Adjoining Declassification and Attack Models by Abstract Interpretation

Roberto Giacobazzi and Isabella Mastroeni

Dipartimento di Informatica,
Università di Verona,
Strada Le Grazie 15, I-37134 Verona, Italy
{roberto.giacobazzi, mastroeni}@sci.univr.it

Abstract. In this paper we prove that attack models and robust declassification in language-based security can be viewed as adjoint transformations of abstract interpretations. This is achieved by interpreting the well known Joshi and Leino's semantic approach to non-interference as a problem of making an abstraction complete relatively to a program's semantics. This observation allows us to prove that the most abstract property on confidential data which flows, here called private observation, and the most concrete harmless attacker observing public data, here called public observable, both modeled as abstractions of the program's semantics, are respectively the adjoint solutions of a completeness problem in standard abstract interpretation theory. In particular declassification corresponds to refining the given model of an attacker with the minimal amount of information in order to achieve completeness, which is non-interference, while the harmless attacker corresponds to remove this information. This proves an adjunction relation between two basic approaches to language-based security: declassification and the construction of suitable attack models, and allows us to apply relevant techniques for abstract domain transformation in language-based security.

Keywords: Abstract interpretation, language-based security, declassification, abstract non-interference, attack models, adjunction, completeness.

1 Introduction

Many security problems in language-based security are problems of confidentiality: If a user wants to keep some information confidential then he/she has to state a policy stipulating that no data visible from other users is affected by confidential data. This policy allows programs to manipulate private data, unless visible/public outputs of those programs do not improperly reveal information about the data [27]. The usual way used to show confidentiality is to prove that an attacker cannot observe any difference between the public outputs of any two executions differing only in their private inputs with the assumption that an attacker (or unauthorized user) is allowed to view only information that is

M. Sagiv (Ed.): ESOP 2005, LNCS 3444, pp. 295–310, 2005.

not confidential. In this case the program is said to satisfy *non-interference* [18], also referred as *secrecy* [27, 30]. In standard non-interference, the attacker can fully analyze concrete computations. In this case, any conservative type/data-flow/control-flow analysis of information flows would discard all the programs which may provide any explicit or implicit concrete flows from confidential to public resources. Standard non-interference is therefore often too strict for practical use in language-based security. In order to adapt security policies to practical cases, it would be essential to know how much an attacker may learn from a program by (statically) analyzing its input/output behavior. This idea has recently lead to the definition of the notion of *abstract non-interference* [13] and robust declassification [32]. Abstract non-interference provides a method for modeling attackers as abstract interpretations [7, 8] of the input/output program behavior, in particular it is used for characterizing the most powerful attacker which is not able to disclose confidential properties, in the following called *harmless*. Declassification corresponds to downgrade the sensitivity of data in order to accommodate with (intentional) information leakage. In [13] and [32] systematic methods have been designed for deriving and analyzing respectively attack models and declassification by characterizing what information flows from confidential to public variables. It is clear that the stronger is the attacker, the more information can be released by the program. Namely, the more concrete is the model of the harmless attacker, the more abstract is the confidential information that can be kept private. This observation gives an intuitive explanation of the adjoint relation existing between the actions of weakening attackers and declassifying private information. In particular, we can note that when we derive the most concrete attack model, then we are looking for the most concrete *public observer*, while when we derive the most abstract property the flows during computation, for characterizing abstract declassification, we are looking for the most abstract *private observable.* Indeed, the most concrete public observer is the model of the most powerful attacker that can observe only public data. While, the most abstract private observable is the minimal amount of information that a program releases during computation.

In this paper, we prove that this duality corresponds precisely to an adjunction in the lattice of abstract interpretations. This is achieved by considering abstract non-interference as a generalization of both declassification for passive attackers and attack models. In this setting we prove that, under non restrictive hypotheses, abstract non-interference corresponds precisely to making abstract interpretation complete (see [17]) relatively to the denotational semantics of programs. This derives directly from an abstract interpretation-based generalization of Joshi and Leino's approach to secure information flows [19], which makes this approach equivalent to a completeness problem. Abstract interpretation plays a key role here, providing the adequate framework where program properties can be compared by considering their relative precision. In particular, we prove that declassification and attack models are adjoint notions and they correspond respectively to the minimal complete refinement, providing the most concrete *public observer* property of the program and the minimal complete simplification, providing the most abstract *private observable* property of the program.

2 Basic Notions

If S and T are sets, then $\wp(S)$ denotes the powerset of S, $S \times T$ denotes the Cartesian product of S and T, $S \smallsetminus T$ denotes the set-difference between S and T, $S \subsetneq T$ denotes strict inclusion, and for a function $f : S \rightarrow T$ and $X \subseteq T$, $f(X) \stackrel{\text{def}}{=} \{f(x) \mid x \in X\}$ and $f^{-1}(X) \stackrel{\text{def}}{=} \left\{ x \,\middle|\, f(x) \in X \right\}$. We will often denote $f(\{x\})$ as $f(x)$ and use lambda notation for functions. Function composition $\lambda x.\ f(g(x))$ is denoted $f \circ g$. $\langle P, \leq \rangle$ denotes a poset P with ordering relation $\leq$, while $\langle P, \leq, \vee, \wedge, \top, \bot \rangle$ denotes a complete lattice P, with ordering $\leq$, *lub* $\vee$, *glb* $\wedge$, greatest element (top) $\top$, and least element (bottom) $\bot$. Often, $\leq_P$ will be used to denote the underlying ordering of a poset P, and $\vee_P$, $\wedge_P$, $\top_P$ and $\bot_P$ denote the basic operations and elements if P is a complete lattice. $\mathtt{id} \stackrel{\text{def}}{=} \lambda x.\ x$ and $\mathbb{T} \stackrel{\text{def}}{=} \lambda x.\ \top$. If $S \subseteq P$ then $\downarrow S \stackrel{\text{def}}{=} \{x \in P \mid \exists y \in S.\ x \leq y\}$. $\downarrow x$ is a shorthand for $\downarrow \{x\}$. $f : C \rightarrow A$ is (completely) additive if f preserves *lub*'s of all subsets of C (emptyset included). Continuity holds when f preserved *lubs*'s of chains. Co-additivity and co-continuity are dually defined.

