

What you lose is what you leak: Information leakage in declassification policies

Anindya Banerjee^{a,1} Roberto Giacobazzi^{b,2} Isabella Mastroeni^{b,2}

^a *Kansas State University, Manhattan, KS 66506, USA*

^b *Università di Verona, Verona, Italy*

Abstract

This paper suggests the following approach for checking whether a program satisfies an information flow policy that may declassify secret information: (a) Compute a finite abstract domain that over-approximates the information released by the policy and (b) Check whether program execution may release more information than what is permitted by the policy by completing the finite abstract domain wrt. weakest liberal preconditions. Moreover, techniques based on the Paige-Tarjan algorithm for partition refinement can be used to generate counterexamples to a declassification policy: the counterexamples demonstrate that more information is released by the program than what the policy permits. Subsequently the policy can be refined so that the least amount of confidential information necessary for making the program secure is declassified.

Keywords: Abstract interpretation, completeness, declassification, information flow

1 Introduction

The secure information flow problem is concerned with protecting data confidentiality by checking that secrets are not leaked during program execution. In the simplest setting, program variables are first partitioned into high security (or private or classified) and low security (or public or unclassified) variables, where high (H) and low (L) are levels in a two point security lattice, $L \leq H$; next, one checks that L output variables do not leak information about the initial values of H input variables. To perform the check, a variety of information flow analyses for confidentiality policies have been developed using technologies like data flow analysis, security type systems, program logics, etc. (See the survey by Sabelfeld and Myers [23] and references therein). The correctness of such analyses is governed by *noninterference* (NI) [13]: for any two runs of a program, L indistinguishable input states yield L indistinguishable output states, where two program states are said to be L indistinguishable iff they agree on the values of the L variables.

Joshi and Leino [15] give a semantic definition of secure information flow (that has been shown equivalent to NI [24]): a program P containing H and L variables (ranged over

¹ Email: ab@cis.ksu.edu

² Email: roberto.giacobazzi@univr.it, isabella.mastroeni@univr.it

by h and l respectively) is secure iff $HH;P;HH = P;HH$, where HH is an assignment of an arbitrary value to h . “The postfix occurrences of HH on each side mean that we are only interested in the final value of l and the prefix HH on the left-hand-side means that the two programs are equal if the final value of l does not depend on the initial value of h ” [24]. In practice, noninterference is too strong a property to be enforced and downgrading of information, or *declassification*, is a necessity. For example, a password checker makes public the (H) result of the comparison between the actual password and the password entered at the login prompt.

This paper is based on the central observation that Joshi and Leino’s semantic definition permits a view of noninterference as *completeness of an abstract interpretation* [10], and the paper explores the consequences of this observation. An abstract interpretation is (backwards) complete for a function, f , if the result obtained when f is applied to any concrete input, x , and the result obtained when f is applied to an abstraction of the concrete input, x , both abstract to the same value. Thus, the essence of completeness is this: an observer who can see only the final abstraction cannot distinguish whether the concrete input value was x or any other concrete value x' with the same abstract value as that of x . The completeness connection is implicit in Joshi and Leino’s definition of secure information flow and the implicit abstraction in their definition is: “each H value is associated with \top , i.e., the set of all possible H values”. (This is discussed in Sects. 2 and 3).

In this paper, we consider more flexible abstractions than the one considered by Joshi and Leino and show that such abstractions naturally describe declassification policies that are concerned with *what* information is declassified [25]. Our *primary contribution* (Sects. 4,5.1) is to show that “declassified NI” (DNI), i.e, NI with a declassification policy, is also a completeness problem: the program points where completeness *fails* are the ones where some private information *is leaked*, thus breaking the policy. Hence, we can mechanically check if a program satisfies a declassification policy by checking whether its semantics is complete wrt. the policy.

Moreover, we show that when a program does not satisfy a declassification policy (i.e, when completeness fails), (a) counterexamples that expose the failure can be generated (Sect. 5.2); (b) there is an algorithm that generates the best refinement of the given policy such that the program respects the refined policy (Sect. 5.3). Finally, (c) we connect *abstract model checking* with secure information flow by showing that the absence of spurious counterexamples in the former can be understood as the absence of information leaks in the latter (Sect. 6).

2 Overview

Notational summary. $\mathbb{V}^H, \mathbb{V}^L$ are the sets of possible H and L values. The set of program states is $\Sigma = \mathbb{V}^H \times \mathbb{V}^L$. Σ is implicitly indexed by the H variables followed by the L variables. For any $X \subseteq \Sigma$, X^H (resp. X^L) is the projection of the H (resp. L) variables. L indistinguishability of states $s_1, s_2 \in \Sigma$, written $s_1 =_L s_2$, denotes that s_1, s_2 agree when indexed by L variables.

Semantic noninterference à la Joshi-Leino. We start with Joshi and Leino’s semantic definition of security [15], $HH;P;HH = P;HH$, where HH assigns to h an arbitrary value. Because of the arbitrary assignment, the semantics of HH can be modelled as an *abstraction function*, \mathcal{H} , on sets of concrete program states, Σ ; that is, $\mathcal{H} : \wp(\Sigma) \rightarrow \wp(\Sigma)$, where

$\wp(\Sigma)$ is ordered by subset inclusion, \sqsubseteq . For each possible value of an L variable, \mathcal{H} associates *all* possible values of the H variables in P . Thus $\mathcal{H}(X) = \mathbb{V}^H \times X^L$, where $\mathbb{V}^H = \top$, the top element of $\wp(\mathbb{V}^H)$. Now the Joshi-Leino definition can be rewritten [10] in the following way, where $\llbracket P \rrbracket$ is the concrete, denotational semantics of P .

$$\mathcal{H} \circ \llbracket P \rrbracket \circ \mathcal{H} = \mathcal{H} \circ \llbracket P \rrbracket \quad (1)$$

For example, let $h_1, h_2 \in \{0, 1\}$ and let $l \in \{0, 1\}$. Then $\mathbb{V}^H = \{0, 1\} \times \{0, 1\}$, $\mathbb{V}^L = \{0, 1\}$. Consider any $X \subseteq \Sigma$; for example, let $X = \{\langle 0, 0, 1 \rangle\}$, i.e., X denotes the state where $h_1 = 0$, $h_2 = 0$, $l = 1$. Then $\mathcal{H}(X) = \mathbb{V}^H \times \{1\}$. Let P be $l := h_1$, so that, $\llbracket P \rrbracket(X) = \{\langle 0, 0, 0 \rangle\}$ and $\mathcal{H}(\llbracket P \rrbracket(X)) = \mathbb{V}^H \times \{0\}$. On the other hand, $\llbracket P \rrbracket(\mathcal{H}(X)) = \{\langle 0, 0, 0 \rangle, \langle 0, 1, 0 \rangle, \langle 1, 0, 1 \rangle, \langle 1, 1, 1 \rangle\}$ so that we have $\mathcal{H}(\llbracket P \rrbracket(\mathcal{H}(X))) = \mathbb{V}^H \times \{0, 1\}$; hence $\mathcal{H}(\llbracket P \rrbracket(\mathcal{H}(X))) \supseteq \mathcal{H}(\llbracket P \rrbracket(X))$. Because $\mathcal{H}(\llbracket P \rrbracket(\mathcal{H}(X)))$ contains triples $\langle 1, 0, 1 \rangle$ and $\langle 1, 1, 1 \rangle$ not present in $\mathcal{H}(\llbracket P \rrbracket(X))$, the dependence of l on h_1 has been exposed. Thus P is insecure: for any two distinct values, 0 and 1 of h_1 in $\mathcal{H}(\llbracket P \rrbracket(\mathcal{H}(X)))$, two *distinct* values, 0 and 1, of l may be associated.

Declassification. For $l := h_1$, had the security policy allowed declassification of h_1 , the program would be secure. Equation (1) must naturally be modified by “filtering” \mathcal{H} through a declassifier, $\phi : \wp(\mathbb{V}^H) \rightarrow \wp(\mathbb{V}^H)$, that provides an abstraction of the secret inputs. The “filtered” \mathcal{H} , written \mathcal{H}^ϕ , models the declassification policy. Thus we enforce the equality

$$\mathcal{H} \circ \llbracket P \rrbracket \circ \mathcal{H}^\phi = \mathcal{H} \circ \llbracket P \rrbracket \quad (2)$$

That is, $\llbracket P \rrbracket$ applied to a concrete input, x , and $\llbracket P \rrbracket$ applied to the abstraction of x where the H component of x has been declassified by ϕ , both abstract to the same value.

