

# Abstract Non-Interference

## Parameterizing Non-Interference by Abstract Interpretation

Roberto Giacobazzi  
Dipartimento di Informatica  
Università di Verona  
Strada Le Grazie 15, I-37134 Verona, Italy  
roberto.giacobazzi@univr.it

Isabella Mastroeni  
Dipartimento di Informatica  
Università di Verona  
Strada Le Grazie 15, I-37134 Verona, Italy  
mastroeni@sci.univr.it

### Abstract

In this paper we generalize the notion of non-interference making it parametric relatively to what an attacker can analyze about the input/output information flow. The idea is to consider attackers as data-flow analyzers, whose task is to reveal properties of confidential resources by analyzing public ones. This means that no unauthorized flow of information is possible from confidential to public data, relatively to the degree of precision of an attacker. We prove that this notion can be fully specified in standard abstract interpretation framework, making the degree of security of a program a property of its semantics. This provides a comprehensive account of non-interference features for language-based security. We introduce systematic methods for extracting attackers from programs, providing domain-theoretic characterizations of the most precise attackers which cannot violate the security of a given program. These methods allow us both to compare attackers and program secrecy by comparing the corresponding abstractions in the lattice of abstract interpretations, and to design automatic program certification tools for language-based security by abstract interpretation.

**Categories and Subject Descriptors:** D.3.1 [Programming Languages]: Formal Definitions and Theory–*Semantics*; F.3.2 [Logics and Meanings of Programs]: Semantics of Programming Languages–*program analysis*; K.6.5 [Management of Computing and Information Systems]: Security and Protection.

**General Terms:** Languages, Security.

**Keywords:** Abstract interpretation, non-interference, language-based security, abstract domains.

### 1 Introduction

A typical problem in language-based security is protecting data confidentiality from erroneous or malicious attacks while data are processed by programs. The standard way to protect confidential

data is *access control* [30]: Higher privileges are required in order to access files containing confidential data. It is well known that access control checks do not put constraints on how the information is propagated. Once information is released from its container, the accessing program may, either by mistake or by purpose, improperly transmit the information in some form. In this case, in order to ensure that information can be used only according to specific security rules, also known as information flow policies, it is necessary to analyze how *information flows* in program's semantics. Goguen and Meseguer [21] firstly introduced *non-interference* as a key feature for specifying security policies:

*“One group of users [...] is noninterfering with another group of users if what the first group does [...] has no effect on what the second group of users can see”.*

Clearly, when computations on public data are non-interfering with those on private resources, no leakage of confidential information is possible by observing public input/output behavior. This principle is the pattern on which most security policies are specified in language-based security (see [30] for an excellent survey).

### The problem

The problem of refining security policies has been addressed by many authors as a major challenge in language-based information flow security [30]. Refining security policies means weakening standard non-interference checks in such a way that these restrictions can be used in practice. 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. The goal is to automatically generate, from security policies, a certificate specifying that the given program has only secure information flows. This is statically achieved to provide programs with their appropriate certificates. In this paper we address the problem of characterizing the degree of secrecy of a program relatively to what an attacker can analyze about the input/output information flow.

Consider the following program written in a simple imperative language, where the `while`-statement iterates until  $x_1$  is 0. Suppose  $x_1 : H$  is a secret variable and  $x_2 : L$  is a public variable:

**while**  $x_1$  **do** ( $x_2 := x_2 + 2$ ;  $x_1 := x_1 - 1$ )

Clearly, in the standard sense of non-interference, there is an implicit flow from  $x_2$  to  $x_1$ , since, due to the `while`-statement,  $x_2$  changes depending on the initial value of  $x_1$ . This represents the case where no restriction is considered on the power of an attacker. However, suppose that the attacker can observe only the parity of a

value (0 is even). It is worth noting that if  $x_2$  is initially even, then it is still even independently from the execution of the while, and therefore from the initial value of  $x_1$ . Similarly if  $x_2$  is initially odd then it remains odd independently from the execution of the while, i.e., from the value of  $x_1$ . This means that there's no information flow concerning parity. In order to model these situations we need to weaken standard non-interference relatively to what an attacker can observe on program information flows.

## The idea: Attackers as abstract interpretations

In this paper we introduce the notion of abstract non-interference by parameterizing standard non-interference relatively to what an attacker can observe. The idea is to consider attackers as static program analyzers whose task is to reveal properties of secret data by statically analyzing public resources. This idea allows us to introduce a notion of secrecy<sup>1</sup> relatively to the attacker's observation. Hence, a program ensures secrecy with respect to a given property, which can be statically analyzed by the attacker, if that property on confidential data cannot be disclosed by analyzing public data. For instance, in the example above, any attacker looking at parity is unable to disclose secrets from confidential data. In this sense the program is secret for parity, while it is not secret relatively to stronger attackers, able to observe more concrete properties of data such as how much a variable growth (e.g. by interval analysis [7]). Because static program analysis can be fully specified as the abstract interpretation of the semantics of the program [7], we can model *attackers as abstract interpretations*.

## Main contribution

In this paper we apply standard techniques from abstract interpretation in order to compare programs according to their relative degree of secrecy and in order to systematically extract attackers from programs by transforming abstractions. We first introduce the notion of abstract non-interference relatively to what an attacker can observe on the input/output of program's behavior. Abstract non-interference is obtained by a step-by-step weakening of standard Goguen and Meseguer's non-interference by making it parametric relatively input/output abstractions. These abstractions model the power of an attacker which analyzes the public input/output behavior of programs. The abstraction plays here the role of observable properties and proving that a program satisfies abstract non-interference relatively to a given abstraction means proving that relatively to that observable property, the program is secure. We characterize the abstract non-interference semantics of a deterministic program as an abstract interpretation of its maximal trace semantics in the corresponding transition system. Based on this observation we give a method for checking abstract non-interference by abstract interpretation of the concrete semantics of the program and we prove that checking abstract non-interference boils down into a standard static program analysis problem. Viewing attackers as abstract interpretations is the key point in order to both make secrecy parametric on what an attacker can observe, and to apply to secrecy the techniques known in abstract interpretation to systematically transform abstract domains. In this view, because attackers are modeled by abstractions, transforming abstract domains means transforming attackers. These can be either refined or simplified in power, following the standard methods in abstract domain design [16, 18]. The main result concerns the possibility of associating programs with attackers in the lattice of abstract interpreta-

---

<sup>1</sup>In the rest of this paper we abuse by considering secrecy and non-interference as synonyms.

tions. We give systematic methods for extracting, when possible, the most precise (viz. most concrete) attacker for which a program ensures secrecy. This is achieved by a fix-point construction which simplifies abstract domains towards the most concrete domain for which the program is secret as regards as the corresponding property. This "canonical" attacker represents, in the lattice of abstract interpretations, the relative degree of security of a program: Any other strictly stronger attacker will violate secrecy. Practical methods for deriving approximate secure attackers are discussed. The paper ends with a discussion on further research in this direction.

## Related work

There is a widespread literature on methods and techniques for checking *secure information flows* in software, ranging from standard data-flow/control-flow analysis techniques to type inference. All of these approaches are devoted to prove that a system as a whole, or parts of it, does not allow confidential data to flow towards public variables. Type-based approaches are designed in such a way that well-typed programs do not leak secrets. In a security-typed language, a type is inductively associated at compile-time with program statements in such a way that any statement showing a potential flow disclosing secrets is rejected [33, 36, 39]. Similarly, data-flow/control-flow analysis techniques are devoted to statically discover flows of secret data into public variables [3, 4, 23, 25, 31]. All these approaches are characterized by the way they model attackers (or unauthorized users). In standard non-interference, the attacker is able to fully analyze concrete computations. In this case, a conservative type/data-flow/control-flow analysis of information flows would discard all the programs which may provide any explicit or implicit concrete flow from confidential to public resources. The problem of refining security policies has been recently addressed by many authors (see Sec. IV-D in [30]). The most related papers are [25, 26, 15, 35]. In these papers some restrictions have been considered on the power of attackers, basically related with complexity issues [25, 35] or with precise quantitative estimation of unauthorized information leakage [26, 15]. While in the second case programs may have an absolute degree of security, corresponding to whether they leak or not secrets under some metric, in the first case, different degrees of security can be associated with programs, e.g. allowing us to accept programs for which leaking secrets would be too complex for any attacker. A lattice-theoretic and property-driven reasoning about the information that violates non-interference is in [24, 40]. In [24] the authors introduce a lattice-theoretic model of information flow based on equivalence relations. This provides necessary and sufficient conditions for standard non-interference of a state machine. In [40] the authors introduce a method for authorizing declassification when confidential information is released, for both passive and active attackers. This approach, even though different, has a strong connection with our work. Robust declassification can be viewed as the dual problem of characterizing the program property (in our case an abstract domain) which violates non-interference. We will show later in Section 6 that this notion can be adapted to abstract non-interference providing a parametric characterization for downgradable information. In [31] the authors use partial equivalence relations (PERs) to model dependencies and therefore information flows. We believe that the PER model can be easily adapted to cope with the weaker notion of abstract non-interference. The use of abstract interpretation in language-based security is not new, even though to the best of our knowledge, no author used the lattice of abstract interpretations for evaluating the degree of secrecy of programs. Abstract interpretation has been basically considered for designing flexible data/control flow analyzers [28] or enhanced security-type systems [39].

## 2 Preliminaries

### 2.1 Basic notions

Sets are usually denoted with capital letters. If  $S$  and  $T$  are sets, then  $\wp(S)$  denotes the powerset of  $S$ ,  $S \setminus T$  denotes the set-difference between  $S$  and  $T$ ,  $S \subset T$  denotes strict inclusion, and for a function  $f : S \rightarrow T$  and  $X \subseteq S$ ,  $f(X) \stackrel{\text{def}}{=} \{f(x) \mid x \in X\}$ . We will often denote  $f(\{x\})$  as  $f(x)$ . By  $g \circ f$  we denote the composition of the functions  $f$  and  $g$ , i.e.,  $g \circ f \stackrel{\text{def}}{=} \lambda x. g(f(x))$ .  $id \stackrel{\text{def}}{=} \lambda x. x$ .  $\langle P, \leq \rangle$  denotes a poset  $P$  with ordering relation  $\leq$ , while  $\langle P, \leq, \vee, \wedge, \top, \perp \rangle$  denotes a complete lattice  $P$ , with ordering  $\leq$ ,  $lub \vee$ ,  $glb \wedge$ , greatest element (top)  $\top$ , and least element (bottom)  $\perp$ . Often,  $\leq_P$  will be used to denote the underlying ordering of a poset  $P$ , and  $\vee_P, \wedge_P, \top_P$  and  $\perp_P$  denote the basic operations and elements if  $P$  is a complete lattice.  $C \cong A$  denotes that  $C$  and  $A$  are isomorphic ordered structures. The downward closure of  $S$  is defined as  $\downarrow S \stackrel{\text{def}}{=} \{x \in P \mid \exists y \in S. x \leq_P y\}$ , and for  $x \in P$ ,  $\downarrow x$  is a shorthand for  $\downarrow \{x\}$ , while the upward closure  $\uparrow$  is dually defined.  $S \rightarrow T$  denotes the set of all functions from  $S$  to  $T$ . We use the symbol  $\sqsubseteq$  to denote point-wise ordering between functions: If  $S$  is any set,  $P$  a poset, and  $f, g : S \rightarrow P$  then  $f \sqsubseteq g$  if for all  $x \in S$ ,  $f(x) \leq_P g(x)$ . Let  $C$  and  $A$  be complete lattices, then,  $C \xrightarrow{m} A$  and  $C \xrightarrow{c} A$ , denote, respectively, the set of all monotone and (Scott-)continuous functions from  $C$  to  $A$ . Recall that  $f \in C \xrightarrow{c} A$  iff  $f$  preserves  $lub$ 's of (nonempty) chains iff  $f$  preserves  $lub$ 's of directed subsets, and  $f : C \rightarrow A$  is (completely) additive if  $f$  preserves  $lub$ 's of all subsets of  $C$  (emptyset included). We denote by  $lfp_{\perp}^{\leq} f$  and  $gfp_{\top}^{\leq} f$ , respectively, the least and greatest fix-point, when they exist, of an operator  $f$  on a poset. If  $f \in C \xrightarrow{c} C$  then  $lfp_{\perp}^{\leq} f = \bigvee_{i \in \mathbb{N}} f^i(\perp_C)$ , where, for any  $i \in \mathbb{N}$  and  $x \in C$ , the  $i$ -th power of  $f$  in  $x$  is inductively defined as follows:  $f^0(x) = x$ ;  $f^{i+1}(x) = f(f^i(x))$ . Dually, if  $f$  is co-continuous then  $gfp_{\top}^{\leq} f = \bigwedge_{i \in \mathbb{N}} f^i(\top_C)$ .  $\{f^i(\perp_C)\}_{i \in \mathbb{N}}$  and  $\{f^i(\top_C)\}_{i \in \mathbb{N}}$  are, respectively, the *upper* and *lower iteration sequences* of  $f$  (see [8]).

