Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 99 (2004) 319–337
www.elsevier.com/locate/entcs

Behind BANANA: Design and Implementation of a Tool for Nesting Analysis of Mobile Ambients[⋆]

C. Braghin[a,1] , A. Cortesi[a,2] , R. Focardi[a,3] , F. L. Luccio[b,4] , and C. Piazza[a,5]

[a] *Dipartimento di Informatica, Università Ca' Foscari di Venezia*
[b] *Dipartimento di Scienze Matematiche, Università di Trieste*

Abstract

We present a survey of the work on control-flow analysis carried on by the Venice Team during the Mefisto project. We study security issues, in particular information leakage detection, in the context of the Mobile Ambient calculus. We describe BANANA, a Java-based tool for ambient nesting analysis, by focussing on analysis accuracy and algorithmic optimizations.

Keywords: Static Analysis, Ambient Calculus, Security.

1 Introduction

In recent years networks have become an essential aspect of our daily computing environment. In such a framework, mobility may be supported in different

[⋆] Partially supported by the MIUR Project "Modelli formali per la sicurezza", the EU Contract IST-2001-32617 "MyThS", and the FIRB project (RBAU018RCZ) "Interpretazione astratta e model checking per la verifica di sistemi embedded".
[1] Email:braghin@dsi.unive.it
[2] Email:cortesi@dsi.unive.it
[3] Email:focardi@dsi.unive.it
[4] Email:luccio@dsm.univ.trieste.it
[5] Email:piazza@dsi.unive.it

forms: physical, e.g., laptops, personal organizers moving around, smart cards entering a network computer slot, and logical, e.g., code or agent mobility.

The main scientific and technological challenge in this setting is to provide security to applications which operate in untrusted and possibly unknown environments. Indeed, distributed applications rely on network communications thus, apart from being potential victims of computer viruses, they can be tampered and eavesdropped in many ways, and for this reason they are vulnerable and insecure. In general, mobile code and communication may need to cross administrative domains and firewalls, and untrusted sites as well. Techniques such as cryptography may solve some problems related to data communication, but, when executable code is exchanged, such techniques may not be sufficient anymore. Much research effort has been devoted to protect servers from untrusted and potentially malicious mobile code. On the converse, mobile code must be protected from potentially malicious servers that may try to disclose or modify its sensible private data.

These security issues constitute a very interesting workbench to evaluate the theoretical and practical impact of static analysis techniques. Giving a way for statically verifying a security property has, in principle, the advantage of making the checking of the properties more efficient; moreover it allows us to write programs which are secure-by-construction, e.g., when the performed analysis is proved to imply some behavioural security properties. A substantial body of research is available, relying on different static techniques and modeling languages, verifying different security properties [3,14,16,25].

In this paper we present a survey of the work on control-flow analysis carried on by the Venice Team during the Mefisto project. The goal was twofold: on one side, to tackle information flow security of mobile code with control flow analysis techniques, on the other side, to optimize time and space complexities of our and existing approaches.

Our approach is based on the calculus of Mobile Ambients (MA), proposed by Cardelli and Gordon [11], where the notion of *ambient* captures the structure and properties of wide area networks, of mobile computing, and of mobile computation. An ambient is an abstraction of the notion of both agent and location. Like an agent, an ambient may move around by exploiting its capabilities. Furthermore, ambients can be arbitrarily nested, forming a tree structure that models the hierarchical organization of administrative domains. In order to study this problem in an abstract manner, we consider the "pure" Mobile Ambients calculus, with no communication channel and where the only possible actions are represented by the moves performed by mobile processes. In this way, we may study a very general notion of information flow which, in our opinion, should be applicable also to more "concrete" versions of the calculus [9,17].

Our focus is on *Multilevel Security*, a particular *Mandatory Access Control* security policy: every entity is bound to a security level (for simplicity, we consider only two levels: *high* and *low*), and information may only flow from the low level to the high one. Typically, two access rules are imposed: (*i*) *No Read Up*, a low level entity cannot access information of a high level entity; (*ii*)*No Write Down*, a high level entity cannot leak information to a low level entity. In order to detect information leakages, a typical approach (see, e.g., [1,2]) consists of directly defining how information may flow from one level to another one. Then, it is sufficient to verify if, in any system execution, no flow of information is possible from level *high* to level *low*. We have followed the above approach.

The starting point of our research was the Control Flow Analysis by Nielson *et al.* presented in [15]. In [7], we refined such an analysis in order to detect information leakage. Both analyses have been implemented in the Banana(Boundary Ambient Nesting ANAlysis) tool [4], a Java applet available at `http://www.dsi.unive.it/~mefisto/BANANA/`. This survey will describe both the technical details of the analyses implemented in the tool, and the implementation issues behind Banana.

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 analyses of [15,20] and of [6,7]. Then, in Section 3, we present the algorithms implemented in the Banana tool, which is described in detail in Section 4. Section 5 concludes the paper.

2 Information Flow in Mobile Ambients

In this section we first introduce the basic terminology of Mobile Ambient calculus, then we briefly report the Control Flow Analysis of [15,20] and its refinement proposed in [6,7].

