Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 128 (2005) 17–25
www.elsevier.com/locate/entcs

Flow-sensitive Leakage Analysis in Mobile Ambients*

Chiara Braghin[1], Agostino Cortesi[2],

Dipartimento di Informatica
Università Ca' Foscari di Venezia
Venice, Italy

Abstract

In this paper, we present a refinement of a Control Flow Analysis aimed at studying information flow security in the the calculus of Mobile Ambients. The improvements are achieved by making the analysis be *flow-sensitive*: the analysis is able to keep track of temporal dependencies of capabilities application when computing a safe approximation of the run-time topology of Mobile Ambient processes.

Keywords: Static Analysis, Ambient Calculus, Security, Information Flow.

1 Introduction

In the context of distributed systems, where resources and data are shared among users located almost everywhere, it is likely that a user gets some (possibly) malicious programs from an untrusted source on the net and executes them inside its own system with unpredictable results. Moreover, it could be the case that a system completely secure inside, results to be insecure when performing critical activities such as electronic commerce or home

* Partially supported by MIUR Project "AIDA Abstract Interpretation: Design and Applications", 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

doi:10.1016/j.entcs.2004.11.039

banking, due to a "weak" mechanism for remote connections.

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. As most non-trivial properties of the run-time behaviour of a program are either undecidable or NP, it is not possible to detect them accurately, and some form of approximation is needed. So, in general, we expect to produce a possibly larger set of possibilities than what will ever happen during execution of the program.

The purpose of *Control Flow Analysis* (CFA) [11] is to statically predict safe and computable approximations to the dynamic behaviour of programs. It can be expressed using different formulations such as the constraint-based formalism popular for the analysis of functional and object-oriented languages, or the *Flow Logic* style [3,10,14]. As in type systems, Flow Logic makes a clear distinction between the *specification* of when an analysis proposed solution is acceptable for a program, and the actual *computation* of the analysis information. By predicting the behaviour of a system, it leads to positive information also when the system under evaluation does not satisfy the property of interest, whereas the type-system approach is given in terms of prescriptive rules (a system is either accepted or discarded). In several recent developments it has been demonstrated that Flow Logic is a robust approach that is able to deal with a variety of calculi of computation: the lambda-calculus, Concurrent ML, the imperative object calculus, the pi-calculus [1], the Mobile Ambients calculus [9,12,13], and the spi-calculus [2].

In [5], we applied the Flow Logic-based control flow analysis proposed in [9] to verify absence of information flow in processes modelled in Mobile Ambients [7]. In order to study this problem as much abstractly as possible, the "pure" Mobile Ambients calculus is considered, in which no communication channels are present, and the only possible actions are represented by the moves performed by mobile processes. This allows the study of a very general notion of information flow which should be applicable also to different versions of the calculus such as, e.g., Boxed Ambients [6], BioAmbients [15], and Safe Ambients [8]. In particular, a new notion of *security boundary* [5] has been introduced to model *multi-level security policies* in this scenario. Information leakage may be expressed in terms of the possibility for a hostile ambient to access confidential data that are not protected inside a security boundary.

With this work, we refine even further the analysis of [5] in order to make

it *flow-sensitive*. More specifically, (*i*) the solution is a power-set of process representations, not just a single flat representation; (*ii*) by tagging new pairs as active and recording which capabilities are enabled, the analysis keeps track of temporal dependencies of capabilities application. The implementation of the refined Control Flow Analysis can be easily integrated to the BANANA tool [4].

The rest of the paper is organised as follows. In Section 2, we introduce the basic terminology of the Mobile Ambient calculus, and briefly report a Control Flow Analysis aiming at computing a safe approximation of the possible ambient nestings occurring the run-time execution of a process. Then, we describe how to formalise multi-level security in the setting of Mobile Ambients. In Section 3, we propose a refinement of the analysis in order to make it flow-sensitive.

2 Background

2.1 The Ambient Calculus

The Mobile Ambient calculus has been introduced in [7] with the main purpose of explicitly modelling 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.

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

Intuitively, the restriction $(\nu n)P$ introduces the new name n and limits its scope to P; $P \mid Q$ is P and Q running in parallel; replication provides recursion and iteration. 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 a sibling ambient n. Labels on ambients and on transitions are introduced as it is customary in static analysis to indicate "program points".

The operational semantics of a process P is given through a suitable reduction relation $\rightarrow$ and a structural congruence $\equiv$ between processes. Intuitively, $P \rightarrow Q$ represents the possibility for P of reducing to Q through some computation (see also [7]).