It is well known that abstract domains can be equivalently formulated either in terms of Galois connections or closure operators [8]. A pair of functions $\alpha : C \rightarrow A$ and $\gamma : A \rightarrow C$ on posets, denoted $\langle C, \alpha, A, \gamma \rangle$, forms an *adjunction* or a *Galois connection* (GC) if for any $x \in C$ and $y \in A$: $\alpha(x) \leq_A y \Leftrightarrow x \leq_C \gamma(y)$. α (resp. γ) is the *left-* (*right-*)*adjoint* to γ (α) and it is an additive (co-additive) function. Additive and co-additive functions f admit respectively right and left adjoint: $f^{+} \stackrel{\text{def}}{=} \lambda x.\ \bigvee \left\{ y \,\middle|\, f(y) \leq x \right\}$ and $f^{-} \stackrel{\text{def}}{=} \lambda x.\ \bigwedge \left\{ y \,\middle|\, x \leq f(y) \right\}$ respectively. Remember that $(f^{+})^{-} = (f^{-})^{+} = f$ [2]. If in addition for any $a \in A$: $\alpha(\gamma(a)) = a$, then we call $\langle C, \alpha, A, \gamma \rangle$ a Galois insertion (GI) of A in C. In GC-based abstract interpretation the concrete and abstract domains, C and A, are complete lattices [7]. An *upper (lower) closure operator* $\rho : P \rightarrow P$ on a poset P is monotone, idempotent, and extensive: $\forall x \in P.\ x \leq_P \rho(x)$ (reductive: $\forall x \in P.\ x \geq_P \rho(x)$). The set of all upper (lower) closure operators on P is denoted by *uco*(P) (*lco*(P)). Let $\langle C, \leq, \vee, \wedge, \top, \bot \rangle$ be a complete lattice. Closure operators are uniquely determined by the set of their fix-points $\rho(C)$. For upper closures, $X \subseteq C$ is the set of fix-points of $\rho \in uco(C)$ iff X is a *Moore-family* of C, i.e., $X = \mathcal{M}(X) \stackrel{\text{def}}{=} \{\wedge S \mid S \subseteq X\}$ — where $\wedge \varnothing = \top \in \mathcal{M}(X)$, iff X is isomorphic to an abstract domain A in a GI (C, α, A, γ), i.e., $A \cong \rho(C)$ with $\iota : \rho(C) \rightarrow A$ and $\iota^{-1} : A \rightarrow \rho(C)$ being an isomorphism, and $(C, \iota \circ \rho, A, \iota^{-1})$ is the GI, i.e., $\rho = \gamma \circ \alpha$. In this case $\rho(C)$ is a complete sub-lattice of C iff ρ is additive. Dual properties can be derived for lower closures. Therefore *uco*(C) is isomorphic to the so called *lattice of abstract interpretations of* C [8]. If C is a complete lattice then *uco*(C) and *lco*(C) ordered point-wise are also complete lattices. For upper closures $\langle uco(C), \sqsubseteq, \sqcup, \sqcap, \mathbb{T}, \mathtt{id} \rangle$ where for every $\rho, \eta \in uco(C)$, $\{\rho_i\}_{i \in I} \subseteq uco(C)$ and $x \in C$: $\rho \sqsubseteq \eta$ iff $\forall y \in C.\ \rho(y) \leq \eta(y)$ iff $\eta(C) \subseteq \rho(C)$; $(\sqcap_{i \in I} \rho_i)(x) = \wedge_{i \in I} \rho_i(x)$; and $(\sqcup_{i \in I} \rho_i)(x) = x \Leftrightarrow \forall i \in I.\ \rho_i(x) = x$. Dual properties can be derived for $\langle lco(C), \sqsubseteq, \sqcup, \sqcap, \lambda x.\ x, \lambda x.\ \bot \rangle$. In the following we will find particularly convenient to identify closure operators (and therefore abstract domains) with their sets of fix-points. The *disjunctive completion* of an abstract domain $\rho \in uco(C)$ is the most abstract domain able to represent the concrete disjunction of its objects: $\curlyvee(\rho) = \sqcup\{\eta \in uco(C) | \eta \sqsubseteq \rho \text{and } \eta \text{ is additive}\}$. ρ is disjunctive iff $\curlyvee(\rho) = \rho$ (cf. [8]). Closure operators and partitions are related concepts. If π is a partition (viz. an equivalence relation), then $[\cdot]_\pi$ is the corresponding equivalence class. A closure $\eta \in uco(\wp(S))$ induces a partition on S: $\{\ [x]_\eta \,|\, x \in S\ \}$, where $[x]_\eta \overset{\text{def}}{=} \{\ y \,|\, \eta(x) = \eta(y)\ \}$. The most concrete closure that induces the same partition of values as η is $\mathcal{P}(\eta) \overset{\text{def}}{=} \curlyvee(\{\ [x]_\eta \,|\, x \in S\ \})$. η is *partitioning* if $\eta = \mathcal{P}(\eta)$ [24]. The idea is that $\mathcal{P}(\eta)$ is the most concrete closure such that for any $y \in \mathcal{P}(\eta(x))$: $\mathcal{P}(\eta(x)) = \mathcal{P}(\eta(y))$, while in general $\eta(y) \subseteq \eta(x)$.

In abstract interpretation there are two equivalent ways to express the soundness of an abstraction [7]. Let C be a complete lattice, $f : C \to C$, (C, α, A, γ) be a Galois insertion, and $f^\sharp : A \to A$. Then (C, α, A, γ) and $f^\sharp$ provide a sound abstraction of f if $\alpha \circ f \le f^\sharp \circ \alpha$, or equivalently (by adjunction) if $f \circ \gamma \le \gamma \circ f^\sharp$. While these two definitions of soundness are equivalent, they are not equivalent when equality is required, i.e., when we consider completeness [8, 17, 15]. In the first case $\alpha \circ f = f^\sharp \circ \alpha$ means that no loss of precision is accumulated by approximating the input arguments of a given semantic function; while $f \circ \gamma = \gamma \circ f^\sharp$ means that no loss of precision is accumulated by approximating the result of computations on abstract objects. We follow [15] where the first is called *backward* ($\mathcal{B}$) and the second is called *forward* ($\mathcal{F}$) completeness. The problem of making abstract domains $\mathcal{B}$-complete has been solved in [17]. These results have been extended to $\mathcal{F}$-completeness in [15]. The key point in this construction is that there exists an either $\mathcal{B}$ or $\mathcal{F}$-complete abstract function $f^\sharp$ in an abstract domain A iff the best correct approximation $\alpha \circ f \circ \gamma$ of f in A is respectively either $\mathcal{B}$ or $\mathcal{F}$ complete. This means that both $\mathcal{F}$ and $\mathcal{B}$ completeness are properties of the underlying abstract domain A relatively to the concrete function f. In a more general setting let $f : C_1 \to C_2$ be a function on complete lattices C_1 and C_2, and $\rho \in uco(C_2)$ and $\eta \in uco(C_1)$ be abstract domains. $\langle \rho, \eta \rangle$ is a pair of $\mathcal{B}(\mathcal{F})$-complete abstractions for f if $\rho \circ f = \rho \circ f \circ \eta$ ($f \circ \eta = \rho \circ f \circ \eta$). In the following we denote by $\mathcal{F}(C_1, C_2, f) \overset{\text{def}}{=} \{\ \langle \rho, \eta \rangle \,|\, f \circ \eta = \rho \circ f \circ \eta\ \}$ and $\mathcal{B}(C_1, C_2, f) \overset{\text{def}}{=} \{\ \langle \rho, \eta \rangle \,|\, \rho \circ f = \rho \circ f \circ \eta\ \}$. A pair of domain transformers can be associated with any completeness problem. We follow [11, 16] by defining a *domain refinement* and *simplification* as any monotone function $\tau : uco(L) \to uco(L)$ such that $X \subseteq \tau(X)$ and $\tau(X) \subseteq X$ respectively. In [17] and [15], a constructive characterization of the most abstract refinement, called *complete shell*, and of the most concrete simplification, called *complete core*, of any domain, making it $\mathcal{F}$ or $\mathcal{B}$ complete, for a given continuous function f, is given as a solution of a simple domain equation. Consider the following basic operators on closures:

$$\boxed{\begin{array}{r|l} R_f^{\mathcal{F}} \overset{\text{def}}{=} \lambda X.\, \mathcal{M}(f(X)) & R_f^{\mathcal{B}} \overset{\text{def}}{=} \lambda X.\, \mathcal{M}(\bigcup_{y \in X} \max(f^{-1}(\downarrow y))) \\ C_f^{\mathcal{F}} \overset{\text{def}}{=} \lambda X.\, \{\ y \in L \,|\, f(y) \subseteq X\ \} & C_f^{\mathcal{B}} \overset{\text{def}}{=} \lambda X.\, \{\ y \in L \,|\, \max(f^{-1}(\downarrow y)) \subseteq X\ \} \end{array}}$$

Let $\ell \in \{\mathcal{F}, \mathcal{B}\}$. In [17] the authors proved that the only interesting cases, as far as the refinement and simplification towards ℓ-completeness are concerned, are respectively the most concrete $\beta \sqsupseteq \rho$ such that $\langle \beta, \eta \rangle$ is ℓ-complete and the most abstract $\beta \sqsubseteq \eta$ such that $\langle \rho, \beta \rangle$ is ℓ-complete. In particular given $\rho \in uco(C_2)$ the ℓ-complete shell of $\eta \in uco(C_1)$ is $\mathcal{R}_f^{\ell,\rho}(\eta) \overset{\text{def}}{=} \eta \sqcap R_f^{\ell}(\rho)$ and given

$\eta \in \mathit{uco}(C_1)$ the ℓ-complete core of $\rho \in \mathit{uco}(C_2)$ is $\mathcal{C}_f^{\ell,\eta}(\rho) \overset{\text{def}}{=} \rho \sqcup C_f^{\ell}(\eta)$. Note that, when f is additive $\max\{\, x \mid f(x) \leq y \,\} = \bigvee\{\, x \mid f(x) \leq y \,\} = f^{+}$, and therefore $\mathcal{B}(C_1, C_2, f) = \mathcal{F}(C_2, C_1, f^{+})$ (cf. [15]). Clearly, when we consider $f : C \to C$ and the constraint $\eta = \rho$, the above construction requires a fixpoint iteration on abstract domains: $\mathcal{R}_f^{\ell}(\rho) = \mathit{gfp}(\lambda X.\, \rho \sqcap R_f^{\ell}(X))$ and $\mathcal{C}_f^{\ell}(\rho) = \mathit{lfp}(\lambda X.\, \rho \sqcup C_f^{\ell}(X))$ are called respectively the *absolute ℓ-complete shell* and *core* of ρ for f. Note that $\mathcal{R}_f^{\ell} \in \mathit{lco}(\mathit{uco}(C))$ and $\mathcal{C}_f^{\ell} \in \mathit{uco}(\mathit{uco}(C))$ (see [17]). It is worth noting that ℓ-complete cores and shells are adjoint abstract domain transformers, i.e., adjoint functions on the lattice of abstract interpretations. For any $\eta \in \mathit{uco}(C_1)$ and $\rho \in \mathit{uco}(C_2)$: $C_f^{\ell}(\eta) \sqsubseteq \rho \;\Leftrightarrow\; \eta \sqsubseteq R_f^{\ell}(\rho)$, which, by definition, implies that $\mathcal{C}_f^{\ell,\eta}(\rho) \sqsubseteq \rho \;\Leftrightarrow\; \eta \sqsubseteq \mathcal{R}_f^{\ell,\rho}(\eta)$.