### 2.2 Abstract interpretation

Abstract domains can be equivalently formulated either in terms of Galois connections or closure operators [9]. An *upper closure operator* on a poset  $P$  is an operator  $\rho : P \rightarrow P$  monotone, idempotent and extensive ( $\forall x \in P. x \leq_P \rho(x)$ ). *Lower closure operators* are defined dually. The set of all upper (lower) closure operators on  $P$  is denoted by  $uco(P)$  ( $lco(C)$ ). Let  $\langle C, \leq, \vee, \wedge, \top, \perp \rangle$  be a complete lattice. A basic property of closure operators is that they 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 an upper closure on  $C$  iff  $X$  is a *Moore-family* of  $C$ , i.e.,  $X = \mathcal{M}(X) \stackrel{\text{def}}{=} \{\wedge S \mid S \subseteq X\}$  — where  $\wedge \emptyset = \top \in \mathcal{M}(X)$ . For any  $X \subseteq C$ ,  $\mathcal{M}(X)$  is called the *Moore-closure* of  $X$  in  $C$ , i.e.,  $\mathcal{M}(X)$  is the least (w.r.t. set-inclusion) subset of  $C$  which contains  $X$  and it is a Moore-family of  $C$ . It turns out that  $\langle \rho(C), \leq \rangle$  is a complete meet sub-semilattice of  $C$  (i.e.,  $\wedge$  is its *glb*), but, in general, it is not a complete sub-lattice of  $C$ , since the  $lub$  in  $\rho(C)$  — defined by  $\lambda Y \subseteq \rho(C). \rho(\bigvee Y)$  — might be different from that in  $C$ . Indeed,  $\rho(C)$  is a complete sub-lattice of  $C$  iff  $\rho$  is additive. Often, we will find particularly convenient to identify closure operators with their sets of fix-points. If  $C$  is a complete lattice then  $uco(C)$  ordered point-wise is also a complete lattice, denoted by  $\langle uco(C), \sqsubseteq, \sqcup, \sqcap, \lambda x. \top, \lambda x. x \rangle$ , 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  $\eta(C) \subseteq \rho(C)$ ;  $(\sqcap_{i \in I} \rho_i)(x) = \bigwedge_{i \in I} \rho_i(x)$ ; and  $(\sqcup_{i \in I} \rho_i)(x) = x \Leftrightarrow \forall i \in I. \rho_i(x) = x$ . The standard abstract interpretation framework is based on the adjoint relation between abstrac-

tion and concretization functions [7]. If  $\alpha : C \xrightarrow{m} A$  and  $\gamma : A \xrightarrow{m} C$  are monotone functions such that  $\lambda x. x \sqsubseteq \gamma \circ \alpha$  and  $\alpha \circ \gamma \sqsubseteq \lambda x. x$ , then  $(A, \alpha, \gamma, C)$  is called a *Galois connection* (GC) between  $C$  and  $A$ . If in addition  $\alpha \circ \gamma = \lambda x. x$ , then  $(A, \alpha, \gamma, C)$  is a *Galois insertion* (GI) of  $A$  in  $C$ . Note that  $A \cong C$  iff both  $(A, \alpha, \gamma, C)$  and  $(C, \gamma, \alpha, A)$  are GI. The concrete and abstract domains,  $C$  and  $A$ , are assumed to be complete lattices and are related by abstraction  $\alpha$  and concretization  $\gamma$  forming a GC  $(A, \alpha, \gamma, C)$ . Following a standard terminology,  $A$  is called an abstraction of  $C$ , and  $C$  is a concretization of  $A$ . If  $(A, \alpha, \gamma, C)$  is a GI, then each value of the abstract domain  $A$  is useful in representing  $C$ , because all the elements of  $A$  represent distinct members of  $C$ , being  $\gamma$  1-1. Any GC may be lifted to a GI by identifying in an equivalence class those values of the abstract domain with the same concretization. This process is known as *reduction* of the abstract domain. If  $(A, \alpha, \gamma, C)$  is a GI and  $f_C \in C \xrightarrow{c} C$ ,  $f_A \in A \xrightarrow{c} A$ , then  $f_A$  is a *sound approximation* of  $f_C$  if  $\alpha \circ f_C \leq_A f_A \circ \alpha$  or equivalently  $\alpha \circ f_C \circ \gamma \leq_A f_A$ .  $f_A^{bca} \stackrel{\text{def}}{=} \alpha \circ f_C \circ \gamma$  is known as the *best correct approximation* (bca) of  $f_C$  in  $A$ . Soundness naturally implies that  $\alpha(lfp_{\perp}^{\leq} f_C) \leq_A lfp_{\perp}^{\leq} f_A$ . If  $\alpha \circ f_C = f_A \circ \alpha$  then we say that  $f_A$  is a *complete approximation* of  $f_C$  [9, 20]. In this case  $\alpha(lfp_{\perp}^{\leq} f_C) = lfp_{\perp}^{\leq} f_A$ .

Note that any Galois insertion  $(A, \alpha, \gamma, C)$  uniquely determines an upper closure operator  $\gamma \circ \alpha \in uco(C)$  and conversely, any closure operator  $\rho \in uco(C)$  uniquely determines a GI  $(\rho(C), \rho, id, C)$ , up to isomorphic representation of domain's objects. Hence, we will identify  $uco(C)$  with the so-called *lattice*  $\mathcal{L}_C$  of *abstract interpretations* of  $C$  (cf. [7, Section 7] and [9, Section 8]), i.e., the complete lattice of all possible abstract domains (modulo isomorphic representation of their objects) of the concrete domain  $C$ . The point-wise ordering on  $uco(C)$  corresponds precisely to the standard ordering used to compare abstract domains with regard to their precision:  $A_1$  is more precise than  $A_2$  (i.e.,  $A_2$  is an abstraction of  $A_1$ ) iff  $A_1 \sqsubseteq A_2$  in  $uco(C)$  iff  $(A_2, \alpha, \gamma, A_1)$  is a GI. Let  $\{A_i\}_{i \in I} \subseteq uco(C)$ :  $\sqcup_{i \in I} A_i$  is the most concrete among the domains in  $\mathcal{L}_C$  which are abstractions of all  $A_i$ 's, i.e.,  $\sqcup_{i \in I} A_i$  is the *least* (w.r.t.  $\sqsubseteq$ ) *common abstraction* of all  $A_i$ 's; and  $\sqcap_{i \in I} A_i$  is (isomorphic to) the well-known *reduced product* (basically cartesian product plus reduction) of all  $A_i$ 's, or, equivalently, it is the most abstract among the domains in  $\mathcal{L}_C$  which are more concrete than every  $A_i$ :  $\sqcap_{i \in I} A_i = \mathcal{M}(\bigcup_{i \in I} A_i)$ . The *disjunctive completion* of a domain is the most abstract domain able to represent the concrete disjunction of its objects:  $\Upsilon(\rho) = \sqcup \{\eta \in uco(L) \mid \eta \sqsubseteq \rho \text{ and } \eta \text{ is additive}\}$ .  $\rho$  is disjunctive iff  $\Upsilon(\rho) = \rho$  (cf. [9, 19]).

### 2.3 The deterministic language

In the following we consider a simple imperative language, IMP [38], where programs are commands with the following syntax:

$$c ::= \mathbf{nil} \mid x := e \mid c; c \mid \mathbf{while} \ x \ \mathbf{do} \ (c)$$

with  $e$  denoting expressions evaluated in the set of values  $\mathbb{V}$  with standard operations, i.e., if  $\mathbb{V} = \mathbb{N}$  then  $e$  can be any arithmetical expression. As usual,  $\mathbb{V}$  can be structured as a flat domains whose bottom element,  $\perp$ , denotes the value of not initialized variables. In the following we will denote by  $Var(P)$  the set of variables of the program  $P \in \text{IMP}$ . Let's consider the well-known operational semantics of IMP in Table 1 [38]. The operational semantics naturally induces a transition relation on a set of states  $\Sigma$ , denoted  $\rightarrow$ , specifying the relation between a state and its possible successors.  $\langle \Sigma, \rightarrow \rangle$  is a transition system. In our case, if  $|Var(P)| = n$  then  $\Sigma$  is a set of  $n$ -tuples of values, i.e.,  $\Sigma = \mathbb{V}^n$ . We follow Cousot's construction [6, 10], defining semantics, at different levels of abstractions, as the abstract interpretation of the maximal trace semantics of a transition system associated with each well-formed program. In the

$\langle \text{nil}, s \rangle \longrightarrow s$	$\frac{\langle e, s \rangle \longrightarrow n \in \mathbb{V}_x}{\langle x := e, s \rangle \longrightarrow s[n/x]}$	$\frac{\langle c_0, s \rangle \longrightarrow s_0, \langle c_1, s_0 \rangle \longrightarrow s_1}{\langle c_0; c_1, s \rangle \longrightarrow s_1}$
$\frac{\langle x, s \rangle \longrightarrow 1, \langle c, s \rangle \longrightarrow s_0, \langle \text{while } x \text{ do } (c), s_0 \rangle \longrightarrow s_1}{\langle \text{while } x \text{ do } (c), s \rangle \longrightarrow s_1}$		
$\frac{\langle x, s \rangle \longrightarrow 0}{\langle \text{while } x \text{ do } (c), s \rangle \longrightarrow s}$		

**Table 1. Operational semantics of IMP**

following,  $\Sigma^+$  and  $\Sigma^\omega \stackrel{\text{def}}{=} \mathbb{N} \longrightarrow \Sigma$  denote respectively the set of finite nonempty and infinite sequences of symbols in  $\Sigma$ . Given a sequence  $\sigma \in \Sigma^\infty \stackrel{\text{def}}{=} \Sigma^+ \cup \Sigma^\omega$ , its length is denoted  $|\sigma| \in \mathbb{N} \cup \{\omega\}$  and its  $i$ -th element is denoted  $\sigma_i$ . A non-empty finite (infinite) *trace*  $\sigma \in \Sigma^\infty$  is a finite (infinite) sequence of program states where two consecutive elements are in the transition relation  $\rightarrow$ , i.e., for all  $i < |\sigma|$ :  $\sigma_i \rightarrow \sigma_{i+1}$ . The *maximal trace semantics* [10] of a transition system associated with a program  $P$  is  $\llbracket P \rrbracket^\infty \stackrel{\text{def}}{=} \llbracket P \rrbracket^+ \cup \llbracket P \rrbracket^\omega$ , where if  $T \subseteq \Sigma$  is a set of final/blocking states then  $\llbracket P \rrbracket^{\omega, T} = \{\sigma \in \Sigma^+ \mid |\sigma| = n, \forall i \in [1, n]. \sigma_{i-1} \rightarrow \sigma_i\}$ ,  $\llbracket P \rrbracket^\omega = \{\sigma \in \Sigma^\omega \mid \forall i \in \mathbb{N}. \sigma_i \rightarrow \sigma_{i+1}\}$ ,  $\llbracket P \rrbracket^+ = \bigcup_{n>0} \{\sigma \in \llbracket P \rrbracket^{\omega, T} \mid \sigma_{n-1} \in T\}$ , and  $\llbracket P \rrbracket^n = \llbracket P \rrbracket^{\omega, T} \cap \llbracket P \rrbracket^+$ . If  $\sigma \in \llbracket P \rrbracket^+$ , then  $\sigma_{-1}$  and  $\sigma_{-}$  denote respectively the final and initial state of  $\sigma$ . The semantics  $\llbracket P \rrbracket^\infty$  has been obtained in [10] as a fix-point of the monotone operator  $F_P^\infty : \wp(\Sigma^\infty) \rightarrow \wp(\Sigma^\infty)$  defined on traces as  $F_P^\infty(X) = \llbracket P \rrbracket^+ \cup \llbracket P \rrbracket^{\omega, T} \frown X$ , where  $\frown$  is sequence concatenation. This operator provides a bi-induction (induction and co-induction) on the complete lattice of the maximal trace semantics  $(\wp(\Sigma^\infty), \sqsubseteq^\infty, \sqcap^\infty, \sqcup^\infty, \sqsupset^\infty, \Sigma^+, \Sigma^\omega)$ , where  $X \sqsubseteq^\infty Y$  if and only if  $X \cap \Sigma^+ \subseteq Y \cap \Sigma^+$  and  $Y \cap \Sigma^\omega \subseteq X \cap \Sigma^\omega$ . In this case:  $\llbracket P \rrbracket^\infty = \text{lfp}_{\Sigma^\infty} F_P^\infty$  (see [6, 10] for details). The *angelic denotational semantics* associates input/output functions with programs, by ignoring non-termination. This semantics is derived in [6] by abstract interpretation from the maximal trace semantics with abstraction  $\alpha^D(X) \stackrel{\text{def}}{=} \lambda s \in \Sigma. \{\sigma_{-1} \mid \sigma \in X \cap \Sigma^+ \wedge s = \sigma_{-}\}$ , such that  $(\langle \Sigma \rightarrow \wp(\Sigma), \sqsubseteq \rangle, \alpha^D, \gamma^D, \langle \wp(\Sigma^\infty), \sqsubseteq \rangle)$  is a GI, for some  $\gamma^D$ . Note that, since our programs are deterministic,  $\alpha^D(X)(s)$  is always a singleton. It is well known that we can associate, inductively on its syntax, with each program  $P \in \text{IMP}$  a function  $\llbracket P \rrbracket$  denoting its input/output relation, such that  $\llbracket P \rrbracket \stackrel{\text{def}}{=} \alpha^D(\llbracket P \rrbracket^+)$ .

### 3 Abstract non-interference

A typical problem in language-based security is checking non-interference. If a user wishes to keep some data confidential, he or she might state a policy stipulating that no data visible to other users is affected by modifying confidential data. This policy allows programs to manipulate and modify private data, as long as visible outputs of those programs do not reveal information about the data. A policy of this sort is a non-interference policy [21], since it states that confidential data may not interfere with public data. Non-interference is also referred as *secrecy* [34], since 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) [14]. Secrecy is usually formulated saying that the *final* value of a low variable does not depend on the *initial* value of high-variables [34]. An attacker (or unauthorized user) is assumed to be allowed to view only information that is not secret. The usual method for showing that secrecy holds is to verify that the attacker cannot ob-

serve *any* difference between two executions that differ only in their secret input [22, 34]. In literature, when a variation of secret input does not cause a variation of public output, we say that the program has only *secure information flows* [1, 2, 5, 12, 13, 14, 23, 34]. Let  $\llbracket P \rrbracket^+$  be the set of terminating traces  $\sigma$  of values for the variables in a program  $P$ . In order to analyze the variables of a program as regards the given set of security classes  $\{H, L\}$ , we consider a typing function  $t \in \text{Var} \longrightarrow \{H, L\}$ , which associates with each variable in a program its security class. In the following, if  $x \in \text{Var}(P)$  then we denote  $x : t(x)$  the corresponding security typing. Moreover, if  $\sigma \in \llbracket P \rrbracket^+$ , we consider  $\sigma^H$  and  $\sigma^L$  as the projections of  $\sigma$  on respectively the values of H and L variables and we assume that any state  $s \in \Sigma$  is such that H values come first. In this case, *standard non-interference* can be formulated as follows: A program  $P$  is *secure* if  $\forall \sigma, \delta \in \llbracket P \rrbracket^+. \sigma_{-}^L = \delta_{-}^L \Rightarrow \sigma_{-}^H = \delta_{-}^H$ . We can reformulate the definition above by using the denotational semantics of  $P$ . In the following, if  $T \in \{H, L\}$ ,  $v \in \mathbb{V}^n$ , and  $n = |\{x \in \text{Var}(P) \mid t(x) = T\}|$ , we abuse notation by denoting  $v \in \mathbb{V}^T$  the fact that  $v$  is a possible value for the vector of variables with security type T.

