

Generalized Abstract Non-Interference

Abstract Secure Information-flow Analysis for Automata

Roberto Giacobazzi and Isabella Mastroeni

Dipartimento di Informatica - Università di Verona, Italy
(roberto.giacobazzi@ | mastroeni@sci.)univr.it

Abstract. Abstract non-interference has been introduced as a weakening non-interference which models attackers as abstract interpretations (i.e., static analyzers) of programming language semantics. In this paper we generalize the notion of abstract non-interference to deal with tree-like models of computation. This allows us to widen the scope of abstract non-interference for modeling security properties in automata, timed automata as models of real-time systems, and concurrent systems. We show that well known definitions of non-interference in these models of computation can be viewed as instances of our generalization. This proves that abstract non-interference can reasonably be considered as a general framework for studying and comparing security properties at different levels of abstraction in both programming languages and systems. Moreover, the most precise harmless attacker of a system is systematically derived by transforming abstract domains, characterizing the security degree of automata and concurrent systems.

Keywords: Abstract interpretation, language-based security, non-interference, real-time systems, concurrency.

1 Introduction

Non-interference [15] is a key notion in language based security. The idea is that no information about confidential data can be obtained by observing public information. The standard methods used for preventing interference are based on access control, i.e. higher privileges are required in order to access files containing confidential data. The problem with these methods is that, after access, there is no further control on how confidential information flows during execution. Hence, many techniques for checking *secure information flows* in software and systems, ranging from standard data-flow/control-flow analysis techniques to type inference, are studied, based on non-interference (see [22] and [11] for excellent surveys). All these approaches are devoted to prove that a system as a whole, or parts of it, does not allow confidential data to flow towards public variables. Type-based approaches are designed in such a way that well-typed programs do not leak secrets. In a security-typed language, a type is inductively associated at compile-time with program statements in such a way that

any statement showing a potential flow disclosing secrets is rejected [25, 27, 28]. Similarly, data-flow/control-flow analysis techniques are devoted to statically discover flows of secret data into public variables [4, 18, 19, 23]. In concurrency, bisimulation is used to prove equivalence between computations where private actions are hidden with respect to the same system’s computations where these actions are avoided [10]. In real-time systems trace equivalence is used to prove that the computations where private actions are avoided are equivalent to the same system’s computations where these actions are admitted with a minimum fixed delay, and then hidden to the attacker [2]. These notions are all based on the same principle, which is non-interference, but they can be hardly recognized as instances of a same construction. This is due to the fact that different aspects of the underlying computational models become crucial in order to provide expressive enough notions of secrecy in sequential, non-deterministic, concurrent and real-time systems. While in sequential programs we are mainly concerned with non-interference in input/output behavior, in concurrency and in real-time we are, respectively, mainly concerned with interleaving actions and time delays.

Standard non-interference is often too strict for any practical use in language-based security: most programs are rejected by static control/data flow analyzers or type checkers for non-interference. In order to adapt security policies to practical cases, it is essential to know how much an attacker may learn from a program by (statically) analyzing its input/output behavior. This idea led to the definition of the notion of *abstract non-interference* [14], which captures a weaker form of non-interference. Namely, non-interference is made parametric relatively to some abstract property, formalized as an abstract interpretation [7], of the input/output behavior. This notion however is not adequate to cope with more complex systems like concurrent and real-time systems. In particular, as stated in [14], abstract non-interference strongly relies upon a denotational model of computation, which is inadequate for modeling security protocols for instance.

Main contribution and related works. In this paper, we prove that the notion of abstract non-interference introduced in [14] can be generalized in order to cope with many well-known models of secrecy in sequential, concurrent and real-time systems and languages. This is achieved by factoring abstractions in order to identify sub-abstractions modeling the different properties of the system on which the notions of non-interference are based. Abstract interpretation [7] and the theory of abstract domain transformers [8] plays a key role in this generalization: The abstraction represents here both what an attacker may observe about a computation (as in abstract non-interference [14]) and which aspects of the computation are relevant for checking non-interference. In this context, non-interference corresponds to asking that the behavior of the chosen relevant aspects of the computation is independent from what an attacker may observe. We prove that both narrow and abstract non-interference in [14] are instances of our generalized abstract non-interference (GANI). Then we prove that NNI (Non-deterministic Non-Interference), SNNI (Strong NNI), NDC (Non-Deducibility on Compositions), BNDC (Bisimulation NDC), BNNI (Bisimulation NNI), and BSNNI (Bisimulation SNNI) in [10] for Security Process Algebras (SPA), are all

instances of GANI. Finally, we prove that decidable notions of non-interference introduced for timed automata in [2] are again instances of GANI. In all these constructions, the model of an attacker is specified as an abstract interpretation of the system semantics. This is a key point in order to introduce systematic methods for deriving attackers by transforming abstract domains. We generalize the method introduced in [14] to derive harmless attackers for GANI, i.e., abstractions of the semantics of systems which guarantee non-interference.

This is not the first attempt neither for deriving general schemes for security policies [12], nor for trying to bridge language-based and process-calculi security [13, 16, 20]. In particular, in [12] the authors provide a uniform method for defining computer security properties for process algebras, obtaining a quite flexible schema for reasoning about different properties. What we do in this paper is something similar since the aim is the same, but in a more general context which ranges from language based-security to process algebras, to timed automata. The problem of studying the link between language-based and process calculi security is well known in literature. In particular, recently [13] this problem has been approached by transforming imperative language in a CCS-like process calculus, and by defining a notion of BNDC which corresponds for imperative languages to the standard notion of non-interference. In our case, we don't have to transform programs, since we consider a general model that can cope with both imperative programming languages and process algebras.