3 Information Flows in Language-Based Security

Confidential data are considered *private*, labeled with `H` (high-level of secrecy), while all other data are public, labeled with `L` (low-level of secrecy) [10]. Non-interference can be naturally expressed by using semantic models of program execution. This idea goes back to Cohen's work on *strong dependency* [6], 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"* [27]. When this happens, we say that the program has only *secure information flows* [1, 6, 9, 10, 19, 30]. 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 (`H`) or public (`L`). Program states in Σ are functions (represented as tuples) mapping variables in the set of values $\mathbb{V}$. Finite traces on Σ are denoted Σ^{+}. If $\mathtt{T} \in \{\mathtt{H}, \mathtt{L}\}$, $n = |\{x \in \mathit{Var}(P) | x : \mathtt{T}\}|$, and $v \in \mathbb{V}^{n}$, we abuse notation by denoting $v \in \mathbb{V}^{\mathtt{T}}$ the fact that v is a possible value for the variables with security type `T`. Moreover, we assume that any input s, can be seen as a pair (h, l), where $s^{\mathtt{H}} = h$ is a value for private data and $s^{\mathtt{L}} = l$ is a value for public data. In this case, *non-interference* can be formulated as follows.

A program P is *secure* if
$\forall$ input $s, t \,.\, s^{\mathtt{L}} = t^{\mathtt{L}} \;\Rightarrow\; (\llbracket P \rrbracket(s))^{\mathtt{L}} = (\llbracket P \rrbracket(t))^{\mathtt{L}}$

This problem has been formulated also as a *Partial Equivalence Relation* (PER) [28]. In this case we have that if the input data are equivalent under a given equivalent relation, then also the outputs are equivalent w.r.t. a corresponding output equivalence relation. The result is a PER on the domain of semantic functions which can be used to model non-interference as above, where the equality on public data can be generalized by considering any equivalence relation. McLean [23] treats possibilistic notions of non-interference for even non-deterministic programs in the context of trace semantics: A program is secure if the set of its traces is closed under a function *purge*, i.e., it is insensible by varying private inputs. In [22] the different notions of possibilistic non-interference are modeled in a odular way. Ryan [25], Focardi and Gorrieri [12] all provide a comprehensive treatment of non-interference for concurrent programs in process algebras, where attackers are modeled as view relations on computation trees. The standard methods for checking non-interference are based on security-type systems and data-flow/control-flow analysis. 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 [29, 31]. Similarly, data-flow/control-flow analysis techniques are devoted to statically discover flows of secret data into public variables [3, 4, 19, 20, 28]. All these approaches are characterized by the way they model attackers (or unauthorized users).

3.1 Joshi and Leino's Semantic-Based Approach

As we said above, a program is secure if any observation of the initial and final values of $l : \mathtt{L}$ do not provide any information about the initial value of $h : \mathtt{H}$ [19]. Assume that the adversary has knowledge of the program text and of the initial and final values of l. The idea of Joshi and Leino's semantic-based approach to language-based security is that of characterizing secure information flow as program equivalence, denoted by $\doteq$. They introduce a program $\mathtt{HH} \overset{\text{def}}{=}$ "assign to h an arbitrary value". Consider a program P for which we want to prove non-interference. The program $\mathtt{HH}; P$ corresponds to run P after having set h to an arbitrary value; while the program $P; \mathtt{HH}$ discards the final value of h resulting from the execution of P. Then a program P is said to be *secure* if

$$\mathtt{HH}\ ;\ P;\ \mathtt{HH}\ \doteq\ P\ ;\ \mathtt{HH} \tag{1}$$

where $\doteq$ is the relational input/output semantic equality between programs, namely for each possible input the two programs have to show the same public output behavior. In order to understand this characterization, note that the occurence of $\mathtt{HH}$ after P on both the sides of the equality indicates that only the final values of l are of interest, whereas the occurence of $\mathtt{HH}$ before P on the left side of the equality indicates that the program starts with an arbitrary assignment to h. Clearly, the two programs are input/output equivalent provided that the final value of l, produced by P, does not depend on the initial value of h, which is indeed standard non-interference.

3.2 Robust Declassification

Declassifying information means downgrading the sensitivity of data in order to accommodate with (intentional) information leakage. Robust declassification has been introduced in [32] as a systematic method to drive declassification by characterizing what information flows from confidential to public variables. In particular the observational attacker's capability is modeled by using equivalence relations as in PER models, and declassification of private data is obtained by

Table 1. Narrow and Abstract Non-Interference

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

manipulating these relations in a semantic-driven way. The semantics considered is the operational semantics, defined on a transition system. The authors provide a systematic method for identifying what the attacker could observe of the concrete execution traces, by iteratively refining the initial equivalence relation on the states of the program. At this point they declassify private data in order to make the attacker *blind*, i.e., they declassify all the information that the attacker can get from the execution of the program.

3.3 Abstract Non-interference: Attack Models and Declassification

In [13], we introduced the notion of abstract non-interference modeling weaker information flows, attack models, and declassification. The idea is that an attacker can observe only some properties, modeled as abstract interpretations of program semantics, of public concrete values. The *model of an attacker*, also called *attacker*, is therefore a pair of abstractions $\langle \eta, \rho \rangle$, with $\eta, \rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}}))$, representing what an observer can see about, respectively, the input and output of a program. The notion of *narrow (abstract) non-interference* (NNI) represents the first weakening of standard non-interference relatively to a given model of an attacker. When a program P satisfies narrow non-interference we write $[\eta]P(\rho)$, see Table 1. The problem with this notion is that it introduces *deceptive flows* [13]. Consider, for instance, $l := l * h^2$, and consider the public input property of being an even number, then we can observe a variation of the output's sign due to the existence of both negative and positive even numbers, revealing flows which does not depend on the private data, here called deceptive. In order to avoid deceptive interference we introduce a weaker notion of non-interference, having no deceptive flows, yet modeling properties of informations flows. Namely, such that, when 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 property ϕ of the private input is observable from the public output. Namely, ϕ represents the confidentialinformation that we want to keep secret. We call this notion *abstract non-interference* (ANI). When a program P satisfies abstract non-interference we write $(\eta)P(\phi \rightsquigarrow\!\!\!| \rho)$, where $\phi \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{H}}))$, see Table 1. Note that $[\mathtt{id}]P(\mathtt{id})$ models exactly (standard) non-interference. Moreover, we have that abstract non-interference is a weakening of both, standard and narrow non-interference: $\forall \eta, \rho \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{L}})), \phi \in \mathit{uco}(\wp(\mathbb{V}^{\mathtt{H}}))$ we have $[\mathtt{id}]P(\mathtt{id}) \ \Rightarrow\ (\eta)P(\phi \rightsquigarrow\!\!\!| \rho)$ and $[\eta]P(\rho) \ \Rightarrow\ (\eta)P(\phi \rightsquigarrow\!\!\!| \rho)$, while standard non-interference is not stronger than the narrow one due to deceptive interference. A proof-system has been introduced, in [14], for checking both narrow and abstract non-interference inductively on program's syntax. Moreover, in [13], 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 collecting in the same abstract 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 $[\eta][\![P]\!](\mathtt{id})$ and $(\eta)[\![P]\!](\phi \rightsquigarrow\!\!\mid \mathtt{id})$, both in $uco(\wp(\mathbb{V}^{\mathtt{L}}))$. The following theorem is proved in [13].