2.1 Mobile Ambients

The Mobile Ambient calculus has been introduced in [10,11] with the main purpose of explicitly modeling mobility. Indeed, ambients are arbitrarily nested boundaries which can move around through suitable capabilities. The syntax of processes is given as follows, 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

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

Fig. 1. Mobile Ambients Syntax

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 $\rightarrow$ between processes. Intuitively, $P \rightarrow Q$ represents the possibility for P of reducing to Q through some computation. (See [10] for more details.) 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". 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 $N_{Lab} = |\mathsf{Lab}(P)| = N_a + N_t$. With N we denote the global number of operators occurring in P. Note that $N_{Lab} < N$, as there is at least one occurrence of $\mathbf{0}$ in every non-empty process.

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.

Example 2.1 Let P_1 be a process modeling an *envelope* sent from *venice* to *pisa*:

$$venice[\ envelope[\ \mathbf{out}\ venice\ .\ \mathbf{in}\ pisa\ .\ \mathbf{0}\ ]\ |\ Q\ ]\ |$$
$$pisa[\ \mathbf{open}\ envelope\ .\ \mathbf{0}\ ]$$

Initially, *envelope* is in site *venice*. Then, it exits *venice* and enters site *pisa* by applying its capabilities **out** *venice* and **in** *pisa*, respectively. Once site *pisa* receives *envelope*, it reads its content by consuming its **open** *envelope* capability. Finally, process P_1 reaches the state: $venice[\ Q\ ]\ |\ pisa[\ \mathbf{0}\ ]$. □

In order to deal with security issues, information is classified into different levels of confidentiality. This is obtained by exploiting the labelling of ambients. In particular, the set of ambient labels is partitioned into three disjoint sets: *high*, *low* and *boundary* labels (Lab^a_H, Lab^a_L and Lab^a_B, respectively). Ambients labelled with boundary labels (*boundary ambients*) are the ones responsible for confining confidential information. Information leakage occurs if a high level ambient exits a boundary, thus becoming possibly exposed to a malicious ambient attack.

Example 2.2 Let P_2 be a labelled extension of process P_1, in which the *envelope* contains confidential data *hdata* (labelled *high*) which need to be safely sent from *venice* to *pisa*.

$$venice^{b_1}[\ envelope^{b_2}[\ \mathbf{out}^{c_1}\ venice\ .\ \mathbf{in}^{c_2}\ pisa\ .\ \mathbf{0}\ |\ hdata^{h}[\ \mathbf{0}\ ]\ ]\ ]\ |$$
$$pisa^{b_3}[\ \mathbf{open}^{c_3}\ envelope\ .\ \mathbf{0}\ ]$$

In this case, *venice*, *pisa* and *envelope* must be labelled *boundary* to protect *hdata* during the whole execution. □

The flow of high level data/ambients outside security boundaries, which we want to detect and avoid, may be formalised as follows.

Definition 2.3 [Information Leakage] Given a process P and a labelling Lab^a(P), P does not leak a secret h $\in \mathsf{Lab}^a_H(P)$ if and only if $\forall Q$ such that $P \to^* Q$, all occurrences of h are contained inside, i.e., "protected by", at least one boundary ambient.

2.2 Control Flow Analysis

The Control Flow Analysis (CFA for short) of a process P described in [15] 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 labelled ℓ^a having inside either a capability or an ambient labelled 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 labelled ℓ^a with name n, then (ℓ^a, n) is expected to belong to $\hat{H}$. [6]
- The pairs are component-wise partially ordered by set inclusion.

The analysis is defined as usual by a representation and a specification functions [21]. They are recalled in Figure 2 and Figure 3, respectively, where $\sqcup$ denotes the component-wise union of the elements of the pairs.

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

Fig. 2. Representation Function for the Control Flow Analysis

The representation function aims at mapping concrete values to their best abstract representation. It is given in terms of a function $\beta^{\mathrm{CF}}_{\ell}(P)$ which maps process P into a pair $(\hat{I},\hat{H})$ corresponding to the initial state of P, with respect to an enclosing ambient labelled with ℓ. The representation of a process P is defined as $\beta^{\mathrm{CF}}_{env}(P)$.

Example 2.4 Let P_3 be the process $n^{\ell^a_1}[\, m^{\ell^a_2}[\mathbf{out}^{\ell^t}\, n.\mathbf{0}\,]\,]$. The representation function of P_3 is $\beta^{\mathrm{CF}}_{env}(P_3) = (\{(env, \ell^a_1), (\ell^a_1, \ell^a_2), (\ell^a_2, \ell^t)\}, \{(\ell^a_1, n), (\ell^a_2, m)\})$. Notice that all ambient nestings are captured by the first component $\{(env, \ell^a_1)$, $(\ell^a_1, \ell^a_2), (\ell^a_2, \ell^t)\}$, while all the correspondences between ambients and labels of P_3 are kept by the second one, i.e., $\{(\ell^a_1, n), (\ell^a_2, m)\}$. $\square$

[6] 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 [20]. In [15,23], 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}) & (\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})\ \Longrightarrow\ (\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})\ \Longrightarrow\ (\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})\quad \Longrightarrow\ (\ell^a,\ell') \in \hat{I}}
\end{array}
$$

Fig. 3. Specification of the Control Flow Analysis

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 labelled ℓ^a has an *open*-capability ℓ^t on an ambient n, that may apply due to the presence of a sibling ambient labelled $\ell^{a'}$ 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.