A program  $P$  is *secure* if  
 $\forall v \in \mathbb{V}^L, \forall v_1, v_2 \in \mathbb{V}^H. (\llbracket P \rrbracket(v_1, v))^L = (\llbracket P \rrbracket(v_2, v))^L$

In order to model secrecy relatively to some fixed observable property, we assume that any program variable can preserve secrecy only as regards a particular amount of information, the one that the attacker can observe. We introduce a weaker notion of information flow, which models an attacker that can observe only some properties of public (i.e., L) concrete values. As usual in abstract interpretation, a property is an *upper closure operator* on the concrete domain of computation [9]. We distinguish between the attacker's observational capability on the inputs and on the outputs of programs, by introducing two distinct properties for respectively input and output on L-variables:  $\eta, \rho \in \text{uco}(\wp(\mathbb{V}^L))$ . This leads us to the first generalization of standard non-interference, called *narrow (abstract) non-interference*. The idea is that a program satisfies narrow abstract non-interference relatively to a typing on security classes and a pair of closures  $\eta$  and  $\rho$ , denoted  $\langle \eta, \rho \rangle$ -*NSecrecy*, if whenever the L input values have the same property  $\eta$  then the L output values have the same  $\rho$  property. This captures precisely the intuition that  $\eta$ -indistinguishable input values provide  $\rho$ -indistinguishable results. The following definition introduces the notion of narrow abstract non-interference as a generalization of the standard one given by Goguen and Meseguer.

**DEFINITION 3.1.** *Let  $\eta, \rho \in \text{uco}(\wp(\mathbb{V}^L))$ . A program  $P \in \text{IMP}$  is  $\langle \eta, \rho \rangle$ -NSecret if*

$$\forall h_1, h_2 \in \mathbb{V}^H, \forall l_1, l_2 \in \mathbb{V}^L. \eta(l_1) = \eta(l_2) \Rightarrow \rho(\llbracket P \rrbracket(h_1, l_1)^L) = \rho(\llbracket P \rrbracket(h_2, l_2)^L).$$

We write  $[\eta]P(\rho)$  to say that a program  $P$  is  $\langle \eta, \rho \rangle$ -NSecret. In this case, if  $[\eta]P(\rho)$  does not hold, written  $P \not\models [\eta]P(\rho)$ , then the attacker may observe an interference due to confidential data-flow.

EXAMPLE 3.2. Consider the property *Sign* and *Par* represented in Figure 1 and the program:

$$P \stackrel{\text{def}}{=} l := 2 * l * h^2;$$

with security typing:  $t = \langle h : \mathbb{H}, l : \mathbb{L} \rangle$  and  $\mathbb{V} = \mathbb{Z}$ . Clearly in the stan-

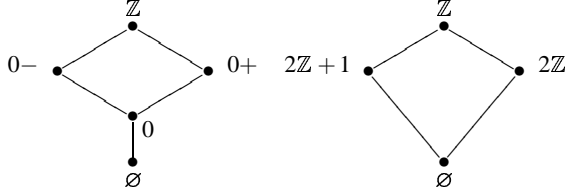


Figure 1. The *Sign* and *Par* domains.

ard notion of secrecy there is a flow of information from variable  $h$  to variable  $l$ , since  $l$  depends on the value of  $h$ , i.e., the statement is not secure. Let's see what happens for the  $\langle \text{Sign}, \text{Par} \rangle$ -NSecrecy. Clearly we have that if the input is such that  $\text{Sign}(l) = 0+$ , then the possible outputs, depending on  $h$ , are always in  $2\mathbb{Z}$ . The same holds if  $\text{Sign}(l) = 0-$ . Therefore any possible output value, with a fixed input  $l$ , has the same abstraction in *Par*, which is  $2\mathbb{Z}$ . Hence we have that  $[\text{Sign}]P(\text{Par})$  holds.

It is worth noting that  $P \not\models [\eta]P(\rho)$  does not necessarily imply an information flow from  $\mathbb{H}$  to  $\mathbb{L}$  values. In this case it is possible a deceptive flow due to the  $\eta$ -undistinguished public values. This means that there may be no information flow from  $\mathbb{H}$  to  $\mathbb{L}$  as regards as the properties  $\eta$  for the input and  $\rho$  for the output. Clearly the more  $\eta$  is precise, the less deceptive flows may appear.

EXAMPLE 3.3. Consider the property *Sign* and *Par* represented in Figure 1 and the program in Example 3.2. Let us consider  $\langle \text{Par}, \text{Sign} \rangle$ -NSecrecy. In this case note that  $\text{Par}(-2) = \text{Par}(4) = 2\mathbb{Z}$ , but  $\text{Sign}(\llbracket P \rrbracket(h, -2)^{\mathbb{L}}) = 0- \neq 0+ = \text{Sign}(\llbracket P \rrbracket(h, 4)^{\mathbb{L}})$ . This means that  $[\text{Par}]P(\text{Sign})$  doesn't hold due to a deceptive flow generated by a variation of low inputs having the same property in *Sign*.

In order to avoid the presence of deceptive flows we consider the possibility of passing abstract  $\mathbb{L}$  values as input for program's semantics. The idea is to model only information flows generated by the variation of  $\mathbb{H}$  values. This is obtained by computing the concrete semantics with abstract values for  $\mathbb{L}$  inputs, denoting  $\eta$ -undistinguished data. The same idea can be applied to confidential data, in order to model properties of confidential resources that does not flow into public variables. In this case we model when the change of a particular property in the private input has effects in what the attacker can observe concerning the properties of public output. We introduce a weaker notion of non-interference having no deceptive flows, such that, when the attacker is able to observe the property  $\eta$  of public input and the property  $\rho$  of public output, then no information flow concerning the property  $\phi$  of private input interferes in the observable property  $\rho$  of the public output, under the assumption that the public input property  $\eta$  doesn't change. In this case the abstraction  $\phi$  represents the property of private data that may flow into the public variables, given an attacker that can observe  $\eta$  on public input and  $\rho$  on public output. We call this notion *abstract non-interference*, denoted  $\langle \eta, \phi, \rho \rangle$ -Secrecy, as regards as the closures  $\eta, \rho \in \text{uco}(\wp(\mathbb{V}^{\mathbb{L}}))$ , and  $\phi \in \text{uco}(\wp(\mathbb{V}^{\mathbb{H}}))$ .

DEFINITION 3.4. Let  $\eta, \rho \in \text{uco}(\wp(\mathbb{V}^{\mathbb{L}}))$  and  $\phi \in \text{uco}(\wp(\mathbb{V}^{\mathbb{H}}))$ .  $P \in \text{IMP}$  is  $\langle \eta, \phi, \rho \rangle$ -Secret if

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

In the following when a program  $P$  is  $\langle \eta, \phi, \rho \rangle$ -Secrecy we will write  $(\eta)P(\phi \rightsquigarrow \rho)$ . Next example shows the difference between narrow and abstract non-interference.

EXAMPLE 3.5. Consider the property *Sign* and *Par* represented in Figure 1 and the program in Example 3.2. Consider  $\langle \text{Par}, \text{id}, \text{Sign} \rangle$ -Secrecy. Then  $\text{Sign}(\llbracket P \rrbracket(h, \text{Par}(-2))^{\mathbb{L}}) = \text{Sign}(\llbracket P \rrbracket(h, \text{Par}(4))^{\mathbb{L}}) = \text{Sign}(2\mathbb{Z} * h^2) = \mathbb{Z}$ . In general we can prove that  $(\text{Par})P(\text{id} \rightsquigarrow [\text{Sign}])$  holds. Hence no more deceptive flows can be revealed.

It is clear that standard secrecy [34] based on Goguen and Meseguer's non-interference is  $\langle \text{id}, \text{id} \rangle$ -NSecrecy, which is equivalent to  $\langle \text{id}, \text{id}, \text{id} \rangle$ -Secrecy, i.e.,  $[\text{id}]P(\text{id}) = (\text{id})P(\text{id} \rightsquigarrow \text{id})$ . More in general it is straightforward to prove the following implications for any  $P \in \text{IMP}$ , where the second implication is consequence of deceptive flows in  $[\eta]P(\rho)$ .

$$\boxed{[\text{id}]P(\text{id}) \Rightarrow [\eta]P(\rho) \Rightarrow (\eta)P(\text{id} \rightsquigarrow \rho) \Rightarrow (\eta)P(\phi \rightsquigarrow \rho)}$$

EXAMPLE 3.6. Consider the properties *Sign* and *Par* described above and the program fragment:

$$P \stackrel{\text{def}}{=} l := l * h^2;$$

with security typing:  $t = \langle h : \mathbb{H}, l : \mathbb{L} \rangle$  and  $\mathbb{V} = \mathbb{Z}$ . Consider  $\langle \text{id}, \text{id}, \text{Par} \rangle$ -Secrecy. Note that  $\text{Par}(\llbracket P \rrbracket(2, 1)^{\mathbb{L}}) = \text{Par}(4) = 2\mathbb{Z}$  while  $\text{Par}(\llbracket P \rrbracket(3, 1)^{\mathbb{L}}) = \text{Par}(9) = 2\mathbb{Z} + 1$ , which are clearly different, therefore in this case  $(\text{id})P(\text{id} \rightsquigarrow \text{Par})$  doesn't hold. On the other hand consider  $\langle \text{id}, \text{Sign}, \text{Par} \rangle$ -Secrecy. Note that  $\text{Par}(\llbracket P \rrbracket(\text{Sign}(2), 1)^{\mathbb{L}}) = \text{Par}(\llbracket P \rrbracket(\text{Sign}(3), 1)^{\mathbb{L}}) = \text{Par}(0+) = \mathbb{Z}$ . In this case it is simple to check that  $(\text{id})P(\text{Sign} \rightsquigarrow \text{Par})$  holds.

Abstract non-interference is parametric on program properties specified as closure operators. In order to better understand the meaning of input/output abstractions in the definitions above, we observe that the property  $(\eta)P(\phi \rightsquigarrow \{\top\})$  always holds. Indeed if a closure identifies some object, then every more abstract closure will identify at least the same objects. From these simple observations we derive the following basic properties of narrow and abstract non-interference.

PROPOSITION 3.7. Let  $\eta, \rho \in \text{uco}(\wp(\mathbb{V}^{\mathbb{L}}))$ ,  $\phi \in \text{uco}(\wp(\mathbb{V}^{\mathbb{H}}))$ , and  $P \in \text{IMP}$ .

1.  $[\eta]P(\rho) \Leftrightarrow \forall \beta \sqsupseteq \rho. [\eta]P(\beta)$ ;
2.  $[\eta]P(\rho) \Leftrightarrow \forall \beta \sqsubseteq \eta. [\beta]P(\rho)$ ;
3.  $\forall i. [\eta]P(\rho_i) \Rightarrow [\eta]P(\sqcap_i \rho_i)$  and  $[\eta]P(\sqcup_i \rho_i)$ ;
4.  $[\text{id}]P(\rho) \Leftrightarrow (\text{id})P(\text{id} \rightsquigarrow \rho)$ ;
5.  $(\eta)P(\phi \rightsquigarrow \rho) \Rightarrow \forall \beta \sqsupseteq \rho. (\eta)P(\phi \rightsquigarrow \beta)$ ;
6.  $\forall i. (\eta)P(\phi \rightsquigarrow \rho_i) \Rightarrow (\eta)P(\phi \rightsquigarrow \sqcap_i \rho_i)$  and  $(\eta)P(\phi \rightsquigarrow \sqcup_i \rho_i)$ .

## 4 Checking abstract non-interference

In this section we derive the abstract non-interference semantics of a programming language by abstract interpretation of its the maximal trace semantics. The abstract non-interference semantics of a program is the set of all the denotations, viz. functions, representing computations for  $P$  that satisfy  $(\eta)P(\phi \rightsquigarrow \rho)$ . Formally we define the domain  $\mathbb{S}(\eta, \phi \rightsquigarrow \rho)$  parametric on the abstract domains  $\eta, \rho \in uco(\wp(\mathbb{V}^L))$  and  $\phi \in uco(\wp(\mathbb{V}^H))$ . This domain is an abstraction of the concrete domain of the angelic denotational semantics  $\Sigma \rightarrow \wp(\Sigma)$  as introduced in [6], where  $\Sigma = \mathbb{V}^n$ .

$$\mathbb{S}(\eta, \phi \rightsquigarrow \rho) \stackrel{\text{def}}{=} \left\{ f \in \Sigma \rightarrow \wp(\Sigma) \left| \begin{array}{l} \forall h_1, h_2 \in \mathbb{V}^H, \forall l_1, l_2 \in \mathbb{V}^L. \\ \eta(l_1) = \eta(l_2) \Rightarrow \\ \rho(f(\phi(h_1), \eta(l_1)))^L = \\ \rho(f(\phi(h_2), \eta(l_2)))^L \end{array} \right. \right\}$$

The key point in order to show that  $\mathbb{S}(\eta, \phi \rightsquigarrow \rho)$  is an abstraction of  $\alpha^{\mathcal{D}}(\wp(\Sigma^\infty))$ , is to note that if  $f$  doesn't satisfy *abstract non-interference*, i.e.,  $f \notin \mathbb{S}(\eta, \phi \rightsquigarrow \rho)$ , then there is no way to make it satisfy that property by adding traces, i.e.,  $\forall g \sqsupseteq f. g \notin \mathbb{S}(\eta, \phi \rightsquigarrow \rho)$ . On the other hand if  $f \in \mathbb{S}(\eta, \phi \rightsquigarrow \rho)$ , then  $\forall g \sqsubseteq f. g \in \mathbb{S}(\eta, \phi \rightsquigarrow \rho)$ .