Theorem 1. *$[\eta][\![P]\!](\mathtt{id}) \sqsubseteq \rho \;\Leftrightarrow\; [\eta]P(\rho)$, $(\eta)[\![P]\!](\phi \rightsquigarrow\!\!\mid \mathtt{id}) \sqsubseteq \rho \;\Leftrightarrow\; (\eta)P(\phi \rightsquigarrow\!\!\mid \rho)$.*

Example 1. Consider the properties $Sign$ and Par, observing, respectively, the sign and the parity of integers, and the program fragment: $P \overset{\text{def}}{=} l := l * h^2$. with security typing: $h : \mathtt{H}$ and $l : \mathtt{L}$ and $\mathbb{V} = \mathbb{Z}$. Let us check if $(\mathtt{id})P(\mathtt{id} \rightsquigarrow\!\!\mid Par)$. Note that $Par([\![P]\!](2,1)^{\mathtt{L}}) = Par(4) = 2\mathbb{Z}$ while $Par([\![P]\!](3,1)^{\mathtt{L}}) = Par(9) = 2\mathbb{Z}+1$, which are clearly different, therefore in this case $(\mathtt{id})P(\mathtt{id} \rightsquigarrow\!\!\mid Par)$ doesn't hold. Consider $(\mathtt{id})P(Sign \rightsquigarrow\!\!\mid Par)$. Note that $Par([\![P]\!](Sign(2),1)^{\mathtt{L}}) = Par([\![P]\!](Sign(3),1)^{\mathtt{L}}) = Par(0+) = \mathbb{Z}$. In this case it is simple to check that $(\mathtt{id})P(Sign \rightsquigarrow\!\!\mid Par)$ holds.

The PER model of non-interference can be easily viewed as a narrow non-interference, where both input and output closures are partitioning, i.e., equivalence relations. This corresponds to narrow non-interference because in PERs the equivalence is checked on the program outputs of concrete computations. Abstract non-interference provides also an abstraction of declassification. The idea is to find the most abstract property on confidential data which has to be declassified in order to guarantee secrecy. For this reason we define the set:

$$\Pi_{\mathtt{P}}(\eta,\rho) \overset{\text{def}}{=} \left\{\, \langle \{\, h \in \mathbb{V}^{\mathtt{H}} \,\middle|\, \rho([\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}}) = A \,\}, \eta(l)\rangle \,\middle|\, l \in \mathbb{V}^{\mathtt{L}},\; A \in \rho \,\right\}$$

This is the set of all the pairs $\langle H, L\rangle \in \wp(\mathbb{V}^{\mathtt{H}}) \times \wp(\mathbb{V}^{\mathtt{L}})$, such that, whenever $\eta(l) = L$, then for any $h_1, h_2 \in H$, no information flows, from private to public, are revealed. We use this set for deriving a partition of private data that guarantees secrecy. For each $L \in \eta$, we define $\Pi_{\mathtt{P}}(\eta,\rho)_{|L} \overset{\text{def}}{=} \left\{\, H \,\middle|\, \langle H, L\rangle \in \Pi_{\mathtt{P}}(\eta,\rho) \,\right\}$. The partition on private data corresponds to the most abstract property that flows when the property observed of the public input is L: $\mathcal{P}(\bigsqcap_{L\in\eta} \mathcal{M}(\Pi_{\mathtt{P}}(\eta,\rho)_{|L}))$. In particular, it is the most abstract property that contains all the possible variations of private inputs that generate insecure information flows, and the most concrete such that each variation generates a flow. namely it uniquely represents the confidential information that flows into the public output.

Example 2. Consider the program fragment: $P = \;\; l := l * h^2$. $\Pi_{\mathtt{P}}(\mathtt{id}, Par)$ is the set $\{\, \langle \mathbb{Z}, l\rangle \mid l \in 2\mathbb{Z} \,\} \cup \{\, \langle 2\mathbb{Z}, l\rangle \mid l \in 2\mathbb{Z}+1 \,\} \cup \{\, \langle 2\mathbb{Z}+1, l\rangle \mid l \in 2\mathbb{Z}+1 \,\}$. Therefore by using the notation above we have that if $l \in 2\mathbb{Z}$ then $\Pi_{\mathtt{P}}(\mathtt{id}, Par)_{|l} = \mathbb{Z}$ and if $l \in 2\mathbb{Z}+1$ then $\Pi_{\mathtt{P}}(\mathtt{id}, Par)_{|l} = \{2\mathbb{Z}, 2\mathbb{Z}+1\}$. Therefore, the most abstract partition on private data that can be declassified is $\{2\mathbb{Z}, 2\mathbb{Z}+1\}$. In other words we have that by looking at the low variables the only information that leaks about the high variables is its parity.

In order to adapt robust declassification in [32] to the abstract non-interference case, we consider passive attackers only and a semantics observing the initial and the final states of computations. We follow [32] in defining the information leaked by an equivalence relation transformer $S[\eta,\rho]$ on Σ for each

$\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$: $s_1 S[\eta, \rho] s_2$ iff $s_1^{\mathtt{L}} \approx_\eta s_2^{\mathtt{L}}$ and $(\forall \sigma, \delta \in \Sigma^+ . \sigma_\vdash = s_1 \wedge \delta_\vdash = s_2 \Rightarrow \sigma_\dashv^{\mathtt{L}} \approx_\rho \delta_\dashv^{\mathtt{L}}))$, where $s_1, s_2 \in \Sigma$ and given $\sigma \in \Sigma^+$ such that $|\sigma| = n \in \mathbb{N}$, then $\sigma_\vdash \overset{\text{def}}{=} \sigma_0$ and $\sigma_\dashv \overset{\text{def}}{=} \sigma_{n-1}$. It is simple to verify that $s_1 S[\eta, \rho] s_2$ iff $\eta(s_1^{\mathtt{L}}) = \eta(s_2^{\mathtt{L}})$ and $\rho([\![P]\!](s_1)^{\mathtt{L}}) = \rho([\![P]\!](s_2)^{\mathtt{L}})$. This means that abstract robust declassification *a la* [32] characterizes the information leaked in narrow abstract non-interference.

4 Abstract Non-interference as Completeness

Joshi and Leino's semantic-based approach to information flows [19] provides a way to interpret abstract non-interference as the problem of making an abstraction complete [17]. By considering the denotational semantics of a program P, $[\![P]\!]$, the Equation (1) becomes a *backward* completeness problem if the semantics of HH could be described as an abstraction. Indeed the program that associates with private variables an arbitrary value can be interpreted as the closure that abstracts the private value to the *"don't know"* abstract value, i.e., the set of all the possible values for private variables. Therefore, we define the function $\mathcal{H} : \wp(\mathbb{V}) \longrightarrow \wp(\mathbb{V})$ in the following way (recall that $\wp(\mathbb{V}) = \wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}})$): $\mathcal{H} = \lambda X. \langle \mathbb{V}^{\mathtt{H}}, X^{\mathtt{L}} \rangle$, where $X^{\mathtt{L}} \overset{\text{def}}{=} \{\, l \,|\, \langle h, l \rangle \in X \,\}$. It is straightforward to prove its monotonicity, idepotence and extensivity. So we can finally conclude that

$$[\![\mathtt{HH}\ ;\ P;\ \mathtt{HH}]\!] = \mathcal{H} \circ [\![P]\!] \circ \mathcal{H} \text{ and } [\![P;\ \mathtt{HH}]\!] = \mathcal{H} \circ [\![P]\!]$$

Hence, non-interference can be equivalentely formalized as $\mathcal{H} \circ [\![P]\!] \circ \mathcal{H} = \mathcal{H} \circ [\![P]\!]$. The idea is to transform $\mathcal{H}$ in order to either refine or simplify the abstraction in order to get completeness, and therefore, abstract non-interference. This can be achieved by observing that $\mathcal{H} = \lambda X. \langle \mathbb{T}(X^{\mathtt{H}}), \mathtt{id}(X^{\mathtt{L}}) \rangle = \lambda X. \langle \mathbb{V}^{\mathtt{H}}, X^{\mathtt{L}} \rangle$ where $X^{\mathtt{H}} \overset{\text{def}}{=} \{\, h \,|\, \langle l, h \rangle \in X \,\}$, i.e., $\mathcal{H}$ is the product of respectively the top and the bottom abstractions in the lattice of abstract interpretations. This means that the private component of $\mathcal{H}$ can only be refined as well as we can only abstract its public one. In this context we prove that shell and core have two different and precise meanings: The core abstracts the public component, viz. characterizes the most concrete attacker that cannot disclose private properties; The shell refines the private component, viz. characterizes the most abstract property that flows.