As before, let P be $l := h_1$ and $X = \{\langle 0, 0, 1 \rangle\}$. We are interested in ϕ 's behavior on $\{\langle 0, 0 \rangle\}$, because $\{\langle 0, 0 \rangle\}$ specifies the values of h_1, h_2 in X . We have, $\phi(\{\langle 0, 0 \rangle\}) = \{\langle 0, 0 \rangle, \langle 0, 1 \rangle\}$: ϕ is the *identity* on what must be declassified – we are releasing the exact value of h_1 – but ϕ is \top on what must be protected, which explains why both $\langle 0, 0 \rangle$ and $\langle 0, 1 \rangle$ appear. Now $\mathcal{H}^\phi(X) = \phi(\{\langle 0, 0 \rangle\}) \times X^L = \{\langle 0, 0, 1 \rangle, \langle 0, 1, 1 \rangle\}$ so that $\llbracket P \rrbracket(\mathcal{H}^\phi(X)) = \{\langle 0, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}$ and $\mathcal{H}(\llbracket P \rrbracket(\mathcal{H}^\phi(X))) = \mathbb{V}^H \times \{0\}$. This is equal to $\mathcal{H}(\llbracket P \rrbracket(X))$. We can show equation (2) for any $X \subseteq \Sigma$; hence $l := h_1$ is secure. Note how ϕ partitions $\wp(\mathbb{V}^H)$ into blocks $\{\langle 0, 0 \rangle, \langle 0, 1 \rangle\}$ (the range of $\langle 0, 0 \rangle$ and $\langle 0, 1 \rangle$) and $\{\langle 1, 0 \rangle, \langle 1, 1 \rangle\}$ (the range of $\langle 1, 0 \rangle$ and $\langle 1, 1 \rangle$). Intuitively, ϕ permits exposing distinctions *between blocks* at the public output, e.g., between $\langle 0, 0 \rangle$ and $\langle 1, 0 \rangle$; in standard NI, ϕ 's range is \top and *no* distinctions should be exposed (as in the earlier example).

3 Review: Completeness of abstract interpretation

Abstract interpretation is typically formulated using Galois connections (GC) [5], but an equivalent framework [6] which we use in this paper, uses *upper closure operators*³. For example, in Sect. 2, $\mathcal{H} : \wp(\Sigma) \rightarrow \wp(\Sigma)$ defined as $\mathcal{H}(X) = \mathbb{V}^H \times X^L$, is an upper closure operator on $\wp(\Sigma)$, because \mathcal{H} is monotone, idempotent and extensive. We often call a closure operator an *abstract domain*. In particular, \mathcal{H} is called the *output* (i.e., *observed abstract domain*), that ignores private information. Likewise, \mathcal{H}^ϕ in Sect. 2 is also an uco.

³ An *upper closure operator* (uco) $\rho : C \rightarrow C$ on a poset C is monotone, idempotent, and extensive, i.e., $\forall x \in C. x \leq_C \rho(x)$. The set of all upper closure operators on C is denoted by $\text{uco}(C)$.

Completeness of abstract interpretation based static analysis has its origins in Cousot’s work, e.g., [5,6], and means that the analysis is as expressive as possible. The following example is taken from Schmidt’s excellent survey [26] on completeness. To validate the Hoare triple, $\{?\} y := -y; x := y + 1 \{isPositive(x)\}$, a *sound* analysis may compute the precondition $isNegative(y)$. But if able to express properties like $isNonNegative$ and $isNonPositive$, a *complete* analysis will calculate the *weakest* precondition property $isNonPositive(y)$.

An abstract domain is complete for a concrete function, f , if the “abstract state transition function precisely mimics the concrete state-transition function modulo the GC between concrete and abstract domains” [26]. There exist two notions of completeness – *backward* (\mathcal{B}) and *forward* (\mathcal{F}) – according as whether the concrete and the abstract computations are compared in the abstract domain or in the concrete domain [11]. Formally, let C be a complete lattice and f be the concrete state transition function, $f : C \rightarrow C$. Abstract domain ρ is a sound abstraction for f provided $\rho \circ f \circ \rho \sqsupseteq \rho \circ f$. For example, in Sect. 2, $\mathcal{H}(\llbracket P \rrbracket(\mathcal{H}(X))) \supseteq \mathcal{H}(\llbracket P \rrbracket(X))$, so \mathcal{H} is a sound abstraction for $\llbracket P \rrbracket$. Completeness is obtained by demanding equality: ρ is a \mathcal{B} (resp. \mathcal{F})-complete abstraction for f iff $\rho \circ f = \rho \circ f \circ \rho$ (resp. $f \circ \rho = \rho \circ f \circ \rho$). Completeness can be generalized to pairs (ρ, η) of abstract domains: \mathcal{B} -completeness holds for (ρ, η) when $\rho \circ f \circ \eta = \rho \circ f$; \mathcal{F} -completeness holds for (ρ, η) when $\rho \circ f \circ \eta = f \circ \eta$ (see [12] for details). For example, in Sect. 2, the declassification example asserts that equation (2) holds, i.e., $(\mathcal{H}, \mathcal{H}^\emptyset)$ is \mathcal{B} -complete for $\llbracket P \rrbracket$. Algorithms for completing abstract domains exist – see [12,11] and Schmidt’s survey [26] for details. Basically, \mathcal{F} -completeness is obtained by adding all the direct images of f to the output abstract domain; \mathcal{B} -completeness is obtained by adding all the maximal of the inverse images of the function to the input domain (see Appendix A for details).

4 \mathcal{F} -completeness and satisfaction of confidentiality policies

Equations (1) and (2) give us a way to dynamically check whether a program satisfies a confidentiality policy: indeed, both equations use the denotational semantics of a program in the process. But can we do this check statically?

We will see presently that static checking involves \mathcal{F} -completeness, instead of \mathcal{B} -completeness, and the use of weakest liberal preconditions instead of the denotational semantics. With weakest liberal preconditions, (written Wlp_p), equation (1) has the following equivalent reformulation:

$$\mathcal{H} \circ Wlp_p \circ \mathcal{H} = Wlp_p \circ \mathcal{H} \quad (3)$$

Equation (3) says that \mathcal{H} is \mathcal{F} -complete for Wlp_p . In other words: consider the abstraction of a concrete input state, X , via \mathcal{H} ; this yields a set of states where the private information is abstracted to “any possible value”. The equation asserts that $Wlp_p(\mathcal{H}(X))$ is a fixpoint of \mathcal{H} , meaning that $Wlp_p(\mathcal{H}(X))$ yields a set of states where each public input is associated with any possible private input: a further abstraction of the fixpoint (c.f., the lhs of equation (3)) yields nothing new. Because no *distinctions among private inputs* get exposed to an observer, the public output is independent of the private input. Hence equation (3) asserts standard NI.

The following theorem asserts that the two ways of describing noninterference by means of \mathcal{B} - and \mathcal{F} -completeness are equivalent.

Theorem 4.1 $\mathcal{H} \circ \llbracket P \rrbracket \circ \mathcal{H} = \mathcal{H} \circ \llbracket P \rrbracket$ iff $\mathcal{H} \circ Wlp_p \circ \mathcal{H} = Wlp_p \circ \mathcal{H}$.

Proof. By [11,12] we know that, if f is additive, then for any ρ we have $\rho \circ f \circ \rho = \rho \circ f$ iff $\rho \circ f^+ \circ \rho = f^+ \circ \rho$. By [4] we have that $\llbracket P \rrbracket^+ = Wlp_P$. Choosing $\mathcal{H}, \llbracket P \rrbracket$ as ρ, f resp., we are done. \square

Example 4.2 Consider the program P , let $\mathbb{V}^H = \{0, 1\} \times \{0, 1\}$, $\mathbb{V}^L = \{0, 1\}$.

$$P \stackrel{\text{def}}{=} \begin{cases} \text{if } h_1 \neq h_2 \text{ then } l := h_1 + h_2 \\ \text{else } l := h_1 - h_2 + 1; \end{cases} \quad \begin{array}{l} \llbracket P \rrbracket : \langle h_1, h_2, l \rangle \mapsto \langle h_1, h_2, 1 \rangle \\ Wlp_P : \begin{cases} \langle h_1, h_2, 1 \rangle \mapsto \{ \langle h_1, h_2 \rangle \} \times \mathbb{V}^L \\ \langle h_1, h_2, l \rangle \mapsto \emptyset \quad l \neq 1 \end{cases} \end{array}$$