**PROPOSITION 4.1.** *Let  $\eta, \rho \in uco(\wp(\mathbb{V}^L))$  and  $\phi \in uco(\wp(\mathbb{V}^H))$ .*

1.  $\mathbb{S}(\eta, \phi \rightsquigarrow \rho)$  is a Moore-family;
2.  $(\mathbb{S}(\eta, \phi \rightsquigarrow \rho), \alpha, \text{id}, \alpha^{\mathcal{D}}(\wp(\Sigma^\infty)))$  is a GI where if we consider  $f \in \alpha^{\mathcal{D}}(\wp(\Sigma^\infty))$  then

$$\alpha(f) = \begin{cases} \lambda x. \Sigma & \text{if } \exists \delta_+, \sigma_- \in \Sigma. \eta(\sigma_-^L) = \eta(\delta_+^L) \wedge \\ & \rho(f(\phi(\sigma_-^H), \eta(\sigma_-^L)))^L \neq \\ & \rho(f(\phi(\delta_+^H), \eta(\delta_+^L)))^L \\ f & \text{otherwise} \end{cases}$$

An analogous result holds in the narrow case. Therefore, checking for  $(\eta)P(\phi \rightsquigarrow \rho)$  means checking whether  $\alpha(\llbracket P \rrbracket) = \llbracket P \rrbracket$ . This check boils down to a standard static program analysis. Let  $P \in \text{IMP}$ ,  $\eta, \rho \in uco(\wp(\mathbb{V}^L))$  and  $\phi \in uco(\wp(\mathbb{V}^H))$  be the observable properties that characterize the attacker. We observe that it is possible to monitor the possible flows of information from variables of type H to variables of type L, by considering the best correct approximation of  $\llbracket P \rrbracket$  given by  $\rho, \eta$ , and  $\phi$ .

**THEOREM 4.2.** *Let  $\eta, \rho \in uco(\wp(\mathbb{V}^L))$  and  $\phi \in uco(\wp(\mathbb{V}^H))$ .*

- $(\eta)P(\phi \rightsquigarrow \rho)$  iff  $\forall Y \in \eta(\wp(\mathbb{V}^L)). \lambda x \in \mathbb{V}^H. \rho(\llbracket P \rrbracket)(\phi(x), Y)^L$  is constant.
- $[\eta]P(\rho)$  iff  $\forall Y \in \eta(\wp(\mathbb{V}^L)). \lambda x \in \mathbb{V}^H, y \in Y. \rho(\llbracket P \rrbracket)(x, y)^L$  is constant.

Hence checking abstract non-interference corresponds to check whether the best correct approximation of the concrete semantics of  $P$ , restricted to H variables in input and L variables in output, is constant. A similar result holds for narrow abstract non-interference, even though checking for the narrow case would not involve the best correct approximation of the concrete semantics but rather the concrete semantics itself. It is clear that, if  $\phi$  and  $\eta$  are complete abstractions for  $\rho$  and  $\llbracket P \rrbracket$  (see [20]), then  $[\eta]P(\rho)$  iff  $(\eta)P(\phi \rightsquigarrow \rho)$ . Finally, note that abstract non-interference, as well as secrecy [34], is not in general a safety property unless no possible change is observable on L variables, i.e., in general only  $\gamma^{\mathcal{D}}(\mathbb{S}(\eta, \phi \rightsquigarrow \top))$  is safety, i.e., it is a prefix-closed set of traces.

## 5 Deriving attackers

In this section we define systematic methods for deriving attackers from programs by abstract interpretation. In particular we are interested in characterizing the most concrete, viz. most precise, attacker for which a given program is secure. This is useful both in automatic program certification for abstract non-interference and in order to classify programs in terms of the properties that make them secure. Since attackers are characterized as pairs of abstract domains, the idea is to define an abstract domain transformer, depending on the program to be analyzed, which is able to transform any non-secret abstraction  $\rho$  into the closest abstraction for which the program is secure. This corresponds to characterize the most concrete, viz. most precise, attacker for which a given program is secure. This clearly has sense, in both narrow and abstract non-interference, if we mean to simplify domains: By Proposition 3.7, any refinement of non-secret output abstraction is still non-secret. Let  $P$  be a program,  $\eta, \rho \in uco(\wp(\mathbb{V}^L))$ , and  $\phi \in uco(\wp(\mathbb{V}^H))$ . Assume that the program  $P$  is not  $\langle \eta, \rho \rangle$ -NSecret [resp.  $\langle \eta, \phi, \rho \rangle$ -Secret]. We assume fixed the input-abstraction  $\eta$  and the private property  $\phi$ . We know by Proposition 3.7 that the most concrete  $\beta \sqsupseteq \rho$  such that  $[\eta]P(\beta)$  [resp.  $(\eta)P(\phi \rightsquigarrow \beta)$ ], always exists unique. We call this domain the *narrow* [resp. *abstract*] *secret kernel* of  $\rho$  for  $P$ . As usual in systematic abstract domain design [9, 18], we specify secret kernels by corresponding abstract domain transformers,  $\mathcal{K}_{P, [\eta]}, \mathcal{K}_{P, (\eta), \phi} : uco(\wp(\mathbb{V}^L)) \rightarrow uco(\wp(\mathbb{V}^L))$ :

$$\begin{aligned} \mathcal{K}_{P, [\eta]} &\stackrel{\text{def}}{=} \lambda \rho. \sqcap \{ \beta \mid \rho \sqsubseteq \beta \wedge [\eta]P(\beta) \} \\ \mathcal{K}_{P, (\eta), \phi} &\stackrel{\text{def}}{=} \lambda \rho. \sqcap \{ \beta \mid \rho \sqsubseteq \beta \wedge (\eta)P(\phi \rightsquigarrow \beta) \} \end{aligned}$$

In order to characterize these abstract domain transformers, we have to identify when a program property, viewed as a collection of values, is secure. Program properties are closure operators on sets of possible values. This means that we have to characterize which set of values are allowed in an abstract domain  $\rho$  such that  $[\eta]P(\rho)$  [resp.  $(\eta)P(\phi \rightsquigarrow \rho)$ ]. For any  $l \in \mathbb{V}^L$  we consider the following sets:

$$\begin{aligned} Y_{\llbracket P \rrbracket}^{\eta} (l) &\stackrel{\text{def}}{=} \{ \llbracket P \rrbracket (h, y)^L \mid \eta(y) = \eta(l), h \in \mathbb{V}^H \} \\ Y_{\llbracket P \rrbracket}^{\eta, \phi} (l) &\stackrel{\text{def}}{=} \{ \llbracket P \rrbracket (\phi(h), \eta(l))^L \mid h \in \mathbb{V}^H \} \end{aligned}$$

These sets represent the collections of the sets of values that any secure property  $\rho$  has not to distinguish, i.e., which have to be abstracted by  $\rho$  into the same object. This means that any secure  $\rho$  has not to distinguish elements in  $Y_{\llbracket P \rrbracket}^{\eta}$  [resp.  $Y_{\llbracket P \rrbracket}^{\eta, \phi}$ ]. Let  $\varepsilon = \eta$  or  $\varepsilon = \eta, \phi$ . We define the predicate  $\text{Secr}_{\llbracket P \rrbracket}^{\varepsilon}(X)$  for any  $X \subseteq \mathbb{V}^L$ :

$$\boxed{\text{Secr}_{\llbracket P \rrbracket}^{\varepsilon}(X) \text{ iff } \forall l \in \mathbb{V}^L. (\exists Z \in Y_{\llbracket P \rrbracket}^{\varepsilon}(l). Z \subseteq X \Rightarrow \forall W \in Y_{\llbracket P \rrbracket}^{\varepsilon}(l). W \subseteq X)}$$

Then  $\text{Secr}_{\llbracket P \rrbracket}^{\varepsilon}(X)$  holds if  $X$  does not brake any  $Y_{\llbracket P \rrbracket}^{\varepsilon}(l)$ , namely  $X$  contains all the sets in  $Y_{\llbracket P \rrbracket}^{\varepsilon}(l)$  or none of them. In the following we denote  $\mathcal{S}_{\llbracket P \rrbracket}^{\varepsilon} : \wp(\wp(\mathbb{V}^L)) \rightarrow \wp(\wp(\mathbb{V}^L))$  the predicate transformer  $\mathcal{S}_{\llbracket P \rrbracket}^{\varepsilon}(\mathbb{X}) = \{ X \in \mathbb{X} \mid \text{Secr}_{\llbracket P \rrbracket}^{\varepsilon}(X) \}$ . The following result proves that  $\text{Secr}$  precisely determines the secret kernel.

**THEOREM 5.1.** *Let  $\eta \in uco(\wp(\mathbb{V}^L))$  and  $\phi \in uco(\wp(\mathbb{V}^H))$ .*

- $\mathcal{K}_{P, [\eta]}(\text{id}) = \{ X \in \wp(\mathbb{V}^L) \mid \text{Secr}_{\llbracket P \rrbracket}^{\eta}(X) \}$ ;
- $\mathcal{K}_{P, (\eta), \phi}(\text{id}) = \{ X \in \wp(\mathbb{V}^L) \mid \text{Secr}_{\llbracket P \rrbracket}^{\eta, \phi}(X) \}$ .

Therefore the secret kernels of a generic domain  $\rho$  can be defined respectively as:  $\mathcal{K}_{P, [\eta]} = \lambda \rho. \mathcal{K}_{P, [\eta]}(\text{id}) \sqcup \rho$  and  $\mathcal{K}_{P, (\eta), \phi} = \lambda \rho. \mathcal{K}_{P, (\eta), \phi}(\text{id}) \sqcup \rho$ .  $\mathcal{K}$  is an abstract domain simplification [18],

namely  $\mathcal{K} \in uco(uco(\wp(C)))$ , mapping insecure abstract domains to the most concrete of their secure abstractions. As observed above, any secure abstraction has not to distinguish the sets in  $\Upsilon_{\llbracket P \rrbracket}^n(l)$  [resp.  $\Upsilon_{\llbracket P \rrbracket}^{n,\phi}(l)$ ]. Hence, for any  $\eta \in uco(\wp(\mathbb{V}^L))$ , we consider the following family of sets of possible values:

$$\mathbb{D}_{\llbracket P \rrbracket}(\eta) \stackrel{\text{def}}{=} \{ \llbracket P \rrbracket(\mathbb{V}^H, \eta(y))^L \mid y \in \mathbb{V}^L \}$$

Note that  $\mathbb{D}_{\llbracket P \rrbracket}(\eta)$  is not in general a Moore-family.

EXAMPLE 5.2. Consider the program fragment

$$P \stackrel{\text{def}}{=} \mathbf{while} \ h \ \mathbf{do} \ (l := l * 2; h := 0)$$

Its single-step standard denotational semantics is:

$$\llbracket P \rrbracket(h, l) = \begin{cases} (h, l) & \text{if } h = 0 \\ (0, l * 2) & \text{otherwise} \end{cases}$$

Then for each  $l \in \mathbb{Z}$  we have  $\llbracket P \rrbracket(\mathbb{Z}, l)^L = \{l, 2l\}$ , hence  $\mathbb{D}_{\llbracket P \rrbracket}(\text{id}) = \{\{0\}, \{1, 2\}, \{2, 4\}, \{3, 6\}, \{4, 8\}, \dots\}$ , which is clearly not a Moore-family.

In order to make  $\mathbb{D}_{\llbracket P \rrbracket}$  an abstract domain we have to add to  $\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\eta))$  all the sets of possible values that don't violate the predicate *Secr*. In particular, it is worth noting that the principal filter of any set in  $\mathbb{D}_{\llbracket P \rrbracket}$  is a possible candidate for being in the secret kernel, unless it violates the predicate *Secr*. Consider  $\mathcal{S}_{\llbracket P \rrbracket}^e(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\eta)))$ . Note that also this set may not be in general a Moore-family.

EXAMPLE 5.3. Let us consider the Example 5.2 above. We build  $\mathcal{S}_{\llbracket P \rrbracket}^{\text{id}}(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\text{id})))$  by checking those  $X \in \uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\text{id}))$  such that  $\text{Secr}_P^{\text{id}}(X)$  holds. Note that the elements that have not to be distinguished by the abstraction are collected in the sets:  $\Upsilon_{\llbracket P \rrbracket}^{\text{id}}(l) = \{l, 2l\}$ . If we denote by  $\{2\}^{\mathbb{N}} \stackrel{\text{def}}{=} \{2^n \mid n \in \mathbb{N}\}$ , then  $\mathcal{S}_{\llbracket P \rrbracket}^e(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\text{id}))) = \Upsilon(\{n\{2\}^{\mathbb{N}} \mid n \in 2\mathbb{N} + 1\} \cup \{0\})$ . It is worth noting that this is a partition of  $\mathbb{N}$ .

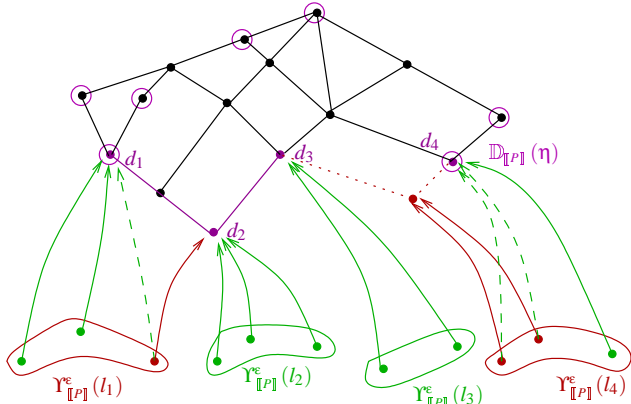


Figure 2. The construction of  $\mathcal{S}_{\llbracket P \rrbracket}^e(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\eta)))$ .

