RESEARCH-ARTICLE

Abstract Interpretation-based Verification for Confidentiality: Information Hiding and Code Protection by Abstract Interpretation

ISABELLA MASTROENI, University of Verona, Verona, VR, Italy

MICHELE PASQUA, University of Verona, Verona, VR, Italy

Published: 11 February 2026
Online AM: 26 December 2025
Accepted: 01 December 2025
Revised: 26 September 2025
Received: 22 April 2025

Abstract Interpretation-based Verification for Confidentiality: Information Hiding and Code Protection by Abstract Interpretation

ISABELLA MASTROENI, Dept. of Computer Science, University of Verona, Verona, Italy
MICHELE PASQUA, Dept. of Computer Science, University of Verona, Verona, Italy

In modern computing systems, preventing sensitive information leakage is a crucial issue. Indeed, to deploy secure computing systems, data protection is an aspect that cannot be ignored. Many security requirements are adopted in this respect, such as *opacity* and *non-interference*. The first assures that the truth value of a predicate is masked during computation, while the second prevents confidential information is leaked through uncontrolled system components. Unfortunately, despite their simple intended meaning, confidentiality notions are quite difficult requirements to enforce. In fact, they are actually *hyperproperties*, and thus require enforcing mechanisms that reason on multiple executions at a time.

To develop effective verification and validation mechanisms for confidentiality notions, it is crucial to precisely characterize the requirements of system executions they dictate. In this article, we investigate the relation between *abstract non-interference* (a weakening of non-interference observing properties of data instead of concrete values) and opacity through the lens of *abstract interpretation*. By adopting such a holistic, abstract approach, we show how to formally characterize the *structure* of confidentiality notions and to compare them in terms of the constraints on system executions they impose and verification complexity. In addition, we show how *code obfuscation* can be restated as a confidentiality problem by defining a corresponding confidentiality notion that can possibly be enforced.

Finally, by exploiting the recently proposed static analysis approach for verifying non-interference, based on *hypersemantics*, we show how to verify abstract non-interference, therefore opacity and other security requirements. Based on *abstract interpretation*, this yields an effective mechanism to enforce a broad range of confidentiality notions.

CCS Concepts: • **Security and privacy** → **Security requirements**; • **Software and its engineering** → *Software verification;*

Additional Key Words and Phrases: Information flow control, opacity, non-interference, abstract interpretation, hyperproperties verification, static analysis, code protection, obfuscation

ACM Reference Format:
Isabella Mastroeni and Michele Pasqua. 2026. Abstract Interpretation-based Verification for Confidentiality: Information Hiding and Code Protection by Abstract Interpretation. *ACM Trans. Priv. Sec.* 29, 1, Article 12 (February 2026), 30 pages. https://doi.org/10.1145/3786347

This work was partially supported by the project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU; by PRIN2022PNRR "RAP-ARA" (PE6) - codice MUR: P2022HXNSC.
Authors' Contact Information: Isabella Mastroeni, Computer Science, University of Verona, Verona, Italy; e-mail: isabella.mastroeni@univr.it; Michele Pasqua (corresponding author), Computer Science, University of Verona, Verona, Italy; e-mail: michele.pasqua@univr.it.

ACM 2471-2566/2026/02-ART12
https://doi.org/10.1145/3786347

1 Introduction

The increasing reliance on computing systems for storing, processing, and transmitting sensitive information has made confidentiality a paramount concern in modern society. From personal health records and financial transactions to national security data and intellectual property, the need to protect sensitive information from unauthorized access and disclosure is critical. Confidentiality, a fundamental principle of information security, aims to ensure that information is accessible only to authorized individuals or entities, preventing its exposure to unintended recipients. In this context, it is crucial to develop computing systems that are resilient to attacks aimed at breaking confidentiality. Either to build secure-by-design systems or to check compliance of already deployed systems, a fundamental issue is to precisely model the security requirements, e.g., confidentiality, of interest. This requires a precise, ideally formal, definition of the requirements the system should fulfill. In literature, two security requirements are well-established and adopted in many contexts: opacity and non-interference.

Opacity [6, 39] is a security requirement assuring the protection of sensitive aspects concerning the behavior of computing systems [41]. Indeed, it is a natural choice in several scenarios, such as privacy, anonymity [25], and geo-privacy [41]. As Bryans et al. [6] have observed, "the essential idea of opacity is that a predicate is made opaque if an observer of the system will never be able to determine the truth of that predicate". In other words, a predicate is made opaque by a system (e.g., , a computer program) when, by observing the system's execution, it is not possible to deduce the predicate truth value (or both truth values in case of *symmetric opacity* [41]). Instead, *non-interference* [8] is a security requirement checking whether and/or how the system's *sensitive* inputs interfere with the *observable* results of system execution. It is a natural choice when information leaks are concerned, as confidential inputs should not flow into non-confidential outputs.

As a common ground between opacity and non-interference, we find that both requirements model a variation of a system's aspect as the effect of system execution, underlying a possible connection between the two notions. Indeed, Schoepe and Sabelfeld [41] intuitively observed that "a program satisfies non-interference if and only if it is opaque for almost all predicates". Unfortunately, this intuition does not allow us to understand how such notions can be compared in the way they tackle confidentiality. Hence, a naturally arising question is: do these notions lay at different levels of formalization, as shown in Reference [41], or do they provide a different perspective on the same security issue? And, in turn: can we use (part of) the verification mechanisms for opacity to verify non-interference (or vice versa)?

To tackle these questions, we can consider *abstract non-interference* [21], a weakening of non-interference observing properties, namely abstractions, of data instead of concrete values (e.g., observing the parity of numbers instead of their actual values). Indeed, opacity already deals with observations/abstractions of system executions (on system outputs only in Reference [6], on system inputs and outputs in Reference [41]). In particular, when we aim at protecting a property of system inputs w.r.t. an observation of system executions (the so-called *initial opacity* [6]), avoiding interference between the variation of the input property and executions observation, then we fall into abstract non-interference. Similarly, when we aim at protecting a property of system outputs (the so-called *final opacity* [6]), then we fall into (semantic) code obfuscation [14] (code is transformed to mask output data properties). Not surprisingly, the latter is strongly related to abstract non-interference [15].

When verification comes into play, things become complex in the context of opacity and (abstract) non-interference. Here, with *verification* is meant the general process of checking weather a system complies with a requirement, i.e., a formal description of what systems are allowed and are not allowed to do. Indeed, the verification process is quite standard when dealing with *trace properties*, i.e., requirements that can be checked inspecting *single* system executions. Problems

arise when the system's aspect of interest requires comparing more than one execution for being verified, as happens in security requirements such as opacity and non-interference. In Reference [7], *hyperproperties* were introduced to formalize precisely those requirements which are not *trace properties*. Usually, a system's behavior (semantics) is modeled as a set of executions, one for each possible input. In this setting, trace properties are sets (properties) of executions, while hyperproperties are sets (properties) of collections of executions, making verification of the latter more challenging. In Reference [34], this challenge has been faced to verify non-interference statically. Specifically, [34] introduced a sound static analysis for checking non-interference based on abstract interpretation [12].