Example 2.5 Consider again process P_3 of Example 2.4. 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_3 is $(\hat{I},\hat{H})$, where $\hat{I} = \{(\mathit{env},\ell_1^a),(\mathit{env},\ell_2^a),(\ell_1^a,\ell_2^a),(\ell_2^a,\ell^t)\}$, $\hat{H} = \{(\ell_1^a,n),(\ell_2^a,m)\}$. Notice that the analysis correctly captures, through the pair (env,ℓ_2^a), the possibility for m to exit from n. □

The correctness of the analysis is proven by showing that every reduction of the semantics is properly mimicked in the analysis, with $\sqsubseteq$ denoting the component-wise set inclusion.

Theorem 2.6 *[15] Let P be a process. If $(\hat{I}, \hat{H}) \models^{\mathrm{CF}} P$ and $\beta^{\mathrm{CF}}_{env}(P) \sqsubseteq (\hat{I}, \hat{H})$ and $P \rightarrow^* P'$, then $\beta^{\mathrm{CF}}_{env}(P') \sqsubseteq (\hat{I}, \hat{H})$.*

2.3 Boundary Ambients Nesting Analysis

The analysis presented in the previous section was not security-oriented, thus it did not exploit the information about "secure" nestings inside boundaries. In [6,7], we propose a more accurate abstract domain that separately considers nesting inside and outside security boundaries, yielding to a much more sophisticated control flow analysis for detecting unwanted boundary crossing, i.e., information leakage. The main idea is to distinguish among nestings either *protected* or *unprotected* by boundaries. More specifically, the $\hat{I}$ component of the abstract domain is split into two (not necessarily disjoint) sets, $\hat{I_B}$ and $\hat{I_E}$, recording the protected and unprotected nestings, respectively, with B standing for Boundary, and E for External environment. Thus the result of the refined analysis is given by means of a triplet $(\hat{I_B}, \hat{I_E}, \hat{H})$.

Also in this case, the analysis is defined by a representation function and a specification. The representation function collects in $\hat{I_B}$ $(\hat{I_E})$ all the nestings of ambients initially (not) contained in at least one boundary ambient.

Example 2.7 Consider again process P_3 of Example 2.4, i.e., of the form: $P_3 = n^{\ell^a_1}[\, m^{\ell^a_2}[\, \mathbf{out}^{\ell^t}\, n\,]\,]$, with $\ell^a_1 \in \mathbf{Lab}^a_B(P)$ and $\ell^a_2 \in \mathbf{Lab}^a_L(P)$, thus the representation function of P is the following: $\beta^{\mathrm{B}}_{env}(P) = (\{(\ell^a_1, \ell^a_2), (\ell^a_2, \ell^t)\}, \{(env, \ell^a_1)\}, \{(\ell^a_1, n), (\ell^a_2, m)\})$. The representation function correctly captures the fact that boundary ambient n protects all of his sub-ambients and capabilities, i.e., both (ℓ^a_1, ℓ^a_2) and (ℓ^a_2, ℓ^t) are recorded in $\hat{I_B}$. □

We do not report the whole analysis specification (see [6,7] for more details). Instead, we explain how it differs from [15] by considering, e.g., the *out*-capability (see Figure 4). Within the specification of the analysis, the predicate $path_B(\ell^a, \ell)$ is used to simplify the notation. Intuitively, it represents a protected path of nestings from ambient labelled ℓ^a to ambient labelled ℓ, in which none of the ambients is a boundary. The rule for the *out*-capability states that if some ambient labelled ℓ^a has an *out*-capability ℓ^t on an ambient n, that may apply due to the presence of a direct ancestor ambient labelled $\ell^{a'}$ whose name is n, then the result of performing that capability should also be recorded in either $\hat{I_B}$ or $\hat{I_E}$, depending on the level of protection of the newly generated nestings. The rule is split into three distinct cases: (i) an ambient exits a boundary, thus moving to an unprotected environment, (ii) all ambients are protected, and finally (iii) all ambients are unprotected. In the first case, all the nestings from the moving ambient ℓ^a to new protecting boundaries have to be copied in $\hat{I_E}$, since after the *out* move they become unprotected. The *in* and *open*-capabilities behave similarly.

$$
\begin{array}{ll}
(out) & (\hat{I_B}, \hat{I_E}, \hat{H}) \models^{\mathrm{B}} \mathbf{out}^{\ell^t}\ n\ .\ P \text{ iff } (\hat{I_B}, \hat{I_E}, \hat{H}) \models^{\mathrm{B}} P \wedge \\
& \forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathsf{Lab}^a(P) : \\
& \text{case } ((\ell^a, \ell^t) \in \hat{I_B} \ \wedge\ (\ell^{a'}, \ell^a) \in \hat{I_E} \cup \hat{I_B} \ \wedge\ (\ell^{a''}, \ell^{a'}) \in \hat{I_E} \\
& \ \wedge (\ell^{a'}, n) \in \hat{H}) \implies \\
& \quad \text{if } (\ \ell^a \in \mathsf{Lab}^a_B(P)) \text{ then } (\ell^{a''}, \ell^a) \in \hat{I_E} \\
& \quad \text{else } (\ell^{a''}, \ell^a) \in \hat{I_E} \ \wedge\ \Big\{(\ell, \ell') \in \hat{I_B} \mid path_B(\ell^a, \ell)\Big\} \subseteq \hat{I_E} \\
& \text{case } ((\ell^a, \ell^t) \in \hat{I_B} \ \wedge\ (\ell^{a'}, \ell^a) \in \hat{I_B} \ \wedge\ (\ell^{a''}, \ell^{a'}) \in \hat{I_B} \ \wedge\ (\ell^{a'}, n) \in \hat{H}) \\
& \quad \implies (\ell^{a''}, \ell^a) \in \hat{I_B} \\
& \text{case } ((\ell^a, \ell^t) \in \hat{I_E} \ \wedge\ (\ell^{a'}, \ell^a) \in \hat{I_E} \ \wedge\ (\ell^{a''}, \ell^{a'}) \in \hat{I_E} \ \wedge\ (\ell^{a'}, n) \in \hat{H}) \\
& \quad \implies (\ell^{a''}, \ell^a) \in \hat{I_E}
\end{array}
$$