2 Information flows in Language-based Security

Non-interference can be naturally expressed by using semantic models of program execution. This idea goes back to Cohen's work on *strong dependency* [5], which uses denotational semantics for modeling how information can be transmitted among variables during the execution of programs. Therefore non-interference for programs essentially means that “*a variation of confidential (high or private) input does not cause a variation of public (low) output*” [22]. When this happens, we say that the program has only *secure information flows* [3, 5, 9, 18, 26]. This situation has been modeled by considering the denotational (input/output) semantics $\llbracket P \rrbracket$ of the program P . In particular, we consider programs where data are typed as private (\mathbb{H}) or public (\mathbb{L}). Program states in Σ are functions (represented as tuples) mapping variables in the set of values \mathbb{V} . If $\mathbb{T} \in \{\mathbb{H}, \mathbb{L}\}$, $n = |\{x \in \text{Var}(P) \mid x : \mathbb{T}\}|$, and $v \in \mathbb{V}^n$, we abuse notation by writing $v \in \mathbb{V}^{\mathbb{T}}$ when v is a value for the variables with security type \mathbb{T} . Moreover, we assume that any input s , can be seen as a pair (h, l) , where $s^{\mathbb{H}} = h$ is a value for private data and $s^{\mathbb{L}} = l$ is a value for public data. In this case, (standard) *non-interference* can be formulated as follows.

$$\boxed{\text{A program } P \text{ is } \textit{secure} \text{ if } \forall \text{ input } s, t . s^{\mathbb{L}} = t^{\mathbb{L}} \Rightarrow (\llbracket P \rrbracket(s))^{\mathbb{L}} = (\llbracket P \rrbracket(t))^{\mathbb{L}}}$$

This problem has been formulated also as a *Partial Equivalence Relation* (PER) [17, 23]. In [14], the notion of abstract non-interference is introduced for modeling both weaker attack models, and declassification. The idea is that, instead

$[\eta]P(\rho) \text{ if } \forall h_1, h_2 \in \mathbb{V}^{\mathbb{H}}, \forall l_1, l_2 \in \mathbb{V}^{\mathbb{L}}. \eta(\{l_1\}) = \eta(\{l_2\}) \Rightarrow \rho(\{\llbracket P \rrbracket(h_1, l_1)^{\mathbb{L}}\}) = \rho(\{\llbracket P \rrbracket(h_2, l_2)^{\mathbb{L}}\})$
$(\eta)P(\phi \rightsquigarrow \rho) \text{ if } \forall h_1, h_2 \in \mathbb{V}^{\mathbb{H}}, \forall l \in \mathbb{V}^{\mathbb{L}}. \rho(\llbracket P \rrbracket(\phi(\{h_1\}), \eta(\{l\}))^{\mathbb{L}}) = \rho(\llbracket P \rrbracket(\phi(\{h_2\}), \eta(\{l\}))^{\mathbb{L}})$

Table 1. Narrow and Abstract Non-Interference.

of observing the concrete semantics of programs, namely the concrete values of public data, the attackers can only observe *properties* of public data, namely *abstract semantics* of the program. For this reason we model attackers by means of abstract domains. Formally, the *lattice of abstract domains* of a concrete domain C is isomorphic to the lattice $uco(C)$ of all the upper closure operators on C [8]. An *upper closure operator* $\rho : C \rightarrow C$ on a poset C is monotone, idempotent, and extensive¹. The *model of an attacker*, also called *attacker*, is therefore a pair of abstractions $\langle \eta, \rho \rangle$, with $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathbb{L}}))$, representing what an observer can see about, respectively, the input and output of a program. The notion of *narrow (abstract) non-interference* (NANI), denoted $[\eta]P(\rho)$, is given in Table 1. It says that if the attacker is able to observe the property η of public input, and the property ρ of public output, then no information flow concerning the private input is observable from the public output. The problem with this notion is that it may introduce *deceptive flows* [14], generated by different public outputs due to different public inputs with the same η property. Consider, for instance, $[Par]l := l * h^2(Sign)^2$, then we can observe a variation of the output's sign due to the existence of both negative and positive even numbers, revealing flows not due to private data, since h cannot affect the sign of the result. Most known models for weakening non-interference (e.g., PER model [23]) and for declassifying information (e.g., robust declassification [29]) corresponds to instances of NANI [14, 17]. In order to avoid deceptive interference we introduce a weaker notion of non-interference. In this case, the *set* of all the elements sharing property η is used as the public input. Moreover we consider also a property $\phi \in uco(\wp(\mathbb{V}^{\mathbb{H}}))$, modeling the private property that has not to be observed by the attacker $\langle \eta, \rho \rangle$. This notion, denoted $(\eta)P(\phi \rightsquigarrow \rho)$, is called *abstract non-interference* (ANI) and is defined in Table 1. Note that $[\text{id}]P(\text{id})$ models exactly (standard) non-interference. Moreover, we have that abstract non-interference is a weakening of both, standard and narrow non-interference: $[\text{id}]P(\text{id}) \Rightarrow (\eta)P(\phi \rightsquigarrow \rho)$ and $[\eta]P(\rho) \Rightarrow (\eta)P(\phi \rightsquigarrow \rho)$, while standard non-interference is not stronger than the narrow version, due to deceptive interference. In [14], two methods for deriving the most concrete output observation for a program, given the input one, for both narrow and abstract non-interference are provided. In particular the idea is that of abstracting in the same object all the elements that, if distinguished, would generate a visible flow. These most concrete output observations, that are not able to get information from the program P observing η in input, are, respectively, denoted $[\eta][\llbracket P \rrbracket](\text{id})$ and $(\eta)[\llbracket P \rrbracket](\phi \rightsquigarrow \text{id})$, both in $uco(\wp(\mathbb{V}^{\mathbb{L}}))$.

¹ $\forall x \in C. x \leq_C \rho(x)$.

² Note that $Par \stackrel{\text{def}}{=} \{\top, ev, od, \perp\}$ and $Sign \stackrel{\text{def}}{=} \{\top, 0+, -, \perp\}$.

3 Generalized Abstract Non-Interference

In this section, we introduce a generalization of abstract non-interference, called *generalized abstract non-interference* (shortly GANI), which subsumes many of the known notions of non-interference based on tree-like computations and automata. Abstract interpretation plays a key role in this generalization: The abstraction represents here both what an attacker may observe about a computation (as in abstract non-interference) and which aspects of the computation are relevant for checking non-interference, aspects determined by the specific notion of non-interference that we have to enforce on the system. Non-interference corresponds to asking that relevant (confidential) aspects of the computation have no effects on what an attacker observes of the computation. Moreover, what an attacker may observe is indeed composed by two aspects: what the particular notion of non-interference *allows* to observe, and what effectively the attacker *can* observe. In the following, we consider computational systems S modeled by their tree semantics $\{\!\{S\}\!\}$, i.e., the set of all the trees of computations of S . The corresponding trace and I/O denotational semantics are, respectively, denoted by $\langle\!\langle S \rangle\!\rangle$ and $\llbracket S \rrbracket$.

