CF}} \mathbf{open}^{\ell^t}\ n\ .\ P$	iff $(\hat{I},\hat{H}) \models^{\mathrm{CF}} P\ \wedge$ $\forall \ell^a, \ell^{a'} \in \mathsf{Lab}^a(P), \forall \ell' \in \mathsf{Lab}(P) : ((\ell^a,\ell^t) \in \hat{I} \ \wedge\ (\ell^a,\ell^{a'}) \in \hat{I}$ $\wedge\ (\ell^{a'},n) \in \hat{H} \ \wedge\ (\ell^{a'},\ell') \in \hat{I}) \implies (\ell^a,\ell') \in \hat{I}$

Fig. 3. Specification of the Control Flow Analysis

The crucial result is that whenever $(\hat{I},\hat{H}) \models^{\mathrm{CF}} P$ and the representation of P is contained in $(\hat{I},\hat{H})$, we are assured that every nesting of ambients and capabilities in every possible derivative of P is also captured in $(\hat{I},\hat{H})$.

Example 3. Consider again process P_2 of Example 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 $\hat{I}= \{(env,\ell_1^a),(env,\ell_2^a),(\ell_1^a,\ell_2^a),(\ell_2^a,\ell^t)\}$, and $\hat{H} = \{(\ell_1^a,n),(\ell_2^a,m)\}$. Notice that the analysis correctly captures through the pair (env,ℓ_2^a) the possibility for m to exit from n. □

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 [19] 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 algorithms of [19] with our new ones that will be given 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 [19]. NS1 is based on a formulation of the analysis which is equivalent to the one presented in the previous section. The constraints in Figure 3 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 $\mathsf{Lab}^t(P)$;
- a constraint for an *open*-capability involves two universal quantifications that range over $\mathsf{Lab}^a(P)$, whose cardinality is N_a, plus another universal quantification that ranges over $\mathsf{Lab}(P)$, whose cardinality is N_L. Constraints for *in* and *out*-capabilities have three universal quantifications ranging over $\mathsf{Lab}^a(P)$.

Since $\mathsf{Lab}^a(P) \subseteq \mathsf{Lab}(P)$, 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 [9] (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 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 4. 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

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

In order to generate the Horn clauses ℓ^a and $\ell^{a'}$ have to be instantiated in all the possible ways in the set $\mathsf{Lab}^a(P) = \{\ell_1^a, \ell_2^a\}$, whose cardinality is $N_a = 2$, and ℓ' ranges over $\mathsf{Lab}(P) = \{\ell_1^a, \ell_2^a, \ell^t\}$, 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.,

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

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 obtain the following:

Corollary 1. *Algorithm NS1 performs at least $N_t \cdot N_a^3$ steps and uses at least $N_t \cdot N_a^3 \log N_L$ bits.*

3.2 The second Algorithm of Nielson and Seidl – NS2

We now consider NS2, the cubic-time algorithm presented in [19]. It is based on an optimization of the analysis of Figure 3 reported in Figure 4 [4]. The equivalence between the analysis of Figures 3 and 4 follows from [19]. 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. It requires to find three labels ℓ^a, $\ell^{a'}$, $\ell^{a''} \in \mathsf{Lab}^a(P)$ 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 $\hat{S} \in \wp(\mathsf{Lab}^a(P) \times \mathsf{Lab}^a(P))$ containing all the pairs of labels corresponding to siblings 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 $\hat{S}$, it is now required a new global constraint (*global*, in Figure 4): $((\ell^{a''}, \ell^a) \in \hat{I} \ \wedge \ (\ell^{a''}, \ell^{a'}) \in \hat{I}) \implies (\ell^a, \ell^{a'}) \in \hat{S}$. Similar optimizations are applied to the other constraints, by introducing the components $\hat{O}, \hat{P} \in \wp(\mathsf{Lab}^a(P) \times \mathsf{Lab}^a(P))$, 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 [9] 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 $\mathsf{Lab}^a(P)$, whose size is N_a;
- the first two constraints in the (*global*) rule involve three universal quantifications over $\mathsf{Lab}^a(P)$;
- the third constraint in the (*global*) rule involves two universal quantifications over $\mathsf{Lab}^a(P)$, and one over $\mathsf{Lab}(P)$, 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 2. *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.*

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

Corollary 2. *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] In [19] 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. 4. The Optimized Control Flow Analysis

4 The new algorithms

In this section, we present our new algorithms for nesting analysis of Figures 3 and 4, 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 1 and 2).