Fig. 4. Specification of the refined Control Flow Analysis - the *out* rule

Example 2.8 Let P be the process of Example 2.7. The least solution of P is the triplet $(\hat{I_B}, \hat{I_E}, \hat{H})$ where $\hat{I_B} = \{(\ell^a_1, \ell^a_2), (\ell^a_2, \ell^t)\}$, $\hat{I_E} = \{(env, \ell^a_1), (env, \ell^a_2), (\ell^a_2, \ell^t)\}$, and $\hat{H} = \{(\ell^a_1, n), (\ell^a_2, m)\}$. □

We now obtain an extension of Theorem 2.6:

Theorem 2.9 *Let P be a process. If $(\hat{I_B}, \hat{I_E}, \hat{H}) \models^{\mathrm{B}} P$ and $\beta^{\mathrm{B}}_{env}(P) \sqsubseteq (\hat{I_B}, \hat{I_E}, \hat{H})$ and $P \longrightarrow^* P'$, then $\beta^{\mathrm{B}}_{env}(P') \sqsubseteq (\hat{I_B}, \hat{I_E}, \hat{H})$.*

The result of the analysis should be read, as expected, in terms of information flows. No leakage of secret data/ambients outside the boundary ambients is possible if in the analysis h (a high level datum) does not appear in any of the pairs belonging to $\hat{I_E}$.

Corollary 2.10 *Let P be a process and $h \in \mathsf{Lab}^a_H(P)$ a high level label. Let $\beta^{\mathrm{B}}_{env}(P) \sqsubseteq (\hat{I_B}, \hat{I_E}, \hat{H})$ and $(\hat{I_B}, \hat{I_E}, \hat{H}) \models^{\mathrm{B}} P$ and $\forall (\ell', \ell'') \in \hat{I_E}, \ell'' \neq h$. Then, P does not leak secret h.*

Example 2.11 Consider, for instance, a further refinement of process P_2 of Example 2.2. In this case, the high level data is willing to enter some translator process, which could possibly be low level code coming from the external environment.

$$
\begin{array}{l}
venice^{b_1}[\ envelope^{b_2}[\ \mathbf{out}^{c_1}\ venice\ .\ \mathbf{in}^{c_2}\ pisa\ .\ \mathbf{0} \mid \\
\qquad\qquad\qquad hdata^{h}[\ \mathbf{in}^{c_3}\ translator\ .\ \mathbf{0}\ ]\ ]\ ] \mid \\
pisa^{b_3}[\ \mathbf{open}^{c_4}\ envelope\ .\ \mathbf{0}\ ] \mid \\
translator^{m}[\ \mathbf{in}^{c_5}\ envelope\ .\ \mathbf{0}\ ] \mid \mathbf{open}^{c_6}\ translator\ .\ \mathbf{0}
\end{array}
$$

In this case, the *translator* behaves correctly with respect to information leakage, i.e., it only enters boundaries. In particular, this means it will never carry high level data outside the security boundaries. However, if we compute the CFA of [15], we obtain the following least solution:

$$\begin{aligned}\hat{I} = \{&(env, b1), (env, b2), (env, h), (env, b3), (env, m), (env, c5), (env, c6),\\ &(b1, b2), (b2, h), (b2, m), (b2, c1), (b2, c2), (h, c3), (b3, b2), (b3, h), (b3, b3),\\ &(b3, m), (b3, c1), (b3, c2), (b3, c4), (m, h), (m, c5)\}\\ \hat{H} = \{&(b1, venice), (b2, envelope), (b3, pisa), (h, hdata), (m, translator)\}\end{aligned}$$

Notice that the pair (env, h) appears in $\hat{I}$, even though there is no execution leading to such a situation. On the other hand, the refined analysis properly captures the fact that the *translator* is entered by confidential data *hdata* only once inside boundary *pisa*, and it never gets out of it, i.e., h appears only inside protected nestings $\hat{I_B}$.

$$\begin{aligned}\hat{I_B} = \{&(b1, b2), (b2, h), (b2, m), (b2, c1), (b2, c2), (h, c3), (b3, b2), (b3, h), (b3, b3),\\ &(b3, m), (b3, c1), (b3, c2), (b3, c4), (m, h), (m, c5)\}\\ \hat{I_E} = \{&(env, b1), (env, b2), (env, b3), (env, m), (env, c5), (env, c6), (m, c5)\}\\ \hat{H} = \{&(b1, venice), (b2, envelope), (b3, pisa), (h, hdata), (m, translator)\}\end{aligned}$$

Thus, the boundary analysis is strictly more precise than the one of [15]. □

3 Algorithms for Nesting Analyses

In this section we describe the different implementations of the analyses described so far, which are the core of the Banana tool.