We define generalized non-interference by means of three abstractions in the standard framework of abstract interpretation, i.e., additive functions, each one with a specific and precise meaning, depending on the given notion of non-interference, and depending on the attacker model. The chosen policy of non-interference decides two of these abstractions:

α_{obs} : The first abstraction α_{obs} abstracts the tree semantics in the model used in the notion of non-interference that has to be enforced. Note that, the abstraction level chosen for defining non-interference corresponds to *delegating particular parts* (i.e., aspects) *of the system to release information* [24]. For instance, if we want to check standard non-interference for imperative programming languages, then α_{obs} corresponds to the denotational semantics abstraction of the computational tree. We call this abstraction the *observation abstraction*. Such an abstraction extracts always an observational property from semantic trees, e.g., all the computational traces, the I/O relations, etc.;

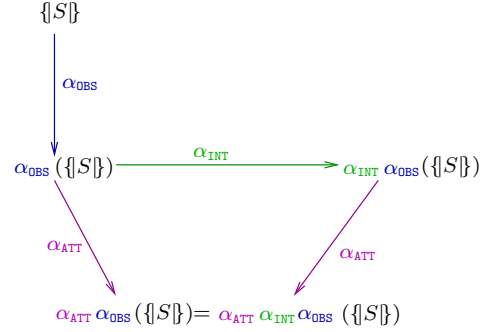
α_{int} : The second abstraction α_{int} characterizes the maximal amount of information that an attacker should observe, in the chosen policy. This abstraction *regulate what information may be released* [24]. For example, if we have to check non-interference in SPA [10], then we want the computations where private actions are hidden to be equivalent to the computations where private actions are avoided. Namely the set of all the computations where private actions are avoided is the maximal information that the attacker is allowed to observe. In this case α_{int} selects only those computations where private actions are not executed. This abstraction, called *interference abstraction*, forgets about all information which should not be observed by an attacker. Such an abstraction always selects the subset of the possible computations that we allow the attacker to observe, namely it is such that for each X in its domain, $\alpha_{\text{int}}(X) \subseteq X$.

These two abstractions tells us that, in general, non-interference holds whenever the amount of information that an attacker can grasp from a computation is precisely what, for the given notion of non-interference, that attacker *is allowed* to observe about it.

Finally, we have to model the observational capability of the *passive* attacker observing the system, and we consider a further abstraction α_{ATT} , called *attacker abstraction*, which characterizes the model of the attacker, namely what it can observe of the system behavior. In this case the attacker is passive since it cannot interfere with the execution of the program, and it cannot control the inputs of the system. By using these three abstractions we define generalized abstract non-interference for the system S as

$$\alpha_{\text{ATT}} \circ \alpha_{\text{OBS}}(\{\!|S|\!\}) = \alpha_{\text{ATT}} \circ \alpha_{\text{INT}} \circ \alpha_{\text{OBS}}(\{\!|S|\!\})$$

This equation says that, in the model chosen by α_{OBS} , the maximal amount of information that the attacker *is allowed to* observe, determined by $\alpha_{\text{ATT}} \circ \alpha_{\text{INT}}$, is exactly what the attacker *does* observe, determined by α_{ATT} . In other words, this definition of non-interference says that the attacker, modelled by the abstraction α_{ATT} , cannot distinguish between the observable computations (α_{OBS}) and the set of only those computations that the attacker should observe ($\alpha_{\text{INT}} \circ \alpha_{\text{OBS}}$).



The Generalized Non-Interference Policy. It is worth noting that this definition of GANI in general is characterized by a *possibilistic* interpretation of equality and doesn't provide an explicit notion of non-interference. Indeed, in ANI [14], the non-interference policy states that all the computations with the same public input has to provide the same public outputs. We can think of generalizing the notion of non-interference by checking the equality of public observations of the outputs for all the computations sharing a common maximal partial execution, instead of sharing only the public input. Consider the tree semantics $\{\!|S|\!\}$ of the system S .

$$\text{A system } S \text{ is } \textit{secure} \text{ if } \forall \sigma \in \alpha_{\text{INT}} \circ \alpha_{\text{OBS}}(\{\!|S|\!\}), \forall \delta \in \alpha_{\text{OBS}}(\{\!|S|\!\}) . \\ \delta \preceq_{\text{max}} \sigma \Rightarrow \alpha_{\text{ATT}}(\delta) = \alpha_{\text{ATT}}(\sigma)$$

where the relation \preceq_{max} , specifies the maximal subtree which δ shares with an element in $\alpha_{\text{OBS}}(\{\!|S|\!\})$. The definition above clearly, depends on the subtree relation \preceq . Consider $\sigma \in \alpha_{\text{INT}} \circ \alpha_{\text{OBS}}(\{\!|S|\!\})$ and consider $\delta \in \alpha_{\text{OBS}}(\{\!|S|\!\})$: $\delta \preceq_{\text{max}} \sigma$ if

$$\exists \pi \preceq \delta . \pi \preceq \sigma \wedge \forall \pi' \neq \pi . \pi \preceq \pi' \preceq \delta, \forall \sigma' \in \alpha_{\text{OBS}}(\{\!|S|\!\}) \text{ then } \pi' \not\preceq \sigma'$$

The definition above is based on the observation that if a *computation* has a maximal partial computation in common with what can be surely observed by

the attacker, then it is in those points, where the common partial computations end, that some private action has interfered in the computation.

Example 1. Let A be a system, if $\alpha_{\text{obs}}(\{A\}) = \{1 \rightarrow 2 \rightarrow 3, 1 \rightarrow 2 \rightarrow 4, 1 \rightarrow 3 \rightarrow 2\}$ and $\alpha_{\text{INT}} \circ \alpha_{\text{obs}}(\{A\}) = \{1 \rightarrow 2 \rightarrow 3, 1 \rightarrow 3 \rightarrow 2\}$ then $1 \rightarrow 2 \rightarrow 3 \preceq_{\text{max}} 1 \rightarrow 2 \rightarrow 3$, $1 \rightarrow 2 \rightarrow 4 \preceq_{\text{max}} 1 \rightarrow 2 \rightarrow 3$ and $1 \rightarrow 3 \rightarrow 2 \preceq_{\text{max}} 1 \rightarrow 3 \rightarrow 2$.