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 not translate constraints into Horn clauses and neither apply the algorithm of [9]. The algorithms start with an empty analysis $\hat{I}$ and with a buffer containing all the pairs corresponding to the initial process representation [5]. Recall that these pairs should be contained in the final $\hat{I}$ in order for the analysis to be correct. 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 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 is 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 [6]. 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 $S_I^a = S_S = S_O = S_P = N_a^2$ 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 3) 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 are presently trying to estimate this number of bad cases.

We now present the two algorithms in detail.

---

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

[6] 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.

Algorithm 1

Our first algorithm, called Algorithm 1, is depicted in Figure 5. 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 [7]. 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 5. Let P be the Firewall Access process of [6], where an agent crosses a firewall by means of previously arranged passwords k, k' and k'' (see [18] for a detailed analysis of the security issues related to this example):

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

The least solution of P, as computed using the specification of the Control Flow Analysis depicted in Figure 3, is the pair $(\hat{I}, \hat{H})$, where:

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

Let us see how Algorithm 1 applies to process P. In this case, $N_a = 5$ and $N_t = 6$, 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 `cap` an array of length 6, initialized as $\langle \mathbf{out}^{t1}\, w, \mathbf{in}^{t2}\, k', \mathbf{in}^{t3}\, w, \mathbf{open}^{t4}\, k', \mathbf{open}^{t5}\, k''$, $\mathbf{open}^{t6}\, k\rangle$. 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 $\langle (env, a1)$, $(env, a3)$, $(a1, a2)$, $(a3, a4), (a1, t4), (a1, t5), (a2, t1), (a2, t2), (a2, t3)$, $(a3, t6)\rangle$.

[7] n here represents an integer $1 \le n \le N_a$ 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^{ℓ^t} 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^{ℓ^t} 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^{ℓ^t} 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. 5. Algorithm 1

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 `cap`[1] is "$\mathbf{out}^{t1}\, w$". Thus, we have l' $= l^t$ and $n = w$. Since l' $\in \mathsf{Lab}^t(P)$ and l' $= l^t$, 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. □

We recall that 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. We can prove the following result:

Theorem 1. *Algorithm 1 is correct. It has time complexity of* $O(S_I^a \cdot N_t \cdot N_L + S_I^t \cdot N_a \cdot N_L)$ *steps and a space complexity of* $O((N_a \cdot N_L) \log N_L)$ *bits.*

Sketch of the Proof. The proof mainly follows two steps. First, we show the following invariant on the outermost `while`−loop:

> *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., required to be in* $\hat{I}$*, but such that* $\mathtt{M}_{\hat{I}}$ *[l,l']=*`false`*) are in* $\mathtt{B}_{\hat{I}}$ *.*

Then we prove that the algorithm terminates when the buffer is empty, i.e., when $\mathtt{M}_{\hat{I}} = \mathtt{B}_{\hat{I}}$. The invariant proves that, in such a situation, $\mathtt{M}_{\hat{I}}$ is a correct analysis. Indeed, assume it is not correct, then all the (l,l') for which the analysis fails must be in $\mathtt{B}_{\hat{I}}$, and thus also in $\mathtt{M}_{\hat{I}}$, giving a contradiction. □

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 $S_t^I = N_a \cdot N_t$.

Algorithm 2

Time complexity of Algorithm 1 can be reduced by applying buffering techniques also to the optimized analysis of Figure 4. This leads to our second algorithm, called Algorithm 2 and depicted in Figure 6. 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. 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 4.

Let S_I^a and S_I^t be as defined above, and S_S, S_O, S_P be the cardinality of $\hat{S}$, $\hat{O}$, $\hat{P}$ at the end of the execution, respectively, then we can prove the following theorem, whose proof follows the lines of Theorem 1:

Theorem 2. *Algorithm 2 is correct. It has a time complexity of* $O(S_I^a \cdot N_L + S_I^t \cdot N_a + S_S \cdot N_t + S_O \cdot N_a + S_P \cdot N_L)$ *steps and a worst-case space complexity of* $O((N_a \cdot N_L) \log N_L)$ *bits.*