3.1 Efficient algorithms for the Control Flow Analysis

The computation of ambient nesting analysis, like [7,15,20], requires considerably high complexities, thus the design of efficient techniques is very important.

This is the first motivation behind the work presented in [23] by Nielson and Seidl. Assuming N is the size of a process, the authors introduce two different algorithms that in $O(N^4)$ and $O(N^3)$ steps, respectively compute the ambient nesting analysis. The algorithms first perform a translation of the Control Flow Analysis constraints into ground Horn clauses. Then, these clauses are processed through satisfiability standard algorithms [12] in order to compute the least solution. As such algorithms always consider all the ground clauses corresponding to the analysis constraints, even in the best case, all the clauses need to be generated.

In [5], we introduce two new algorithms which improve both time and space complexities of the techniques proposed by Nielson and Seidl. Space complexity of those algorithms is $O(N^4 \log N)$ and $O(N^3 \log N)$ bits, respectively, while the one of both our algorithms is $O(N^2 \log N)$ bits. Regarding time, the worst case complexity is of the same order, i.e., $O(N^4)$ and $O(N^3)$ steps, respectively. In the Nielson and Seidl techniques, those complexities are tight because of the need of generating all the ground Horn clauses, as explained above. Instead, our schemes perform better for some particular small solutions (e.g., solutions that are linear to the size of the process), reaching a complexity of $O(N^3)$ and $O(N^2)$ steps, respectively.

These improvements are first achieved by enhancing the data structure representations. Then, we attack the problem with a direct operational approach, i.e., without passing through Horn formulas. Finally, we reduce the computation to the Control Flow Analysis constraints that are effectively necessary to get to the least solution. In fact, the algorithms dynamically choose, in an *on the fly* manner, only the constraints that are effectively necessary for the computation of the analysis. This implies that no useless repetition occurs and there is no need of representing in memory all the possible instantiation of constraints as done in the Nielson and Seidel approach.

Let us now shortly describe our $O(N^4)$ algorithm, called Algorithm 1, and depicted in Figure 5. The algorithm starts with an empty analysis $\hat{I}$ and with a set of buffers containing all the pairs corresponding to the initial process representation. 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 allows to use each pair of labels in the buffer *exactly once* to generate new pairs.

Let us now analyse Algorithm 1 in more details. 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 labelled with ℓ^t and with n as target. [7] During the parsing, the representation $\beta^{\text{CF}}_{env}(P)$ is computed giving two initial sets $\hat{I}_0$ and $\hat{H}_0$ that are stored into an $N_a \times N_{Lab}$ bit matrix