The public output l is always 1, hence P is secure, as the following calculation shows. Given $\mathbb{V}^H \times \{l\} \in \mathcal{H}$, we can prove that \mathcal{B} -completeness for $\llbracket P \rrbracket$ holds:

$$\mathcal{H}(\llbracket P \rrbracket(\mathbb{V}^H \times \{l\})) = \mathcal{H}(\mathbb{V}^H \times \{1\}) = \mathbb{V}^H \times \{1\} = \mathcal{H}(\langle h_1, h_2, 1 \rangle) = \mathcal{H}(\llbracket P \rrbracket(\langle h_1, h_2, l \rangle))$$

\mathcal{F} -completeness for Wlp_P holds also:

$$\begin{aligned} \mathcal{H}(Wlp_P(\mathbb{V}^H \times \{1\})) &= \mathcal{H}(\mathbb{V}^H \times \mathbb{V}^L) = \mathbb{V}^H \times \mathbb{V}^L = Wlp_P(\mathbb{V}^H \times \{1\}) \\ \mathcal{H}(Wlp_P(\mathbb{V}^H \times \{l \neq 1\})) &= \mathcal{H}(\emptyset) = \emptyset = Wlp_P(\mathbb{V}^H \times \{l \neq 1\}) \end{aligned}$$

5 Completeness and Declassified NI (DNI)

When does a program P satisfy noninterference *declassified* by ϕ ? Consider any two runs with states $s_1, s_2 \in \Sigma$. Suppose $s_1 =_L s_2$. Let s_1^H and s_2^H denote the secret values in s_1, s_2 that are declassified by ϕ and suppose that the distinction between the declassified values is not exposed in the two runs of P , i.e. $\phi(s_1^H) = \phi(s_2^H)$. Then P satisfies noninterference declassified by ϕ provided $\llbracket P \rrbracket(s_1) =_L \llbracket P \rrbracket(s_2)$. Formally:

$$s_1 =_L s_2 \wedge \phi(s_1^H) = \phi(s_2^H) \Rightarrow \llbracket P \rrbracket(s_1) =_L \llbracket P \rrbracket(s_2)$$

Note that ordinary noninterference is obtained by setting $\phi(s_1^H) = \top = \phi(s_2^H)$.

5.1 Modelling declassification

The discussion in the previous section has not motivated *why* we might want Wlp_P and this is what we proceed to do in the context of declassification.

Consider secrets $h_1, h_2 \in \{0, 1\}$ and the declassification policy “at most one of the secrets h_1, h_2 is 1”. The policy releases a relation between h_1 and h_2 but not their exact values. Does the program $P \stackrel{\text{def}}{=} l := h_1 + h_2$ satisfy the policy?

Here $\mathbb{V}^H = \{0, 1\} \times \{0, 1\}$ and the declassifier, ϕ , is defined as: $\phi(\emptyset) = \emptyset$; $\phi(\langle 0, 0 \rangle) = \phi(\langle 0, 1 \rangle) = \phi(\langle 1, 0 \rangle) = \{ \langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle \}$ (i.e., we collect together all the elements with the same declassified property) and $\phi(\langle 1, 1 \rangle) = \mathbb{V}^H$; $\phi(X) = \bigcup_{x \in X} (\phi(\{x\}))$. A program that respects the above policy *should not expose the distinctions* between inputs $\langle 0, 0 \rangle$, $\langle 0, 1 \rangle$ and $\langle 1, 0 \rangle$ at the public output. But it *is* permissible to expose the distinction between $\langle 1, 1 \rangle$

and any pair from the partition block $\{\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle\}$, because this latter distinction is supported by the policy. Does P expose distinctions it should not?

To answer, we consider $Wlp_P(l = a)$, where a is some generic output value. Why Wlp ? Because then we can statically simulate the kind of analysis an attacker would do for obtaining initial values of (or initial relations among) secret information. Why $l = a$? Because this gives us the most general Wlp , parametric on the output value. Now, note that $Wlp_P(l = a) = (h_1 + h_2 = a)$; let $H_a \stackrel{\text{def}}{=} (h_1 + h_2 = a)$. Because $a \in \{0, 1\}$, we have $H_0 \stackrel{\text{def}}{=} (h_1 + h_2 = 0)$. This allows the attacker to solve for h_1, h_2 : $h_1 = 0, h_2 = 0$. Thus when $l = 0$, a distinction, $\{\langle 0, 0 \rangle\}$, in the partition block, $\{\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle\}$ gets exposed. Hence the program does not satisfy the policy.

So consider a declassified confidentiality policy, and model the declassified information by means of the abstraction ϕ , of the private inputs, which collects together all the elements with the same property, declassified by the policy. Let $\mathcal{H}^\phi : \wp(\Sigma) \rightarrow \wp(\Sigma)$ be the corresponding abstraction function. Accordingly, let $X \in \wp(\Sigma)$ be a concrete set of states and let X^L be the L slice of X . Consider any $l \in X^L$. Define set $H_l \stackrel{\text{def}}{=} \{h \in \mathbb{V}^H \mid \langle h, l \rangle \in X\}$; i.e., given an l , H_l contains all the H values associated with l in X . Then the ‘‘declassified’’ abstract domain, $\mathcal{H}^\phi(X)$, corresponding to X is defined as $\mathcal{H}^\phi(X) = \bigcup_{l \in X^L} \phi(H_l) \times \{l\}$. Note that the domain, \mathcal{H} , for ordinary noninterference is the instantiation of \mathcal{H}^ϕ , where ϕ maps any set to \top . The analogue of equation (2)

$$\mathcal{H}^\phi \circ Wlp_P \circ \mathcal{H} = Wlp_P \circ \mathcal{H} \quad (4)$$

asserts that $(\mathcal{H}^\phi, \mathcal{H})$ is \mathcal{F} -complete for Wlp_P . For example, \mathcal{F} -completeness fails for the program P . With $X = \langle 0, 0, 0 \rangle$, we have $\mathcal{H}(X) = \mathbb{V}^H \times \{0\}$ and $Wlp_P(\mathcal{H}(X)) = \{\langle 0, 0, 0 \rangle\}$. But $\mathcal{H}^\phi(Wlp_P(\mathcal{H}(X))) = \{\langle 0, 0, 0 \rangle, \langle 0, 1, 0 \rangle, \langle 1, 0, 0 \rangle\} \supset Wlp_P(\mathcal{H}(X))$.

We are now in a position, via Theorem 5.1 below, to connect \mathcal{H}^ϕ to NI: the only caveat is that ϕ must *partition* the input abstract domain, i.e., $\forall x. \phi(x) = \{y \mid \phi(x) = \phi(y)\}$. The intuition behind partitioning is that ϕ ’s image on singletons is all we need for deriving the property of any possible set.

Theorem 5.1 *Consider a partitioning ϕ . Then P satisfies noninterference declassified by ϕ iff $\mathcal{H} \circ \llbracket P \rrbracket \circ \mathcal{H}^\phi = \mathcal{H} \circ \llbracket P \rrbracket$.*

Together with Theorem 4.1 we are led to

Corollary 5.2 *Consider a partitioning ϕ . Then P satisfies noninterference declassified by ϕ iff $\mathcal{H}^\phi \circ Wlp_P \circ \mathcal{H} = Wlp_P \circ \mathcal{H}$, i.e., $(\mathcal{H}^\phi, \mathcal{H})$ is \mathcal{F} -complete for Wlp_P .*

The equality in the corollary asserts that *nothing more is released* by the Wlp than what is already released by ϕ . If \mathcal{F} -completeness did not hold, but $(\mathcal{H}^\phi, \mathcal{H})$ was merely sound, then $\mathcal{H}^\phi \circ Wlp_P \circ \mathcal{H} \sqsupseteq Wlp_P \circ \mathcal{H}$. In this case Wlp (i.e., the rhs) releases *more information* (technically: is more concrete) than that declassified (i.e., the lhs). Our goal is not only to check whether a program satisfies a particular confidentiality policy, but also to find the public observations that may breach the confidentiality policy and also the associated secret that each offending observation reveals. Consider, for example, the following program [7] where $l, h \in \text{Nats}$.

$$P \stackrel{\text{def}}{=} \mathbf{while} (h > 0) \mathbf{do} (h := h - 1; l := h) \mathbf{endw}$$

If we observe $l = 0$ at output, all we can say about input h is $h \geq 0$. But with output observation $l \neq 0$, we can deduce $h = 0$ in the input: the loop must not have been executed.

Because Wlp relates the observed (public) output to the private (secret) inputs, therefore, from the final observation we can derive the exact secret which is released by that observation in the following manner: (a) Compute wlp wrt. each observation obtaining a most general predicate on the input states. (b) Check whether the states described by the wlp are “more abstract”, i.e., do not permit more distinctions of private input than those permitted by the policy. If so, there is no breach.