In this article, we will show that opacity and abstract non-interference represent a different perspective on the same security issue, and this strong relation provides us the perfect field where moving the verification approach presented in Reference [34] from non-interference to abstract non-interference and opacity. Indeed, later in the article, we will provide a static verification mechanism for confidentiality notions, such as abstract non-interference and opacity, based on abstract interpretation. More in detail, we will first introduce a formalism for describing many flavors of abstract non-interference and opacity in the same setting, allowing us to compare the two notions formally. Once the connection between opacity and abstract non-interference is clear, we will identify a version of the latter that implies the former. Then, by adapting the verification mechanism of Reference [34] to abstract non-interference, we can provide a static analysis technique for soundly verifying opacity. In addition, we will leverage the proposed formalism to define confidentiality notions modeling code obfuscation as a combination of opacity and abstract non-interference. The latter can be useful to check whether a system is already obfuscating.

Structure of the article. In Section 2, we provide the background material, introducing abstract interpretation, as well as presenting the classic notions of opacity and abstract non-interference. At the end of the section, we introduce the reader to hyperproperties, and we depict a high-level view of the verification process for hyperproperties using abstract interpretation. Section 3 provides a general formalism for defining and comparing confidentiality notions, where opacity and abstract non-interference are instantiated and compared. In Section 4, we use again the proposed formalism to define confidentiality notions in code protection, specifically code obfuscation. In Section 5, we deep dive into the verification process for abstract non-interference and opacity. In particular, we investigate how to apply abstract interpretation, adapting the approach followed by [34], to analyze generalized abstract non-interference and symmetric opacity. Finally, Section 6 concludes the article and discusses the related work.

2 Background

2.1 Equivalences and Closure Operators

The *equivalence relations* on a set C form a lattice $\langle Eq(\mathsf{C}), \sqsubseteq, \sqcap, \sqcup, Id_{\mathsf{C}}, All_{\mathsf{C}}\rangle$ where: $Id_{\mathsf{C}} \triangleq \{(\mathsf{c},\mathsf{c}) \mid \mathsf{c} \in \mathsf{C}\}$ is the relation distinguishing all the elements in C (i.e., the bottom element); $All_{\mathsf{C}} \triangleq \mathsf{C} \times \mathsf{C}$ is the relation that cannot distinguish any elements in C (i.e., the top element); for all relations Q and R in $Eq(\mathsf{C})$, $\mathsf{Q} \sqsubseteq \mathsf{R} \triangleq \mathsf{Q} \subseteq \mathsf{R}$; $\mathsf{Q} \sqcap \mathsf{R} \triangleq \mathsf{Q} \cap \mathsf{R}$; $\mathsf{Q} \sqcup \mathsf{R} \triangleq (\mathsf{Q} \cup \mathsf{R})^+$, where $(\mathsf{Q} \cup \mathsf{R})^+$ is the transitive closure of $\mathsf{Q} \cup \mathsf{R}$. In the following, $[\mathsf{c}]_{\mathsf{R}} \triangleq \{\mathsf{c}' \mid \mathsf{c}\ \mathsf{R}\ \mathsf{c}'\}$ denotes the equivalence class of c under the relation R in $Eq(\mathsf{C})$.

An *upper closure operator* on a poset $\langle \mathsf{C}, \leq_{\mathsf{C}}\rangle$ is a function $\rho : \mathsf{C} \longrightarrow \mathsf{C}$ having the following properties.

- $\forall \mathsf{c}, \mathsf{c}' \in \mathsf{C}$, if $\mathsf{c} \leq_{\mathsf{C}} \mathsf{c}'$ then $\rho(\mathsf{c}) \leq_{\mathsf{C}} \rho(\mathsf{c}')$ (monotonicity);
- $\forall \mathsf{c} \in \mathsf{C}.\ \rho(\mathsf{c}) = \rho(\rho(\mathsf{c}))$ (idempotence);
- $\forall \mathsf{c} \in \mathsf{C}.\ \mathsf{c} \leq_{\mathsf{C}} \rho(\mathsf{c})$ (extensitivity).

A closure[1] $\rho : \mathsf{C} \longrightarrow \mathsf{C}$ is uniquely identifiable with the set of its fixpoints $\rho(\mathsf{C}) \triangleq \{\mathsf{c} \in \mathsf{C} \mid \rho(\mathsf{c}) = \mathsf{c}\}$. With a little abuse of notation, in the following, we will often use ρ to denote the closure as a function and to denote its set of fixpoints $\rho(\mathsf{C})$. If C is a complete lattice then $\langle uco(\mathsf{C}), \sqsubseteq, \sqcap, \sqcup, id, \top\rangle$ is the lattice of upper closure operators on C, where the top element is $\top \triangleq \lambda \mathsf{C}\,.\,\top_{\mathsf{C}}$ (here $\top_{\mathsf{C}}$ is the top element of C) and the bottom element is $id \triangleq \lambda \mathsf{c}\,.\,\mathsf{c}$. For every ρ and η in $uco(\mathsf{C})$ and for any subset $\{\rho_i\}_{i\in I}$ of $uco(\mathsf{C})$ we have that: $\rho \sqsubseteq \eta \triangleq \eta(\mathsf{C}) \subseteq \rho(\mathsf{C})$; $\bigsqcup_{i\in I} \rho_i \triangleq \bigcap_{i\in I} \rho_i$; and $\bigsqcap_{i\in I} \rho_i \triangleq \mathcal{M}(\bigcup_{i\in I} \rho_i)$. In particular, $\mathcal{M}$ is the operation of closing a domain by a concrete greatest lower bound, e.g., intersection on power domains. Given a closure ρ in $uco(\mathsf{C})$, $X \subseteq C$ is the set of fixpoints of ρ iff X is a *Moore-family* of C, i.e., $X = \mathcal{M}(X) \triangleq \{\bigwedge_{\mathsf{C}} Y \mid Y \subseteq X\}$, where $\bigwedge_{\mathsf{C}} \varnothing = \top_{\mathsf{C}}$ is in $\mathcal{M}(X)$.

When $\rho : \wp(\mathsf{C}) \longrightarrow \wp(\mathsf{C})$, meaning that ρ is an upper closure operator defined on a power set domain, then ρ is called *partitioning* [38] iff it is complemented[2], i.e., for all subsets $X \subseteq \mathsf{C}$, if X belongs to $\rho(\wp(\mathsf{C}))$ then also $\overline{X}$ belongs to $\rho(\wp(\mathsf{C}))$, where $\overline{X} \triangleq \mathsf{C} \smallsetminus X$ is the complement of X. The *atoms* of a closure ρ, denoted $Atom(\rho)$, are the fixpoints covering the bottom of the corresponding Moore family.