[7] 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$_{\hat{I}}$ != NIL do
  (l,l') := pop$_{\hat{I}}$ (); M$_{\hat{I}}$ [l,l'] := true;
  for i := 1 to $N_t$ do
    case cap[i] of:
     in$^{\ell^t}$ n: if (l' $\in$ Lab$^t$(P) and l' = $\ell^t$)
                then for j := 1 to $N_a$ do
                       for k := 1 to $N_a$ do
                          if (M$_{\hat{I}}$ [k,l] and M$_{\hat{I}}$ [k,j] and M$_{\hat{H}}$ [j,n]) then push_c$_{\hat{I}}$ (j,l)
                else if (l' $\in$ Lab$^a$(P) and M$_{\hat{I}}$ [l',$\ell^t$]) then for j:= 1 to $N_a$ do
                            if (M$_{\hat{I}}$ [l,j] and M$_{\hat{H}}$ [j,n]) then push_c$_{\hat{I}}$ (j,l');
                       if (l' $\in$ Lab$^a$(P) and M$_{\hat{H}}$ [l',n]) then for j:= 1 to $N_a$ do
                            if (M$_{\hat{I}}$ [j,$\ell^t$] and M$_{\hat{I}}$ [l,j]) then push_c$_{\hat{I}}$ (l',j);
     out$^{\ell^t}$ n: if (l' $\in$ Lab$^t$(P) and l' = $\ell^t$)
                then for j := 1 to $N_a$ do
                       for k := 1 to $N_a$ do
                          if (M$_{\hat{I}}$ [j,l] and M$_{\hat{I}}$ [k,j] and M$_{\hat{H}}$ [j,n]) then push_c$_{\hat{I}}$ (k,l)
                else if (l' $\in$ Lab$^a$(P) and M$_{\hat{I}}$ [l',$\ell^t$] and M$_{\hat{H}}$ [l,n]) then for j:= 1 to $N_a$ do
                            if M$_{\hat{I}}$ [j,l] then push_c$_{\hat{I}}$ (j,l');
                       if (l' $\in$ Lab$^a$(P) and M$_{\hat{H}}$ [l',n]) then for j:= 1 to $N_a$ do
                            if (M$_{\hat{I}}$ [j,$\ell^t$] and M$_{\hat{I}}$ [l',j]) then push_c$_{\hat{I}}$ (l,j);
     open$^{\ell^t}$ n: if (l' $\in$ Lab$^t$(P) and l' = $\ell^t$)
                then for j := 1 to $N_a$ do
                       for k := 1 to $N_{Lab}$ do
                          if (M$_{\hat{I}}$ [l,j] and M$_{\hat{I}}$ [j,k] and M$_{\hat{H}}$ [j,n]) then push_c$_{\hat{I}}$ (l,k)
                else if (l' $\in$ Lab$^a$(P) and M$_{\hat{I}}$ [l,$\ell^t$] and M$_{\hat{H}}$ [l',n]) then for j:= 1 to $N_{Lab}$ do
                            if M$_{\hat{I}}$ [l',j] then push_c$_{\hat{I}}$ (l,j);
                       if (l' $\in$ Lab$^a$(P) and M$_{\hat{H}}$ [l,n]) then for j:= 1 to $N_a$ do
                            if (M$_{\hat{I}}$ [j,$\ell^t$] and M$_{\hat{I}}$ [j,l]) then push_c$_{\hat{I}}$ (j,l');
```

Fig. 5. Algorithm 1

$\mathtt{B}_{\hat{I}}$, 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 $N_a \times N_{Lab}$ 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_{Lab} by capability labels. This initialization phase requires only $O(N)$ steps, since two parsings of P are sufficient.

Example 3.1 Let P be the Firewall Access process of [10,11], where an agent crosses a firewall by means of previously arranged passwords k, k' and k''

(see [22] 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:

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

Let us see how Algorithm 1 applies to process P. In this case, $N_a = 5$ and $N_t = 6$, 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 $\langle (env, a1),\ (env, a3),\ (a1, a2),\ (a3, a4), (a1, t4), (a1, t5), (a2, t1), (a2, t2),$ $(a2, t3), (a3, t6)\rangle$.

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' $= \ell^t$ and $n = w$. Since l' $\in$ **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 24th round, when $\mathtt{buf}_{\hat{I}}$ is empty. □

More details may be found in [5], where we also present an improved algorithm, called Algorithm 2, based on an optimization of the analysis depicted in Figure 3. For space reasons, we omit its description.

3.2 Boundary Ambients Nesting Analysis

In Section 2.3 we described the analysis in terms of a representation and a specification function. It is possible to prove that a least solution for this analysis always exists and it may be computed as follows: first apply the representation function to the process P, then apply the analysis to validate the correctness of the proposed solution, adding, if needed, new information to the triplet until a fix-point is reached. The iterative procedure computes the least solution independent of the iteration order.

3.3 Boundary Inference for Enforcing Security Policies in Mobile Ambients

The notion of "boundary ambient" and the refined control flow analysis can be further enhanced to infer which ambients should be "protected" to guarantee the absence of information leakage for a given process. More specifically, in [8] we consider a process P wherein only high level data are known (i.e, Lab_H is fixed) and the aim of the analysis is to detect which ambients among the "untrusted" ones should be protected and labelled "boundary" to guarantee that the system is secure. This problem can be properly addressed by re-executing the Boundary Ambients Nesting Analysis of Section 2.3. A successful analysis infers boundary ambients until a fix-point is reached, returning the set of ambients that should be "protected".

The algorithm is described Figure 6. It analyses process P starting from the initial labeling $\mathcal{L}_0$. It may either succeed (in this case a labeling $\mathcal{L}_k$ is reached that fulfills the security property we are interested in) or it may fail. The latter case simply means that the process P cannot be guaranteed to be secure by our analysis.

The analysis is currently implemented with a simple fix-point algorithm, but we are working to extend the optimizations presented in Section 3.1 also to the Nesting Analysis.

Example 3.2 Consider process P_2 of Example 2.2:

$$venice^x [\, envelope^y [\, \mathbf{out}^c\, venice \,.\, \mathbf{in}^c\, pisa \,] \mid hdata^h [\, \mathbf{in}^c\, envelope \,]\,] \mid$$
$$\mid pisa^z [\, \mathbf{open}^c\, envelope \,]$$

- Given the set of high level labels Lab_H in P, the initial label partitioning $\mathcal{L}_0$ is computed. $\mathcal{L}_0 = (\mathsf{Lab}_H = \{h\}, \mathsf{Lab}^0_B = \{x\}, \mathsf{Lab}^0_L = \{y, z\})$