Example 5.3 Consider the following code [22]

$$P \stackrel{\text{def}}{=} h := h \bmod 2; \text{ if } h = 0 \text{ then } (h := 0; l := 0) \text{ else } (h := 1; l := 1);$$

Let $\mathbb{V}^H = \text{Nats} = \mathbb{V}^L$. Suppose we wish to declassify the test, $h = 0$. Then $\phi(\{0\}) = \{0\}$ and $\phi(\{h\}) = \text{Nats} \setminus \{0\}$. Thus $\{h \mid h \neq 0\}, \{0\}$ is the partition induced by ϕ on \mathbb{V}^H and we obtain $\mathcal{H}^\phi = \{\emptyset, \top\} \cup \{\{h \mid h \neq 0\} \times \mathbb{V}^L\} \cup \{\{0\} \times \mathbb{V}^L\}$.

Let $H_a \stackrel{\text{def}}{=} \mathbb{V}^H \times \{a\}$. Consider now the wlp of the program, $Wlp_P(l = a)$, where $a \in \mathbb{V}^L$.

$$\begin{aligned} & \{(a = 0 \wedge h \bmod 2 = 0) \vee (a = 1 \wedge h \bmod 2 = 1)\} \\ & h := h \bmod 2; \\ & \{(a = 0 \wedge h = 0) \vee (a = 1 \wedge h = 1)\} \\ & \text{if } (h = 0) \text{ then } (h := 0; l := 0) \text{ else } (h := 1; l := 1) \\ & \{l = a\} \end{aligned}$$

Thus, Wlp maps output set of states H_0 to the input states $\{\langle h, l \rangle \mid h \bmod 2 = 0, l \in \mathbb{V}^L\}$. But this state is not more abstract than the state, $\{h \mid h \neq 0\} \times \mathbb{V}^L$, specified by \mathcal{H}^ϕ : it distinguishes, e.g., $(8, 1)$ from $(7, 1)$ - a distinction not permitted under the policy. Indeed, consider two runs of P with initial values 8 and 7 of h and 1 for l ; $\phi(8) = \phi(7)$; yet we get two distinct output values of l .

Example 5.4 Consider $P \stackrel{\text{def}}{=} \text{if } (h \geq k) \text{ then } (h := h - k; l := l + k) \text{ else skip}$ [22], and its Wlp semantics. Consider $H_{a,b} \stackrel{\text{def}}{=} \{\langle h, l, k \rangle \mid h \in \mathbb{V}^H, l = a, k = b\}$. Suppose the declassification policy is \top , i.e., nothing has to be released.

$$\begin{aligned} & \{(h \geq b \wedge l = a - b \wedge k = b) \vee (h < b \wedge l = a \wedge k = b)\} \\ & \text{if } (h \geq k) \text{ then } (h := h - k; l := l + k) \text{ else skip} \\ & \{l = a \wedge k = b\} \end{aligned}$$

$$Wlp_P : H_{a,b} \mapsto \{\langle h, a - b, b \rangle \mid h \geq b\} \cup \{\langle h, a, b \rangle \mid h < b\}$$

In this case, we can say that the program does not satisfy the security policy. In fact, in presence of the same public inputs we can distinguish between values h greater than the initial value of k , and lower than this value. Note that, in this case the way $Wlp_P(H_{a,b})$ partitions the private value domain depends also on the public input. This is not a problem,

since by completing the input domain with these elements we are able to induce a partition of the private domain only. In this way, $H_{a,b} \stackrel{\text{def}}{=} \{\langle h, l, k \rangle \mid h \in \mathbb{V}^H, l = a, k = b\}$ has to be split in the elements $H'_{a,b} \stackrel{\text{def}}{=} \{\langle h, l, k \rangle \mid h \in \mathbb{V}^H, l = a, h \geq k = b\}$ distinguishing $h \geq k$, and $H''_{a,b} \stackrel{\text{def}}{=} \{\langle h, l, k \rangle \mid h \in \mathbb{V}^H, l = a, h < k = b\}$ distinguishing $h < k$, and hence the initial policy \top does not guarantee security.

Example 5.5 Consider the Oblivious Transfer Protocol [21], with principals Alice and Bob. Alice has two messages. Bob knows the messages by name but not by content. Bob asks for a message by name. But Alice does not know which message Bob asked for, and Bob has not to find out the content of the other message.

$$P \stackrel{\text{def}}{=} \left[\begin{array}{l} r_0, r_1 : \in M; d : \in \{0, 1\}; \\ r := r_d; \\ e := c \oplus d; \\ f_0, f_1 := m_0 \oplus r_e, m_1 \oplus r_{1 \oplus e}; \\ m := f_c \oplus r; \end{array} \right. \begin{array}{c} \begin{array}{|c|} \hline \mathbf{Alice} \\ \hline \end{array} \\ \hline \text{Hid: } r; d; c \in \{0, 1\}; m \\ \text{Vis: } m_0; m_1; r_0; r_1; f_0; f_1; e \\ \hline \end{array} \begin{array}{c} \begin{array}{|c|} \hline \mathbf{Bob} \\ \hline \end{array} \\ \hline m_0; m_1; r_0; r_1 \\ c; m; f_0; f_1; e; d; r \\ \hline \end{array} \end{array}$$

The protocol is implemented via a trusted third party, Ted, who sends the random messages r_0, r_1 to Alice and the random bit d to Bob. In the implementation (due to C. Morgan) M denotes the set of messages and \oplus is xor; the table above shows what is *Hid* (“hidden” or H) and *Vis* (“visible” or L) for Alice and Bob. Bob randomly chooses bit c and sends Alice $e = c \oplus d$. Alice sends Bob f_0, f_1 whence Bob can now obtain m_c as $f_c \oplus r$.

If we compute the *Wlp* we derive that the relations disclosed are $f_0 = m_0 \oplus r_0$ and $f_1 = m_1 \oplus r_1$ when $c = d$, and $f_0 = m_0 \oplus r_1$ and $f_1 = m_1 \oplus r_0$ when $c \neq d$. In both cases, the message m_c that Bob can read is combined with the random message Bob knows (since $r = r_d$ and r public to Bob). We can summarize the *Wlp* in the following way: $f_c = m_c \oplus r_d$ and $f_{1 \oplus c} = m_{1 \oplus c} \oplus r_{1 \oplus d}$. Hence, $f_{1 \oplus c}$ tells *almost nothing* about the hidden message $m_{1 \oplus c}$, expressing only if it is equal or not with an unknown random message, $r_{1 \oplus d}$ ⁴.

5.2 Deriving counterexamples

Can we mechanize the derivation of counterexamples? That is, can we derive exactly where the policy fails by demonstrating two input states that break noninterference?

We have advanced the thesis that noninterference is a completeness problem in abstract interpretation. Ranzato and Tapparo [20] studied completeness in abstract interpretation from a more algorithmic point of view. They show a correspondence between completeness and the Paige-Tarjan (PT) algorithm [19] for partition refinement, that derives the coarsest bisimulation of a given partition. Hence, we have a correspondence between completeness and absence of *unstable* elements of a closure wrt. a function f : Given a partition $\Pi \subseteq \wp(C)$ and $f : \wp(C) \rightarrow \wp(C)$, an element $X \in \Pi$ is *stable* for f with respect to $Y \in \Pi$ if $X \subseteq f(Y)$ or $X \cap f(Y) = \emptyset$; otherwise X is *unstable*. The understanding of completeness in terms of stability guarantees that if an abstract domain is not complete than there exist at least two of its elements which are not stable. In our context, f , is *Wlp*; the element for which we want

⁴ We leave a probabilistic analysis of “almost nothing” as future work.

to check stability is a set of private inputs in the partition of \mathbb{V}^H induced by the declassifier, ϕ ; and the element against which we check stability (Y in the definition) is the particular output observation (e.g., $l = a$).