while ($\mathtt{buf}_{\hat{I}}$!= NIL or $\mathtt{buf}_{\hat{S}}$!= NIL or $\mathtt{buf}_{\hat{O}}$!= NIL or $\mathtt{buf}_{\hat{P}}$!= NIL) do
  if $\mathtt{buf}_{\hat{I}}$!= NIL then
    (l,l') := $\mathtt{pop}_{\hat{I}}$ (); $\mathtt{M}_{\hat{I}}$ [l,l'] := true; b:= "I"
  else if $\mathtt{buf}_{\hat{S}}$!= NIL then
    (l,l') := $\mathtt{pop}_{\hat{S}}$ (); $\mathtt{M}_{\hat{S}}$ [l,l'] := true; b:= "S"
  else if $\mathtt{buf}_{\hat{O}}$!= NIL then
    (l,l') := $\mathtt{pop}_{\hat{O}}$ (); $\mathtt{M}_{\hat{O}}$ [l,l'] := true; b:= "O"
  else if $\mathtt{buf}_{\hat{P}}$!= NIL then
    (l,l') := $\mathtt{pop}_{\hat{P}}$ (); $\mathtt{M}_{\hat{P}}$ [l,l'] := true; b:= "P";
  if (b="I" or b="S") then for i := 1 to N_t do
    case cap[i] of:
      $\mathbf{in}^{\ell^t}$ n: if (b="I" and l' $\in$ Lab^t(P) and l' = l^t)
        then for j := 1 to N_a do
          if ($\mathtt{M}_{\hat{S}}$ [l,j] and $\mathtt{M}_{\hat{H}}$ [j,n]) then $\mathtt{push\_c}_{\hat{I}}$ (j,l)
        else if (b="S" and $\mathtt{M}_{\hat{I}}$ [l,l^t] and $\mathtt{M}_{\hat{H}}$ [l',n] then $\mathtt{push\_c}_{\hat{I}}$ (l',l);
      $\mathbf{out}^{\ell^t}$ n: if (b="I" and l' $\in$ Lab^t(P) and l' = l^t)
        then for j := 1 to N_a do
          if ($\mathtt{M}_{\hat{I}}$ [j,l] and $\mathtt{M}_{\hat{H}}$ [j,n]) then $\mathtt{push\_c}_{\hat{O}}$ (j,l)
        else if (b="I" and l' $\in$ Lab^a(P) and $\mathtt{M}_{\hat{I}}$ [l',l^t] and $\mathtt{M}_{\hat{H}}$ [l,n] then $\mathtt{push\_c}_{\hat{O}}$ (l,l');
      $\mathbf{open}^{\ell^t}$ n: if (b="I" and l' $\in$ Lab^t(P) and l' = l^t)
        then for j := 1 to N_a do
          if ($\mathtt{M}_{\hat{I}}$ [l,j] and $\mathtt{M}_{\hat{H}}$ [j,n]) then $\mathtt{push\_c}_{\hat{P}}$ (l,j)
        else if (b="I" and l' $\in$ Lab^a(P) and $\mathtt{M}_{\hat{I}}$ [l,l^t] and $\mathtt{M}_{\hat{H}}$ [l',n] then $\mathtt{push\_c}_{\hat{P}}$ (l,l');
  if b="I" then for j:= 1 to N_a do
    if $\mathtt{M}_{\hat{I}}$ [l,j] then $\mathtt{push\_c}_{\hat{S}}$ (l',j);
    if $\mathtt{M}_{\hat{O}}$ [l',j] then $\mathtt{push\_c}_{\hat{I}}$ (l,j);
    if $\mathtt{M}_{\hat{P}}$ [j,l] then $\mathtt{push\_c}_{\hat{I}}$ (j,l');
  if b="O" then for j:= 1 to N_a do
    if $\mathtt{M}_{\hat{I}}$ [j,l] then $\mathtt{push\_c}_{\hat{I}}$ (j,l');
  if b="P" then for j:= 1 to N_L do
    if $\mathtt{M}_{\hat{I}}$ [l',j] then $\mathtt{push\_c}_{\hat{I}}$ (l,j);

Fig. 6. Algorithm 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 $N_L = N_a + N_t$.

The following corollary further refines the result of Theorem 2, showing that the actual impact of the multiplicative constants is negligible.

Corollary 3. *Algorithm 2 has time complexity smaller than* $5 \cdot N_a^2 \cdot N_t + 3 \cdot N_a^3$ *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.*

5 Related Works and Conclusions

Complexity of static analysis is an issue that has attracted many researchers, since seminal papers like [12]. Decidability of analysis has been considered in [13], while the question why certain dataflow analysis problems can be solved efficiently, but not others, is treated in [15]. 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, [14] investigates bottom-up logic programming as a formalism for expressing and analyzing static analysis, while [7, 8] investigate the complexity of model checking Mobile Ambients.

As we mentioned in the introduction, [19] is the first contribution tackling the issue of estimating the complexity of Control Flow Analysis for Mobile Ambients [6], by combining a new optimization technique (sharing and tiling) with previous results on Horn clauses [9]. In [20], Nielson et al. improve on [19] 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 allows very easy and efficient implementations of the algorithm. We are presently working on a prototype verifier that will be soon available on-line.