3.1 Abstract non-interference as GANI

In this section, we show that abstract non-interference [14], which generalizes standard non-interference [5, 15], is an instance of GANI. For deterministic programs the standard denotational semantics is given as the abstraction approximating traces with input/output relations (functions for deterministic programs) [6]³. Let X be a set of traces, the denotational semantics is defined: $\alpha^{\mathcal{D}}(X) = \lambda \sigma_{\perp}. \{ \sigma_{\perp} \mid \sigma \in X, |\sigma| < \omega \} \cup \{ \perp \mid |\sigma| = \omega \}$, where σ_{\perp} and σ_{\dashv} denote respectively the initial and the final states of the trace σ . Given two closures $\phi \in \text{uco}(\mathbb{V}^{\text{H}})$ and $\eta \in \text{uco}(\mathbb{V}^{\text{L}})$, we define the abstraction $\alpha_{\phi}^{\eta} : (\Sigma \rightarrow \Sigma) \rightarrow \wp(\wp(\Sigma) \times \wp(\Sigma))$ such that for any $f : \Sigma \rightarrow \Sigma$:

$$\alpha_{\phi}^{\eta}(f) = \{ \langle S_{\perp}, S_{\dashv} \rangle \mid S_{\perp} = \langle \phi(h), \eta(l) \rangle, h \in \mathbb{V}^{\text{H}}, l \in \mathbb{V}^{\text{L}}, S_{\dashv} = f(\phi(h), \eta(l)) \}$$

The idea is to abstract denotational input/output semantics to the set of all the possible associations between the corresponding input/output abstract states. In this way, we model the observation made by the attacker, which consists precisely in the ability to observe input/output abstract values. Consider the function $\mathcal{C}_{\text{H}} : \wp(\mathbb{V}^{\text{H}}) \rightarrow \mathbb{V}^{\text{H}}$ that uniquely chooses an element in the domain of values \mathbb{V}^{H} . Note that the equation $\forall h_1, h_2. \rho(\llbracket P \rrbracket(\langle \phi(h_1), \eta(l) \rangle)^{\text{L}}) = \rho(\llbracket P \rrbracket(\langle \phi(h_2), \eta(l) \rangle)^{\text{L}})$ is equivalent to the equation $\forall h. \rho(\llbracket P \rrbracket(\langle \phi(h), \eta(l) \rangle)^{\text{L}}) = \rho(\llbracket P \rrbracket(\langle \phi(\mathcal{C}_{\text{H}}(\mathbb{V}^{\text{H}})), \eta(l) \rangle)^{\text{L}})$. Therefore abstract non-interference can be formulated as follows:

$$\forall h \in \mathbb{V}^{\text{H}}. \rho(\llbracket P \rrbracket(\phi(h), \eta(l))^{\text{L}}) = \rho(\llbracket P \rrbracket(\phi(\mathcal{C}_{\text{H}}(\mathbb{V}^{\text{H}})), \eta(l))^{\text{L}})$$

At this point, we can define the *interference* abstraction $\alpha_{\text{ANI}} : \wp(\wp(\Sigma) \times \wp(\Sigma)) \rightarrow \wp(\wp(\Sigma) \times \wp(\Sigma))$, which selects only the observation with the fixed private input, hence for any $\mathcal{F} \in \wp(\wp(\Sigma) \times \wp(\Sigma))$:

$$\alpha_{\text{ANI}}(\mathcal{F}) = \{ \langle S_{\perp}, S_{\dashv} \rangle \in \mathcal{F} \mid \exists l \in \mathbb{V}^{\text{L}}. S_{\perp} = \langle \phi(\mathcal{C}_{\text{H}}(\mathbb{V}^{\text{H}})), \eta(l) \rangle \}$$

In order to obtain abstract non-interference, we assume that the attacker may observe only the ρ abstraction of the low output. This process is encoded by the attacker abstraction, which depends upon the input/output abstractions $\eta, \rho \in \text{uco}(\wp(\mathbb{V}^{\text{L}}))$, i.e., $\alpha_{\text{ATT}}^{\rho\eta} : \wp(\wp(\Sigma) \times \wp(\Sigma)) \rightarrow \wp(\wp(\mathbb{V}^{\text{L}}) \times \wp(\mathbb{V}^{\text{L}}))$ where

$$\alpha_{\text{ATT}}^{\rho\eta}(\mathcal{F}) = \{ \langle \eta(X_{\text{L}}), \rho(Y_{\text{L}}) \rangle \mid \langle \langle X_{\text{H}}, X_{\text{L}} \rangle, \langle Y_{\text{H}}, Y_{\text{L}} \rangle \rangle \in \mathcal{F} \}$$

Then, we can specify abstract non-interference in the following theorem.

³ For deterministic systems the trace semantics coincides with the tree semantics.

Theorem 1. $\alpha_{\text{ATT}}^{\rho\eta} \circ \alpha_{\phi}^{\eta}(\llbracket P \rrbracket) = \alpha_{\text{ATT}}^{\rho\eta} \circ \alpha_{\text{NANI}} \circ \alpha_{\phi}^{\eta}(\llbracket P \rrbracket)$ iff $(\eta)P(\phi \rightsquigarrow \rho)$.

Note here that the observation abstraction is the composition $\alpha_{\phi}^{\eta} \circ \alpha^{\mathcal{P}}$.

As far as the narrow case is concerned, we have to check if the possible executions with the high variable ranging on the whole concrete domain \mathbb{V}^{H} and the low variables ranging on the set of values with the same property η are equal to the interference abstraction obtained by setting the high variable to $\mathcal{C}_{\text{H}}(\mathbb{V}^{\text{H}})$ and the low one to any fixed value in the given property of low variables. This means that we have to change the interference abstraction given above as follows, where $\mathcal{C}_{\text{L}} : \wp(\mathbb{V}^{\text{L}}) \rightarrow \mathbb{V}^{\text{L}}$ is a function that uniquely selects an element from sets of values: $\alpha_{\text{NANI}} : \wp(\wp(\Sigma) \times \wp(\Sigma)) \rightarrow \wp(\wp(\Sigma) \times \wp(\Sigma))$

$$\alpha_{\text{NANI}}(\mathcal{F}) = \left\{ f \mid \begin{array}{l} \exists l \in \mathbb{V}^{\text{L}} . f = \langle \langle \mathcal{C}_{\text{H}}(\mathbb{V}^{\text{H}}), \eta(l) \rangle, S_{\downarrow} \rangle \langle \langle \mathcal{C}_{\text{H}}(\mathbb{V}^{\text{H}}), l' \rangle, S_{\downarrow} \rangle \in \mathcal{F} \\ l' = \mathcal{C}_{\text{L}}(\{ y \in \mathbb{V}^{\text{L}} \mid \eta(y) = \eta(l) \}) \end{array} \right\}$$

Therefore, we can rewrite also narrow abstract non-interference (NANI).