We recall that there exists an isomorphism between equivalence relations on a domain C and a subset of upper closure operators [26, 38] on $\wp(\mathsf{C})$. For each equivalence relation $\mathsf{R} \subseteq \mathsf{C} \times \mathsf{C}$ on a set C, we can define a closure Clo^{R} in $uco(\wp(\mathsf{C}))$ and, vice versa, from each closure ρ in $uco(\wp(\mathsf{C}))$ we can define an equivalence relation $Rel^{\rho} \subseteq \mathsf{C} \times \mathsf{C}$ on C. Consider a closure ρ in $uco(\wp(\mathsf{C}))$, we define $Rel^{\rho} \subseteq \mathsf{C} \times \mathsf{C}$ as: given c and c′ in C, $\mathsf{c}\ Rel^{\rho}\ \mathsf{c}' \triangleq \rho(\{\mathsf{c}\}) = \rho(\{\mathsf{c}'\})$. Proving that Rel^{ρ} is an equivalence relation is immediate and does not depend on the fact that ρ is a closure, but only on the fact that it is a function. Consider now an equivalence relation $\mathsf{R} \subseteq \mathsf{C} \times \mathsf{C}$, we define Clo^{R} in $uco(\wp(\mathsf{C}))$ as: given $X \subseteq \mathsf{C}$, $Clo^{\mathsf{R}}(X) \triangleq \bigcup_{\mathsf{c}\in X} [\mathsf{c}]_{\mathsf{R}}$. Note that Clo^{R} is partitioning by construction, and $[\mathsf{c}]_{\mathsf{R}}$, given c in C, are its atoms. In [26], the authors observed that for each R in $Eq(\mathsf{C})$ we have $Rel^{Clo^{\mathsf{R}}} = \mathsf{R}$. In the following, for the sake of readability, when a closure ρ in $uco(\wp(\mathsf{C}))$ is applied to a single element c in C, we abuse notation writing $\rho(\mathsf{c})$ instead of $\rho(\{\mathsf{c}\})$.

2.2 Abstract Interpretation

In the standard framework of *abstract interpretation* [12], abstract domains can be equivalently formulated either in terms of Galois connections or upper closure operators [13]. A pair of functions $\alpha : \mathsf{C} \longrightarrow \mathsf{A}$ and $\gamma : \mathsf{A} \longrightarrow \mathsf{C}$ on posets, denoted $\langle \mathsf{C}, \alpha, \gamma, \mathsf{A}\rangle$, forms an *adjunction* or a *Galois connection* (GC) if for any concrete element c in C and abstract element a in A we have $\alpha(\mathsf{c}) \leq_{\mathsf{A}} \mathsf{a}$ iff $\mathsf{c} \leq_{\mathsf{C}} \gamma(\mathsf{a})$. In this case, α (resp., γ) is the *left-* (resp., *right-*) *adjoint* of γ (resp., α), and it is additive (resp. co-additive), i.e., it preserves existing least upper bounds (resp., greatest lower bounds) of arbitrary non-empty sets. A *Galois insertion* (GI) is a Galois connection such that $\alpha \circ \gamma$, where $\circ$ denotes function composition from left to right[3], is the identity on A. Every Galois insertion $\langle \mathsf{C}, \alpha, \gamma, \mathsf{A}\rangle$ is uniquely identifiable with a closure $\gamma \circ \alpha$ in $uco(\mathsf{C})$. Since it is always possible to reduce a GC to a GI by eliminating redundant elements [13], we can identify any abstract domain of C with closure in $uco(\mathsf{C})$. In the following, given a concrete domain C, we will use the terms abstract domain, abstraction, and closure of C interchangeably.

Abstract interpretation is often used for abstracting computations, e.g., system/program semantics. Given a monotone function $f : \mathsf{C} \longrightarrow \mathsf{C}$ and an abstraction ρ in $uco(\mathsf{C})$ of C, we call *best correct approximation* (bca for short) of f in ρ the function $\rho \circ f \circ \rho$. When the domains of input (say C) and output (say D) of f are different, that is, $f : \mathsf{C} \longrightarrow \mathsf{D}$, then also the input abstraction η in $uco(\mathsf{C})$

[1] We will often just say 'closure' in place of 'upper closure operator'.
[2] Here, $\wp(\cdot)$ denotes the powerset operator, that is, $\wp(\mathsf{C})$ is the st of all subsets of C.
[3] We will often omit the composition operator for brevity.

may be different from the output abstraction ρ in $uco(\mathsf{D})$. In this case, the bca is $\rho \circ f \circ \eta$ [13]. Bca computation is always correct, i.e., for every c in C, $\rho \circ f(\mathsf{c}) \leq \rho \circ f \circ \eta(\mathsf{c})$, meaning that we may lose precision when computing on abstract inputs, but no erroneous information is added. The bca is *complete* when for every c in C, $\rho \circ f(\mathsf{c}) = \rho \circ f \circ \eta(\mathsf{c})$ [13, 23], i.e., when there is no loss of information when computing on abstract inputs (no loss of precision in abstract computations). Completeness is termed *local* [5] w.r.t. a concrete element c in C, when it is verified on that fixed element, i.e., when $\rho \circ f(\mathsf{c}) = \rho \circ f \circ \eta(\mathsf{c})$.

2.3 Opacity

Opacity is a security requirement that aims at protecting sensitive predicates on system executions [41], making them *opaque*. Intuitively, a predicate is opaque if, for any execution that satisfies the predicate, another execution does not satisfy the predicate and is indistinguishable by an attacker observing the system. Opacity is particularly useful in scenarios like privacy, anonimity [25] and *geo-privacy* [41]. For instance, opacity is the natural choice in the context of *geo-fences* for geo-tagging policies in sensitive areas [41].

In Reference [6] the authors provide an extensive study of the notion of opacity on labeled transition systems, but the same notion can be easily moved on any system semantics (e.g., program semantics). The original notion concerns the protection of a predicate that can be defined on initial states (*initial opacity*) or on final states (*final opacity*), or on the whole execution, namely on the execution trace (*total opacity*). When moving from transition systems to system semantics modeled by a function f, the last two notions can be collapsed into a single one (final opacity), since the results of the function can be either values or a trace of values (or any other denotation for modeling executions). We combine here the work of [6] with [41], where an input observation is added to a notion corresponding to initial opacity on program semantics and where *symmetric opacity* is introduced, i.e., opacity protecting both predicate truth values. We recall the original definitions of initial opacity [6], which we simply call Opacity as in [41]. When doing so, we use the notation we will formally introduce in Section 3. In particular, we assume program semantics to be modeled as functions $f : \mathbb{I} \to \mathbb{O}$ from an input set $\mathbb{I}$ to an output set $\mathbb{O}$. In this context, the restricted universal (resp., existential) quantification $\forall_{\mathrm{I}}\, x \,.\, \Pi$ (resp., $\exists_{\mathrm{I}}\, x \,.\, \Pi$) limits the scope of the variable x in the formula Π to elements belonging to $\mathrm{I} \subseteq \mathbb{I}$, that is, x is not quantified on the whole $\mathbb{I}$ but on the elements of its subset I only (see Section 3 for more details).