Example 3. Consider $P \overset{\text{def}}{=} l := 2 * h$, where $l : \mathtt{L}$ and $h : \mathtt{H}$. P violates non-interference, e.g., $\mathcal{H} \circ [\![P]\!] \circ \mathcal{H}(\langle 2, 3 \rangle) = \mathcal{H} \circ [\![P]\!](\langle \mathbb{Z}, 3 \rangle) = \mathcal{H}(\langle \mathbb{Z}, 2\mathbb{Z} \rangle) = \langle \mathbb{Z}, 2\mathbb{Z} \rangle$ while $\mathcal{H} \circ [\![P]\!](\langle 2, 3 \rangle) = \mathcal{H}(\langle 2, 4 \rangle) = \langle \mathbb{Z}, 4 \rangle$, where $2\mathbb{Z} \neq 4$. We can derive the complete core of $\mathcal{H}$, which makes the program secure. From [17] we have to keep only those elements whose inverse image is a fix-point of $\mathcal{H}$: $\mathcal{C}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H}) = \{\, \langle \mathbb{Z}, L \rangle \,|\, \{\, \langle h, l \rangle \,|\, \langle h, 2h \rangle \subseteq \langle \mathbb{Z}, L \rangle \,\} \subseteq \mathcal{H} \,\}$. Note that $\langle H, L \rangle \in \mathcal{H}$ iff $H = \mathbb{Z}$ and $\langle \mathbb{Z}, L' \rangle \supseteq \langle \mathbb{Z}, 2\mathbb{Z} \rangle$ iff $L' \supseteq 2\mathbb{Z}$. More generally, if $L' \subseteq 2\mathbb{Z} + 1$ then $\langle h, 2h \rangle \in \langle \mathbb{Z}, L' \rangle$ is false for each possible L', namely $\{\, \langle h, l \rangle \,|\, [\![P]\!](\langle h, l \rangle) \subseteq \langle H', L' \rangle \,\} = \varnothing \subseteq \mathcal{H}$ which means that in this case $\langle \mathbb{Z}, L' \rangle$ is kept. Therefore, $\mathcal{C}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H}) = \{\, \langle \mathbb{Z}, L \rangle \,|\, L \cap 2\mathbb{Z} \in \{2\mathbb{Z}, \varnothing\} \,\}$, which corresponds to abstracting the public output in the domain that is not able to distinguish even numbers. Let $\overline{\mathcal{H}} \overset{\text{def}}{=} \mathcal{C}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H})$, then in the previous case, we have $\overline{\mathcal{H}} \circ [\![P]\!] \circ \overline{\mathcal{H}}(\langle 2, 3 \rangle) = \overline{\mathcal{H}} \circ [\![P]\!](\langle \mathbb{Z}, 3 \rangle) = \overline{\mathcal{H}}(\langle \mathbb{Z}, 2\mathbb{Z} \rangle) = \langle \mathbb{Z}, 2\mathbb{Z} \rangle$ and $\overline{\mathcal{H}} \circ [\![P]\!](\langle 2, 3 \rangle) = \overline{\mathcal{H}}(\langle 2, 4 \rangle) = \langle \mathbb{Z}, 2\mathbb{Z} \rangle$.

Example 4. Consider $P \stackrel{\text{def}}{=} l := (2h+1) \;\mathtt{mod}\; 2$, where $l : \mathtt{L}$ and $h : \mathtt{H}$. The program violates non-interference, since, for instance, $\mathcal{H}\circ[\![P]\!]\circ\mathcal{H}(\langle 2,3\rangle) = \mathcal{H}\circ[\![P]\!](\langle \mathbb{Z},3\rangle) = \mathcal{H}(\langle \mathbb{Z},\{-1,1\}\rangle) = \langle \mathbb{Z},\{-1,1\}\rangle$ while $\mathcal{H}\circ[\![P]\!](\langle 2,3\rangle) = \mathcal{H}(\langle 2,1\rangle) = \langle \mathbb{Z},1\rangle$ and $\{-1,1\} \neq 1$. We compute the complete shell of $\mathcal{H}$, characterizing the flowing property of private information, namely we add all the inverse images of the elements in $\mathcal{H}$.

$$\mathcal{R}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H}) = \mathcal{H} \sqcap \mathcal{M}(\textstyle\bigcup_{L' \in \wp(\mathbb{V}^{\mathtt{L}})} \left\{ \langle h,l\rangle \,\middle|\, \langle h, 2h+1 \;\mathtt{mod}\; 2\rangle \in \langle \mathbb{Z}, L'\rangle \right\})$$

If $-1 \notin L'$, then $\left\{ \langle h,l\rangle \,\middle|\, \langle h, 2h+1 \;\mathtt{mod}\; 2\rangle \in \langle \mathbb{Z}, L'\rangle \right\} = \langle \mathbb{Z}^{+}_{0}, \mathbb{Z}\rangle$, $\mathbb{Z}^{+}_{0} \stackrel{\text{def}}{=} \mathbb{Z}^{+} \cup \{0\}$. If $1 \notin L'$, then $\left\{ \langle h,l\rangle \,\middle|\, \langle h, 2h+1 \;\mathtt{mod}\; 2\rangle \in \langle \mathbb{Z}, L'\rangle \right\} = \langle \mathbb{Z}^{-}, \mathbb{Z}\rangle$. Finally, if $1, -1 \notin L'$ we have $\left\{ \langle h,l\rangle \,\middle|\, \langle h, 2h+1 \;\mathtt{mod}\; 2\rangle \in \langle \mathbb{Z}, L'\rangle \right\} = \varnothing$.
Hence $\bigcup_{L' \in \wp(\mathbb{V}^{\mathtt{L}})} \left\{ \langle h,l\rangle \,\middle|\, \langle h, 2h+1 \;\mathtt{mod}\; 2\rangle \in \langle \mathbb{Z}, L'\rangle \right\} = \{\langle \mathbb{Z}^{+}_{0}, \mathbb{Z}\rangle, \langle \mathbb{Z}^{-}, \mathbb{Z}\rangle, \varnothing\}$, which implies $\mathcal{R}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H}) = \mathcal{H} \cup \left\{ \langle H, L\rangle \,\middle|\, H \in \{\mathbb{Z}^{+}_{0}, \mathbb{Z}^{-}\},\ L \in \wp(\mathbb{V}^{\mathtt{L}}) \right\}$. Let $\overline{\mathcal{H}} \stackrel{\text{def}}{=} \mathcal{R}^{\mathcal{B}}_{[\![P]\!]}(\mathcal{H})$, then $\overline{\mathcal{H}}\circ[\![P]\!]\circ\overline{\mathcal{H}}(\langle 2,3\rangle) = \overline{\mathcal{H}}\circ[\![P]\!](\langle \mathbb{Z}^{+}_{0},3\rangle) = \overline{\mathcal{H}}(\langle \mathbb{Z}^{+}_{0},\{1\}\rangle) = \langle \mathbb{Z}^{+}_{0},\{1\}\rangle$ and $\overline{\mathcal{H}}\circ[\![P]\!](\langle 2,3\rangle) = \overline{\mathcal{H}}(\langle 2,1\rangle) = \langle \mathbb{Z}^{+}_{0},\{1\}\rangle$.

The idea is to embed the model of an attacker as given in ANI, i.e., as a pair of input/output abstractions, in $\mathcal{H}$. Consider $\langle \wp(\mathbb{V}^{\mathtt{H}}) \times \wp(\mathbb{V}^{\mathtt{L}}), \varnothing, \langle \mathbb{V}^{\mathtt{H}}, \mathbb{V}^{\mathtt{L}}\rangle, \Cup, \cap, \subseteq\rangle$, where $\langle H_1, L_1\rangle \Cup \langle H_2, L_2\rangle \stackrel{\text{def}}{=} \langle H_1 \cup H_2, L_1 \cup L_2\rangle$. It is well known that there exists an obvious GI of $\wp(\mathbb{V}^{\mathtt{H}}) \times \wp(\mathbb{V}^{\mathtt{L}})$ in $\wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}})$, corresponding to the closure: $\mathtt{Split} \stackrel{\text{def}}{=} \lambda X.\ \left\{ \langle x_1, x_2\rangle \,\middle|\, \exists y.\ \langle x_1, y\rangle \in X,\ \exists z.\ \langle z, x_2\rangle \in X \right\}$. Consider the closure $\rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$. We define $\mathcal{H}_\rho \in uco(\wp(\mathbb{V}^{\mathtt{H}}) \times \wp(\mathbb{V}^{\mathtt{L}}))$:

$$\mathcal{H}_\rho \stackrel{\text{def}}{=} \lambda X.\ \langle \mathbb{V}^{\mathtt{H}}, \rho(X^{\mathtt{L}})\rangle$$

Note that $\mathcal{H} = \mathcal{H}_{\mathtt{id}}$, $\mathcal{H}_\rho \in uco(\wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}}))$, and for any pair of disjunctive closures $\eta, \rho \in uco(\mathbb{V}^{\mathtt{L}})$ and for all $\langle h,l\rangle \in \mathbb{V}$: $\mathcal{H}_\rho\circ[\![P]\!]\circ\mathcal{H}_\eta(\langle h,l\rangle) = \mathcal{H}_\rho\circ[\![P]\!](\langle h,l\rangle) \;\Leftrightarrow\; \mathcal{H}_\rho\circ[\![P]\!]\circ\mathcal{H}_\eta = \mathcal{H}_\rho\circ[\![P]\!]$.