For instance, let P_1 be a process modelling an *envelope* sent from *venice* to *london*:

$$venice[\, envelope[\, \mathbf{out}\, venice.\mathbf{in}\, london.\mathbf{0}\,] \mid Q\,] \mid london[\, \mathbf{open}\, envelope.\mathbf{0}\,]$$

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

To express multi-level security policies, 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. Ambients labelled with boundary labels (*boundary ambients*) are the ones responsible for confining confidential information. Information leakage occurs if, during the execution of the process, a high level ambient is not confined inside a boundary, thus being possibly exposed to a malicious ambient attack. For instance, let P_2 be an extension of process P_1, in which the *envelope* contains confidential data *hdata* (labelled *high*) which needs to be safely sent from *venice* to *london*.

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

In this case, *venice*, *envelope* and *london* must be labelled *boundary* to protect *hdata* during the whole execution. (See [5] for more detail.)

2.2 A Control Flow Analysis for Mobile Ambients

A first Control Flow Analysis aiming at modelling the possible nesting of processes occurring at run-time was proposed in [9]. In the case of Mobile Ambients, the control structure computed by the analysis is expressed by the hierarchical structure of ambients, given by the *father-son* relationship between the nodes of the tree structure. The analysis was not security-oriented, thus it did not exploit the information about "secure" nestings inside boundaries.

In [5], we proposed 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 analysis is expressed in terms of tuple $(\hat{I}_B, \hat{I}_E, \hat{H})$, where:

- The first and the second component ($\hat{I}_B$ and $\hat{I}_E$) are elements of $\wp(\mathbf{Lab}^a \times \mathbf{Lab})$. 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 $\hat{I}_B$ or $\hat{I}_E$ depending on the level of protection of the nesting (with B standing for Boundary, and E standing for External environment).

- The third component $\hat{H} \in \wp(\mathbf{Lab}^a \times \mathbf{Amb})$ 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 $\hat{H}$.

The analysis is defined by a representation and a specification functions [11]. The representation function aims at mapping concrete values to their best abstract representation. 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. It is given in terms of a function $\beta^{\mathrm{B}}_{\ell}(P)$ which maps process P into a triplet $(\hat{I_B}, \hat{I_E}, \hat{H})$ corresponding to the initial state of P, with respect to an enclosing ambient labelled with ℓ.

The specification states a closure condition of a triplet $(\hat{I_B}, \hat{I_E}, \hat{H})$ 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 lack of space, in Figure 1 we report only the rule for *out*-capability. Within the specification, the predicate $path_B(\ell^a, \ell)$ is used to simplify the notation: it represents an 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 $\hat{I_E}$ or $\hat{I_B}$, 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.

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

Fig. 1. Specification of the Control Flow Analysis.

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 $\hat{I}_E$.

Example 2.1 Let P_3 be a process modelling the fact that confidential data *hdata* is inserted inside a secure *envelope* before, possibly, being sent in the insecure external environment.

Notice that in the pure Ambient calculus there are not communication primitives. In the absence of such primitives, the exchange of an *envelope* between a *Sender* (site *venice*, in our example) and a *Receiver* may be modelled as a sequence of **out** *Sender* and **in** *Receiver* actions, performed by ambient *envelope*. In this way *envelope* moves from *Sender* to *Receiver*, where it will be **open**ed in order to read its contents (ambient *hdata*, in our case).

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

In this case, ambient *hdata* behaves correctly with respect to information leakage since it never exits the boundary *venice*. However, if we compute the CFA, we obtain that the pair (env, h) appears in $\hat{I}_E$, *env* denoting the external environment, thus leading to a false alarm. □

3 A Flow-Sensitive Control Flow Analysis

The CFA of Section 2.2 is both *context* and *flow-insensitive*: it does not consider the temporal order in which the capabilities are executed, and it is not able to distinguish between blocking and non-blocking capabilities in a capability path. Consider again process P_3 of Example 2.1: in this case, capability $\mathbf{out}^{c_2}$ *venice* is executed only if capability $\mathbf{in}^{c_1}$ *envelope* has been consumed, i.e., only when ambient *hdata* is inside *envelope*, and consequently the *out*-capability is not enabled. On the contrary, the analysis considers also the case in which the *out*-capability is consumed, both when ambient *hdata* is inside *venice* (i.e., an old snapshot of the system), and when it is inside *envelope* (i.e., the system's situation after the in-movement).

In order to refine the analysis, so that it recognizes when a capability is *enabled* or not (i.e., it is ready to be consumed), and if a pair in $\hat{I}_B$ or $\hat{I}_E$ represents either an old snapshot of the system or the current situation, we enrich the abstract domains by:

- adding a fourth component, $\mathcal{L}$, which is a set of lists of capabilities' labels. More specifically, each list represents an ordered sequence of the capabilities' labels along a path, thus the head of each list is an enabled capability

keeping blocked the capabilities in the tail of the list. For example, in process P_3, at the initial state, $\mathcal{L} = \{[c_1, c_2], [c_3]\}$, with $\mathbf{in}^{c_1}$ *envelope* and $\mathbf{open}^{c_3}$ *hdata* both ready to be consumed.