Definition 2.1 (Opacity). Let $\overset{\imath}{\sim}$ in $Eq(\mathbb{I})$ and $\overset{o}{\sim}$ in $Eq(\mathbb{O})$ be equivalence relations on input $\mathbb{I}$ and output $\mathbb{O}$, respectively. A predicate $\mathsf{P} \subseteq \mathbb{I}$ is *opaque* for $f : \mathbb{I} \to \mathbb{O}$ w.r.t. $\overset{\imath}{\sim}$ and $\overset{o}{\sim}$, denoted $\mathsf{P} \in \mathrm{Op}(f, \overset{\imath}{\sim}, \overset{o}{\sim})$, if:

$$\forall_{\mathsf{P}}\, x_1 \,.\, \exists_{\overline{\mathsf{P}}}\, x_2 \,.\, \left(x_1 \overset{\imath}{\sim} x_2 \wedge f(x_1) \overset{o}{\sim} f(x_2)\right), \tag{1}$$

where $\overline{\mathsf{P}} \triangleq \mathbb{I} \setminus \mathsf{P}$. When both $\mathsf{P}, \overline{\mathsf{P}} \in \mathrm{Op}(f, \overset{\imath}{\sim}, \overset{o}{\sim})$, P is said *symmetrically opaque*, written $\mathsf{P} \in \mathrm{SOp}(f, \overset{\imath}{\sim}, \overset{o}{\sim})$.

The writing in Equation (1) tantamount to say that for every input value v_1 that satisfies P (i.e., v_1 is in P) there exists another input value v_2 that does not satisfy P (i.e., v_2 is not in P or, equivalently, v_2 is in $\overline{\mathsf{P}}$) such that $\mathsf{v}_1 \overset{\imath}{\sim} \mathsf{v}_2$ and $f(\mathsf{v}_1) \overset{o}{\sim} f(\mathsf{v}_2)$. This is exactly the definition of Opacity given in [41].

The classic Opacity [6, 39] without the input observation means that for any input whose execution satisfies the predicate, there is an execution that does not satisfy the predicate. The way the input observation is introduced in [41] consists in requiring that for each possible input observation, the predicate must be opaque. It is a strengthening process that requires Opacity inside each equivalence class of input observations. In the following, we instantiate Opacity to show its usefulness in geo-privacy scenarios.

Example 2.2. Consider the *Program 1* of [41], that we list below. Variables hX and hY store the user's (potentially sensitive) location coordinates. Suppose that a clinic represents a sensitive location that occupies a rectangular area with corners $(200, 50)$ and $(400, 150)$. The program outputs on a public channel, and it aims to prevent privacy leakage. In particular, when the user is inside the clinic, the program outputs a default location $(100, 200)$ and otherwise the user's real location.

```
/*  Program 1  */
clinicXmin := 200; clinicXmax := 400;
clinicYmin := 50; clinicYmax := 150;
if (hX >= clinicXmin and hX <= clinicXmax and hY >= clinicYmin and hY <= clinicYmax) {
   out (100, 200);
} else {
   out (hX, hY);
}
```

Given a pair of natural numbers $(\mathsf{v}, \mathsf{v}')$, we denote with $(\mathsf{v}, \mathsf{v}')_{\vdash} = \mathsf{v}$ and $(\mathsf{v}, \mathsf{v}')_{\dashv} = \mathsf{v}'$ the projection on the first and the second element, respectively. The behavior of the program above can be modeled with the function $f_1 : \mathbb{I} \to \mathbb{O}$, where $\mathbb{I} = \mathbb{O} = \mathbb{N} \times \mathbb{N}$ and $\mathbb{N}$ are the natural numbers, as follows:

$$f_1(x) = \begin{cases} (100, 200) & \text{if } 200 \leq x_{\vdash} \leq 400 \text{ and } 50 \leq x_{\dashv} \leq 150 \\ x & \text{otherwise} \end{cases}$$

The predicate to protect is the one indicating whether the user is inside the clinic, i.e., the predicate $\mathsf{P} \subseteq \mathbb{I}$, defined as $\mathsf{P} \triangleq \{(\mathsf{v}, \mathsf{v}') \mid 200 \leq \mathsf{v} \leq 400 \wedge 50 \leq \mathsf{v}' \leq 150\} \subseteq \mathbb{I}$.

Consider the equivalence relations $\overset{o}{\sim}$ (on output) and $\overset{\iota}{\sim}$ (on input): $\overset{o}{\sim} \triangleq \{((\mathsf{v}, \mathsf{v}'), (\mathsf{v}, \mathsf{v}') \mid (\mathsf{v}, \mathsf{v}') \in \mathbb{O}\}$ and $\overset{\iota}{\sim} \triangleq \mathbb{I} \times \mathbb{I}$. In particular, $\overset{o}{\sim}$ models the fact that the attacker can see everything about outputs, while $\overset{\iota}{\sim}$ models the fact that the attacker cannot distinguish inputs. Then, we have that P is opaque (Definition 2.1) for f_1, i.e., $\mathsf{P} \in \mathrm{Op}(f_1, \overset{\iota}{\sim}, \overset{o}{\sim})$, namely for every x_1 in P there exists x_2 not in P (or, equivalently, in $\overline{\mathsf{P}}$) such that $x_1 \overset{\iota}{\sim} x_2$ and $f_1(x_1) \overset{o}{\sim} f_1(x_2)$. Indeed, for every x_1 in P (for instance, $x = (250, 100)$) take $x_2 = (100, 200)$, that is not in P. We have that $x \overset{\iota}{\sim} (100, 200)$ and $f_1(x) \overset{o}{\sim} f_1(100, 200)$. Indeed, $f_1(x) = (100, 200)$, for any x belonging to P, and $f_1(100, 200) = (100, 200)$. Hence, Program 1 protects the predicate P.

Let now consider the predicate $\overline{\mathsf{P}} = \mathbb{I} \setminus \mathsf{P}$. In this case, $\overline{\mathsf{P}}$ is not opaque for f_1, i.e., $\overline{\mathsf{P}} \notin \mathrm{Op}(f_1, \overset{\iota}{\sim}, \overset{o}{\sim})$. Indeed, consider $x_1 = (50, 250)$, which is in $\overline{\mathsf{P}}$. For every x_2 not in $\overline{\mathsf{P}}$, i.e., x_2 is in P, we have $f_1(x_2) = (100, 200)$. In this case, $f_1(x_1) = f_1(50, 250) = (50, 250) \not\overset{o}{\sim} (100, 200) = f_1(x_2)$. Hence, Program 1 does not protect $\overline{\mathsf{P}}$.