Theorem 2. *Let $\rho, \eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$.*

1. $[\eta]P(\rho) \;\Leftarrow\; \mathcal{H}_\rho\circ[\![P]\!]\circ\mathcal{H}_\eta = \mathcal{H}_\rho\circ[\![P]\!]$*;*
2. *If ρ is disjunctive and η is partitioning:* $[\eta]P(\rho) \;\Rightarrow\; \mathcal{H}_\rho\circ[\![P]\!]\circ\mathcal{H}_\eta = \mathcal{H}_\rho\circ[\![P]\!]$.

This result proves that narrow abstract non-interference is weaker than the generalization of Joshi and Leino's semantics-based approach to non-interference, which is a problem of completeness. Moreover, when both η and ρ are equivalence relations on public data as in the PER model, i.e., partitioning closures, then the narrow abstract non-interference is equivalent to the PER model of non-interference, which is in turn an instance of a completeness problem. Theorem 2 gives a slightly weaker condition, because all partitioning closures (viz. equivalence relations) are disjunctive, but the converse does not hold in general. In order to extend Theorem 2 to model abstract non-interference we have to modify the program semantics. The idea is to consider an abstract semantics that is applied to abstract (public and private) data. Consider the closures $\eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$. We define the abstract semantics as $[\![P]\!]^{\eta,\phi} \stackrel{\text{def}}{=} \lambda\langle h,l\rangle.\ [\![P]\!](\phi(h), \eta(l))$. Note that, for any pair of disjunctive closures $\eta, \rho \in uco(\mathbb{V}^{\mathtt{L}})$ and for all $\langle h,l\rangle \in \mathbb{V}$: $\mathcal{H}_\rho\circ[\![P]\!]^{\eta,\phi}\circ\mathcal{H}_\eta(\langle h,l\rangle) = \mathcal{H}_\rho\circ[\![P]\!]^{\eta,\phi}(\langle h,l\rangle) \;\Leftrightarrow\; \mathcal{H}_\rho\circ[\![P]\!]^{\eta,\phi}\circ\mathcal{H}_\eta = \mathcal{H}_\rho\circ[\![P]\!]^{\eta,\phi}$.

Theorem 3. *Consider $\eta, \rho \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ and $\phi \in uco(\wp(\mathbb{V}^{\mathtt{H}}))$:*

1. $(\rho)P(\phi \rightsquigarrow\!\!| \eta) \;\Leftarrow\; \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi} \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi}$*;*
2. *If ρ and η are disjunctive then:* $(\eta)P(\phi \rightsquigarrow\!\!| \rho) \;\Rightarrow\; \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi} \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\phi}$.

Once again abstract non-interference is weaker than the generalization of Joshi and Leino's approach to non-interference. Additivity is here sufficient in order to let these two approaches equivalent. Because the only difference in the corresponding completeness problems is due to the program semantics: $[\![P]\!]^{\eta,\phi}$ in the case of abstract non-interference and $[\![P]\!]^{\mathtt{id},\mathtt{id}}$ in the narrow case, in the following, without loss of generality we consider the abstract non-interference case being more general. In the following we omit the apex $\mathcal{B}$ from shells and cores, since we will consider always backward completeness.

5 The Most Concrete *Observer* as Completeness Core

In [13] we gave a method for systematically deriving the most concrete harmless attacker (canonical attacker) associated with a given program. By Theorem 3, the most concrete public observer, which is the canonical attacker, can be derived as the most concrete abstraction satisfying the following completeness problem:

$$\mathcal{H} \circ [\![P]\!]^{\eta,\phi} \circ \mathcal{H}_\eta = \mathcal{H} \circ [\![P]\!]^{\eta,\phi} \tag{2}$$

Then we have the following result which allows us to specify the canonical attacker as the fix-point of an abstract domain simplification.

Theorem 4. *Let $\eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ be disjunctive and $\phi \in uco(\mathbb{V}^{\mathtt{H}})$. Then we have* $\mathcal{C}^{\mathcal{H}_\eta}_{[\![P]\!]^{\eta,\phi}}(\mathcal{H}) = \left\{ \langle \mathbb{V}^{\mathtt{H}}, L\rangle \,\middle|\, \left\{ \langle h, l\rangle \,\middle|\, [\![P]\!](\langle \phi(h), \eta(l)\rangle) \subseteq \langle \mathbb{V}^{\mathtt{H}}, L\rangle \right\} \in \mathcal{H}_\eta \right\}$ *and*

$$\left\{ L \in \wp(\mathbb{V}^{\mathtt{L}}) \,\middle|\, \langle \mathbb{V}^{\mathtt{H}}, L\rangle \in \mathcal{C}^{\mathcal{H}_\eta}_{[\![P]\!]^{\eta,\phi}}(\mathcal{H}) \right\} = (\eta)[\![P]\!](\phi \rightsquigarrow\!\!| \mathtt{id}).$$

Example 5. Consider the following program fragment, with $l : \mathtt{L}$ and $h : \mathtt{H}$.

$$P \overset{\text{def}}{=} \textbf{while } h \textbf{ do } l := 2l;\ h := 0 \textbf{ endw} \qquad [\![P]\!](\langle h, l\rangle) = \begin{cases} \langle h, l\rangle & \text{if } h = 0 \\ \langle h, 2l\rangle & \text{otherwise} \end{cases}$$

We look for the core in order to make $\langle \mathcal{H}, \mathcal{H}\rangle$ complete for the map $[\![P]\!]^{\mathtt{id},\mathtt{id}} = [\![P]\!]$.

$$\mathcal{C}^{\mathcal{H}}_{[\![P]\!]}(\mathcal{H}) = \left\{ \langle \mathbb{Z}, L\rangle \,\middle|\, \forall l \in \mathbb{V}^{\mathtt{L}} .\, l \in L \;\Leftrightarrow\; 2l \in L \right\}$$

It is straightforward to show that $\mathcal{C}^{\mathcal{H}}_{[\![P]\!]}(\mathcal{H})$ is the domain that abstracts the public data in the domain $\curlyvee(\{ n\{2\}^{\mathbb{N}} \,|\, n \in 2\mathbb{Z}+1 \})$, where $\{2\}^{\mathbb{N}} \overset{\text{def}}{=} \{ 2^k \,|\, k \in \mathbb{N} \}$. Let $\overline{\mathcal{H}} \overset{\text{def}}{=} \mathcal{C}^{\mathcal{H}}_{[\![P]\!]}(\mathcal{H})$, then, $\overline{\mathcal{H}} \circ [\![P]\!] \circ \mathcal{H}(\langle 3, 5\rangle) = \overline{\mathcal{H}} \circ [\![P]\!](\mathbb{Z}, 5) = \overline{\mathcal{H}}(\langle \mathbb{Z}, \{5, 10\}\rangle) = \langle \mathbb{Z}, 5\{2\}^{\mathbb{N}}\rangle$, and $\overline{\mathcal{H}} \circ [\![P]\!](\langle 3, 5\rangle) = \overline{\mathcal{H}}(\langle \mathbb{Z}, \{10\}\rangle) = \langle \mathbb{Z}, 5\{2\}^{\mathbb{N}}\rangle$ while we have that $\mathcal{H} \circ [\![P]\!] \circ \mathcal{H}(\langle 3, 5\rangle) = \mathcal{H} \circ [\![P]\!](\mathbb{Z}, 5) = \mathcal{H}(\langle \mathbb{Z}, \{5, 10\}\rangle) = \langle \mathbb{Z}, \{5, 10\}\rangle$ and, on the other hand, $\mathcal{H} \circ [\![P]\!](\langle 3, 5\rangle) = \mathcal{H}(\langle \mathbb{Z}, \{10\}\rangle) = \langle \mathbb{Z}, \{10\}\rangle$.