Theorem 2. $\alpha_{\text{ATT}}^{\rho\eta} \circ \alpha_{\text{id}}^{\text{id}}(\llbracket P \rrbracket) = \alpha_{\text{ATT}}^{\rho\eta} \circ \alpha_{\text{NANI}} \circ \alpha_{\text{id}}^{\text{id}}(\llbracket P \rrbracket)$ iff $[\eta]P(\rho)$.

3.2 GANI in concurrency

In [10], the authors introduced a classification of security properties for *security process algebras*. Since process algebras can be modeled by computational trees, we show how different security properties defined in [10] can be re-interpreted as instances of the generalized abstract non-interference. In the following we consider the process algebra SPA introduced in [10]. We only remind the reader that, if $L \subseteq \text{Act}$, then $P \setminus L$ can execute all the actions P is able to do, provided that they do not belong to L , $P \setminus_I L$ can execute all the actions P is able to do, provided that they do not belong to $L \cap I$, and P/L hides the actions in L .

Consider a process $P \in \text{SPA}$, whose computational tree is $\{\!\{P\}\!\}$. We start by considering NNI (non-deterministic non-interference) which is defined by using the trace equivalence \approx_{T} in the following way: $(P \setminus_I \text{H})/\text{H} \approx_{\text{T}} P/\text{H}$, where P/H means that all the action in H (high) are hidden, i.e., they are substituted by the internal action ε , while $P \setminus_I \text{H}$ means that all the actions in H which are input actions cannot be executed by P . Then, we can translate this definition as GANI. It is clear that the definition of NNI considers the concrete system P , this means that $\alpha_{\text{OBS}} \stackrel{\text{def}}{=} \text{id}$. On the other hand, we have that what an external user can observe is the system having the high actions hidden. Therefore, we have to define the attacker abstraction that hides high-level actions. In the following let \mathbb{T}_{Act} be the set of all the semantic trees on the set of actions Act , while let \mathbb{T}_{L} be the set of all the semantic trees where the private actions are hidden. Let $\tau \in \mathbb{T}_{\text{Act}}$ and consider the following function: $\text{low} : \mathbb{T}_{\text{Act}} \rightarrow \mathbb{T}_{\text{L}}$ such that $\text{low}(\tau)$ is the tree where any label $\sigma \in \text{H}$ in τ is substituted by ε . Let $\{\!\{P\}\!\}$ the semantics of P specified as a computational tree. We define the function α_{low} as follows: $\alpha_{\text{low}}(\{\!\{P\}\!\}) = \{ \text{low}(\tau) \mid \tau \in \{\!\{P\}\!\} \}$. This specifies the attacker abstraction in GANI and it is such that: $\{\!\{P/\text{H}\}\!\} = \alpha_{\text{low}}(\{\!\{P\}\!\})$. Moreover, we can note that NNI is defined by using trace equivalence, this means that the

attacker can analyze traces of computations only. By definition two systems are trace equivalent if they accept the same language, therefore we have to make equal the α_T abstraction of the result, namely $\alpha_{\text{ATT}} \stackrel{\text{def}}{=} \alpha_T \circ \alpha_{1\text{ow}}$. Finally, consider the operation $P \setminus_I \mathbb{H}$ which avoids high-level inputs. Let I be the set of input actions, then we can define the abstraction: $\alpha_L^I : \wp(\mathbb{T}_{Act}) \longrightarrow \wp(\mathbb{T}_{Act})$ such that for any $T \subseteq \mathbb{T}_{Act}$: $\alpha_L^I(T) = \{ \tau \in T \mid \forall \sigma \in \tau . \sigma \notin \mathbb{H} \cap I \}$, where $\sigma \in \tau$ is a shorthand notation for σ being an action (node) in τ . Then we have that $\wp(P \setminus_I \mathbb{H}) = \alpha_L^I(\wp(P))$. At this point, we can derive the NNI as:

$$\boxed{\alpha_T \circ \alpha_{1\text{ow}}(\wp(P)) = (\alpha_T \circ \alpha_{1\text{ow}}) \circ \alpha_L^I(\wp(P))}$$

Consider now the notion of *Strong Non-deterministic Non-Interference* (SNNI) defined in [10]: P satisfies SNNI iff $P/\mathbb{H} \approx_T P \setminus \mathbb{H}$. In order to define SNNI as an instance of the generalized abstract non-interference, we have to define the operator P/\mathbb{H} , that hides all the high-level actions. Let $\alpha_L : \wp(\mathbb{T}_{Act}) \longrightarrow \wp(\mathbb{T}_{Act})$ be the map such that $\forall T \subseteq \mathbb{T}_{Act} : \alpha_L(T) = \{ \tau \in T \mid \forall \sigma \in \tau . \sigma \in \mathbb{L} \}$. This defines the interference abstraction in GANI and it is such that $\wp(P \setminus \mathbb{H}) = \alpha_L(\wp(P))$. The standard notion of SNNI introduced in [10] can be defined as

$$\boxed{\alpha_T \circ \alpha_{1\text{ow}}(\wp(P)) = \alpha_T \circ \alpha_{1\text{ow}} \circ \alpha_L(\wp(P))}$$

At this point, since bisimulations are equivalence relations [21], they can be viewed as abstractions of computational trees, i.e., a tree is abstracted into the equivalence class of all the trees bisimilar to it, then we can model both BNNI and BSNNI. In this context, we obtain this by substituting to α_T , the abstraction α_B , corresponding to the given chosen bisimulation, which associates with a computation the set of all the computations bisimilar to the given one.