End Example.

Example 2.3. Consider the *Program 2* of [41], that we list below. It is similar to Program 1 except that a pseudo-random location outside the clinic is returned whenever the user is inside it. When observing the output, the attacker cannot deduce whether it is the user's real location outside the clinic or a (pseudo) randomly generated location while the user is inside.

```
/*  Program 2  */
clinicXmin := 200; clinicXmax := 400;
clinicYmin := 50; clinicYmax := 150;
if (hX >= clinicXmin and hX <= clinicXmax and hY >= clinicYmin and hY <= clinicYmax) {
   randomize(x);
   randomize(y);
   while (x >= clinicXmin and x <= clinicXmax and hY >= clinicYmin and hY <= clinicYmax) {
     randomize(x);
     randomize(y);
   }
   out (x, y);
} else {
   out (hX, hY);
}
```

The behavior of the program can be modeled with the function $f_2 : \mathbb{I} \to \mathbb{O}$, where $\mathbb{I} = \mathbb{O} = \mathbb{N} \times \mathbb{N}$, as follows:

$$\text{let} Rnd \triangleq \text{rand}(\mathbb{N} \times \mathbb{N} \setminus [200..400] \times [50..150]) \text{ in}$$
$$f_2(x) = \begin{cases} Rnd & \text{if } 200 \leq x_{\vdash} \leq 400 \text{ and } 50 \leq x_{\dashv} \leq 150 \\ x & \text{otherwise} \end{cases}$$

Here, $\text{rand}(N)$ returns a pseudo-random element of the set N, while $[min..max]$ is the set of numbers in $\mathbb{N}$ between min and max (included), following the usual less than or equal ordering between naturals. Note that, since rand() uses a pseudo-random number generator with fixed seed, hence deterministic, f_2 is indeed a function.

Consider the equivalence relations $\overset{\iota}{\sim}$ and $\overset{o}{\sim}$ and the predicate P of Example 2.2. In this case, both P and $\overline{\mathsf{P}}$ are opaque (Definition 2.1) for f_2, since the attacker cannot realize whether a user is inside or outside the clinic. Hence, P is symmetrically opaque, i.e., $\mathsf{P} \in \text{SOp}(f_2, \overset{\iota}{\sim}, \overset{o}{\sim})$. In this case, Program 2 makes it more difficult for an attacker to learn anything about the user's location, since both the sensitive predicate (whether the user is inside the clinic) and its negation are protected.

End Example.

As far as verification is concerned, we have to remark that opacity is not a trace property since it requires comparing at least two system executions. Therefore, standard approaches are either too imprecise or not applicable, in the case of opacity verification. Indeed, opacity is a hyperproperty [7], that requires hyper-level verification approach [1, 34], e.g., a hyperanalysis, as introduced in Subsection 2.6.

2.4 Abstract Non-Interference

In Reference [21] the classic notion of *non-interference* [8], checking whether and/or how the sensitive input interferes with the observable result of a function, e.g., a program semantics, has been generalized, allowing to state *when* to check interference, *how* an attacker observes the result and *what* has to be protected. The central idea is to track interference between properties (or abstractions) of data instead of concrete values, where properties/abstractions are modeled by means of upper closure operators. In Reference [21], the most general (so far) version of non-interference, there called *abstract non-interference*, has been given in terms of the best correct approximation of the function to be checked. Similarly to Opacity, we use the notation we will formally introduce in Section 3 to recall abstract non-interference as defined in Reference [21].

Definition 2.4 (Abstract Non-Interference). Let ϕ, η in $uco(\wp(\mathbb{I}))$ be abstractions on input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. A function $f : \mathbb{I} \to \mathbb{O}$ satisfies *Abstract Non-Interference* (ANI in short) w.r.t. ϕ, η in input and ρ in output, written $f \models \text{ANI}[\langle \phi, \eta \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$, if [4]:

$$\forall x_1 \,.\, \forall x_2 \,.\, (\phi(x_1) = \phi(x_2) \Rightarrow \rho f \eta(x_1) = \rho f \eta(x_2)). \tag{2}$$

The input observation ϕ is a weakening fixing when (i.e., for which inputs) we ask for non-interference. It behaves as a selection function, namely, we check non-interference only for the inputs having the property ϕ. Instead, ρ is the observation of the computation; again, it is a weakening, meaning that we can observe the property ρ of results and not concrete values. Finally, η models an input data abstraction whose variation has not to be observable, that is, variety η on inputs must not be conveyed to outputs. This is the input property protected when ANI holds [21]. Note that ANI, in how it is formalized, can only model the protection of *input* data properties, not output data property.

[4]Recall that, here we use the more concise notation $\phi(x_1)$ in place of $\phi(\{x_1\})$.

Example 2.5. Consider a predicate $\mathsf{P} \subseteq \mathbb{I}$ on $\mathbb{I} = \mathbb{N} \times \mathbb{N}$ (pairs of natural numbers) and let $\mathbb{O} = \mathbb{I}$. For instance, P could be the predicate protecting whether the user is inside the clinic of Example 2.2. Consider the following closures on $\wp(\mathbb{I}) = \wp(\mathbb{O})$, i.e., data abstractions[5]: $\phi = \{\mathbb{I}\}$, $\rho = \wp(\mathbb{I})$ and $\eta = \{\varnothing, \mathsf{P}, \overline{\mathsf{P}}, \mathbb{I}\}$. Here, ϕ and ρ are the top and bottom abstractions, respectively: ϕ cannot distinguish any element of $\mathbb{I}$, while ρ can distinguish every element of $\mathbb{I}$. In particular, $\phi(\{(\mathsf{v}_1, \mathsf{v}_2)\}) = \mathbb{I}$ and $\rho(\{(\mathsf{v}_1, \mathsf{v}_2)\} = \{(\mathsf{v}_1, \mathsf{v}_2)\})$, for every $(\mathsf{v}_1, \mathsf{v}_2) \in \mathbb{I}$. Furthermore, $\eta(\{(\mathsf{v}_1, \mathsf{v}_2)\}) = \mathsf{P}$ if $(\mathsf{v}_1, \mathsf{v}_2)$ belongs to P, and $\eta(\{(\mathsf{v}_1, \mathsf{v}_2)\}) = \overline{\mathsf{P}}$ otherwise.

Consider ANI for the function f_2 of the *Program 2* in Example 2.3. We have that $f_2 \models \mathsf{ANI}[\langle\phi, \eta\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$. Indeed, the premise of the implication in Equation (2) is always trivially true, since $\phi(x_1) = \phi(x_2)$ holds for any x_1 and x_2 in $\mathbb{I}$ when $\phi = \{\mathbb{I}\}$. Then, when $\eta(x_1) = \eta(x_2)$ the whole implication is true, being f_2 a function. Conversely, when $\eta(x_1) \neq \eta(x_2)$, we have that x_1 is in P and x_2 is in $\overline{\mathsf{P}}$, or vice versa. But, by definition of f_2, we have that both[6] $f_2(\mathsf{P})$ and $f_2(\overline{\mathsf{P}})$ are equal to $\mathbb{N} \times \mathbb{N} \setminus [200..400] \times [50..150]$. Hence, the whole implication is true, and *Program 2* satisfies ANI.