6 The Most Abstract *Observable* as Completeness Shell

We are now interested in applying the same construction for characterizing the most abstract private observable, used for defining abstract declassification as a solution of a completeness problem in abstract interpretation. Namely, we are interested in the most abstract property that can be declassified in order to guarantee abstract non-interference. By Theorem 3, this information can be obtained by solving the following completeness problem:

$$\mathcal{H}_\rho \circ [\![P]\!]^{\eta,\mathtt{id}} \circ \mathcal{H}_\eta = \mathcal{H}_\rho \circ [\![P]\!]^{\eta,\mathtt{id}} \tag{3}$$

Lemma 1. *Let $\rho, \eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$. Then we have*

$$\mathcal{R}^{\mathcal{H}_\rho}_{[\![P]\!]^{\eta,\mathtt{id}}}(\mathcal{H}_\eta) = \mathcal{H}_\eta \sqcap \mathcal{M}(\{\ \{\ \langle h,l\rangle \,|\, \rho([\![P]\!](\langle h,\eta(l)\rangle)^{\mathtt{L}}) \subseteq L\ \} \,|\, L \in \rho\ \}\,.$$

Moreover, let $\mathcal{R} \overset{def}{=} \mathcal{R}^{\mathcal{H}_\rho}_{[\![P]\!]^{\eta,\mathtt{id}}}(\mathcal{H}_\eta)$, then forall $l, l' \in \mathbb{V}^{\mathtt{L}}, h, h' \in \mathbb{V}^{\mathtt{H}}$ we have

$$\mathcal{R}(\langle h,l\rangle) = \mathcal{R}(\langle h',l'\rangle) \qquad \textit{iff} \qquad \rho[\![P]\!](\langle h,\eta(l)\rangle)^{\mathtt{L}} = \rho[\![P]\!](\langle h',\eta(l')\rangle)^{\mathtt{L}}$$

It is worth noting that, by Lemma 1, the partition induced by the complete shell of $\mathcal{H}_\eta$ on $\wp(\mathbb{V}^{\mathtt{H}} \times \mathbb{V}^{\mathtt{L}})$ for Equation 3 does not affect the closure η. This means that the only component which is actually refined is the abstraction on private data, and this corresponds to the most abstract partitioning of private data which can be declassified. This means that any change between equivalent elements does not produce insecure flows, as stated in the following theorem.

Theorem 5. *Let $\rho, \eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ then for each $l, l' \in \mathbb{V}^{\mathtt{L}}, h, h' \in \mathbb{V}^{\mathtt{H}}$ we have $\eta(l) = \eta(l') = Y \ \Rightarrow\ (\mathcal{R}(\langle h,l\rangle) = \mathcal{R}(\langle h',l'\rangle)$ iff $h' \in [h]_{\Pi_{\mathtt{P}}(\eta,\rho)_{|Y}})$.*

Next examples show how declassification can be obtained as solutions of completeness problems.

Example 6. Consider the program fragment: $P \overset{\text{def}}{=}\ l := l * h^2$, with $l : \mathtt{L}$ and $h : \mathtt{H}$. We want to find the shell in order to make $\langle \mathcal{H}, \mathcal{H}_{Par}\rangle$ complete for the map $[\![P]\!]^{\mathtt{id},\mathtt{id}} = [\![P]\!]$.

$$\mathcal{R}^{\mathcal{H}_{Par}}_{[\![P]\!]}(\mathcal{H}) = \mathcal{H} \sqcap \left(\left\{\begin{array}{l}\langle \mathbb{Z},\mathbb{Z}\rangle, \langle \mathbb{Z}, 2\mathbb{Z}\rangle \cup \langle 2\mathbb{Z}, 2\mathbb{Z}+1\rangle, \langle 2\mathbb{Z}+1, 2\mathbb{Z}+1\rangle,\\ \langle 2\mathbb{Z}+1, 2\mathbb{Z}\rangle, \varnothing\end{array}\right\}\right)$$

This means that the reduced product generates also $\langle 2\mathbb{Z}, 2\mathbb{Z}+1\rangle$ and therefore $\langle 2\mathbb{Z}, l\rangle$ for each $l \in 2\mathbb{Z}+1$. Let $\overline{\mathcal{H}} \overset{\text{def}}{=} \mathcal{R}^{\mathcal{H}_{Par}}_{[\![P]\!]}(\mathcal{H})$, then for instance, we have $\mathcal{H}_{Par} \circ [\![P]\!] \circ \overline{\mathcal{H}}(\langle 2,3\rangle) = \mathcal{H}_{Par} \circ [\![P]\!](\langle 2\mathbb{Z}, 3\rangle) = \langle \mathbb{Z}, 2\mathbb{Z}\rangle$, and $\mathcal{H}_{Par} \circ [\![P]\!](\langle 2,3\rangle) = \langle \mathbb{Z}, 2\mathbb{Z}\rangle$, while $\mathcal{H}_{Par} \circ [\![P]\!] \circ \mathcal{H}(\langle 2,3\rangle) = \mathcal{H}_{Par} \circ [\![P]\!](\mathbb{Z}, 3) = \langle \mathbb{Z}, \mathbb{Z}\rangle$. As in abstract desclassification [13], this means that it is the variation of parity of the private input that generates the flow.

Example 7. Consider $\rho \overset{\text{def}}{=} \{\mathbb{Z}, 2\mathbb{Z}, 4\mathbb{Z}, 2\mathbb{Z}{+}1, \varnothing\}$ and $\eta \overset{\text{def}}{=} \{\mathbb{Z}, 2\mathbb{Z}, 5\mathbb{Z}, 10\mathbb{Z}, \varnothing\}$, and consider $P \overset{\text{def}}{=}$ **if** $(h \mathtt{~mod~} 4) = 0$ **then** $l := l * h$ **else** $l := l * (h+1)$ **fi**. Compute, first, the abstract robust declassification, as was introduced in [13]:

$$\Pi_{\mathtt{P}}(\eta, \rho) = \left\{ \begin{array}{l} \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 10\mathbb{Z}\rangle, \langle 4\mathbb{Z}+1 \cup 4\mathbb{Z}+2, 10\mathbb{Z}\rangle, \\ \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 5\mathbb{Z}\rangle, \langle 4\mathbb{Z}+1, 5\mathbb{Z}\rangle, \langle 4\mathbb{Z}+2, 5\mathbb{Z}\rangle, \\ \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 2\mathbb{Z}\rangle, \langle 4\mathbb{Z}+1 \cup 4\mathbb{Z}+2, 2\mathbb{Z}\rangle, \\ \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, \mathbb{Z}\rangle, \langle 4\mathbb{Z}+1, \mathbb{Z}\rangle, \langle 4\mathbb{Z}+2, \mathbb{Z}\rangle \end{array} \right\}$$

Therefore we obtain $\Pi_{\mathtt{P}}(\eta, \rho)_{|10\mathbb{Z}} = \Pi_{\mathtt{P}}(\eta, \rho)_{|2\mathbb{Z}} = \{4\mathbb{Z} \cup 4\mathbb{Z}+3, 4\mathbb{Z}+1 \cup 4\mathbb{Z}+2\}$ and $\Pi_{\mathtt{P}}(\eta, \rho)_{|5\mathbb{Z}} = \Pi_{\mathtt{P}}(\eta, \rho)_{|\mathbb{Z}} = \{4\mathbb{Z} \cup 4\mathbb{Z}+3, 4\mathbb{Z}+1, 4\mathbb{Z}+2\}$. Consider now the completeness shell:

$$\begin{array}{r} \left\{ \langle h, l\rangle \,\middle|\, [\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}} \subseteq 4\mathbb{Z} \right\} = \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, \mathbb{Z}\rangle \\ \left\{ \langle h, l\rangle \,\middle|\, [\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}} \subseteq 2\mathbb{Z} \right\} = \langle \mathbb{Z} \smallsetminus 4\mathbb{Z}+2, \mathbb{Z}\rangle \cup \langle 4\mathbb{Z}+2, 2\mathbb{Z}\rangle \\ \left\{ \langle h, l\rangle \,\middle|\, [\![P]\!](\langle h, \eta(l)\rangle)^{\mathtt{L}} \subseteq 2\mathbb{Z}+1 \right\} = \varnothing \end{array}$$

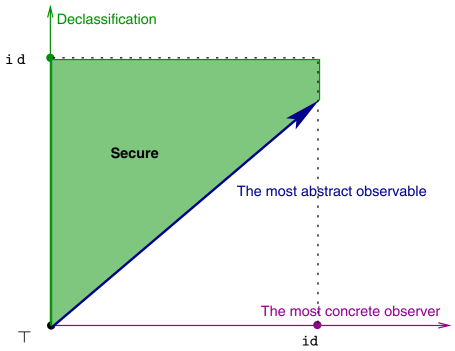
Then we have:

$$\mathcal{R}^{\mathcal{H}_\rho}_{[\![P]\!]}(\mathcal{H}_\eta) = \mathcal{H}_\eta \cup \left\{ \begin{array}{l} \langle \mathbb{Z} \smallsetminus 4\mathbb{Z}+2, \mathbb{Z}\rangle \cup \langle 4\mathbb{Z}+2, 2\mathbb{Z}\rangle, \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, \mathbb{Z}\rangle, \\ \langle \mathbb{Z} \smallsetminus 4\mathbb{Z}+2, 5\mathbb{Z}\rangle \cup \langle 4\mathbb{Z}+2, 10\mathbb{Z}\rangle, \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 5\mathbb{Z}\rangle, \\ \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 2\mathbb{Z}\rangle, \langle 4\mathbb{Z} \cup 4\mathbb{Z}+3, 10\mathbb{Z}\rangle \end{array} \right\}$$