In Figure 2 we have an example on how  $\mathcal{S}_{\llbracket P \rrbracket}^e(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\eta)))$  is constructed. Assume  $\mathbb{D}_P(\eta) = \{d_1, d_2, d_3, d_4\}$ . We note that  $d_2$  brakes the set  $\Upsilon_{\llbracket P \rrbracket}^e(l_1)$  and  $d_3$  cause the brake of  $\Upsilon_{\llbracket P \rrbracket}^e(l_4)$ , therefore  $\text{Secr}(d_2)$  and  $\text{Secr}(d_3)$  don't hold. The operation  $\mathcal{S}_{\llbracket P \rrbracket}^e$  erases  $d_2$  and  $d_3$  from  $\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\eta))$ . With similar argument we erase from  $\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\eta))$  all the points that do not satisfy *Secr*. The remaining points are circled in the picture, and they form the set  $\mathcal{S}_{\llbracket P \rrbracket}^e(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\eta)))$ . We call these points *relevant elements* of the secret kernel. This is not sufficient in order to get the whole secret kernel. Indeed there are elements of the concrete domain of computation that are not involved in the verification of secrecy, namely

in the observed output of the program. We call these elements *irrelevant*:

$$\text{Irr}_{\llbracket P \rrbracket}^{\phi, \eta} \stackrel{\text{def}}{=} \left\{ X \in \wp(\mathbb{V}^L) \mid \begin{array}{l} \forall h \in \mathbb{V}^H, l \in \mathbb{V}^L. \\ X \not\subseteq \uparrow(\llbracket P \rrbracket(\phi(h), \eta(l))^L) \end{array} \right\}$$

Irrelevant elements are also important in order to characterize the elements that reveal deceptive flows in narrow abstract non-interference. Suppose the output closure we are checking for secrecy contains an element  $X \subset \llbracket P \rrbracket(h, \eta(l))^L$ , for some  $h \in \mathbb{V}^H$ ,  $l \in \mathbb{V}^L$ , and  $X \neq \emptyset$ , then clearly there exists  $y_1 \in \eta(l)$  such that  $\llbracket P \rrbracket(h, y_1)^L \in X$  and  $y_2 \in \eta(l)$  such that  $\llbracket P \rrbracket(h, y_2)^L \notin X$ . In this situation the revealed flow for  $\llbracket P \rrbracket(h, y_1)^L$  and  $\llbracket P \rrbracket(h, y_2)^L$ , due to the presence of  $X$  in the closure, is clearly a deceptive flow. These sets, revealing deceptive flows, are in general irrelevant elements in the abstract non-interference case. This because in the abstract non-interference case we are interested in abstracting  $\llbracket P \rrbracket(h, \eta(l))^L$ . Since  $X \subset \llbracket P \rrbracket(h, \eta(l))^L \subseteq \llbracket P \rrbracket(\phi(h), \eta(l))^L$ ,  $X$  falls in the set of irrelevant elements for abstract non-interference. In particular, the sets  $X$  which reveal deceptive flows are contained in  $\text{Irr}_{\llbracket P \rrbracket}^{\text{id}, \eta} \setminus \text{Irr}_{\llbracket P \rrbracket}^{\text{id}, \text{id}}$ . Note also that for the narrow case we have to consider the more concrete closure that induces the same partition of values as  $\eta \in uco(\wp(\mathbb{V}^L))$ , i.e.,  $\mathcal{P}(\eta) \stackrel{\text{def}}{=} \Upsilon(\{[x]_\eta \mid x \in \mathbb{V}^L\})$ , where  $[x]_\eta \stackrel{\text{def}}{=} \{y \mid \eta(x) = \eta(y)\}$ . This is essential in order get the secret kernel, i.e., the most concrete domain  $\beta$  such that  $[\eta]P(\beta)$ . 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  $\eta(y) \subseteq \eta(x)$ .

EXAMPLE 5.4. Consider the following program fragment:

$$P \stackrel{\text{def}}{=} \mathbf{while} \ h \ \mathbf{do} \ (l := l + 6; h := h - 1)$$

with security typing  $t = \langle h : \mathbb{H}, l : \mathbb{L} \rangle$  and  $\mathbb{V} = \mathbb{Z}$ . Let us consider  $[\eta]\llbracket P \rrbracket(\text{id})$ , where  $\eta(\wp(\mathbb{Z})) = \{\mathbb{Z}, 3\mathbb{Z}, 6\mathbb{Z}, \emptyset\}$ . We show that if we consider  $\eta$  instead of  $\mathcal{P}(\eta)$  we don't find the secret kernel of the program. First of all note that  $\forall l \in 6\mathbb{Z}. \Upsilon_{\llbracket P \rrbracket}^n(l) = 6\mathbb{Z}$  and  $\forall l. \eta(l) = 3\mathbb{Z}. \Upsilon_{\llbracket P \rrbracket}^n(l) = 6\mathbb{Z} + 3$ . Finally if  $l \notin 3\mathbb{Z}$ , then  $\Upsilon_{\llbracket P \rrbracket}^n(l) = \bigcup \{6\mathbb{Z} + l \mid l \notin 3\mathbb{Z}\}$  which is  $\mathbb{Z} \setminus 3\mathbb{Z}$ . At this point  $\mathbb{D}_{\llbracket P \rrbracket}(\eta) = \{6\mathbb{Z}, 3\mathbb{Z}, \mathbb{Z}\}$ , and it is simple to verify that  $\mathcal{S}_{\llbracket P \rrbracket}^n(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\eta))) = \{\mathbb{Z}, \mathbb{Z} \setminus (3(2\mathbb{Z} + 1)), 3\mathbb{Z}, 6\mathbb{Z}\}$ . Let us consider now  $\mathcal{P}(\eta) = \Upsilon(\{\mathbb{Z}, \mathbb{Z} \setminus 3\mathbb{Z}, 3(2\mathbb{Z} + 1), 6\mathbb{Z}, \emptyset\})$ . Then we have that  $\mathbb{D}_{\llbracket P \rrbracket}(\mathcal{P}(\eta)) = \Upsilon(\{6\mathbb{Z}, 3(2\mathbb{Z} + 1), \mathbb{Z} \setminus 3\mathbb{Z}\})$ . At this point it is simple to show that  $\mathcal{S}_{\llbracket P \rrbracket}^n(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\mathcal{P}(\eta)))) = \mathbb{D}_{\llbracket P \rrbracket}(\mathcal{P}(\eta))$  which strictly contains the set  $\mathcal{S}_{\llbracket P \rrbracket}^n(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\eta)))$  obtained above.

We are now in the position to specify the secret kernel for both the narrow and abstract non-interference.

$$\begin{aligned} [\eta]\llbracket P \rrbracket(\text{id}) &\stackrel{\text{def}}{=} \mathcal{S}_{\llbracket P \rrbracket}^n(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\mathcal{P}(\eta)))) \cup \text{Irr}_{\llbracket P \rrbracket}^{\text{id}, \text{id}} \\ (\eta)\llbracket P \rrbracket(\phi \rightsquigarrow \text{id}) &\stackrel{\text{def}}{=} \mathcal{S}_{\llbracket P \rrbracket}^{n, \phi}(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\eta))) \cup \text{Irr}_{\llbracket P \rrbracket}^{\phi, \eta} \end{aligned}$$

Note that these definitions introduce a slight abuse of notation: While  $(\eta)\llbracket P \rrbracket(\phi \rightsquigarrow \text{id}) \in uco(\wp(\mathbb{V}^L))$  is an abstract domain,  $(\eta)P(\phi \rightsquigarrow \text{id})$  is a program property. The same holds for the narrow case. These notions have a strong relation as specified by the following result which provides a domain-theoretic characterization of the objects in the secret kernels.

THEOREM 5.5. Let  $\eta, \rho \in uco(\wp(\mathbb{V}^L))$  and  $\phi \in uco(\wp(\mathbb{V}^H))$ .

1.  $\mathcal{K}_{\rho, [\eta]}(\text{id}) = [\eta]\llbracket P \rrbracket(\text{id})$ ;
2.  $\mathcal{K}_{\rho, (\eta), \phi}(\text{id}) = (\eta)\llbracket P \rrbracket(\phi \rightsquigarrow \text{id})$ .

The following examples show the complete construction for both narrow and abstract non-interference.

EXAMPLE 5.6. Consider the program fragment:

$$P \stackrel{\text{def}}{=} \text{while } h \text{ do } (l := l * 2; h := h - 1)$$

with security typing  $\langle h : \mathbb{H}, l : \mathbb{L} \rangle$  and  $\mathbb{V} = \mathbb{N}$ . We noted in the introduction that this fragment is secure as regards as the property **Sign**, while it is not secure as regards as **Parity**. We find here the most concrete property (that clearly has to contain **Sign**) that makes  $P$  secure. The denotational semantics is:

$$\llbracket P \rrbracket(h, l) = \begin{cases} (h, l) & \text{if } h = 0 \\ (0, l * 2^h) & \text{otherwise} \end{cases}$$

We compute the closure  $\llbracket \text{id} \rrbracket \llbracket P \rrbracket(\text{id})$ . If  $l = 1$  we have  $\llbracket P \rrbracket(\mathbb{V}^{\mathbb{H}}, 1)^{\mathbb{L}} = \{2\}^{\mathbb{N}}$ , if  $l = 2$  then  $\llbracket P \rrbracket(\mathbb{V}^{\mathbb{H}}, 2)^{\mathbb{L}} = 2\{2\}^{\mathbb{N}}$ , and for each  $l \in \mathbb{V}^{\mathbb{L}}$  we have  $\llbracket P \rrbracket(\mathbb{V}^{\mathbb{H}}, l)^{\mathbb{L}} = l\{2\}^{\mathbb{N}}$ , i.e.,  $\mathbb{D}_{\llbracket P \rrbracket}(\text{id}) = \{n\{2\}^{\mathbb{N}} \mid n \in \mathbb{N}\}$ . Moreover for each  $n$  we have that  $\Upsilon_{\llbracket P \rrbracket}^{\text{id}}(n) = n\{2\}^{\mathbb{N}}$ . Clearly not all these elements are secret, indeed for each  $n$  we have  $n\{2\}^{\mathbb{N}} \supseteq 2n\{2\}^{\mathbb{N}}$  and this implies that  $n \in \Upsilon(n)$  but  $n \notin 2n\{2\}^{\mathbb{N}}$ , while  $2n \in \Upsilon(n)$  and  $2n \in 2n\{2\}^{\mathbb{N}}$ . Hence  $\neg \text{Secr}_{\llbracket P \rrbracket}^{\text{id}}(2n\{2\}^{\mathbb{N}})$ . We apply  $\mathcal{S}_{\llbracket P \rrbracket}^{\text{id}}$  and we obtain the abstract domain  $\mathcal{S}_{\llbracket P \rrbracket}^{\text{id}}(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\text{id}))) = \Upsilon(\{n\{2\}^{\mathbb{N}} \mid n \in 2\mathbb{N} + 1\} \cup \{\emptyset\})$ . Note that in this case the set of irrelevants is  $\{\emptyset\}$ . The resulting set, by Theorem 5.5, is the most concrete domain making  $P$  secure.

EXAMPLE 5.7. Let us consider the program fragment:

$$P \stackrel{\text{def}}{=} l := l + (h \bmod 3);$$

with security typing  $\langle h : \mathbb{H}, l : \mathbb{L} \rangle$  and  $\mathbb{V} = \mathbb{Z}$ . Let us consider  $(\eta)P(\text{id} \rightsquigarrow \text{id})$  where  $\eta$  is the abstract domain  $\eta(\wp(\mathbb{Z})) = \{\mathbb{Z}, [2, 4], [5, 8], \{5\}, \emptyset\}$ . We have to compute  $\llbracket P \rrbracket(\mathbb{Z}, \{5\})^{\mathbb{L}} = [5, 7]$ ,  $\llbracket P \rrbracket(\mathbb{Z}, [2, 4])^{\mathbb{L}} = [2, 6]$ ,  $\llbracket P \rrbracket(\mathbb{Z}, [5, 8])^{\mathbb{L}} = [5, 10]$ , and also  $\llbracket P \rrbracket(\mathbb{Z}, \mathbb{Z})^{\mathbb{L}} = \mathbb{Z}$ . Therefore  $\mathbb{D}_{\llbracket P \rrbracket}(\eta) = \{[5, 7], [2, 6], [5, 10], \mathbb{Z}\}$ . On the other hand we have  $\Upsilon_{\llbracket P \rrbracket}^{\eta, \text{id}}([2, 4]) = \{[2, 4], [3, 5], [4, 6]\}$ ,  $\Upsilon_{\llbracket P \rrbracket}^{\eta, \text{id}}([5, 8]) = \{[5, 8], [6, 9], [7, 10]\}$ ,  $\Upsilon_{\llbracket P \rrbracket}^{\eta, \text{id}}(\{5\}) = \{5, 6, 7\}$  and, for all other possible low input  $l$ , we have  $\Upsilon_{\llbracket P \rrbracket}^{\eta, \text{id}}(l) = \{\mathbb{Z}\}$  that cannot create problems to secrecy. It is possible to verify that  $\mathcal{S}_{\llbracket P \rrbracket}^{\eta, \text{id}}(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\eta)))$  is the following collection of objects:

$$\left\{ Y \in \wp(\mathbb{V}) \left| \begin{array}{l} Y \supseteq [2, 10] \vee \\ Y \not\supseteq [2, 10], Y \supseteq [2, 7], \\ Y \cap [5, 10] \notin \{[5, 9], [5, 8], [5, 8] \cup \{10\}\} \vee \\ Y \not\supseteq [2, 10], Y \supseteq [5, 10], \\ Y \cap [2, 7] \notin \{[4, 7], [3, 7], [4, 7] \cup \{2\}\} \end{array} \right. \right\}$$

An approximation can be introduced in the derivation of the secret kernel by separately analyzing program fragments. In the following we show two methods for approximating the secret kernel of a closure  $\rho$  for a program  $P$ , by inductively deriving the kernels of program components.

### Bounded iterations

Let  $\llbracket P \rrbracket^{(n)}$  represents the partial semantics of the program at the  $n$ -th step of evaluation, supposing that all while's are unfolded: If  $P = c_0; c_1; \dots; c_m$ , then for any  $n$  define:

$$\begin{aligned} \llbracket P \rrbracket^{(0)} &\stackrel{\text{def}}{=} \llbracket c_0 \rrbracket \\ \llbracket P \rrbracket^{(n+1)} &\stackrel{\text{def}}{=} \llbracket c_{n+1} \rrbracket \circ \llbracket P \rrbracket^{(n)} \end{aligned}$$

For instance,  $\llbracket P \rrbracket^{(2)}$  is the semantics of  $c_0; c_1; c_2$ , that is  $\llbracket P \rrbracket^{(2)} = \llbracket c_2 \rrbracket \circ \llbracket c_1 \rrbracket \circ \llbracket c_0 \rrbracket$ . We define an abstract domain transformer  $\mathcal{X}_{P, [\eta], \phi}^{\text{par}}$  [resp.  $\mathcal{X}_{P, (\eta), \phi}^{\text{par}}$ ] denoting the common abstraction among all the domains  $\rho_i$  such that the first  $i$ -steps are  $\langle \eta, \rho_i \rangle$ -*NSecret*, [resp.