Consider now non-deducibility on compositions (NDC) and the bisimulation-based NDC (BNDC) notions of non-interference. NDC is: $\forall II . P/\mathbb{H} \approx_T (P \parallel II) \setminus \mathbb{H}$, where II is a process that can execute only high-level actions. In [10] it is proved that NDC=SNNI, therefore also NDC can be modeled as a generalized abstract non-interference. The situation is different when we consider BNDC, i.e., $\forall II . P/\mathbb{H} \approx_B (P \parallel II) \setminus \mathbb{H}$, for the bisimulation relation B . In this case, we have that BNDC \neq BSNNI, and therefore we have to explicitly model it as generalized abstract non-interference. In [10] the authors also prove that BNDC can be equivalently formalized as: $\forall II . P \setminus \mathbb{H} \approx_B (P \parallel II) \setminus \mathbb{H}$. At this point, we note that we have to consider $\alpha_B \circ \alpha_L$ as α_{ATT} , since in this definition it is only observable what a low-level user (i.e., a user that can execute only low level actions) can see, which is only the computation without high-level actions. Moreover, we have that α_{OBS} is the identity, since, in this case, non-interference is defined on computational trees. Finally, we define α_{INT} noting that the semantics (computational tree) of $P \parallel II$ contains the semantics of P (which doesn't execute synchronizations), therefore we can define $\alpha_{\text{INT}}(\wp(P \parallel II)) = \wp(P)$, modeling BNDC as follows:

$$\boxed{\forall II . (\alpha_B \circ \alpha_L) \circ \alpha_{\text{INT}}(\wp(P \parallel II)) = \alpha_B \circ \alpha_L(\wp(P \parallel II))}$$

This is BNDC since in the right side of the equality α_L is applied to the semantics of $P \parallel II$, and therefore executes only the high-level actions of P .

Theorem 3. *Given a system P then:*

- P satisfies SNNI iff $\alpha_T \circ \alpha_{low}(\{P\}) = \alpha_T \circ \alpha_{low} \circ \alpha_L(\{P\})$;
- P satisfies BNDC iff $\forall II. (\alpha_B \circ \alpha_L) \circ \alpha_{int}(\{P||II\}) = \alpha_B \circ \alpha_L(\{P||II\})$.

3.3 GANI in real-time systems

Let A be a timed automaton [1], $\{A\}$ the corresponding computational tree semantics, and $\mathcal{L} \stackrel{\text{def}}{=} \alpha_T(\{A\})$ the corresponding timed accepted language, i.e., the sequence of all the computational traces of states $\langle \sigma, t \rangle$, where σ is an action executed at the time t . In [2] a notion of non-interference for timed automata is introduced. Given a natural number n , the authors say that high-level actions do not interfere with the system, by considering minimum delay n , if the system behaviour in absence of high-level actions is equivalent to the system behaviour, observed on low-level actions only, when high-level actions can occur with a delay between them greater than n . Let Σ be the alphabet of actions of A . We suppose that Σ is partitioned into two disjoint sets of actions H and L : H is the set of the high-level actions, while L is the set of the low-level ones. Consider the following languages:

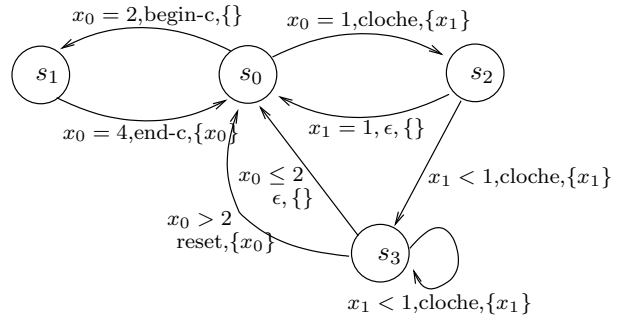
$$\begin{aligned} \mathcal{L}|_L &\stackrel{\text{def}}{=} \left\{ \overline{\langle \sigma, t \rangle} \in \mathcal{L} \mid \forall \langle \sigma_i, t_i \rangle \in \overline{\langle \sigma, t \rangle} . \sigma_i \in L \right\} \\ \mathcal{L}/H &\stackrel{\text{def}}{=} \left\{ w \mid \exists \overline{\langle \sigma, t \rangle} \in \mathcal{L} \text{ such that } w \text{ is the projection of } \overline{\langle \sigma, t \rangle} \right. \\ &\quad \left. \text{on the pairs } \{ \langle \sigma, t \rangle \mid \sigma \in L \} \right\} \\ \mathcal{L}_H^n &\stackrel{\text{def}}{=} \left\{ \overline{\langle \sigma, t \rangle} \in \mathcal{L} \mid \forall \langle \sigma_i, t_i \rangle, \langle \sigma_j, t_j \rangle \in \overline{\langle \sigma, t \rangle} . i \neq j, \right. \\ &\quad \left. \sigma_i, \sigma_j \in H \Rightarrow |t_i - t_j| \geq n \right\} \end{aligned}$$

So, $\mathcal{L}|_L$ avoids high-level actions, i.e., it takes only the traces of the system that make only low-level actions. On the other hand, \mathcal{L}/H hides all the high-level actions, i.e., it executes them and then it hides them. Finally, \mathcal{L}_H^n selects only those traces where the high-level actions are distant at least n . Then in [2] a system is said to be n -non-interfering iff $\mathcal{L}_H^n/H = \mathcal{L}|_L$.

Consider the example below [2]. This timed automaton have $L = \{\text{begin-c}, \text{end-c}\}$ and $H = \{\text{cloche}, \text{reset}\}$. There is only one possible trace of only low-level actions:

$$\langle \text{begin-c}, 2 \rangle \langle \text{end-c}, 4 \rangle \dots \langle \text{begin-c}, 2 + 4i \rangle \langle \text{end-c}, 4 + 4i \rangle \dots$$

If more than one cloche action is executed and the time elapsed between them is less than 1, then it is possible to execute the action reset, which can change the moment of the execution of begin-c, and therefore in this case we have an interference.



In particular, for example, we can have the trace

$$\langle \text{begin-c}, 2 \rangle \langle \text{end-c}, 4 \rangle \langle \text{cloche}, 5 \rangle \langle \text{cloche}, 5.6 \rangle \langle \text{cloche}, 6.3 \rangle \langle \text{begin-c}, 8.3 \rangle \dots$$

whose projection $\langle \text{begin-c}, 2 \rangle \langle \text{end-c}, 4 \rangle \langle \text{begin-c}, 6.3 \rangle \langle \text{end-c}, 8.3 \rangle$, on the low-level actions, is not the one described above. This means that in this system there is interference.