For instance, consider $5, 9 \in 4\mathbb{Z}+1$, $6, 10 \in 4\mathbb{Z}+2$, and note that $\eta(10) = \eta(30) = 10\mathbb{Z}$, and $\eta(5) = \eta(15) = 5\mathbb{Z}$. Note that, 5 and 6 are in the same equivalence class in the partition induced by $\Pi_{\mathtt{P}}(\eta, \rho)_{|10\mathbb{Z}}$, written $5 \in [6]_{10\mathbb{Z}}$, and indeed $\mathcal{R}(\langle 5, 10\rangle) = \mathcal{R}(\langle 6, 30\rangle) = \langle \mathbb{Z}, 10\mathbb{Z}\rangle \in \mathcal{H}_\eta$. While $5 \in [9]_{5\mathbb{Z}} \neq [6]_{5\mathbb{Z}}$, namely the partition induced by $\Pi_{\mathtt{P}}(\eta, \rho)_{|5\mathbb{Z}}$ distinguishes 5 and 6, while 5 is together with 9 and 6 is together with 10, i.e., $10 \in [6]_{5\mathbb{Z}}$. On the other hand, we have $\mathcal{R}(\langle 5, 5\rangle) = \mathcal{R}(\langle 9, 15\rangle) = \langle \mathbb{Z} \smallsetminus 4\mathbb{Z}+2, 5\mathbb{Z}\rangle \cup \langle 4\mathbb{Z}+2, 10\mathbb{Z}\rangle$ and $\mathcal{R}(\langle 6, 5\rangle) = \mathcal{R}(\langle 10, 15\rangle) = \langle \mathbb{Z}, 5\mathbb{Z}\rangle \in \mathcal{H}_\eta$.

7 Adjoining Observer and Observable Properties

Modeling attackers means characterizing the maximal power of an harmless attacker, i.e., an attacker which cannot disclose confidential information. Declassification, instead, means characterizing the information revealed to a fixed attacker. As we have seen in the previous sections, the model of the most concrete harmless attacker corresponds to the most concrete public observer, while abstract declassification is characterized by the most abstract private observable. Clearly there is a strong relation between these two notions, since the more powerful is the attacker and the less is the confidential information that can be kept



private. In other words the index of the partition of private data for declassification is proportional to the cardinality of the abstract domain which models the precision of the property that the attacker can observe. This phenomenon can be precisely characterized in the lattice of abstract interpretations as an adjunction. In the picture on the right we provide a graphical representation of the relation existing between the most concrete property modeling the public observer and the most abstract property modeling the private observable. In particular, this picture represents the fact that the more powerful is the attacker, i.e., the more concrete is the observer property, the less confidential information can be kept private, i.e., the more concrete is the private observable. The picture also shows that, if the arrow represents the most abstract private observable, then when we declassify a confidential property which lays in the white area we cannot guarantee the secrecy of the program, since we are declassifying less than what is released by the semantics. When we declassify a property in the filled area instead, then we guarantee that no confidential information leakage may happen. Moreover, note that, even if the attacker is able to observe the value of public variables, then the observable property can be more abstract then the identity since the program itself can behave as a firewall for certain confidential properties, such as the square operation hides the sign. In Section 5 and 6 we proved that both problems can be viewed as instances of the problem of making abstractions complete. While the private observable for declassification is obtained by computing the completeness shell, the public observer, modeling the attacker, is obtained by computing the completeness core in the same completeness problem. These abstract domain transformers have been proved in [17] to be adjoint functions (see Sect. 2) on the lattice of abstract interpretations. The following result is therefore a consequence of Theorem 4 and 5.

Theorem 6. *Let $\eta \in uco(\wp(\mathbb{V}^{\mathtt{L}}))$ be a disjunctive property, and P a program. Then we have that* $\mathtt{id} \sqsubset (\eta)[\![P]\!](\mathtt{id} \rightsquigarrow\!\!\| \mathtt{id}) \;\Leftrightarrow\; \mathcal{P}(\sqcap_{L\in\eta}\mathcal{M}(\mathit{\Pi}_{\mathtt{P}}(\eta, \mathtt{id})_{|L})) \sqsubset \mathbb{T}$.

This result provides a precise mathematical framework where declassification and attack models can be systematically derived and compared with each other in the lattice of abstract interpretations by applying well known methods for abstract domain design. This framework can be the basis for applying quantitative methods and metrics [5] for measuring the amount of information leaked relatively to a given attack model, or by adjunction, the precision of an attacker under the hypothesis that some information can be declassified. Recently, several papers treated the problem of defining non-interference for programs where confidential information can be explicitly declassified [26, 21]. In these cases the authors define weaker notions of non-interference in order to model confinement problem for programs where there are intentional releases of information. Abstract non-interference, instead does not consider explicit declassification, but allows to characterize which confidential information should be declassified, at least, in order to guarantee only secure information flows.

References

1. D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, MITRE Corp. Badford, MA, 1973.
2. T.S. Blyth and M.F. Janowitz. *Residuation theory.* Pergamon Press, 1972.
3. C. Bodei, P. Degano, F. Nielson, and H.R. Nielson. Static analysis for secrecy and non-interference in networks of processes. In *Proc. of PaCT'01*, volume 2127 of *Lecture Notes in Computer Science*, pages 27–41. Springer-Verlag, 2001.
4. D. Clark, C. Hankin, and S. Hunt. Information flow for algol-like languages. *Computer Languages*, 28(1):3–28, 2002.
5. D. Clark, S. Hunt, and P. Malacaria. Quantitative analysis of the leakage of confidential data. In *Workshop on Quantitative Aspects of Programming Laguages (QAPL '01)*, volume 59 of *Electronic Notes in Theoretical Computer Science*. Elsevier, Amsterdam, 2001.
6. E. S. Cohen. Information transmission in computational systems. *ACM SIGOPS Operating System Review*, 11(5):133–139, 1977.
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. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–242, 1976.
10. D. E. Denning and P. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.
11. G. Filé, R. Giacobazzi, and F. Ranzato. A unifying view of abstract domain design. *ACM Comput. Surv.*, 28(2):333–336, 1996.
12. R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer security*, 3(1):5–33, 1995.
13. 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.
14. R. Giacobazzi and I. Mastroeni. Proving abstract non-interference. In *Annual Conference of the European Association for Computer Science Logic (CSL'04)*, volume 3210, pages 280–294. Springer-Verlag, 2004.
15. R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples and refinements in abstract model-checking. In P. Cousot, editor, *Proc. of The 8th Internat. Static Analysis Symp. (SAS'01)*, volume 2126 of *Lecture Notes in Computer Science*, pages 356–373. Springer-Verlag, 2001.

16. R. Giacobazzi and F. Ranzato. Refining and compressing abstract domains. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. of the 24th Internat. Colloq. on Automata, Languages and Programming* (*ICALP '97*), volume 1256 of *Lecture Notes in Computer Science*, pages 771–781. Springer-Verlag, Berlin, 1997.
17. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. of the ACM.*, 47(2):361–416, 2000.
18. 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.
19. R. Joshi and K. R. M. Leino. A semantic approach to secure information flow. *Science of Computer Programming*, 37:113–138, 2000.
20. P. Laud. Semantics and program analysis of computationally secure information flow. 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.
21. P. Li and S. Zdancewic. Downgrading policies and relaxed noninterference. In *Proc. of the 32st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '05).* ACM-Press, NY, 2005. To appear.
22. H. Mantel. Possibilistic definitions of security – an assemply kit –. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 185–199. IEEE Computer Society Press, 2000.
23. J. McLean. Proving noninterference and functional correcteness using traces. *Journal of Computer security*, 1(1):37–58, 1992.
24. F. Ranzato and F. Tapparo. Strong preservation as completeness in abstract interpretation. In D. Schmidt, editor, *Proc. of the 13th European Symposium on Programming (ESOP'04)*, volume 2986 of *Lecture Notes in Computer Science*, pages 18–32. Springer-Verlag, 2004.
25. P. Ryan. Mathematical models of computer security – tutorial lectures. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design*, volume 2171 of *Lecture Notes in Computer Science*, pages 1–62. Springer-Verlag, 2001.
26. A. Sabelfeld and A. C. Myers. A model for delimited information release. In *Proc. of the International Symp. on Software Security (ISSS'03)*, Lecture Notes in Computer Science. Springer-Verlag, 2004.
27. A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE J. on selected ares in communications*, 21(1):5–19, 2003.
28. 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.
29. C. Skalka and S. Smith. Static enforcement of security with types. In *ICFP'00*, pages 254–267. ACM Press, New York, 2000.
30. 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.
31. 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.
32. 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.