Proposition 5.6 *Unstable elements of \mathcal{H}^ϕ provide counterexamples to ϕ .*

Proof. Suppose that $\exists l \in \mathbb{V}^L$ such that (the input states described by) $Wlp(H_l) \notin \mathcal{H}^\phi$. Then there exist $x \in Wlp(H_l)$, and $h \in \phi(x^H)$ such that $\langle h, x^L \rangle \notin Wlp(H_l)$. Note that $\phi(x^H) \times \{x^L\} \cap Wlp(H_l) \neq \emptyset$ since x is in both; and, $\phi(x^H) \times \{x^L\} \not\subseteq Wlp(H_l)$ since $\langle h, x^L \rangle \in \phi(x^H) \times \{x^L\}$ and $\langle h, x^L \rangle \notin Wlp(H_l)$. Hence, the abstract domain \mathcal{H}^ϕ is not *stable*. To find a counterexample consider $h_1 \in \phi(x^H) \setminus \{k \mid \langle k, x^L \rangle \in Wlp(H_l)\}$ and $h_2 \in \{k \mid \langle k, x^L \rangle \in Wlp(H_l)\}$. The latter set is obtained by wlp for the output observation l , hence any of its elements, e.g., h_2 , leads to the observation l , while all the elements outside the set, e.g., h_1 , cannot lead to l . \square

Example 5.7 Consider the following program with h 's parity declassified. We can compute Wlp_P wrt. $l = a \in \mathbb{Z}$, and $H_a \stackrel{\text{def}}{=} \{\langle h, l \rangle \mid h \in \mathbb{V}^H, l = a\}$.

$$\begin{aligned} & \{(h = 0 \wedge l = a) \vee (h > 0 \wedge a = 0)\} \\ & \mathbf{while} (h > 0) \mathbf{do} (h := h - 1; l := h) \mathbf{endw} \\ & \{l = a\} \\ Wlp : & \begin{cases} H_0 \mapsto \{\langle h, l \rangle \mid h > 0, l \in \mathbb{V}^L\} \cup \{\langle 0, 0 \rangle\} \\ H_a \mapsto \{\langle 0, a \rangle\} \quad (a \neq 0) \end{cases} \end{aligned}$$

Hence, $Wlp(H_0) = (\mathbb{V}^H \times \{0\}) \cup \{\langle h, l \rangle \mid h > 0, l \neq 0\}$. Thus, all the input states where $l = 0$ are not counterexamples to the declassification policy. On the contrary, for any two runs agreeing on input $l \neq 0$, whenever $h_1 = 0$ and $h_2 \in \text{Evens} \setminus \{0\}$, we observe different outputs. Hence, we can distinguish more than the declassified partition $\{\text{Evens}, \text{Odds}\}$.

The following example [16] shows that this approach provides a weakening of noninterference which corresponds to relaxed noninterference. Both approaches provide a method for characterizing the information that flows and that have to be declassified, indeed they both give the same result since they are driven by (parametric on) the particular output observation. However, let us underline that the abstract interpretation-based approach allows also to derive the maximal information disclosed *independently from the observed public output* [17].

Example 5.8 Consider the program P [16] with $sec, x, y : H$, and $in, out : L$, where $hash$ is a function:

$$P \stackrel{\text{def}}{=} \begin{cases} x := hash(sec); y := x \bmod 2^{64}; \\ \mathbf{if} y = in \mathbf{then} out := 1 \mathbf{else} out := 0; \\ z := x \bmod 3; \end{cases}$$

Consider its *Wlp* semantics where *out*, *in* and *z* are respectively $a, b, c \in \mathbb{Z}$:

$$\begin{aligned}
 & \{(a = 1, out = a, hash(sec) \bmod 2^{64} = b, hash(sec) \bmod 3 = c) \vee \\
 & \quad (a = 0, out = a, hash(sec) \bmod 2^{64} \neq b, hash(sec) \bmod 3 = c)\} \\
 & x := hash(sec); y := x \bmod 2^{64}; \\
 & \{(a = 1, out = a, y = b, x \bmod 3 = c) \vee (a = 0, out = a, y \neq b, x \bmod 3 = c)\} \\
 & \text{if } y = in \text{ then } out := 1 \text{ else } out := 0; \\
 & \{out = a, in = b, x \bmod 3 = c\} \\
 & z := x \bmod 3; \\
 & \{out = a, in = b, z = c\}
 \end{aligned}$$

Let us consider first P without the last assignment to z and consider the domain formed by the sets $H_{in,1} \stackrel{\text{def}}{=} \{\langle sec, in, out, x, y \rangle \mid in = hash(sec) \bmod 2^{64}, out = 1\}$ and $H_{in,0} \stackrel{\text{def}}{=} \{\langle sec, in, out, x, y \rangle \mid in \neq hash(sec) \bmod 2^{64}, out = 0\}$. The set of all these domains embodies the declassification policy since it collects together all the tuples such that sec has the same value for $hash(sec) \bmod 2^{64}$. At this point note that the Wlp_P semantics does the following associations: $Wlp : H_{in,a} \mapsto H_{in,a}$ and this clearly means that the domain is complete, i.e., the declassification policy is sufficient to protect the program.

Let us consider now also the last assignment, then we have one more variable and we redefine $H_{in,a}$ as sets of tuples containing also z but without any condition on z since it is not considered in the declassification policy. In this case, the *Wlp* semantics does the following associations:

$$Wlp : \begin{cases} H_{in,1} \mapsto H_{in,1} \cap \left\{ \langle sec, in, out, x, y, z \rangle \mid hash(sec) \bmod 3 = z \right\} \\ H_{in,0} \mapsto H_{in,0} \cap \left\{ \langle sec, in, out, x, y, z \rangle \mid hash(sec) \bmod 3 = z \right\} \end{cases}$$

The new elements added to the domain have one more condition on the private variable sec , which can distinguish further the private inputs by observing the public output. This makes the initial declassification policy unsatisfied.

5.3 Refining confidentiality policies

The natural use of the method previously described is for a semantic driven *refinement* of confidentiality policies. The idea is to start with a confidentiality policy stating what can be released in terms of abstract domains (or equivalence relations). In the extreme case, the policy could state that nothing about private information must be released.

A consequence of Corollary 5.2 is that whenever \mathcal{H}^ϕ is *not* forward complete for Wlp_P , more information is released than what declassification ϕ permits. Thus the partition induced on the private domain by ϕ must be *refined* by the completion process. To derive the refined policy, ϕ' , we perform the following steps: (a) Consider the domain, $\mathcal{H}^{\phi'}$, obtained by completion from \mathcal{H}^ϕ ; (b) for each $Y \in \mathcal{H}^{\phi'}$ compute sets, $\pi_l(Y)$, that are parametric on a fixed public value $l \in \mathbb{V}^L$, where: $\pi_l(Y) \stackrel{\text{def}}{=} \{h \in \mathbb{V}^H \mid \langle h, l \rangle \in Y\}$; (c) for

each l , compute the partition, $\bar{\pi}_l$, induced on the private domain as $\bar{\pi}_l \stackrel{\text{def}}{=} \bigwedge_{X \in \mathcal{H}^{\phi'}} \pi_l(X)$; (d) let $\pi \stackrel{\text{def}}{=} \bigwedge_{l \in \mathbb{V}^L} \bar{\pi}_l$. The declassification policy, ϕ' , can now be defined as a refinement, $\mathcal{R}(\phi)$ of ϕ , by computing the *partitioning closure corresponding to π* [14], i.e., the *disjunctive completion*, Υ , of the sets forming the partition: $\phi' = \mathcal{R}(\phi) \stackrel{\text{def}}{=} \Upsilon(\pi)$.

For instance, in Example 5.4, each output observation $k = b$ induces the partition $\pi_b = \{\{h \mid h \geq b\}, \{h \mid h < b\}\}$, which is the information released by the single observation. If we consider the set of all the possible observations, then we derive $\pi = \bigwedge_b \pi_b = id$, namely we have $\phi = id$.

Proposition 5.9 *Let ϕ model the information declassified. If $\mathcal{H}^\phi \circ Wlp_P \circ \mathcal{H} \sqsupseteq Wlp_P \circ \mathcal{H}$, then $\mathcal{R}(\phi) \sqsupseteq \phi$, i.e., $\mathcal{R}(\phi)$ is a refinement of ϕ , and it is the closest to ϕ ⁵*

Example 5.10 Consider the program P [22] with $\mathbb{V}^H = \mathbb{V}^L = \mathbb{Z}$ and its Wlp semantics

$$\begin{aligned}
 P &\stackrel{\text{def}}{=} h_1 := h_1; h_2 := h_1; \dots h_n := h_1; avg := \text{declassify}((h_1 + h_2 + \dots + h_n)/n) \\
 &\{h_1 = a\} \\
 h_1 := h_1; h_2 := h_1; \dots h_n := h_1; & \quad Wlp: \left\{ \begin{array}{l} X \mapsto \emptyset \quad \text{if } \forall a \in \mathbb{V}^L. X \neq H_a \\ H_a \mapsto \left\{ \langle a, h_2, \dots, h_n, a \rangle \mid \begin{array}{l} \forall i. h_i \in \mathbb{Z} \\ avg = a \end{array} \right\} \end{array} \right. \\
 \{ (h_1 + h_2 + \dots + h_n)/n = a \} & \\
 avg := (h_1 + h_2 + \dots + h_n)/n & \\
 \{ avg = a \} &
 \end{aligned}$$