Note that, $f_1 \models \mathsf{ANI}[\langle\phi, \eta\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$ does not hold for f_1 (of the *Program 1*) defined in Example 2.2. Consider the following counterexample. Take $x_1 = (300, 100)$, which is in P, and $x_2 = (20, 20)$, which is in $\overline{\mathsf{P}}$. Then, $\phi(x_1) = \phi(300, 100) = \mathbb{I} = \phi(20, 20) = \phi(x_2)$, $\eta(x_1) = \eta(300, 100) = \mathsf{P}$, $\eta(x_2) = \eta(20, 20) = \overline{\mathsf{P}}$, $\rho f_1 \eta(x_1) = \rho f_1(\mathsf{P}) = \{(100, 200)\}$ and $\rho f_1 \eta(x_2) = \rho f_1(\overline{\mathsf{P}}) = \mathbb{N} \times \mathbb{N} \setminus [200..400] \times [50..150]$. Hence, $\phi(x_1) = \phi(x_2)$ but $\rho f \eta(x_1) \neq \rho f \eta(x_2)$, and *Program 1* does not satisfy ANI.

End Example.

To tune non-interference for coping with different information-flow protection requirements, several flavors of Abstract Non-Interference have been defined in literature [20, 21, 30]. We report here the most relevant for our work.

A *narrow* version of ANI fixes under which input and output observations we have not to detect interference, resulting in a property of an abstract observation of the concrete executions (the abstraction η is not present).

Definition 2.6 (Narrow Abstract Non-Interference). Let ϕ in $uco(\wp(\mathbb{I}))$ be an abstraction on input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. A function $f : \mathbb{I} \to \mathbb{O}$ satisfies *Narrow Abstract Non-Interference* (NANI in short) w.r.t. ϕ in input and ρ in output, written $f \models \mathsf{NANI}[\phi, \rho]$, if:

$$\forall x_1 \,.\, \forall x_2 \,.\, (\phi(x_1) = \phi(x_2) \Rightarrow \rho f(x_1) = \rho f(x_2))$$

It is easy to note that NANI is a special case of ANI, indeed $f \models \mathsf{NANI}[\phi, \rho]$ iff $f \models \mathsf{ANI}[\langle\phi, id\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$. A more interesting notion is a *declassified* version of ANI, fixing which input data property must be protected, namely, which property must not affect the output observation. Note that, also in ANI, the property η is protected; what changes here is the way computation is observed.

Definition 2.7 (Declassified Abstract Non-Interference). Let η in $uco(\wp(\mathbb{I}))$ be an abstraction on input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. A function $f : \mathbb{I} \to \mathbb{O}$ satisfies *Declassified Abstract Non-Interference* (DANI in short) w.r.t. η in input and ρ in output, written $f \models \mathsf{DANI}[\eta, \rho]$, if:

$$\forall x_1 \,.\, \forall x_2 \,.\, (\eta(x_1) \neq \eta(x_2) \Rightarrow \rho f(x_1) = \rho f(x_2))$$

In this case, DANI is not a special case of ANI for any *concrete* function f. Still, we can correlate the two notions when we consider *abstract* functions of the form $f \circ \eta$.

Proposition 2.8. *Let η in $uco(\wp(\mathbb{I}))$ be an abstraction on input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. Then we have that $f \circ \eta \models \mathsf{DANI}[\eta, \rho]$ iff $f \circ \eta \models \mathsf{ANI}[\langle\top, \eta\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$.*

[5]Recall that, a closure can be uniquely identified by the set of its fixpoints.

[6]In the following, we abuse notation denoting with f a function in $\mathbb{I} \to \mathbb{O}$ and its additive lift to sets, defined as: $f(X) \triangleq \{f(x) \mid x \in X\}$, given $X \subseteq \mathbb{I}$.

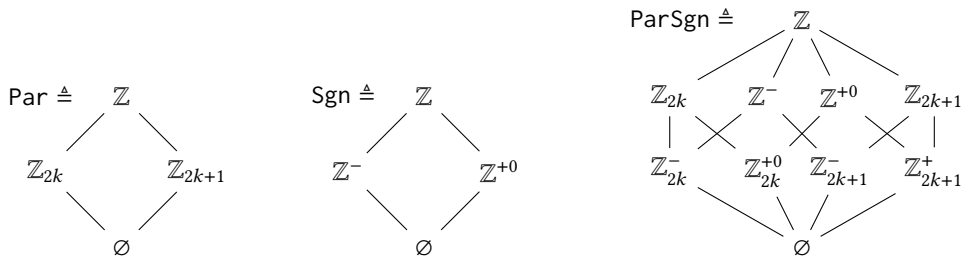

Fig. 1. From left to right: Parity domain, Sign domain, and the reduced product between Parity and Sign domains.

2.5 Program Semantics and Static Analysis

With program verification, we mean the general process of checking whether a system complies with a requirement, i.e., , (formal) description of what the system is allowed and not allowed to do. When considering programs as systems, the behavior (semantics) of a program P is usually modeled as *the observation* of a set of executions at a fixed degree of precision, one for each possible input for P. The semantics of a program is usually computed inductively on the program syntax by means of a language interpreter, computing the intended and observable behavior for each step of computation determined by the program instructions.

In this setting, the set-theoretic *semantics* of a program P manipulating data in $\mathbb{D}$, is a subset of $\mathbb{I} \times \mathbb{O}$, where $\mathbb{I} = \mathbb{D}$ is usually the domain of program inputs, while $\mathbb{O}$ corresponds to the denotation used to observe the program behavior. In standard cases, the semantics can be the small-step trace semantics, denoted $[\![P]\!]^{\mathcal{T}}$, with $\mathbb{O} = \mathbb{D}^*$ (i.e., each input is associated with a small-step finite execution trace) or the I/O semantics, denoted $[\![P]\!]$, with $\mathbb{O} = \mathbb{D}$ (i.e., each input is associated with the computed output). In the following, we abuse notation by indicating with the same term a semantics in its set-theoretic form (e.g., $[\![P]\!] \subseteq \mathbb{D} \times \mathbb{D}$) and its equivalent functional form (e.g., $[\![P]\!] : \mathbb{D} \to \mathbb{D}$).

When dealing with program verification and analysis, a program semantics is usually lifted to sets of inputs, in order to analyze properties of executions.