- tagging pairs in $\hat{I_B}$ or $\hat{I_E}$ with A or NA, denoting if a pair is *active* or not (i.e., it occurs, but it cannot be used to generate further nestings since it belongs to an old snapshot of the system). To this aim, we define a function Φ which, given a pair, complements the tag (e.g, $\Phi((\ell, \ell')_A) = (\ell, \ell')_{NA}$). For example, in process P_3, at the beginning $(b_1, h)_A$, but once *hdata* moves inside *envelope* it must change to $(b_1, h)_{NA}$.

The result of the analysis is a power-set of representations of processes: $\mathcal{S} = \{(\hat{I_B}, \hat{I_E}, \hat{H}, \mathcal{L})\}$. Also in this case, the analysis is given in terms of a representation and of a specification function. The representation function collects all the initial nesting among ambients and capabilities, tagging with A only ambient nestings and the pairs (ℓ, ℓ^t) where ℓ^t is the label of an enabled capability. Again, we do not report the whole analysis specification. Instead, we explain how it differs from [5] by considering, e.g., the *out*-capability (see Figure 2). The rule for the *out*-capability states that the capability is considered only if it is enabled. If this is the case, then the result of performing that capability should also be recorded in either $\hat{I_E}$ or $\hat{I_B}$, depending on the level of protection of the newly generated nestings. In addition, the tags of the pairs involved in the movement should be updated consequently, and ℓ^t deleted from the head of a list in $\mathcal{L}$.

$$
\begin{array}{l}
(\mathbf{out})\ \mathcal{S} \models^{\mathrm{B}} \mathbf{out}^{\ell^t}\ n.P \text{ iff } \mathcal{S} \models^{\mathrm{B}} P \wedge \\
\quad \forall(\hat{I_B}, \hat{I_E}, \hat{H}, \mathcal{L}) \in \mathcal{S}: \text{ if } (\ell^t \in \mathrm{head}(\mathcal{L})) \text{ then} \\
\quad \forall \ell^a, \ell^{a'}, \ell^{a''} \in \mathbf{Lab}^a: \\
\quad \text{case } ((\ell^a, \ell^t)_A \in \hat{I_B} \ \wedge\ (\ell^{a'}, \ell^a)_A \in \hat{I_E} \cup \hat{I_B} \ \wedge\ (\ell^{a''}, \ell^{a'})_A \in \hat{I_E} \wedge \\
\quad (\ell^{a'}, n)_A \in \hat{H}) \implies \Phi((\ell^a, \ell^t)_A) \ \wedge\ \Phi((\ell^{a'}, \ell^a)_A) \ \wedge\ \mathrm{top}(\ell^t, \mathcal{L}) \wedge \\
\qquad \text{if } (\ \ell^a \in \mathbf{Lab}^a_B) \text{ then } (\ell^{a''}, \ell^a)_A \in \hat{I_E} \\
\qquad \text{else } (\ell^{a''}, \ell^a)_A \in \hat{I_E} \ \wedge\ \left\{(\ell, \ell')_A \in \hat{I_B} \mid path_B(\ell^a, \ell)\right\} \subseteq \hat{I_E} \\
\quad \text{case } ((\ell^a, \ell^t)_A \in \hat{I_B} \ \wedge\ (\ell^{a'}, \ell^a)_A \in \hat{I_B} \ \wedge\ (\ell^{a''}, \ell^{a'})_A \in \hat{I_B} \ \wedge\ (\ell^{a'}, n)_A \in \hat{H}) \\
\quad\ \implies (\ell^{a''}, \ell^a)_A \in \hat{I_B} \ \wedge\ \Phi((\ell^a, \ell^t)_A) \ \wedge\ \Phi((\ell^{a'}, \ell^a)_A) \ \wedge\ \mathrm{top}(\ell^t, \mathcal{L}) \\
\quad \text{case } ((\ell^a, \ell^t)_A \in \hat{I_E} \ \wedge\ (\ell^{a'}, \ell^a)_A \in \hat{I_E} \ \wedge\ (\ell^{a''}, \ell^{a'})_A \in \hat{I_E} \ \wedge\ (\ell^{a'}, n)_A \in \hat{H}) \\
\quad\ \implies (\ell^{a''}, \ell^a)_A \in \hat{I_E} \ \wedge\ \Phi((\ell^a, \ell^t)_A) \ \wedge\ \Phi((\ell^{a'}, \ell^a)_A) \ \wedge\ \mathrm{top}(\ell^t, \mathcal{L})
\end{array}
$$

Fig. 2. Specification of the Refined Control Flow Analysis.