Consider the attacker abstraction α_{low} , defined in Sec. 3.2, the interference abstraction α_{L} and the language \mathcal{L}_{H}^n . We define the family of abstractions α_n , with $n \in \mathbb{N}$, as follows:

$$\alpha_n(\mathcal{L}) = \{ \tau \in \mathcal{L} \mid \forall \langle \sigma_i, t_i \rangle, \langle \sigma_j, t_j \rangle \in \tau. i \neq j, \sigma_i, \sigma_j \in \mathbb{H} \Rightarrow |t_i - t_j| \geq n \}$$

where each map α_n is additive and $\mathcal{L}_{\text{H}}^n = \alpha_n(\mathcal{L})$. Then the notion of non-interference introduced in [2] for timed automata can be specified as follows:

$$\boxed{\alpha_{\text{low}} \circ \alpha_n(\mathcal{L}) = \alpha_{\text{low}} \circ \alpha_{\text{L}} \circ \alpha_n(\mathcal{L})},$$

where $\alpha_{\text{L}} \circ \alpha_n = \alpha_{\text{L}}$. Note that, in this case, $\alpha_{\text{obs}} = \alpha_n \circ \alpha_{\text{T}}$.

Theorem 4. *A timed automaton, with timed language \mathcal{L} , satisfies n -non interference iff $\alpha_{\text{low}} \circ \alpha_n(\mathcal{L}) = \alpha_{\text{low}} \circ \alpha_{\text{L}} \circ \alpha_n(\mathcal{L})$.*

4 Deriving GANI attackers

In this section, we generalize the construction to GANI of the most powerful attacker [14]. Let A be a system and let α_{obs} and α_{int} be abstractions defining the chosen notion of non-interference for which A results insecure whenever observed by the attacker modeled by α_{att} . As we said, α_{obs} and α_{int} depend on the definition of non-interference that we chose, while α_{att} depends on what we decide to observe about the computation. Therefore if non-interference is not satisfied, i.e., the system is not secure as regards the chosen notion of non-interference, we can think of further abstracting the attacker abstraction in order to achieve security. The resulting abstraction provides a certificate of the security level of the system A with respect to the fixed observation and private abstractions. In order to find an abstraction that makes equal the sets $\alpha_{\text{att}} \circ \alpha_{\text{obs}}$ and $\alpha_{\text{att}} \circ \alpha_{\text{int}} \circ \alpha_{\text{obs}}$ we have to merge elements in both sets in order to make them contain the same new abstract objects. Hence, given a security policy determined by what is observable (α_{obs}), and what at most the attacker should observe ($\alpha_{\text{int}} \circ \alpha_{\text{obs}}$), we derive *from the program* the most concrete harmless attacker α_{att} for the given policy. Namely, we derive the minimal abstraction necessary in order to make GANI hold. Note that, in abstract non-interference [14] there is a clear criterion for collecting elements in order to build the abstraction: abstracting to the same object all the elements resulting from computations that differ only for private inputs. The corresponding construction for GANI is provided by the relation \preceq_{max} defined in the previous section. Hence, we use \preceq_{max} for defining the sets of objects that need to have the same abstraction in order to achieve secrecy.

Hence, $\forall \sigma \in \alpha_{\text{INT}} \circ \alpha_{\text{OBS}}(\{A\})$ the following set collects all the trees that have to be indistinguishable from the tree σ :

$$\Upsilon(\sigma) = [\sigma] \stackrel{\text{def}}{=} \{ \delta \in \alpha_{\text{OBS}}(\{A\}) \mid \delta \preceq_{\text{max}} \sigma \}$$

Example 2. Consider a system A , if we have $\alpha_{\text{OBS}}(\{A\}) = \{1 \rightarrow 2 \rightarrow 3, 1 \rightarrow 2 \rightarrow 4, 1 \rightarrow 3 \rightarrow 2\}$ and $\alpha_{\text{INT}} \circ \alpha_{\text{OBS}}(\{A\}) = \{1 \rightarrow 2 \rightarrow 3, 1 \rightarrow 3 \rightarrow 2\}$ then $[1 \rightarrow 2 \rightarrow 3] = \{1 \rightarrow 2 \rightarrow 3, 1 \rightarrow 2 \rightarrow 4\}$, $[1 \rightarrow 3 \rightarrow 2] = \{1 \rightarrow 3 \rightarrow 2\}$. While, if we have $\alpha_{\text{OBS}}(\{A\}) = \{1 \rightarrow 2 \rightarrow 4, 1 \rightarrow 2 \rightarrow 3, 1 \rightarrow 5 \rightarrow 3, 3 \rightarrow 5 \rightarrow 4, 1 \rightarrow 2 \rightarrow 5, 1 \rightarrow 3 \rightarrow 2, 3 \rightarrow 2 \rightarrow 1\}$ and $\alpha_{\text{INT}} \circ \alpha_{\text{OBS}}(\{A\}) = \{1 \rightarrow 2 \rightarrow 3, 1 \rightarrow 3 \rightarrow 2, 3 \rightarrow 2 \rightarrow 1\}$ then $[1 \rightarrow 2 \rightarrow 3] = \{1 \rightarrow 5 \rightarrow 3, 1 \rightarrow 2 \rightarrow 3, 1 \rightarrow 2 \rightarrow 4\}$, $[1 \rightarrow 3 \rightarrow 2] = \{1 \rightarrow 5 \rightarrow 3, 1 \rightarrow 3 \rightarrow 2\}$ and, finally, $[3 \rightarrow 2 \rightarrow 1] = \{3 \rightarrow 2 \rightarrow 1, 3 \rightarrow 5 \rightarrow 4\}$.

Similarly to [14], we define the set $\mathbb{D}_{\{A\}}$ collecting all computations that may fail secrecy, and $\text{Irr}_{\{A\}}$ collecting all computations for which secrecy cannot fail.

$$\begin{aligned} \mathbb{D}_{\{A\}} &= \{ [\sigma] \mid \sigma \in \alpha_{\text{INT}} \circ \alpha_{\text{OBS}}(\{A\}) \} \\ \text{Irr}_{\{A\}} &= \{ X \mid \forall \sigma \in \alpha_{\text{INT}} \circ \alpha_{\text{OBS}}(\{A\}) . X \notin \uparrow([\sigma]) \} \end{aligned}$$

The predicate $\text{Secr}_{\{A\}}$, which characterizes all the elements that should be contained in the abstraction modeling the most concrete harmless attacker, is defined as

$$\text{Secr}_{\{A\}}(X) \text{ iff } \forall \sigma \in \alpha_{\text{INT}} \circ \alpha_{\text{OBS}}(\{A\}) . (\exists Z \in [\sigma] . Z \subseteq X \Rightarrow \forall W \in [\sigma] . W \subseteq X)$$

Following the construction in [14], we can prove that $\mathcal{S}\{A\} \stackrel{\text{def}}{=} \{ X \mid \text{Secr}_{\{A\}}(X) \}$ is the most concrete abstraction that enforces the notion of GANI to hold, w.r.t. the relation \preceq_{max} .