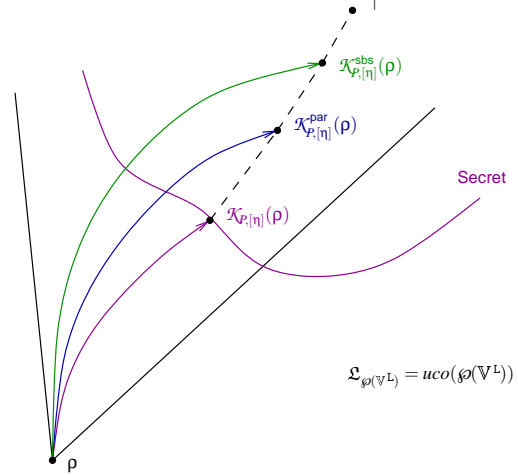


Figure 3. Deriving secret kernels

$\langle \eta, \phi, \rho_i \rangle$ -*Secret*] i.e., for any  $i \leq n$ :  $\rho_i = [\eta] \llbracket P \rrbracket^{(i)}(\text{id})$  [resp.  $(\eta) \llbracket P \rrbracket^{(n)}(\phi \rightsquigarrow \rho_i)$ ], it is defined in the following way:

$$\begin{aligned} \mathcal{X}_{P, [\eta], \phi}^{\text{par}} &\stackrel{\text{def}}{=} \lambda \rho. \rho \sqcup \bigsqcup_{i \leq n} [\eta] \llbracket P \rrbracket^{(i)}(\text{id}) \\ \mathcal{X}_{P, (\eta), \phi}^{\text{par}} &\stackrel{\text{def}}{=} \lambda \rho. \rho \sqcup \bigsqcup_{i \leq n} (\eta) \llbracket P \rrbracket^{(i)}(\phi \rightsquigarrow \text{id}) \end{aligned}$$

It is clear that  $\mathcal{X}_{P, [\eta]} \sqsubseteq \mathcal{X}_{P, [\eta], \phi}^{\text{par}}$ . By Proposition 3.7,  $[\eta]P(\mathcal{X}_{P, [\eta], \phi}^{\text{par}}(\rho))$  still holds. The same holds in the abstract non-interference case.

### Independent composition

An even coarser approximation of the secret kernel of  $\rho$  can be obtained by considering the most concrete abstraction making all program statements in  $P$   $\langle \eta, \rho \rangle$ -*NSecret* [resp.  $\langle \eta, \phi, \rho \rangle$ -*Secret*]. Suppose that for each statement  $c$  in  $P$ ,  $\llbracket c \rrbracket$  is the denotational semantics of  $c$ . Let  $P = c_0; c_1; \dots; c_m$  and define:

$$\begin{aligned} \mathcal{X}_{P, [\eta]}^{\text{sbs}} &\stackrel{\text{def}}{=} \lambda \rho. \rho \sqcup \bigsqcup_{i \leq m} [\eta] \llbracket c_i \rrbracket(\text{id}) \\ \mathcal{X}_{P, (\eta), \phi}^{\text{sbs}} &\stackrel{\text{def}}{=} \lambda \rho. \rho \sqcup \bigsqcup_{i \leq m} (\eta) \llbracket c_i \rrbracket(\phi \rightsquigarrow \text{id}) \end{aligned}$$

Intuitively  $\mathcal{X}_{P, [\eta]}^{\text{sbs}}$  [resp.  $\mathcal{X}_{P, (\eta), \phi}^{\text{sbs}}$ ] is the property which is not disclosed with respect to each slice of the program  $P$  relatively to sequential composition. Note that, in this case, if there is at least one statement  $c$  such that  $[\eta] \llbracket c \rrbracket(\text{id}) = \{\top\}$  then  $\mathcal{X}_{P, [\eta]}^{\text{sbs}} = \{\top\}$ . Also in this case we have  $\mathcal{X}_{P, [\eta]} \sqsubseteq \mathcal{X}_{P, [\eta]}^{\text{sbs}}$ . It is clear that this is an upper approximation of both  $\mathcal{X}_{P, [\eta], \phi}^{\text{par}}$  and  $\mathcal{X}_{P, (\eta), \phi}^{\text{par}}$ , i.e., as shown in Figure 3:

$$\mathcal{X}_{P, [\eta]} \sqsubseteq \mathcal{X}_{P, [\eta], \phi}^{\text{par}} \sqsubseteq \mathcal{X}_{P, [\eta]}^{\text{sbs}}$$

The same holds for the abstract non-interference case. By Proposition 3.7,  $[\eta]P(\mathcal{X}_{P, [\eta]}^{\text{sbs}}(\rho))$  and  $(\eta)P(\phi \rightsquigarrow \llbracket \mathcal{X}_{P, [\eta]}^{\text{sbs}}(\rho) \rrbracket)$  still hold. We now show a simple example for the narrow case, where the relations  $\mathcal{X}_{P, [\eta]} \sqsubseteq \mathcal{X}_{P, [\eta], \phi}^{\text{par}} \sqsubseteq \mathcal{X}_{P, [\eta]}^{\text{sbs}}$  are strict inclusions.

EXAMPLE 5.8. Consider the program fragment

$$P \stackrel{\text{def}}{=} l := 2 * h; l := l * h$$

with typing  $\langle h : \mathbb{H}, l : \mathbb{L} \rangle$  and  $\mathbb{V} = \mathbb{Z}$  and  $\llbracket P \rrbracket(h, l) = (h, 2 * h^2)$ . We consider  $\llbracket \text{id} \rrbracket P(\text{id})$ . We have  $\mathbb{D}_{\llbracket P \rrbracket}(\text{id}) = \{\{2n^2 \mid n \in \mathbb{Z}\}\}$ . Note that in this case the operator  $\mathcal{S}_{\llbracket P \rrbracket}^{\text{id}}$  is the identity on  $\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\text{id}))$ . Hence we obtain the secret kernel by computing the irrelevant elements which are  $\Upsilon(\{\{m\} \mid m \in \mathbb{Z} \cup \cup \{2n^2 \mid n \in \mathbb{Z}\}\})$ . In this case, the secret kernel of  $P$  is the closure  $\rho = \uparrow(\{\{2n^2 \mid n \in \mathbb{Z}\}\} \cup \Upsilon(\{\{m\} \mid m \in \mathbb{Z} \cup \cup \{2n^2 \mid n \in \mathbb{Z}\}\}))$ .

Consider now  $\mathcal{K}_{\rho_1, [\text{id}]}^{\text{par}}(\text{id})$ . In order to obtain this domain we have to find the closure  $\rho_1$  such that  $[\text{id}]l := 2h(\rho_1)$ . This domain is  $\rho_1 = \uparrow(\{2\mathbb{Z}\} \cup \{\{n\} \mid n \in 2\mathbb{Z} + 1\})$ . Clearly  $\llbracket P \rrbracket^{(2)} = \llbracket P \rrbracket$ , therefore  $\mathcal{K}_{\rho_1, [\text{id}]}^{\text{par}}(\text{id}) = \rho_1 \sqcup \rho$ . It is worth noting that  $\rho_1 \sqsupset \rho$ , therefore  $\mathcal{K}_{\rho_1, [\text{id}]}^{\text{par}}(\text{id}) = \rho_1 \sqsupset \rho$ . Finally consider  $\mathcal{K}_{\rho_1, [\text{id}]}^{\text{abs}}(\text{id})$ . Clearly, in this case, the secret kernel for the first statement is exactly  $\rho_1$  as defined in the previous case. We have to compute  $\rho_2$  such that  $[\text{id}]l := l * h(\rho_2)$ . It is worth noting that  $\mathbb{D}_{\llbracket P \rrbracket}(\text{id})$  is the set of congruences of the kind  $n\mathbb{Z}$  with  $n \in \mathbb{Z}$ . At this point the operator  $\mathcal{S}_{\llbracket P \rrbracket}^e$  determines the domain  $\rho_2 = \{\mathbb{Z}\}$ . It is worth noting that  $\rho_2 \sqsupset \rho_1$ , therefore we have that  $\mathcal{K}_{\rho_1, [\text{id}]}^{\text{abs}}(\text{id}) = \rho_1 \sqcup \rho_2 = \rho_2 \sqsupset \mathcal{K}_{\rho_1, [\text{id}]}^{\text{par}}(\text{id})$ .

### Fix-point attackers

We finally consider the problem of finding the most concrete domain  $\rho \in \text{uco}(\wp(\mathbb{V}^L))$  such that  $[\rho]P(\rho)$  [resp.  $(\rho)P(\phi \rightsquigarrow \llbracket \rho \rrbracket)$ ]. This problem is slightly more complex and requires an iterative solution. Indeed, while simplifying the output observation we do not brake secrecy, the same simplified abstraction applied to the input may reveal secrets. An abstract domain  $\rho$  such that  $[\rho]P(\rho)$  [ $(\rho)P(\phi \rightsquigarrow \llbracket \rho \rrbracket)$ ] represents a possible attacker which is unable to disclose confidential data by analyzing the same property on input/output relations. We are interested in characterizing the strongest of such attackers if it exists unique, later called the (canonical) secure attacker for  $P$ .

**THEOREM 5.9.** *Let  $\rho \in \text{uco}(\wp(\mathbb{V}^L))$  and  $\phi \in \text{uco}(\wp(\mathbb{V}^H))$ .*

- $[\rho]P(\rho)$  iff  $\rho = [\rho]\llbracket P \rrbracket(\text{id})$ ;
- $(\rho)P(\phi \rightsquigarrow \llbracket \rho \rrbracket)$  iff  $\rho = (\rho)\llbracket P \rrbracket(\phi \rightsquigarrow \llbracket \rho \rrbracket)$ .

In order to constructively characterize the secure attackers of a program, we consider standard domain-theoretic arguments. It is worth noting that in the narrow case  $[\eta]\llbracket P \rrbracket(\text{id})$ , the set of irrelevants does not depend on the input observation  $\eta$ . Therefore, the change of the input property does not have any effect on the irrelevants  $\text{Irr}_{\llbracket P \rrbracket}^{\text{id}, \text{id}}$ . Moreover, by construction, the set  $\mathbb{D}_{\llbracket P \rrbracket}$  monotonically depends upon the input property  $\eta$ . Therefore we have the following result.

**PROPOSITION 5.10.** *Let  $P$  be a program.  $\lambda X. [X]\llbracket P \rrbracket(\text{id})$  is monotone on  $\langle \text{uco}(\wp(\mathbb{V}^L)), \sqsubseteq \rangle$ .*

Therefore, by Tarski fix-point theorems, an iterative solution to the problem of deriving the most concrete secure attacker for a program  $P$  in the narrow case can be found such that, at each step  $n$ , we find the most concrete domain  $\rho_n$  that satisfies  $[\rho_{n-1}]P(\rho_n)$ , which is  $\mathcal{K}_{\rho_{n-1}, [\text{id}]}(\text{id})$ . By Proposition 5.10 we have that, given a program  $P$ , then  $\lambda X. ([X]\llbracket P \rrbracket(\text{id}))$  is monotone on  $\text{uco}(\wp(\mathbb{V}^L))$ . The most concrete secure attacker for the narrow case is therefore the least fix-point of  $\lambda X. \mathcal{K}_{\rho, [X]}(\text{id})$ .

**COROLLARY 5.11.** *lfp $_{\text{id}}^{\sqsubseteq} \lambda X. ([X]\llbracket P \rrbracket(\text{id}))$  is the (unique) most concrete narrow secure attacker for  $P$ .*

In the following we denote  $\mathcal{F}_P \stackrel{\text{def}}{=} \text{lfp}_{\text{id}}^{\sqsubseteq} \lambda X. ([X]\llbracket P \rrbracket(\text{id}))$ .

**EXAMPLE 5.12.** *Consider the fragment given in Example 5.6 together with its denotational semantics. In Example 5.6 we derived the most concrete domain  $\rho_1 \stackrel{\text{def}}{=} \mathcal{S}_{\llbracket P \rrbracket}^e(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\text{id})))$  such that  $[\text{id}]P(\rho_1)$ . Consider the closure  $[\rho_1]\llbracket P \rrbracket(\text{id})$ , namely the most concrete closure that makes  $P$  secure with input observation  $\rho_1$ . Note that  $\rho_1 = \mathcal{P}(\rho_1)$ . We have to compute  $\llbracket P \rrbracket(\mathbb{V}^H, Y)$  for each  $Y \in \rho_1$ . Namely, consider for example  $Y = \{2\}^{\mathbb{N}}$ , then  $\llbracket P \rrbracket(\mathbb{V}^H, \{2\}^{\mathbb{N}})^L =$*

$\{2\}^{\mathbb{N}}$ , if  $Y = 3\{2\}^{\mathbb{N}}$  then  $\llbracket P \rrbracket(\mathbb{V}^H, 3\{2\}^{\mathbb{N}})^L = 3\{2\}^{\mathbb{N}}$ , and so on. It is clear that we reached the fix-point. Therefore, in this example  $\mathcal{F}_P = \Upsilon(\{\{n\{2\}^{\mathbb{N}} \mid n \in 2\mathbb{N} + 1\} \cup \{\{0\}\}\})$ . We now apply bounded iterations to approximate the closure  $\mathcal{F}_P$ , denoted  $\mathcal{F}_P^{\text{par}}$ . The single-step semantics is:

$$\llbracket P \rrbracket(h, l) = \begin{cases} (h, l) & \text{if } h = 0 \\ (h-1, l * 2) & \text{otherwise} \end{cases}$$

Then, for each value  $l \in \mathbb{V}^L$ , we have  $\llbracket P \rrbracket(\mathbb{V}^H, l)^L = \{l, 2l\}$ . Hence it is possible to verify that  $\mathcal{S}_{\llbracket P \rrbracket}^{\text{id}}(\uparrow(\mathbb{D}_{\llbracket P \rrbracket}(\text{id}))) = \Upsilon(\{\{n\{2\}^{\mathbb{N}} \mid n \in 2\mathbb{N} + 1\} \cup \{\{0\}\}\})$ . As for the second step is concerned for each  $l \in \mathbb{V}^L$  we have that  $\llbracket P \rrbracket^{(2)}(\mathbb{V}^H, l)^L = \{l, 2l, 4l\}$ . Again it is possible to verify that the resulting domain is  $\Upsilon(\{\{n\{2\}^{\mathbb{N}} \mid n \in 2\mathbb{N} + 1\} \cup \{\{0\}\}\})$  which is the closure  $\mathcal{F}_P^{\text{par}}$ . In this case the irrelevants are  $\{\emptyset\}$ , since the range of  $\llbracket P \rrbracket$  covers the whole domain of values.