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 (high level datum) does not appear in any of the pairs belonging to any of the $\hat{I_E}$ sets in $\mathcal{S}$.

Theorem 3.1 *Let P be a process, $h \in \mathbf{Lab}^a_H$ a high level label, and $\mathcal{S} = \{(\hat{I_B}, \hat{I_E}, \hat{H}, \mathcal{L})\}$. Let $\beta^{\mathrm{B}}_{env}(P) \subseteq \mathcal{S}$, $\mathcal{S} \models^{\mathrm{B}} P$, and $\forall(\ell', \ell'') \in \hat{I_E}, \ell'' \neq h$. Then, P does not leak secret h.*

Example 3.2 Consider again process P_3 of Example 2.1. The representation function gives a snapshot of process P_3 in its initial state:

$$\mathcal{S}_0 = \{\{(b_1,h)_A,(h,c_1)_A,(h,c_2)_{NA},(b_1,b_2)_A,(b_1,c_3)_A\},\{(env,b_1)_A\},\{[c_1,c_2],[c_3]\},\\ \{(b_1,venice),(h,hdata),(b_2,envelope)\}$$

At this point, both $\mathbf{in}^{c_1}$ *envelope* and $\mathbf{open}^{c_3}$ *hdata* are enabled. If the open-rule is applied, we obtain:

$$\mathcal{S}_1 = \mathcal{S}_0 \cup \{\{(b_1,h)_{NA},(h,c_1)_{NA},(h,c_2)_{NA},(b_1,b_2)_A,(b_1,c_3)_{NA},(b_1,c_1)_A,(b_1,c_2)_{NA}\},\\ \{(env,b_1)_A\},\{[c_1,c_2]\},\{(b_1,venice),(h,hdata),(b_2,envelope)\}$$

Now, only $\mathbf{in}^{c_1}$ *envelope* is enabled:

$$\mathcal{S}_2 = \mathcal{S}_1 \cup \{\{(b_1,h)_{NA},(h,c_1)_{NA},(h,c_2)_A,(b_1,b_2)_A,(b_1,c_3)_A,(b_2,h)_A\},\{(env,b_1)_A\},\\ \{[c_1],[c_3]\},\{(b_1,venice),(h,hdata),(b_2,envelope)\}$$

Finally, none of the capabilities enabled can be consumed, thus the fixed-point algorithm stops. The two snapshots added to the solution represent two possible traces of the run-time execution of process P_3. Also in this case, the results should be read in term of information flow: since h does not appear in $\hat{I_E}$, the analysis correctly verifies that there is no information leakage. □

References

[1] C. Bodei, P. Degano, F. Nielson, and H. R. 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. R. Nielson. Flow Logic for Dolev-Yao Secrecy in Cryptographic Processes. *Future Generation Computer Systems*, 18(6):747–756, 2002.

[3] C. Bodei, P. Degano, H. R. Nielson, and F. Nielson. Security Analysis using Flow Logics. EATCS Bulletin vol. 70; an extended version is to be published by World Scientific, 2000.

[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 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 437–441. Springer–Verlag, Berlin, 2003.

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

[6] M. Bugliesi, G. Castagna, and S. Crafa. Boxed Ambients. In *Proc. of 4th Int. Conference on Theoretical, Aspects of Computer Science (TACS'01)*, number 2215 in Lecture Notes in Computer Science, pages 38–63. Springer–Verlag, Berlin, 2001.

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

[8] F. Levi and D. Sangiorgi. Mobile Safe Ambients. *ACM Transactions on Programming Languages and Systems*, 25(1):1–69, 2003.

[9] F. Nielson, R. R. Hansen, and H. R. Nielson. Abstract Interpretation of Mobile Ambients. *Science of Computer Programming, Special Issue on Static Analysis*, 47(2–3):145–175, 2003.

[10] F. Nielson and H. R. Nielson. Flow Logics and Operational Semantics. *Electronic Notes on Theoretical Computer Science*, 10, 1998.

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

[12] F. Nielson, H. R. Nielson, and M. Sagiv. A Kleene Analysis of Mobile Ambients. In *Proc. of 9th European Symposium On Programming (ESOP'00)*, number 1782 in Lecture Notes in Computer Science, pages 305–319. Springer–Verlag, Berlin, 2000.

[13] H. R. Nielson and F. Nielson. Shape Analysis for Mobile Ambients. *Nordic Journal of Computing*, 8(2):233–275, 2001.

[14] H. R. Nielson and F. Nielson. Flow Logic: a Multi-paradigmatic Approach to Static Analysis. pages 223–244, 2002.

[15] A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Shapiro. BioAmbients: An Abstraction for Biological Compartments. *Theoretical Computer Science*, 325(1):141–167, 2004.