```
Boundary Inference Algorithm
Input: a process $P$ and a partition labeling $\mathcal{L}_0$.
  (i) Compute the Nesting Analysis with input $P$ and labels $\mathcal{L}_i$.
  (ii) During the execution of the Fix-Point Algorithm, whenever a high level ambient
       $n$ labelled $h$ gets into an unprotected environment, i.e. $\exists \ell\ :\ (\ell, h) \in \hat{I_E}$ do:
         1. if $(env, h) \in \hat{I_E}$, the analysis terminates with failure, as it cannot infer a
            satisfactory labeling that guarantees absence of information leakage;
         2. otherwise, if $(env, h) \notin \hat{I_E}$:
              – a new labeling $\mathcal{L}_{i+1}$ should be considered, labeling every $\ell$ such that
                $(\ell, h) \in \hat{I_E}$ as a boundary. Let $\mathbf{L} = \{\ell |\ (\ell, h) \in \hat{I_E}\}$ then $\mathcal{L}_{i+1} =$
                $(\mathsf{Lab}_H, \mathsf{Lab}^i_B \cup \{\mathbf{L}\}, \mathsf{Lab}^i_L \setminus \{\mathbf{L}\})$.
              – go to (i) with $i = i + 1$.
```

Fig. 6. Boundary inference algorithm

- Applying the representation function $\beta^{\mathcal{L}_0}$ to P, it returns the triplet $(\hat{I^o_B}, \hat{I^o_E}, \hat{H})$:

$$\hat{I^o_B} = \{(x, y), (x, h), (y, c), (h, c)\}$$
$$\hat{I^o_E} = \{(env, x), (env, z), (z, c)\}$$
$$\hat{H} = \{(h, hdata), (x, venice), (y, envelope), (z, pisa)\}$$

  Executing the Fix-Point Algorithm, the pair (y, h) is introduced in $\hat{I_E}$, reflecting the fact that ambient *envelope* leaves ambient *venice* during the execution of process P.

- At this point, a new label partitioning should be considered:
  $\mathcal{L}_1 = (\mathsf{Lab}(P)_H = \{h\}, \mathsf{Lab}^1_B = \{x, y\}, \mathsf{Lab}^1_L = \{z\})$
  is computed again. During its execution, the pair $(z, h) \in \hat{I_E}$, reflecting the fact that the boundary *envelope*, containing confidential data, is opened inside the low ambient *pisa* during the execution of process P.

- At this point, the following new label partitioning is considered:
  $\mathcal{L}_2 = (\mathsf{Lab}(P)_H = \{h\}, \mathsf{Lab}^2_B = \{x, y, z\}, \mathsf{Lab}^2_L = \emptyset)$
  the Fix-Point Algorithm is computed again, and a fix-point is finally reached. Thus, the set of ambients that should be labelled as boundaries is $\{venice, envelope, pisa\}$.

□

4 Tool Overview

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/~mefisto/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 [15], the one by Braghin et al. in [6] (called Focardi Cortesi Braghin in the tool), and the one by Braghin et al. in [8] (FCB Boundary Inference). Five different implementations of the analysis described in [15] are available in the tool. They correspond to:
  - a fix-point computation of the least solution of the constraints in Figure 3 (called Nielson in the tool) [8];
  - a fix-point computation of the least solution of the constraints of the optimized algorithm of Nielson and Siedl (Nielson Optimized);
  - Algorithm 1 of Figure 5 (Buffered Boundary Analysis, B.B.A., v1);
  - Algorithm 2 of [5] (B.B.A. v2);
  - Algorithm 2 of [5] with some code optimizations (B.B.A. v2 Optimized).
- A post-processing module, that interprets the results of the analysis in terms of the boundary-based information-flow model proposed in [6], 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.

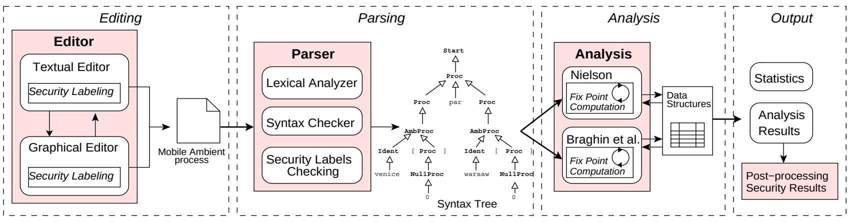
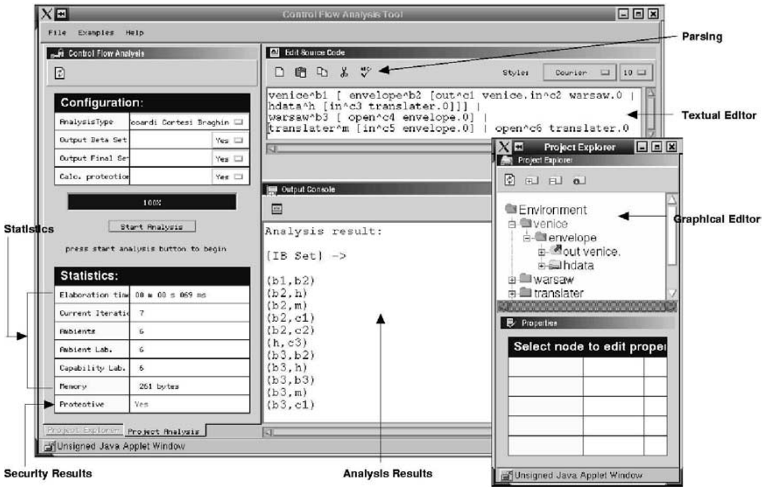


Fig. 7. Overview of the Banana tool.

[8] This implementation does not use the algorithm in [12] and it has a $O(N^5)$ worst-case time complexity.

Figure 7 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., [17]). Moreover, Banana is conceived as an applet based on AWT and thus compatible with the majority of current web browsers supporting Java.

A screen-shot of the Banana tool is shown in Figure 8. A user can edit



```
venice^b1 [ envelope^b2 [out^c1 venice.in^c2 warsaw.0 |
hdata^h [in^c3 translater.0]]] |
warsaw^b3 [ open^c4 envelope.0] |
translater^m [in^c5 envelope.0] | open^c6 translater.0
```

Configuration:

AnalysisType	oardi Cortesi Braghin
Output Beta Set	Yes
Output Final Se	Yes
Calc. protectio	Yes

100%

press start analysis button to begin

Statistics:

Elaboration tim	00 m 00 s 069 ms
Current Iterati	7
Ambients	6
Ambient Lab.	6
Capability Lab.	6
Memory	261 bytes
Protective	Yes

```
Analysis result:

(IB Set) ->

(b1,b2)
(b2,h)
(b2,m)
(b2,c1)
(b2,c2)
(h,c3)
(b3,b2)
(b3,h)
(b3,b3)
(b3,m)
(b3,c1)
```