For decidability reasons, such a (concrete) semantics can be approximated by building an *abstract* interpreter [12, 13]. In this case, we can fix an observation (i.e., an abstraction) ρ modeling what we aim to observe of program computations. Then, we can build the interpreter computing the program on properties of data instead of concrete values, namely on subsets of $\mathbb{D}$. Specifically, we consider computations on $\rho(\wp(\mathbb{D}))$ yielding an abstract semantics $[\![P]\!]^{\rho} : \rho(\wp(\mathbb{D})) \to \rho(\wp(\mathbb{D}))$. Since we will use this construction only in a few examples, we avoid technical details and show the meaning with a simple example. Consider the following program P, and consider the classic parity `Par` and sign `Sgn` abstractions on numeric values reported in Figure 1 (on the left and center, respectively).

```
in num;
if (num % 2 == 0) {
   res := num * num;
} else {
   res := -(num * num);
}
out res;
```

The program manipulates integer numbers, thus $\mathbb{D} = \mathbb{Z}$. For every even integer $x \in \mathbb{D}$, we have that $[\![P]\!](x) = x^2$, while for every odd integer $x \in \mathbb{D}$, we have that $[\![P]\!](x) = -x^2$. That is, the I/O semantics of the program is $[\![P]\!] = \left\{(x, x^2) \;\middle|\; x \in \mathbb{Z}_{2k}\right\} \cup \left\{(x, -x^2) \;\middle|\; x \in \mathbb{Z}_{2k+1}\right\}$.

Let us compute the (abstract) semantics $[\![P]\!]^{\mathsf{Sgn}} : \mathsf{Sgn} \to \mathsf{Sgn}$ evaluating in the abstract the sign of expressions, only knowing the sign of input data. In this case, information may be lost, for instance, when dealing with sums or tests not implying any sign information, such as parity (the parity of a number does not provide any information on its sign). In particular, we have both even and odd numbers among positive and negative numbers, hence we cannot decide the test at the first line of program P when we only know the sign of the input. This means that we have to conservatively merge both branches: in the true branch `res` is always positive, while in the false one it is always negative, meaning that $[\![P]\!]^{\mathsf{Sgn}}(\mathbb{Z}^{+0}) = \mathbb{Z}$ and $[\![P]\!]^{\mathsf{Sgn}}(\mathbb{Z}^{-}) = \mathbb{Z}$. That is, for any $X \in \mathsf{Sgn}$ in input, the result is always *any sign* (the sign is unknown), modeled by $\mathbb{Z}$. On the other hand, when computing $[\![P]\!]^{\mathsf{Par}} : \mathsf{Par} \to \mathsf{Par}$, if the input is even ($\mathbb{Z}_{2k}$), we can say that `res` is even, and the same for the odd input ($\mathbb{Z}_{2k+1}$). Formally, $[\![P]\!]^{\mathsf{Par}}(\mathbb{Z}_{2k}) = \mathbb{Z}_{2k}$ and $[\![P]\!]^{\mathsf{Par}}(\mathbb{Z}_{2k+1}) = \mathbb{Z}_{2k+1}$.

Finally, consider the reduced product of the parity and sign domains, depicted in Figure 1 (on the right), keeping track of both parity and sign of variables. When computing the (abstract) semantics $[\![P]\!]^{\mathsf{ParSgn}} : \mathsf{ParSgn} \to \mathsf{ParSgn}$, then we know that if the input is even (either positive or negative), then the result is positive even, while if it is odd (either positive or negative), the result is negative odd. That is, $[\![P]\!]^{\mathsf{ParSgn}}(X) = \mathbb{Z}^{+0}_{2k}$ when $\mathsf{Par}(X) = \mathbb{Z}_{2k}$, and $[\![P]\!]^{\mathsf{ParSgn}}(X) = \mathbb{Z}^{-}_{2k+1}$ when $\mathsf{Par}(X) = \mathbb{Z}_{2k+1}$. For the sake of simplicity, we considered here as abstract semantics $[\![P]\!]^{\rho}$ the abstract I/O computation (returning the property at the exit program point), but, in general, we can compute the data abstract property at each program point [13].

2.6 Hyper Static Analysis

In Reference [7], *hyperproperties* were introduced to formalize those requirements which are not *trace properties*, i.e., that cannot be checked observing *single* system executions. In this setting, hyperproperties are sets of sets of executions, and no more sets of executions (like trace properties). Hence, a program P satisfies a trace property P iff $[\![P]\!]^{\mathcal{T}} \subseteq \mathsf{P}$, and it satisfies a hyperproperty HP iff $[\![P]\!]^{\mathcal{T}} \in \mathsf{HP}$ or, equivalently, iff $\{[\![P]\!]^{\mathcal{T}}\} \subseteq \mathsf{HP}$. The same holds when we observe the I/O only instead of the whole execution trace. That is, the same lift can be done for any chosen execution denotation.

In Reference [33], the authors define k-bounded subset-closed hyperproperties: these requirements can be refuted just by exhibiting a counterexample set consisting in at most k executions. It turns out that many interesting hyperproperties are k-bounded, like Abstract Non-Interference (which is 2-bounded). Conversely, Opacity is not a subset-closed hyperproperty, hence its verification is particularly challenging. In Section 5, we will see how we can approximate the verification of Opacity, using a verification mechanism for Abstract Non-Interference. The latter is obtained by adapting the abstract interpretation-based verification mechanism for non-interference proposed in Reference [34]. We report here the background needed to understand the constructions in Section 5.

Hypersemantics. In the rest of the section, we briefly describe the principles of hyperproperties verification [33]. From now on, we consider as concrete semantics the I/O semantics $[\![P]\!]$ of a program P. As outlined in Reference [33], in order to verify hyperproperties, we need to lift the concrete semantics to sets of sets, namely, we need a *hypersemantics* $\{\!|P|\!\}_h : \wp(\wp(\mathbb{D})) \to \wp(\wp(\mathbb{D}))$. In Reference [33], the authors show how to define a *correct* hypersemantics, starting from a given concrete semantics. Correct here means that it contains the concrete semantics, namely $[\![P]\!](X) \triangleq \left\{ [\![P]\!](x) \;\middle|\; x \in X \right\} \in \{\!|P|\!\}_h(\{X\})$, for any $X \subseteq \mathbb{D}$. With an over-approximation of a (correct) hypersemantics, we can soundly verify hyperproperties. The over-approximation is given by an *abstract* hypersemantics, computing on a suitable abstract domain, exploiting *abstract interpretation* [12]. Indeed, abstract interpretation is a framework for approximating programs semantics, which is sound (i.e., correct) by construction. An actual verification mechanism requires a (computable) abstract semantics computed on a (machine representable) abstract domain. In particular, we first

have to define an abstract domain of computation $\langle \mathbb{D}^\natural, \subseteq^\natural \rangle$, forming a Galois connection with the concrete domain of computation $\langle \wp(\wp(\mathbb{D})), \subseteq \rangle$, namely[7]:

$$\langle \wp(\wp(\mathbb{D})), \subseteq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \mathbb{D}^\natural, \subseteq^\natural \rangle$$