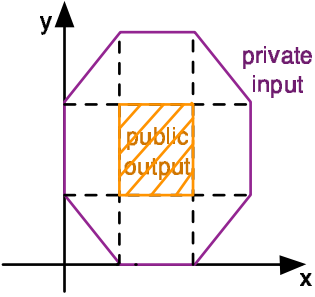
where $H_a \stackrel{\text{def}}{=} \{\langle h_1, \dots, h_n, avg \rangle \mid h_i \in \mathbb{Z}, (h_1 + h_2 + \dots + h_n)/n = avg = a \in \mathbb{Z}\}$. Suppose the input declassification policy releases the average of the private values, i.e., $\phi(\langle h_1, \dots, h_n \rangle) \stackrel{\text{def}}{=} \{\langle h'_1, \dots, h'_n \rangle \mid (h'_1 + \dots + h'_n)/n = (h_1 + \dots + h_n)/n\}$; the policy collects together all the possible private inputs with the same average value. Hence, the average is the only property that this partition of states allows to observe. Clearly, the program releases more. Consider $n = 4$, $h_i \in \{1, \dots, 8\}$, and $X = H_4$. The partition induced by $\mathcal{H}^\phi(X)$ on the states with $avg = 4$ is $\{\langle 5, 2, 3, 6, 4 \rangle, \langle 7, 3, 1, 5, 4 \rangle, \dots\}$. But $Wlp_P(H_4) = \{\langle 4, 3, 7, 2, 4 \rangle, \langle 4, 8, 3, 1, 4 \rangle, \dots\}$. Thus, we need to refine the original policy, completing $\mathcal{H}^\phi(X)$ wrt. Wlp_P : we add elements $Wlp_P(H_a)$ for all $a \in \mathbb{Z}$. In each such element, h_1 has the particular value a . Formally, the domain, $\mathcal{H}^{\phi'}(X)$, contains all the sets $\{\langle h_1, h_2, \dots, h_n, avg \rangle \mid h_1 = avg = a, \forall i > 1. h_i \in \mathbb{Z}\}$; $\mathcal{H}^{\phi'}(X)$ distinguishes all tuples that differ in the first private input, where ϕ' is obtained as disjunctive completion of the computed partition and declassifies the value of h_1 . This is the closest domain to ϕ since, if we add any other element in the resulting domain, we would distinguish more than what is necessary, i.e., more than the distinction on the value of h_1 . Indeed, still abstracting the average of the elements, we could add other sets of tuples with the same average value, but the only ones that we can add (i.e, which are not yet in the domain) must add some new distinctions. For example, if we add sets like $\{\langle 4, 6, 1, 5, 4 \rangle, \langle 4, 6, 3, 3, 4 \rangle, \dots\}$, where also h_2 is fixed, then we allow also to distinguish the value of h_2 , which is not released by the program.

⁵ In theory, this refinement can also be computed as the intersection between the policy ϕ and the refinement of the unde-classified policy \top . Efficiency comparison between these two approaches is left to the implementation phase of our work.

5.4 Refining Abstract Noninterference policies

The method described for checking and refining a security policy is parametric on public observations, but one could carry out the same process on *properties*. If some information about the execution context of the program is present then we can restrict (abstract) the possible observations. These restrictions can be modeled as abstract domains, and therefore by means of abstract noninterference policies. In particular, it has been proved [10] that the more we observe about public information, the less private information can be kept secret. This means that a security policy, unsafe in a general context, can become safe if we consider a weaker observation of the public output.

Consider, for example, the following program P with two private inputs x, y , and two public outputs x_L, y_L . d, d_x, d_y are constant public inputs with $d_x > d$ and $d_y > d$:

$$P \stackrel{\text{def}}{=} \left[\begin{array}{l} \text{if}(d \leq x + y \leq d + d_x + d_y \wedge -d_y \leq x - y \leq d_x) \text{ then} \\ \quad \text{if}(x \geq 0 \wedge x \leq d) \text{ then } x_L := d; \\ \quad \text{if}(x > d \wedge x \leq d_x) \text{ then } x_L := x; \\ \quad \text{if}(x > d_x \wedge x \leq d_x + d) \text{ then } x_L := d_x; \\ \quad \text{if}(y \geq 0 \wedge y \leq d) \text{ then } y_L := d; \\ \quad \text{if}(y > d \wedge y \leq d_y) \text{ then } y_L := y; \\ \quad \text{if}(y > d_y \wedge y \leq d_y + d) \text{ then } y_L := d_y; \end{array} \right.$$


Instead of concrete inputs and outputs, we might want to track *properties*, e.g., in what *interval* a particular variable lies. The figure above represents the input and output of the program in graphical form: the program transforms an input property, namely, an *octagon* (in the private variables x, y , represented by sets of constraints of the form $\pm x \pm y \leq c$) to an output property, namely, a *rectangle* (in the variables x_L, y_L): Thus if we take Wlp_P wrt. the *property of intervals* – this corresponds to rectangles in the 2-dimensional space – then the Wlp_P semantics returns an *octagon abstract domain* [18], i.e., we derive an octagonal relation between the two private inputs. Thus the security policy has to declassify at least the octagon domain in order to make the program secure.

Moreover, *abstract noninterference* policies can be useful in order to make the algorithm *computable*. In fact by abstracting the public domain we can make finite the amount of possible observations of the attacker; in practice, this means that when we compute $Wlp(\rho(l) = A)$ we are guaranteed that there will be finitely many Wlp computations whenever the abstract domain is finite.

This example shows that we can combine (narrow) abstract non interference [9] with declassification in the following completeness equation: Let $\mathcal{H}_\rho \stackrel{\text{def}}{=} \lambda X. \mathbb{V}^H \times \rho(X^L)$ [10] and $\mathcal{H}_\eta^\phi \stackrel{\text{def}}{=} \lambda X. \phi(H_{\eta(l)}) \times \eta(l)$, where $H_{\eta(l)} \stackrel{\text{def}}{=} \{h \mid \eta(l') = \eta(l), \langle h, l' \rangle \in X\}$

$$\mathcal{H}_\eta^\phi \circ Wlp_P \circ \mathcal{H}_\rho = Wlp_P \circ \mathcal{H}_\rho$$

6 Abstract model checking and information flow

In the previous sections we have seen how we can verify and refine confidentiality policies that admit some leak of private information. The whole study is done by considering I/O semantics (denotational and wlp) and modelling DNI as a completeness problem. On the other hand, the strong relationship between completeness and stability (in the Paige-Tarjan sense) existing in the framework of abstract model checking (AMC) has been studied [20,11]: the completeness in question is \mathcal{B} -completeness for the *post* function induced by an equivalence relation on the domain of states. Since the denotational semantics is the *post* for a transition system where all traces are two-states long – because they are I/O states – a straightforward generalization of our work and of the notion of noninterference can be obtained via a generic *post* function. Hence, two traces are L indistinguishable, i.e., $=_L$, if they have the same public projection.

Theorem 6.1 *Let $\langle\!\langle P \rangle\!\rangle$ the standard trace semantics of P (deterministic program). The noninterference on traces, i.e., $\forall \sigma_1, \sigma_2 \in \Sigma. \sigma_1 =_L \sigma_2 \Rightarrow \langle\!\langle P \rangle\!\rangle(\sigma_1) =_L \langle\!\langle P \rangle\!\rangle(\sigma_2)$, holds iff $\mathcal{H} \circ \text{post}_p \circ \mathcal{H} = \mathcal{H} \circ \widetilde{\text{pre}}_p \circ \mathcal{H}$, where post_p is the post function associated with the transition system modelling P .*

This theorem implies that we can characterize the declassification property on the private information of states also when we have to protect the whole trace semantics from malicious observations. Moreover, the completeness equation, rewritten as $\mathcal{H} \circ \widetilde{\text{pre}} \circ \mathcal{H} = \widetilde{\text{pre}} \circ \mathcal{H}$ (via Theorem 4.1), asserts (in the context of AMC) that there are *no spurious counterexamples*. In the NI context, this means that there is no leakage of information. In particular, for *declassification*, if $\mathcal{H}^\phi \circ \widetilde{\text{pre}}_p \circ \mathcal{H} = \widetilde{\text{pre}}_p \circ \mathcal{H}$ holds, there is no need to further declassify private information via refinement, even if we suppose that the attacker can observe every intermediate step of computation.