```
Environment
  venice
    envelope
      out venice.
      hdata
  warsaw
  translater
```

Select node to edit proper

Fig. 8. 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 [6,8,15]. 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.

5 Conclusion

Security is a major concern for computation in open networks, and is often considered a serious source of potential limitation to a widespread use of mobile code technologies. In this paper, we have studied information flow security in the calculus of Mobile Ambients with a static approach.

Plans of future work include the refinement of the analysis abstract domains, and the study of the relationship among types, abstract interpretation, and control flow analysis. Enhancing the precision of the analysis result can affect the efficiency of the analysis: a careful study is therefore needed to balance the precision and the efficiency of the analysis. In addition, we believe that the formalization of the CFA and types in a common setting would allow us to compare their expressive power, understand the relative merits of each approach, and, possibly, for which class of properties one method is more adequate than another. We are presently studying how to extend our approach to the full calculus and to more "concrete" version of Ambients, e.g., Boxed and Safe Ambients [9,17].

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*, 168:68–92, 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'98)*, LNCS 1466, pages 84–98. Springer-Verlag, 1998.

[3] C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Static Analysis of Processes for No Read-Up and No Write-Down. In *Proc. FoSSaCS'99*, number 1578 in Lecture Notes in Computer Science, pages 120–134. Springer-Verlag, 1999.

[4] C. Braghin, A. Cortesi, S. Filippone, R. Focardi, F.L. Luccio, and C. Piazza. BANANA: A tool for Boundary Ambients Nesting ANAlysis. In H. Garavel and J. Hatcliff, editors, *Proc. of the Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, LNCS 2619, pages 437–441. Springer-Verlag, 2003.

[5] C. Braghin, A. Cortesi, R. Focardi, F.L. Luccio, C. Piazza. Complexity of Nesting Analysis in Mobile Ambients. In L.D. Zuck, P.C. Attie, A. Cortesi, S. Mukhopadhyay, editors, *Proc. of the 4-th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'03)*, LNCS 2575, pages 86–101. Springer-Verlag, 2003.

[6] C. Braghin, A. Cortesi, and R. Focardi. Security Boundaries in Mobile Ambients. *Computer Languages*, Elsevier, 28(1):101–127, Nov. 2002.

[7] 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 (FMOODS'02)*, pages 197–212. Kluwer Academic Publisher, 2002.

[8] 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 (IFIP TCS'02)*. Kluwer Academic Publisher, pages 383–395, 2002.

[9] M. Bugliesi, G. Castagna and S. Crafa, *Boxed Ambients*, in: *Proc. of the 4th Int. Conference on Theoretical Aspects of Computer Science (TACS'01)*, LNCS **2215** (2001), pp. 38–63.

[10] L. Cardelli and A.D. Gordon. Mobile Ambients. In M. Nivat, editor, *Proc. of Foundations of Software Science and Computation Structures (FoSSaCS'98)*, LNCS 1378, pages 140–155. Springer-Verlag, 1998.

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

[12] 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.

[13] 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'00)*, LNCS 1961, pp. 199–214. Springer, 2000.

[14] R. Focardi and R. Gorrieri. A Classification of Security Properties for Process Algebras. *Journal of Computer Security*, 3(1):5–33, 1995.

[15] R.R. Hansen, J.G. Jensen, F. Nielson, and H. Riis Nielson. Abstract Interpretation of Mobile Ambients. In A. Cortesi and G. File', editors, *Proc. of Static Analysis Symposium (SAS'99)*, LNCS 1694, pp. 134–148. Springer, 1999.

[16] M. Hennessy and J. Riely. Information Flow vs. Resource Access in the Asynchronous Pi-Calculus. In U. Montanari, J. Rolim, and E. Welzl, editors, *Proc. of the International Colloquium on Automata, Languages and Programming*, volume 1853 of *Lecture Notes in Computer Science*, pages 415–427, Geneva, August 2000. Springer-Verlag.

[17] F. Levi and D. Sangiorgi. Controlling Interference in Ambients. In *Proc. 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 352–364, 2000.

[18] D.A. McAllester. On the Complexity Analysis of Static Analyses. *Journal of the ACM*, 49(4): 512-537, July 2002.

[19] 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'00)*, pages 67–80. ACM Press, N.Y., U.S.A., 2000.

[20] F. Nielson, R. R. Hansen, and H. Riis Nielson. Abstract Interpretation of Mobile Ambients. *Science of Computer Programming*, Special Issue on Static Analysis edited by A. Cortesi and G. File', vol. 47(2-3), pp. 145-175, 2003.

[21] F. Nielson, H. Riis Nielson, C.L. Hankin. Principles of Program Analysis. Springer, 1999.

[22] 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'99)*, LNCS 1664, pages 463–477. Springer-Verlag, 1999.

[23] F. Nielson and H. Seidl. Control-flow Analysis in Cubic Time. In D. Sands, ed., *Proc. of European Symposium On Programming (ESOP'01)*, LNCS 2028, pp. 252–268. Springer, 2001.

[24] F. Nielson, H. Riis Nielson, and H. Seidl. Automatic Complexity Analysis. In D. Le Metayer, ed.,*Proc. of European Symposium On Programming (ESOP'02)*, LNCS 2305, pp.243–261. Springer, 2002.

[25] G. Smith and D. Volpano. Secure Information Flow in a Multi-Threaded Imperative Language. In *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, pages 355–364, New York, NY, 1998.