In the example above we reached the fix-point in one step. Let us see an example where an infinite iteration is necessary.

**EXAMPLE 5.13.** *Consider the program fragment:*

$$P \stackrel{\text{def}}{=} l := l^2 + (h \bmod l)$$

where  $\bmod$  is the rest of the integer division, the security typing is  $t = \langle h : \mathbb{H}, l : \mathbb{L} \rangle$ , and  $\mathbb{V} = \mathbb{N}$ . The denotational semantics  $\llbracket P \rrbracket$  of  $P$  is immediate from its definition. Then let us consider the closure  $[\text{id}]P(\text{id})$ , in particular let us derive  $\mathbb{D}_{\llbracket P \rrbracket}(\text{id})$ . Consider for example  $l = 1$ , then  $\llbracket P \rrbracket(\mathbb{V}^H, 1)^L = \{1\}$ . If  $l = 2$  then we obtain the integer interval  $\llbracket P \rrbracket(\mathbb{V}^H, 2)^L = [4, 5]$ , and so on. The resulting intervals are all disjoint, therefore  $\mathcal{S}_{\llbracket P \rrbracket}^{\text{id}}$  is the identity in this case. The resulting closure contains  $\Upsilon(\mathbb{D}_{\llbracket P \rrbracket}(\text{id}))$  together with all the elements that are not contained in any of these intervals, i.e., the irrelevants. Let  $\rho_1$  be this closure. We consider now the closure  $[\rho_1]\llbracket P \rrbracket(\text{id})$ . We show only the abstraction of the singleton  $\{4\}$  since the first three natural numbers behave exactly as in the previous case. If we consider  $l = 4$  we have, for example, that  $\llbracket P \rrbracket(\mathbb{V}^H, [4, 5])^L = [16, 19] \cup [25, 29]$ , and so on for the other values. In this way we may continue until the fix-point characterizing the canonical secure attacker. Note that  $\mathcal{S}_{\llbracket P \rrbracket}^{\text{id}}$  on these domains is always the identity, since all the generated intervals are disjoint.

For the weaker case of abstract non-interference, there are two facts that avoid in general monotonicity. First the set of irrelevants grows by abstracting  $\eta$ , namely it is anti-monotone; second the set of relevants is not comparable with the set of relevants obtained by abstracting  $\eta$ . Therefore, in general,  $\lambda X. (X)\llbracket P \rrbracket(\phi \rightsquigarrow \llbracket X \rrbracket)$  is not monotone on  $\langle \text{uco}(\wp(\mathbb{V}^L)), \sqsubseteq \rangle$ .

**EXAMPLE 5.14.** *Let us consider the program fragment  $P$  in Example 5.7. Let us consider two closures  $\eta \sqsubseteq \beta$  with sets of fix-points:  $\eta(\mathbb{Z}) = \{\mathbb{Z}, [2, 4], [5, 8], \{5\}, \emptyset\}$  and  $\beta(\mathbb{Z}) = \{\mathbb{Z}, [2, 4], [5, 8], \emptyset\}$ . Consider first  $\eta$ , then in Example 5.7 we obtained that  $\mathbb{D}_{\llbracket P \rrbracket}(\eta) = \{\{5, 7\}, [2, 6], [5, 10], \mathbb{Z}\}$ . We can note that  $\neg \text{Secr}_{\llbracket P \rrbracket}^{\eta, \text{id}}([2, 6])$  since  $[2, 6]$  contains some elements of  $\Upsilon_{\llbracket P \rrbracket}^{\eta, \text{id}}(5)$  (see Example 5.7), but not all of them. This means that surely  $[2, 6]$  is not included in the final domain, as seen in Example 5.7. Consider now  $\beta$ , then we have to compute  $\llbracket P \rrbracket(\mathbb{Z}, [2, 4])^L = [2, 6]$ ,  $\llbracket P \rrbracket(\mathbb{Z}, [5, 8])^L = [5, 10]$  and  $\llbracket P \rrbracket(\mathbb{Z}, \mathbb{Z})^L = \mathbb{Z}$ . Therefore  $\mathbb{D}_{\llbracket P \rrbracket}(\beta) = \{[2, 6], [5, 10], \mathbb{Z}\}$ . While  $\Upsilon_{\llbracket P \rrbracket}^{\beta, \text{id}}([2, 4]) = \{[2, 4], [3, 5], [4, 6]\}$ ,  $\Upsilon_{\llbracket P \rrbracket}^{\beta, \text{id}}([5, 8]) = \{[5, 8], [6, 9], [7, 10]\}$  and for all other possible low input  $y$  we have  $\Upsilon_{\llbracket P \rrbracket}^{\beta, \text{id}}(y) = \{\mathbb{Z}\}$  that clearly cannot create problems to secrecy. In this case  $\text{Secr}_{\llbracket P \rrbracket}^{\beta, \text{id}}([2, 6])$ , which implies that  $[2, 6]$  is now left in the domain.*

This example shows that in abstract non-interference, in general, by further abstracting the input observation we don't necessarily further abstract the output. Hence, in the abstract non-interference case, the fact that  $\lambda X. (X) \llbracket P \rrbracket (\phi \rightsquigarrow \text{id})$  is not monotone, avoids us to characterize the canonical secure attacker by means of Tarski fix-point theorems. In order to constructively characterize a secure attacker as the limit of a possible transfinite upper iteration sequence of  $\lambda X. (X) \llbracket P \rrbracket (\phi \rightsquigarrow \text{id})$ , we make this function extensive on  $(uco(\wp(\mathbb{V}^L)), \sqsubseteq)$ , i.e., we consider the function:  $\lambda X. (X) \llbracket P \rrbracket (\phi \rightsquigarrow \text{id}) \sqcup X$ . By a well known result in lattice-theory we know that any extensive function has a fix-point [11]. The following result proves that any fix-point of  $\lambda X. (X) \llbracket P \rrbracket (\phi \rightsquigarrow \text{id}) \sqcup X$  is a secure attacker for  $P$ .

**THEOREM 5.15.** *Let  $\rho \in uco(\wp(\mathbb{V}^L))$  and  $\phi \in uco(\wp(\mathbb{V}^H))$ . If  $\rho = (\rho) \llbracket P \rrbracket (\phi \rightsquigarrow \text{id}) \sqcup \rho$  then  $(\rho)P(\phi \rightsquigarrow \text{id})$ .*

## 6 Abstract robust declassification

Declassifying information means downgrading the sensitivity of data in order to accommodate with (intentional) information leakage. Robust declassification has been introduced in [40] as a systematic method to drive declassification by characterizing what information flows from confidential to public variables. In this section we generalize the robust declassification to security properties based on abstract non-interference. Let  $P \in \text{IMP}$  be a program and  $\eta, \rho \in uco(\wp(\mathbb{V}^L))$  be abstract domains. Recall that the set of partitions of a set  $S$  is isomorphic to the set of closures  $\mathcal{P}(uco(\wp(S)))$  [29]. In particular if  $\pi$  is a partition of  $S$  and for any  $\delta \in uco(\wp(S)) \approx_\delta$  is the equivalence relation such that  $v_1 \approx_\delta v_2$  iff  $\delta(v_1) = \delta(v_2)$ , then  $\mathcal{P}(\eta)$ , where  $\eta = \pi \cup \{S, \emptyset\} = \mathcal{M}(\pi)$  is the flat domain ordered by set-inclusion, is the (unique) most concrete abstract domain such that  $\approx_{\mathcal{P}(\eta)}$  induces the same partition as  $\pi$ . We are interested in characterizing an abstraction  $\phi \in uco(\wp(\mathbb{V}^H))$  such that any pair of values which can be distinguished by  $\phi$  violates  $(\eta, \phi, \rho)$ -*Secrecy*. Among these abstract domains we are interested in the most concrete one. This domain provides precisely the maximal amount of information concerning H-values that flows in  $P$  when the attacker is given by the pair of input/output abstractions:  $(\eta, \rho)$ . Note that  $\mathcal{P}(\phi)$  is the most concrete domain that induces the same partition as  $\approx_\phi$ , and because in abstract non-interference we abstract single values, it is clear that we can consider  $\phi \in \mathcal{P}(uco(\wp(\mathbb{V}^H)))$ . A closure  $\phi \in \mathcal{P}(uco(\wp(\mathbb{V}^H)))$  is called *flow-irredundant* with respect to a pair of input/output abstractions  $(\eta, \rho)$  if  $\forall X_1, X_2 \in \phi(\wp(\mathbb{V}^H))$ ,  $\exists Y \in \eta(\wp(\mathbb{V}^L))$  such that:  $\rho(\llbracket P \rrbracket(X_1, Y)^L) \neq \rho(\llbracket P \rrbracket(X_2, Y)^L)$ . The most concrete flow-irredundant domain  $\phi \in \mathcal{P}(uco(\wp(\mathbb{V}^H)))$ , when it exists relatively to  $(\eta, \rho)$ , defines the *maximal abstract interference* of  $P$ , denoted  $(\eta)P(\phi \Rightarrow \rho)$ . The idea for deriving the most concrete flow-irredundant domain  $\phi$ , such that  $(\eta)P(\phi \Rightarrow \rho)$  relies upon a similar construction as the one used in Section 5. Let:

$$\Pi(\eta, \rho) \stackrel{\text{def}}{=} \left\{ \bigvee \left\{ \langle h, \eta(l) \rangle \mid \left. \begin{array}{l} \rho(\llbracket P \rrbracket(h, \eta(l))^L) = \\ \rho(\llbracket P \rrbracket(h', \eta(l))^L) \end{array} \right\} \mid \begin{array}{l} h \in \mathbb{V}^H \\ y \in \mathbb{V}^L \end{array} \right\} \right\}$$

where  $\langle X, Y \rangle \vee \langle X', Y' \rangle = \langle X \cup X', Y \cup Y' \rangle$ . It is worth noting that for any  $Y \in \eta(\wp(\mathbb{V}^L))$ , the set  $\pi_Y \stackrel{\text{def}}{=} \{X \in \wp(\mathbb{V}^H) \mid \langle X, Y \rangle \in \Pi(\eta, \rho)\}$  is a partition of  $\mathbb{V}^H$ .

**THEOREM 6.1.** *If  $P \not\models (\eta)P(\text{id} \rightsquigarrow \rho)$  then:*

$$(\eta)P(\mathcal{P}(\prod_{Y \in \eta(\wp(\mathbb{V}^L))} \mathcal{M}(\pi_Y)) \Rightarrow \rho).$$

**EXAMPLE 6.2.** *Consider the program fragment:*

$$P = l := l * h^2;$$

*We can compute  $\Pi(\text{id}, \text{Par})$  which 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  $\pi_l = \mathbb{Z}$  and if  $l \in 2\mathbb{Z} + 1$  then  $\pi_l = \{2\mathbb{Z}, 2\mathbb{Z} + 1\}$ . This means that  $\mathcal{P}(\prod_{l \in \mathbb{Z}} \mathcal{M}(\pi_l)) = \text{Par}$ . 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 [40] to the abstract non-interference case, we consider a semantic variation of what proposed in [40], which considers passive attackers only and a semantics observing only the initial and the final states of computations. We follow [40] in defining the information leaked by an equivalence relation transformer  $S[\eta, \rho]$  on  $\Sigma$  for each  $\eta, \rho \in uco(\wp(\mathbb{V}^L))$ :

$$s_1 S[\eta, \rho] s_2 \text{ iff } s_1^L \approx_\eta s_2^L \text{ and } (\forall \sigma, \delta \in \Sigma^+. \sigma_- = s_1 \wedge \delta_- = s_2 \Rightarrow \sigma_-^L \approx_\rho \delta_-^L)$$

where  $s_1, s_2 \in \Sigma$ . It is simple to verify that  $s_1 S[\eta, \rho] s_2$  iff  $\eta(s_1^L) = \eta(s_2^L)$  and  $\rho(\llbracket P \rrbracket(s_1)^L) = \rho(\llbracket P \rrbracket(s_2)^L)$ . This means that abstract robust declassification *a la* [40] is based on a formulation of non-interference which is stronger than abstract non-interference, since it may allow also deceptive flows, as we have in the narrow abstract non-interference case. This means that abstract robust declassification characterizes the information leaked in narrow abstract non-interference.