Theorem 5. *Let A be a system. $\mathcal{S}\{A\}$ is the most concrete abstraction such that $\forall \sigma \in \alpha_{\text{INT}} \circ \alpha_{\text{OBS}}(\{A\}), \forall \delta \in \alpha_{\text{OBS}}(\{A\}) . \delta \preceq_{\text{max}} \sigma \Rightarrow \mathcal{S}\{A\}(\delta) = \mathcal{S}\{A\}(\sigma)$.*

Proposition 1. $\mathcal{S}(\{A\}) = \mathcal{S}(\uparrow(\mathbb{D}_{\{A\}})) \cup \text{Irr}_{\{A\}}$.

5 Conclusion

We introduced GANI as a generalization of abstract non-interference for automata and concurrent systems. We believe that the combination of abstract interpretation and non-interference may provide advanced techniques for analyzing, in a *modular* way, how sub-components of complex systems interact during computation and how, analyses at different levels of abstraction can be combined in a useful way. On one side, abstract interpretation has been proved to be the most appropriate framework for reasoning about properties of computations at different levels of abstraction. On the other side, strong-dependency, and in particular non-interference, is the most appropriate notion to disclose information-flows among sub-components of a system, when a variation of some

of them can be conveyed to the others. GANI is intended to bridge these two notions in order to provide adequate methods for studying properties of complex systems by analyzing the properties of computations that are conveyed among system sub-components. In this sense, GANI may provide a framework for studying the relation between different and interacting entities which may be reciprocally influenced by the action of computing, giving advanced techniques for systematically classifying the information leakage in the lattice of abstractions. Moreover, the advantage of specifying different notions of non-interference for sequential, concurrent and timed systems as GANI relies upon the possibility offered by abstract interpretation to systematically derive abstractions. This paper does not contain any tool support for the analysis, clearly such a tool would provide an evidence on how the framework can be used, and indeed this problem deserves further work. However, the definition of a general schema for defining security properties allows to study relationship among different properties. Moreover, by using a unique model and unique schema, parametric on the security policy and on the computational system, it is possible to develop more general theories which could then be applied to a number of definitions by simply instantiating them, in the same spirit as [12].

References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. R. Barbuti, N. De Francesco, A. Santone, and L. Tesei. A notion of non-interference for timed automata. *Fundamenta Informaticae*, 51:1–11, 2002.
3. D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, MITRE Corp. Bedford, MA, 1973.
4. D. Clark, C. Hankin, and S. Hunt. Information flow for algol-like languages. *Computer Languages*, 28(1):3–28, 2002.
5. E. S. Cohen. Information transmission in sequential programs. *Foundations of Secure Computation*, pages 297–335, 1978.
6. P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theor. Comput. Sci.*, 277(1-2):47,103, 2002.
7. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of Conf. Record of the 4th ACM Symp. on Principles of Programming Languages (POPL '77)*, pages 238–252. ACM Press, New York, 1977.
8. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of Conf. Record of the 6th ACM Symp. on Principles of Programming Languages (POPL '79)*, pages 269–282. ACM Press, New York, 1979.
9. D. E. Denning and P. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.
10. R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer security*, 3(1):5–33, 1995.
11. R. Focardi and R. Gorrieri. Classification of security properties (part i: Information flow). In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

12. Riccardo Focardi and Fabio Martinelli. A uniform approach for the definition of security properties. In *World Congress on Formal Methods (1)*, pages 794–813, 1999.
13. Riccardo Focardi, Sabina Rossi, and Andrei Sabelfeld. Bridging language-based and process calculi security. In *FoSSaCS*, pages 299–315, 2005.
14. R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *Proc. of the 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '04)*, pages 186–197. ACM-Press, NY, 2004.
15. J. A. Goguen and J. Meseguer. Security policies and security models. In *Proc. IEEE Symp. on Security and Privacy*, pages 11–20. IEEE Computer Society Press, 1982.
16. Kohei Honda, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Secure information flow as typed process behaviour. In *ESOP*, pages 180–199, 2000.
17. S. Hunt and I. Mastroeni. The per model of abstract non-interference. In *Proc. of The 12th Internat. Static Analysis Symp. (SAS'05)*, Lecture Notes in Computer Science. Springer-Verlag, 2005. To appear.
18. R. Joshi and K. R. M. Leino. A semantic approach to secure information flow. *Science of Computer Programming*, 37:113–138, 2000.
19. P. Laud. Semantics and program analysis of computationally secure information flow. In *In Programming Languages and Systems, 10th European Symp. On Programming, ESOP*, volume 2028 of *Lecture Notes in Computer Science*, pages 77–91. Springer-Verlag, 2001.
20. Heiko Mantel and Andrei Sabelfeld. A unifying approach to the security of distributed and multi-threaded programs. *Journal of Computer Security*, 11(4):615–676, 2003.
21. R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1989.
22. A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE J. on selected areas in communications*, 21(1):5–19, 2003.
23. A. Sabelfeld and D. Sands. A PER model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation*, 14(1):59–91, 2001.
24. A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *Proc. of 18th IEEE Computer Security Foundations Workshop (CSFW-18)*. IEEE Comp. Soc. Press, 2005.
25. C. Skalka and S. Smith. Static enforcement of security with types. In *ICFP'00*, pages 254–267. ACM Press, New York, 2000.
26. D. Volpano. Safety versus secrecy. In *Proc. of the 6th Static Analysis Symp. (SAS'99)*, volume 1694 of *Lecture Notes in Computer Science*, pages 303–311. Springer-Verlag, 1999.
27. D. Volpano, G. Smith, and C. Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2,3):167–187, 1996.
28. M. Zanotti. Security typings by abstract interpretation. In *Proc. of The 9th Internat. Static Analysis Symp. (SAS'02)*, volume 2477 of *Lecture Notes in Computer Science*, pages 360–375. Springer-Verlag, 2002.
29. S. Zdancewic and A. C. Myers. Robust declassification. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 15–23. IEEE Computer Society Press, 2001.