In the following simple example we show how this approach works by also providing its relationship with the equivalence relation transformer defined by Zdancewic and Myers [28] for characterizing leakages of private information. First, we rewrite their transformer as follows: Let \approx be an equivalence relation, we define $\sigma_1 S(\approx) \sigma_2$ if and only if $\forall i > 0. [\text{post}^i([\sigma_1]_{\approx})]_{\approx} = [\text{post}^i([\sigma_2]_{\approx})]_{\approx}$. (post^i is the composition of *post* with itself i times.) This new equivalence relation, if different from \approx , tells us that something is released. With our method, we can characterize exactly what is released. When we deal with equivalence relations, the backward completeness equation for *post* can be rewritten as [14]: $[\text{post}[\sigma_1]_{\approx}]_{\approx} = \text{post}[\sigma_1]_{\approx}$; and so we are led to

Theorem 6.2 $S(\approx) =_{\approx}$ iff \approx is backward complete for *post*.

Example 6.3 Consider the following transition system [28] which uses a password system to launder confidential information:

$$\begin{aligned} \langle t, h, p, q, r \rangle &\mapsto \langle t, h, p, q, r \rangle \\ \langle 0, h, p, p, 0 \rangle &\mapsto \langle 1, h, p, p, 1 \rangle \\ \langle 0, h, p, p, 1 \rangle &\mapsto \langle 1, h, p, p, 0 \rangle \\ \langle 0, h, p, q, 0 \rangle &\mapsto \langle 1, h, p, q, 0 \rangle \quad p \neq q \\ \langle 0, h, p, q, 1 \rangle &\mapsto \langle 1, h, p, q, 1 \rangle \quad p \neq q \end{aligned}$$

where $t \in \{0, 1\}$ is the time (1 indicates that the program has been executed), $r \in \{0, 1\}$ denotes the result of the test (it is left unchanged if the test of equality between the password p and the query q fails).

The public variables are t, q, r , hence the partition induced by \mathcal{H} is:

$$\langle t, h, p, q, r \rangle \equiv \langle t', h', p', q', r' \rangle \text{ iff } t = t' \wedge q = q' \wedge r = r'$$

The above says we are considering two states that are L indistinguishable (as in ordinary NI). By checking completeness we characterize the information that can be released. For example, consider the set of possible input states which are in the same equivalence class and for which the state $\langle 0, h, p, q, 0 \rangle$ is a representative. Applying the transition rules, we see (below, left) that this state reveals a different public output. Thus there is a leakage of confidential information. In order to characterize *what* information is released we complete the domain \mathcal{H} by \widetilde{pre}_p (below, right):

$$\langle 0, h, p, q, 0 \rangle \mapsto \begin{cases} \langle 1, h, p, p, 1 \rangle \\ \langle 1, h, p, q, 0 \rangle \end{cases} \quad \widetilde{pre}_p : \begin{cases} \langle 1, h, p, p, 1 \rangle \mapsto \langle 0, h, p, p, 0 \rangle \\ \langle 1, h, p, q, 0 \rangle \mapsto \langle 0, h, p, q, 0 \rangle \quad p \neq q \end{cases}$$

Hence we have to refine the original partition by adding the new blocks $\langle 0, h, p, p, 0 \rangle$ and $\langle 0, h, p, q, 0 \rangle$ where $p \neq q$, i.e., we release the information whether $p = q$ or $p \neq q$.

AMC techniques are usually applied to Kripke structures. A Kripke structure consists of a set of states, a set of transitions between states, and a function that labels each state with a set of properties that are true in the state. The Kripke model for a program corresponds to the standard transition system associated with the program where states are labelled with the values of the variables. The connection between declassification and AMC suggests the use of existing algorithms for AMC in order to derive the information released by a system, whenever the confidential information is fixed. Indeed, the existence of a spurious counterexample in the AMC (abstraction corresponding to the declassification policy) corresponds to the existence of an insecure information flow in the concrete system. Suppose we interpret the initial abstract domain of a system as a declassification policy (the distinction between all the states mapped to different properties is declassified). Then whenever an AMC algorithm finds a spurious counterexample it means that there is a breach in the security, and hence some more secrets, i.e., some more distinctions among states, are released. For instance, in the example above, the given trace (from $\langle 0, h, p, q, 0 \rangle$) would be identified as a spurious counterexample, and the refinement for erasing it is exactly the refinement we describe. When no more spurious counterexamples exist, then we have characterized, in the resulting abstract domain, the secure declassification policy.

7 Discussion

In this paper we exploit completeness of abstract interpretation for modelling noninterference for confidentiality policies based on declassification. Starting with Joshi and Leino's semantic formulation of NI [15], it is possible to characterize NI as a problem of \mathcal{B} -completeness for denotational semantics [10]. This paper provides an equivalent formulation of NI as \mathcal{F} -completeness for the wlp semantics, and extends the formulation to declassification. Semantically, we represent a declassification policy as an abstraction of the H inputs that induces a partition on them. A program that satisfies the policy is guaranteed not to expose distinctions within a partition block. \mathcal{F} -completeness formalizes "not exposing distinctions". The advantage of our formalization, compared to other approaches, is

that we can associate with each possible public observation the exact secret released. Moreover, the strong connection between completeness and declassification, together with the connection between completeness and abstract model checking, allows the use of standard techniques in abstract model checking for checking and refining declassification policies. In particular, model checking can be applied to generic finite state systems, and abstractions allow to consider even infinite state systems. As future work, we are studying the practical use of these techniques applied to more complex systems.

The relation between the abstract interpretation approach to NI [9,10] and many extant approaches for noninterference and declassification has been studied by means of examples [14,17]. Sabelfeld and Sands note that most extant proposals suffer from lack of a compelling semantics for declassification. In earlier work they use the PER model [24] for defining selective dependency [3] by means of equivalence relations instead of abstract domains. They also show, via an example, that the PER model can be used to show that nothing more is learnt by an attacker than what the policy itself releases [25]; in our model we derive this formally (Corollary 5.2) and also show how, in the case where a policy is not satisfied, counterexamples may be generated and the policy may be refined. Joshi and Leino [15] introduce *abstract variables* in order to obtain a more general notion of security. In this case they substitute the private variables with functions, i.e., properties, of them. This corresponds to abstract noninterference where we fix what we want to protect instead of what we admit to flow [9,17], hence it is not helpful for computing what information is released. Dárvas et al. [7] use dynamic logic to dynamically analyze the declassification property. The information flow property is modelled as a dynamic logic formula. Next, they fix some declassifying preconditions and execute the analysis. If the analysis succeeds then there is an upper bound on the information disclosed; otherwise the precondition must be refined. Because of the connections of completeness to PT, our approach can provide a more systematic method for designing and refining these preconditions. Our approach differs from quantitative characterizations [2,8] of the information released since we provide a *qualitative* analysis of the leaked secrets.

In a recent paper, Unno et al. [27] have proposed a method for automatically finding counterexamples of secure information flow, which combines security type-based analysis for standard NI and model checking. Our context is more general, since standard NI is a particular case of DNI. Nevertheless, as future work, we plan to investigate whether their approach can be directly derived from ours.

Alur et al. [1] consider preservation of secrecy under refinement and present a simulation-based technique to show when one system is a refinement of another wrt. secrecy. They contend that their approach is flexible because it can express arbitrary secrecy requirements. In particular, if the specification does not maintain secrecy of a property then the implementation does not need to either. Our notion of refinement is slightly different: if a program leaks more information than the policy, we consider how the policy might have to be refined to admit the program. It is possible that there might be strong connections to their work and we plan to explore these connections.

In other future work, we plan to further exploit the strong relation of NI with AMC and stability. One direction is to implement algorithms for deriving the maximal amount of information disclosed and for refining declassification policies, by erasing counterexamples. Moreover, the example above shows that it is possible to combine both abstract noninterference and declassification. So existing abstract model checking techniques can

be used not only to derive the amount of information disclosed, but also to characterize the strongest harmless attacker. Finally, we plan to extend the framework in this paper to handle heap-manipulating programs.

Acknowledgements. Thanks to the anonymous referees for their suggestions. Banerjee was supported in part by NSF grants CCR-0209205, CCR-0296182, ITR-0326577, CNS-0627748 and by the AIDA Project (MIUR-COFIN 2005-2007). Mastroeni was supported by NSF grant CCR-0209205 and Giacobazzi by the AIDA Project (MIUR-COFIN 2005-2007).