## 7 Discussion

In this paper we have weakened the standard notion of non-interference, making it parametric relatively to input/output observations made by abstract interpretations. This provides a number of systematic methods for mapping programs to properties according to their secure information flow. There are many points that deserves further discussion and research. (1) Possible applications of abstract non-interference are security program analysis, when we are interested in analyzing how much of information flows from confidential to public variables, and in automatic program certification, where we are interested in releasing certificates on secure information flows. In the latter case we prove that if  $P \models [\eta]P(\rho)$  [resp.  $P \models (\eta)P(\phi \rightsquigarrow \rho)$ ] then for all  $\beta \sqsupseteq \rho$ :  $P \models [\eta]P(\beta)$  [resp.  $P \models (\eta)P(\phi \rightsquigarrow \beta)$ ]. As shown in Section 5, certificates can be systematically derived by transforming abstract domains in the standard abstract interpretation framework. In particular the derivation of certificates for secure information flows strictly depends upon the way we model secure attackers. The canonical secure attacker (when it exists) represents therefore the most precise property certifying a program as secure. While this attacker always exists unique in the narrow case, we are only able to provide a fix-point representative attacker for the weaker abstract case. It would be interesting here to consider quantitative methods and metrics in order to enforce convergence towards an optimal fix-point under some estimation of abstract domain's structure and complexity. (2) Checking abstract non-interference corresponds to check whether an attribute independent analysis between H and L values is equivalent to its relational counterpart. This observation makes stronger the relation between static program analysis and language-based security. Abstract model checking can be used to check or derive certificates. The fact that in the narrow case the secret kernel of an abstract domain is mostly a partition is not a chance. We know from [17] and [29] that partitioning abstract domains, i.e.,  $\rho$  such that  $\rho = \mathcal{P}(\rho)$ , have the same precision as abstract model-checking. Therefore, abstract model checking can be used to check parametric non-interference without any additional loss of precision in the narrow case. (3) The standard notion of non-interference is based on

a strict independence between confidential and public data-flows. This means that it is possible to isolate the program's confidential data-flow by program slicing [37]. The definition of abstract non-interference weaken this independence, providing the base for weaker notions of program slicing relatively to observations specified by abstract domains. (4) Both notions of narrow and abstract non-interference can easily be generalized to multi-level, *a la* Denning, security policies. We remind that the purpose of a security policy is to declare which information flows are not permitted [21, 32]. For example in standard secrecy, downward flows are not allowed ( $H \not\rightarrow L$ ). If  $\mathbb{S}$  is the set of security classes for a set of variables, then a security policy  $\mathbb{P}$  is a set of flow constraints of the form  $S_1 \not\rightarrow S_2$ , with  $S_1, S_2 \in \mathbb{S}$ , which states that any information stored in variables of type  $S_1$  cannot flow into variables of type  $S_2$ . In particular, if  $(\mathbb{S}, \leq_{\mathbb{S}})$  is a lattice as in Denning's model [13], then the security policy is implicitly determined by avoiding upwards information flows:  $\mathbb{P} = \{ S_1 \not\rightarrow S_2 \mid S_1, S_2 \in \mathbb{S}, S_1 \not\leq_{\mathbb{S}} S_2 \}$ . Given a typing  $t \in Var \rightarrow \mathbb{S}$ , if for each  $S \in \mathbb{S}$ , we denote by  $v^S$  the projection of the state  $v \in \Sigma$  on the values of variables of type  $S$ , then we can generalize abstract non-interference to arbitrary multi-level security policies  $\mathbb{P}$  as follows: Let  $\{\rho_S\}_{S \in \mathbb{S}}$  be a family of abstract domains such that  $\rho_S \in uco(\wp(\mathbb{V}^S))$ . We say that a program  $P \in \text{IMP}$  is  $(\{\rho_S\}_{S \in \mathbb{S}})$ -Secret if for each  $S_1 \not\rightarrow S_2 \in \mathbb{P}$ ,  $v_1, v_2 \in \Sigma$ :

$$\forall S \neq S_1. \rho_S(v_1^S) = \rho_S(v_2^S) \Rightarrow \rho_{S_2}(\llbracket P \rrbracket((\prod_{S \in \mathbb{S}} \rho_S)(v_1))^{S_2}) = \rho_{S_2}(\llbracket P \rrbracket((\prod_{S \in \mathbb{S}} \rho_S)(v_2))^{S_2})$$

where for any  $S' \in \mathbb{S}$ :  $(\prod_{S \in \mathbb{S}} \rho_S)(v)^{S'} = \rho_{S'}(v^{S'})$ . (5) In this paper we have been mostly interested in checking program security relatively to the observation made by an attacker. This is justified by our interest in releasing security certificates for programs. The dual problem of deriving the most abstract attacker  $\rho$  such that  $P$  is not  $(\eta, \rho)$ -NSecret [resp.  $(\eta, \phi, \rho)$ -Secret] has not been considered here. This is an interesting problem for hackers, which may be interested in the least efforts (viz. the most abstract domain) necessary to disclose secrets. Unfortunately this problem may not have a solution. In general the concrete domain  $\text{id}$  can be secure, and therefore there are no closures that can make the program insecure. Moreover, while for making closures secure we merge sets of possible values, in this case we would have to split sets. This is a major problem when proving that a most abstract output domain exists making a given program insecure (the dual of secret kernels). Indeed, in most of the cases, any possible partition of a secure domain may reveal interference. Consider for instance the program in Example 5.12. In this example we proved that a closure that makes the program secure is  $\rho \stackrel{\text{def}}{=} \Upsilon(\{n\{2\}^{\mathbb{N}} \mid n \in 2\mathbb{N}+1\} \cup \{\{0\}\})$ . Then by the construction it is worth noting that if we distinguish the element  $\{1\}$ , the program is no more secure, namely by considering  $\rho' \stackrel{\text{def}}{=} \Upsilon(\{n\{2\}^{\mathbb{N}} \mid n \in 2\mathbb{N}+1 \setminus \{1\}\} \cup \{\{2\}^{\mathbb{N} \setminus \{0\}}\} \cup \{\{0\}, \{1\}\})$ . The same situation happens if we distinguish the element  $\{2\}$  instead, namely if we consider the abstract domain  $\rho'' \stackrel{\text{def}}{=} \Upsilon(\{n\{2\}^{\mathbb{N}} \mid n \in 2\mathbb{N}+1 \setminus \{1\}\} \cup \{\{2\}^{\mathbb{N} \setminus \{1\}}\} \cup \{\{0\}, \{2\}\})$ . The problem here is that  $\rho'$  and  $\rho''$  are unrelated, and their common abstraction still makes the program secure:  $\rho = \rho' \sqcup \rho''$ . (6) Our construction captures precisely the idea of modeling attackers as abstract interpretations, namely as processes acting as static program analyzers whose aim is to disclose confidential data. The language we consider is rather restricted, even though the idea of making non-interference parametric on input/output abstractions can be in principle extended to all programming languages having a formally defined semantics and for which abstract interpretation applies. In this respect we believe that (narrow) abstract non-interference and secrecy can be extended to non-deterministic and concurrent (synchronous or asynchronous) programs, as well as to programs with

probabilistic choice. The latter case is more delicate, as covert channels may influence secrecy. Probabilistic, termination, timing, and resource exhaustion channels require an adequate semantics modeling stochastic properties [27], termination [6], time properties or resource usage. This is particularly evident when we are interested in extending attackers, viewed as abstract interpretations, to other programming languages where the attacker can benefit from specific control features to improve its precision. In this case more concrete semantics, such as the natural semantics modeling terminating and non-terminating computations of a transition system [6], would provide a more adequate semantics-base.

## Acknowledgments

The interest in weakening non-interference by abstract interpretation came to our attention during the final meeting of the Italian-MURST Project *COFIN99: Automatic Program Certification by Abstract Interpretation*. We wish to thank the anonymous referees, Samir Genaim, and Massimo Merro for their helpful comments, and all the colleagues at the Dagstuhl seminar 03411 on *Language-Based Security* for pointing us relevant literature and comments on an early version of this paper. This work is partly supported by the Italian-MIUR Projects *COFIN02-CoVer: Constraint-based verification of reactive systems* and *FIRB: Abstract interpretation and model checking for the verification of embedded systems*.

## 8 References

- [1] BELL, D. E., AND BURKE, E. A software validation technique for certification: The methodology. Tech. Rep. MTR-2932, MITRE Corp. Bedford, MA, 1974.
- [2] BELL, D. E., AND LAPADULA, L. J. Secure computer systems: Mathematical foundations and model. Tech. Rep. M74-244, MITRE Corp. Bedford, MA, 1973.
- [3] BODEI, C., DEGANI, P., NIELSON, F., AND NIELSON, H. Static analysis for secrecy and non-interference in networks of processes. In *Proc. of 6th Internat. Conference on Parallel Computing Technologies : (PaCT'01)* (2001), vol. 2127 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 27–41.
- [4] CLARK, D., HANKIN, C., AND HUNT, S. Information flow for algol-like languages. *Computer Languages* 28, 1 (2002), 3–28.
- [5] COHEN, E. S. Information transmission in computational systems. *ACM SIGOPS Operating System Review* 11, 5 (1977), 133–139.
- [6] COUSOT, P. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theor. Comput. Sci.* 277, 1-2 (2002), 47,103.
- [7] COUSOT, P., AND COUSOT, R. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the 4th ACM Symp. on Principles of Programming Languages (POPL '77)* (1977), ACM Press, New York, pp. 238–252.
- [8] COUSOT, P., AND COUSOT, R. Constructive versions of Tarski's fixed point theorems. *Pacific J. Math.* 82, 1 (1979), 43–57.
- [9] COUSOT, P., AND COUSOT, R. Systematic design of program analysis frameworks. In *Conference Record of the 6th ACM*

- Symp. on Principles of Programming Languages (POPL '79)* (1979), ACM Press, New York, pp. 269–282.
- [10] COUSOT, P., AND COUSOT, R. Inductive definitions, semantics and abstract interpretation. In *Conference Record of the 19th ACM Symp. on Principles of Programming Languages (POPL '92)* (1992), ACM Press, New York, pp. 83–94.
- [11] DAVEY, B. A., AND PRIESTLEY, H. A. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, U.K., 1990.
- [12] DENNING, D. E. *Secure Information Flow in Computer Systems*. PhD thesis, Purdue University, 1975.
- [13] DENNING, D. E. A lattice model of secure information flow. *Communications of the ACM* 19, 5 (1976), 236–242.
- [14] DENNING, D. E., AND DENNING, P. Certification of programs for secure information flow. *Communications of the ACM* 20, 7 (1977), 504–513.
- [15] DI PIERRO, A., HANKIN, C., AND WIKLICKY, H. Approximate non-interference. In *Proc. of the IEEE Computer Security Foundations Workshop* (2002), IEEE Computer Society Press, pp. 1–17.
- [16] FILÉ, G., GIACOBazzi, R., AND RANZATO, F. A unifying view of abstract domain design. *ACM Comput. Surv.* 28, 2 (1996), 333–336.
- [17] GIACOBazzi, R., AND QUINTARELLI, E. Incompleteness, counterexamples and refinements in abstract model-checking. In *Proc. of The 8th Internat. Static Analysis Symposium, (SAS'01)* (2001), P. Cousot, Ed., vol. 2126 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 356–373.
- [18] GIACOBazzi, R., AND RANZATO, F. Refining and compressing abstract domains. In *Proc. of the 24th Internat. Colloq. on Automata, Languages and Programming (ICALP '97)* (1997), P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, Eds., vol. 1256 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, pp. 771–781.
- [19] GIACOBazzi, R., AND RANZATO, F. Optimal domains for disjunctive abstract interpretation. *Sci. Comput. Program.* 32, 1-3 (1998), 177–210.
- [20] GIACOBazzi, R., RANZATO, F., AND SCOZZARI, F. Making abstract interpretations complete. *J. of the ACM.* 47, 2 (2000), 361–416.
- [21] GOGUEN, J. A., AND MESEGUER, J. Security policies and security models. In *Proc. IEEE Symp. on Security and Privacy* (1982), IEEE Computer Society Press, pp. 11–20.
- [22] GOGUEN, J. A., AND MESEGUER, J. Unwinding and inference control. In *Proc. IEEE Symp. on Security and Privacy* (1984), IEEE Computer Society Press, pp. 75–86.
- [23] JOSHI, R., AND LEINO, K. R. M. A semantic approach to secure information flow. *Sci. Comput. Program.* 37 (2000), 113–138.
- [24] LANDAUER, J., AND REDMOND, T. A lattice of information. In *Proc. of the IEEE Computer Security Foundations Workshop* (1993), IEEE Computer Society Press, pp. 65–70.
- [25] LAUD, P. Semantics and program analysis of computationally secure information flow. In *In Programming Languages and Systems, 10th European Symposium On Programming, (ESOP'01)* (2001), vol. 2028 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 77–91.
- [26] LOWE, G. Quantifying information flow. In *Proc. of the IEEE Computer Security Foundations Workshop* (2002), IEEE Computer Society Press, pp. 18–31.
- [27] MONNIAUX, D. An abstract analysis of the probabilistic termination programs. In *Proc. of The 8th Internat. Static Analysis Symposium, (SAS'01)* (2001), vol. 2126 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 111–127.
- [28] ØRBÆK, P. Can you trust your data? In *Proc. of the 6th Internat. Joint Conf. CAAP/FASE, Theory and Practice of Software Development (TAPSOFT '95)* (1995), P. Mosses, M. Nielsen, and M. Schwartzbach, Eds., vol. 915 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 575–589.
- [29] RANZATO, F., AND TAPPARO, F. Making abstract model checking strongly preserving. In *Proc. of The 9th Internat. Static Analysis Symposium, (SAS'02)* (2002), M. Hermenegildo and G. Puebla, Eds., vol. 2477 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 411–427.
- [30] SABELFELD, A., AND MYERS, A. Language-based information-flow security. *IEEE J. on selected areas in communications* 21, 1 (2003), 5–19.
- [31] SABELFELD, A., AND SANDS, D. A PER model of secure information flow in sequential programs. *Higher-Order and Symbolic Computation* 14, 1 (2001), 59–91.
- [32] SCHNEIDER, F., MORRISSETT, G., AND HARPER, R. A language-based approach to security. In *In Informatics: 10 Years Back, 10 Years Ahead* (2000), vol. 2000 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 86–101.
- [33] SKALKA, C., AND SMITH, S. Static enforcement of security with types. In *Proc. Internat. Conference on Functional Programming (ICFP'00)* (2000), ACM Press, New York, pp. 254–267.
- [34] VOLPANO, D. Safety versus secrecy. In *Proc. of The 6th Internat. Static Analysis Symposium (SAS'99)* (1999), A. Cortesi and G. Filé, Eds., vol. 1694 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 303–311.
- [35] VOLPANO, D., AND SMITH, G. Verifying secrets and relative secrecy. In *Conference Record of the 27th ACM Symp. on Principles of Programming Languages (POPL '00)* (2000), ACM Press, New York, pp. 268–276.
- [36] VOLPANO, D., SMITH, G., AND IRVINE, C. A sound type system for secure flow analysis. *J. of Computer Security* 4, 2,3 (1996), 167–187.
- [37] WEISER, M. Program slicing. *IEEE Transactions on software engineering* 10, 4 (1984), 352–357.
- [38] WINSKEL, G. *The formal semantics of programming languages: an introduction*. MIT Press, 1993.
- [39] ZANOTTI, M. Security typings by abstract interpretation. In *Proc. of The 9th Internat. Static Analysis Symposium, (SAS'02)* (2002), M. Hermenegildo and H. Puebla, Eds., vol. 2477 of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 360–375.
- [40] ZDANCEWIC, S., AND MYERS, A. C. Robust declassification. In *Proc. of the IEEE Computer Security Foundations Workshop* (2001), IEEE Computer Society Press, pp. 15–23.