The extension of our results to more general scenarios is the main topic of our current research.

References

1. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static Analysis for the π-calculus with Applications to Security. *Information and Computation*, 165:68–92, 2001.
2. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Control Flow Analysis for the π-calculus. In D. Sangiorgi, R. de Simone, editors, *Proc. of International Conference on Concurrency Theory (CONCUR)*, LNCS 1466, pages 84–98. Springer-Verlag, 1998.
3. C. Braghin, A. Cortesi, and R. Focardi. Security Boundaries in Mobile Ambients. *Computer Languages*, Elsevier, 28(1):101–127, Nov. 2002.
4. C. Braghin, A. Cortesi, and R. Focardi. Control Flow Analysis of Mobile Ambients with Security Boundaries. In B. Jacobs and A. Rensink, editors, *Proc. of Fifth Int. Conf. on Formal Methods for Open Object-Based Distributed Systems*, pages 197–212. Kluwer Academic Publisher, 2002.
5. C. Braghin, A. Cortesi, R. Focardi, and S. van Bakel. Boundary Inference for Enforcing Security Policies in Mobile Ambients. In *Proc. of The 2nd IFIP International Conference on Theoretical Computer Science (TCS)*, Kluwer Academic Publisher, pages 383–395, 2002.

6. L. Cardelli and A.D. Gordon. Mobile Ambients. Theoretical Computer Science (TCS), 240(1):177–213, 2000.
7. W. Charatonik and J. Talbot. The Decidability of Model Checking Mobile Ambients. In L. Fribourg, editor, *Proc. of Annual Conference of the European Association for Computer Science Logic (CSL)*, LNCS 2142, pages 339–354. Springer-Verlag, 2001.
8. W. Charatonik, S. Dal Zilio, A.D. Gordon, S. Mukhopadhyay, and J. Talbot. The Complexity of Model Checking Mobile Ambients. In F. Honsell, M. Miculan, eds., *Prof. of Int. Conf. on Foundations of Software Science and Computation Structures (FoSSaCS)*, LNCS 2030, pp. 152–167. Springer, 2001.
9. 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.
10. P. Degano, F. Levi, and C. Bodei. Safe Ambients: Control Flow Analysis and Security. In Jifeng He and Masahiko Sato, editors, *Proc. of Advances in Computing Science - 6th Asian Computing Science Conference, Penang, Malaysia (ASIAN)*, LNCS 1961, pp. 199–214. Springer, 2000.
11. R. R. Hansen, J. G. Jensen, F. Nielson, and H. Riis Nielson. Abstract Interpretation of Mobile Ambients. In A. Cortesi and G. File', eds., *Proc. of Static Analysis Symposium (SAS)*, LNCS 1694, pp. 134–148. Springer, 1999.
12. N.D. Jones and S.S. Muchnick. Complexity of flow analysis, inductive assertion synthesis, and a language due to Dijkstra. In S.S. Muchnick and N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, chapter 12, pages 380–393. Prentice-Hall, 1981.
13. W. Landi. Undecidability of static analysis. *ACM Letters on Programming Languages and Systems*, 1(4):323–337, December 1992.
14. D.A.McAllester. On the Complexity Analysis of Static Analyses. In A. Cortesi and G. File', eds., *Proc. of Static Analysis Symposium (SAS)*, LNCS 1694, pp. 312–329. Springer, 1999.
15. R. Muth and S. K. Debray. On the Complexity of Flow-Sensitive Dataflow Analyses. In *Proc. of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 67–80. ACM Press, N.Y., U.S.A., January 2000.
16. F. Nielson, R. R. Hansen, and H. Riis Nielson. Abstract Interpretation of Mobile Ambients. *Science of Computer Programming*, Issue on Static Analysis edited by A. Cortesi and G. File', to appear, 2003.
17. F. Nielson, H. Riis Nielson, C.L. 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 J.C.M. Baeten, S. Mauw, editors, *Proc. of International Conference on Concurrency Theory (CONCUR)*, LNCS 1664, pages 463–477. Springer-Verlag, 1999.
19. F. Nielson and H. Seidl. Control-flow Analysis in Cubic Time. In D. Sands, ed., *Proc. of European Symposium On Programming (ESOP)*, LNCS 2028, pp. 252–268. Springer, 2001.
20. F. Nielson, H. Riis Nielson, and H. Seidl. Automatic Complexity Analysis. In D. Le Metayer, ed.,*Proc. of European Symposium On Programming (ESOP)*, LNCS 2305, pp.243–261. Springer, 2002.