References

- [1] R. Alur, P. Cerny, and S. Zdancewic. Preserving secrecy under refinement. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Proc. of the 33rd Internat. Colloq. on Automata, Languages and Programming (ICALP '06)*, volume 4052 of *Lecture Notes in Computer Science*, pages 107–118, Berlin, 2006. Springer-Verlag.
- [2] D. Clark, S. Hunt, and P. Malacaria. Quantified interference: Information theory and information flow (extended abstract), 2004.
- [3] E. S. Cohen. Information transmission in computational systems. *ACM SIGOPS Operating System Review*, 11(5):133–139, 1977.
- [4] 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.
- [5] 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, New York, 1977. ACM Press.
- [6] 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, New York, 1979. ACM Press.
- [7] A. Darvas, R. Hähnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In D. Hutter and M. Ullmann, editors, *Security in Pervasive Computing: Second International Conference (SPC 2005)*, volume 3450, pages 193–209, Berlin, 2005. Springer-Verlag.
- [8] A. Di Pierro, C. Hankin, and H. Wiklicky. Approximate non-interference. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 1–17, Los Alamitos, Calif., 2002. IEEE Comp. Soc. Press.
- [9] R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *Proc. of the 31st Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL '04)*, pages 186–197, New York, 2004. ACM-Press.
- [10] R. Giacobazzi and I. Mastroeni. Adjoining declassification and attack models by abstract interpretation. In S. Sagiv, editor, *Proc. of the European Symp. on Programming (ESOP '05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 295–310, Berlin, 2005. Springer-Verlag.
- [11] 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, Berlin, 2001. Springer-Verlag.
- [12] R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. of the ACM.*, 47(2):361–416, 2000.
- [13] J. A. Goguen and J. Meseguer. Unwinding and inference control. In *Proc. IEEE Symp. on Security and Privacy*, pages 75–86, Los Alamitos, Calif., 1984. IEEE Comp. Soc. Press.
- [14] S. Hunt and I. Mastroeni. The PER model of abstract non-interference. In C. Hankin and I. Siveroni, editors, *Proc. of The 12th Internat. Static Analysis Symp. (SAS '05)*, volume 3672 of *Lecture Notes in Computer Science*, pages 171–185, Berlin, 2005. Springer-Verlag.
- [15] R. Joshi and K. R. M. Leino. A semantic approach to secure information flow. *Science of Computer Programming*, 37:113–138, 2000.
- [16] P. Li and S. Zdancewic. Downgrading policies and relaxed noninterference. In *Proc. of the 32nd Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL '05)*, pages 158–170, New York, 2005. ACM-Press.
- [17] I. Mastroeni. On the role of abstract non-interference in language-based security. In K. Yi, editor, *Third Asian Symp. on Programming Languages and Systems (APLAS '05)*, volume 3780 of *Lecture Notes in Computer Science*, pages 418–433, Berlin, 2005. Springer-Verlag.
- [18] A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19:31–100, 2006.

- [19] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):977–982, 1987.
- [20] F. Ranzato and F. Tapparo. An abstract interpretation-based refinement algorithm for strong preservation. In N. Halbwachs and L. Zuck, editors, *Proc. of TACAS: Tools and Algorithms for the Construction and Analysis of Systems*, volume 3440 of *Lecture Notes in Computer Science*, pages 140–156, Berlin, 2005. Springer-Verlag.
- [21] Ronald Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. Unpublished note, 1999.
- [22] A. Sabelfeld and A. C. Myers. A model for delimited information release. In N. Yonezaki K. Futatsugi, F. Mizoguchi, editor, *Proc. of the International Symp. on Software Security (ISSS'03)*, volume 3233 of *Lecture Notes in Computer Science*, pages 174–191, Berlin, 2004. Springer-Verlag.
- [23] A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE J. on Selected Areas in Communications*, 21(1):5–19, 2003.
- [24] 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.
- [25] A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *Proc. of the IEEE Computer Security Foundations Workshop (CSFW-18)*, pages 255–269, Los Alamitos, Calif., 2005. IEEE Comp. Soc. Press.
- [26] D. A. Schmidt. Comparing completeness properties of static analyses and their logics. In N. Kobayashi, editor, *Proc. 2006 Asian Programming Languages and Systems Symposium (APLAS'06)*, volume 4279 of *Lecture Notes in Computer Science*, pages 183–199, Berlin, 2006. Springer-Verlag.
- [27] H. Unno, N. Kobayashi, and A. Yonezawa. Combining type-based analysis and model checking for finding counterexamples against non-interference. In *Proc. of the 2006 Workshop on Programming Languages and Analyses for Security (PLAS'06)*, pages 17–26, New York, NY, USA, 2006. ACM Press.
- [28] S. Zdancewic and A. C. Myers. Robust declassification. In *Proc. of the IEEE Computer Security Foundations Workshop*, pages 15–23, Los Alamitos, Calif., 2001. IEEE Comp. Soc. Press.

A Relevant background

Making abstract domain complete

The problem of making abstract domains \mathcal{B} -complete and \mathcal{F} -complete has been solved [12,11]. The key point in these constructions is that both \mathcal{F} and \mathcal{B} completeness are properties of the underlying abstract domain A relative to the concrete function f . To make a domain \mathcal{F} -complete, one adds all the direct images of f to the output abstract domain; to make a domain \mathcal{B} -complete, one adds all the maximal of the inverse images of the function to the input domain. (see Fig. A.1). In a more general setting let $f : C_1 \rightarrow C_2$ 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 any case the idea of making a domain complete is to add all the direct images of the concrete function to the output abstract domain for \mathcal{F} -completeness, and to add all the maximal of the inverse images of the function to the input domain for \mathcal{B} -completeness (see Fig. A.1). Formally, we refine the corresponding domains wrt., a generic function

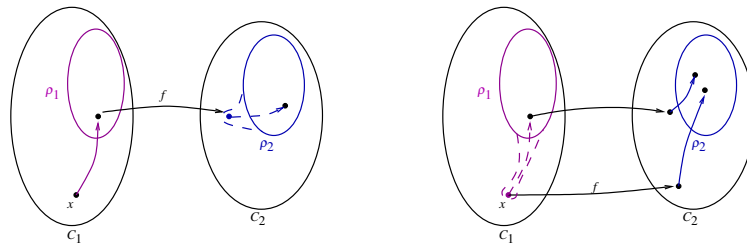


Fig. A.1. Making \mathcal{F} and \mathcal{B} complete.

$f : C_1 \longrightarrow C_2$ by using the following operations:

$$R_f^{\mathcal{F}} \stackrel{\text{def}}{=} \lambda X. \mathcal{M}(f(X)) \quad \Big| \quad R_f^{\mathcal{B}} \stackrel{\text{def}}{=} \lambda X. \mathcal{M}(\bigcup_{y \in X} \max(f^{-1}(\downarrow y)))$$

Let $\ell \in \{\mathcal{F}, \mathcal{B}\}$. In [12] the authors proved that the most abstract $\beta \sqsubseteq \eta$ such that $\langle \rho, \beta \rangle$ is ℓ -complete, i.e., given $\rho \in uco(C_2)$ the ℓ -complete shell of $\eta \in uco(C_1)$, is $\mathcal{R}_f^{\ell, P}(\eta) \stackrel{\text{def}}{=} \eta \sqcap R_f^\ell(\rho)$.

The Paige-Tarjan algorithm

The Paige Tarjan algorithm is a well known algorithm for computing the coarsest bisimulation of a given partition. Consider a relation R such that $f = \text{pre}(R)$. The algorithm is provided below, where P is a partition, $\text{PTSplit}_R(S, P)$ partitions each unstable block in P wrt. R with $B \cap f(S)$ and $B \setminus f(S)$, while $\text{PTRefiners}_R(P)$ is the set of all the blocks in P , or obtained as union of blocks in P , which make other blocks unstable.

$$\begin{aligned}
 & P : \text{Partition} \\
 \text{PTSplit}_R(S, P) : & \left\{ \begin{array}{l} \text{Partition obtained from } P \text{ by replacing} \\ \text{each block } B \in P \text{ with } B \cap f(S) \text{ and } B \setminus f(S) \end{array} \right. \\
 \text{PTRefiners}_R(P) \stackrel{\text{def}}{=} & \left\{ B \mid P \neq \text{PTSplit}_R(S, P) \wedge \exists \{B_i\}_i \subseteq P. S = \bigcup_i B_i \right\} \\
 \text{PT-Algorithm}_R : & \left\{ \begin{array}{l} \mathbf{while} (P \text{ is not } R\text{-stable}) \mathbf{do} \\ \quad \mathbf{choose} S \in \text{PTRefiners}_R(P); \\ \quad P := \text{PTSplit}_R(S, P); \\ \mathbf{endwhile} \end{array} \right.
 \end{aligned}$$

Fig. A.2. A generalized version of the PT algorithm.

This algorithm has been shown to be a forward completeness problem for the function f [20].