Then, with an over-approximation $\{\!|P|\!\}_h^\natural : \mathbb{D}^\natural \to \mathbb{D}^\natural$ of the concrete hypersemantics, i.e., such that $\alpha(\{\!|P|\!\}_h(\{X\})) \subseteq^\natural \{\!|P|\!\}_h^\natural(\alpha(\{X\}))$, for any $X \subseteq \mathbb{D}$, and an under-approximation $\mathsf{HP}^\natural$ of the hyperproperty to check, i.e., such that $\gamma(\mathsf{HP}^\natural) \subseteq \mathsf{HP}$, we can verify HP totally in the abstract domain [34]:

$$\{\!|P|\!\}_h^\natural(\alpha(\{\mathbb{D}\})) \subseteq^\natural \mathsf{HP}^\natural \text{ implies } \{[\![P]\!]\} \subseteq \mathsf{HP}$$

3 Generalizing Confidentiality Notions

In this section, we show how existing notions of confidentiality on systems, in particular Abstract Non-Interference and Opacity, can be generalized (abstracted) and how they correlate. Without loss of generality, from now on we consider programs as the systems of interest. The idea is to build up what we will call confidentiality properties in terms of simpler execution requirements, that is, constraints on program executions. This process has the two-fold objective of better understanding the *essence* of known confidentiality notions and, contingently, of highlighting their mutual correlation.

In the following, we provide a general process of defining semantic (program) properties, that we then instantiate to confidentiality. Let $\mathcal{S} \triangleq \mathbb{I} \to \mathbb{O}$ be the set of semantic functions of the programs under consideration. An *execution requirement* of $\mathcal{S}$ is a formula π defined on single functions (i.e., programs) in $\mathcal{S}$. Formally, when π contains occurrences of a free variable, say x, and the function symbol f, then π stipulates a constraint for the (semantic) function f in $\mathcal{S}$ executed on x when x is substituted with an input value in $\mathbb{I}$. More generally, π may contain more than one free variable, stipulating a constraint on multiple program executions (dubbed *hyperproperties* [7] in literature). We assume an underlying first-order logic with equality where f is the only uninterpreted function symbol and variables are interpreted in $\mathbb{I}$. We denote with $\mathsf{free}(\pi) = \{x_1, \ldots, x_n\}$ the set of variables appearing free in π and we denote by $x \in \mathsf{free}(\pi)$ the fact the x is among such variables. Given an execution requirement π, having x as a free variable, we denote with $\pi[{}^{\mathsf{v}}/_{x}]$ the fact that the variable x has been substituted with the value v belonging to $\mathbb{I}$ inside π. We then define a *semantic formula* Π inductively on its free variables as follows.

- An execution requirement π is a semantic formula.
- If Π is a semantic formula and $x \in \mathsf{free}(\Pi)$, then the following are semantic formulae: $\Pi[{}^{\mathsf{v}}/_{x}]$, with v belonging to $\mathbb{I}$; $\exists x\,.\,\Pi$; and $\forall x\,.\,\Pi$.

A semantic formula Π is *closed* when it does not contain free variables, namely when $\mathsf{free}(\Pi) = \varnothing$. In this setting, we define a *semantic property* as a closed semantic formula, and we denote with $f \models \Pi$ the fact that the function f in $\mathcal{S}$ satisfies (i.e., it is a model of) the semantic property Π. Intuitively: f satisfies a semantic property of the form $\Pi[{}^{\mathsf{v}}/_{x}]$ if f fulfills the constraints stipulated by Π when a given value v belonging to $\mathbb{I}$ is substituted to x inside Π; f satisfies a semantic property of the form $\exists x\,.\,\Pi$ if there exits at least one value v belonging to $\mathbb{I}$ that substituted to x inside Π makes f fulfilling the constraints stipulated by Π; and f satisfies a semantic property of the form $\forall x\,.\,\Pi$ if every value v belonging to $\mathbb{I}$ substituted to x inside Π makes f fulfilling the constraints stipulated by Π.

We call *local semantic property* a closed semantic formula in which universal ($\forall$) and existential ($\exists$) quantifiers do not appear. Formally, given an execution requirement π, with $\mathsf{free}(\pi) = \{x_1, \ldots, x_n\}$,

[7]The following is an alternative representation for Galois connections

we have that $\Pi \triangleq \pi[{}^{v_1,\ldots,v_n}/_{x_1,\ldots,x_n}]$, for some values $v_1, \ldots, v_n$ belonging to $\mathbb{I}$, is a local semantic property.

In addition, we define a notion of *restricted semantic formula*, where variables are interpreted in $I \subseteq \mathbb{I}$ in place of $\mathbb{I}$. This intuitively means that the constraints imposed by Π should hold for a subset of input values only. Of course, the restriction makes sense only when Π contains free variables. Formally, given a semantic formula Π with $x \in \mathsf{free}(\Pi)$, we have that $\exists_I x \,.\, \Pi$ and $\forall_I x \,.\, \Pi$ are restricted semantic formulae whose intended meaning is as follows: f satisfies $\exists_I x \,.\, \Pi$ if there exits at least one value v belonging to $I \subseteq \mathbb{I}$ that substituted to x inside Π makes f fulfilling the constraints stipulated by Π; and f satisfies $\forall_I x \,.\, \Pi$ if every value v belonging to $I \subseteq \mathbb{I}$ substituted to x inside Π makes f fulfilling the constraints stipulated by Π.

In this work, we will focus on semantic properties modeling confidentiality, which we call *confidentiality properties*, but the schema above can also be applied to other contexts. For instance, precision in static analysis is formalized by a notion of *completeness* [23] w.r.t. an input observation η and an output observation ρ, and it can be defined in terms of the execution requirement $\pi_c \triangleq \rho f \eta(x) = \rho f(x)$, with $\mathsf{free}(\pi_c) = \{x\}$. Then, the classic notion of Completeness [13] is the semantic property $\Pi_{uc} \triangleq \forall x \,.\, \pi_c$, while Local Completeness [5] on the input value v belonging to $\mathbb{I}$ is the (local) semantic property[8] $\Pi_{lc} = \pi_c[{}^{v}/_{x}]$. Note that, the semantic property $\Pi_{ec} = \exists x \,.\, \pi_c$ asks a different question from Local Completeness. The former checks completeness for a given, fixed input value, while the latter checks whether such an input value exists or not, and, to the best of our knowledge, it has not been investigated in the literature.

3.1 Generalizing Abstract Non-Interference

We now generalize Abstract Non-Interference, in that we define a confidentiality property capturing the Abstract Non-Interference notions introduced before as special cases. To do so, we introduce a new general notion of Abstract Non-Interference, modeling all different confidentiality aspects considered so far into a unique setting. Following the schema described above, we build a confidentiality property by identifying the execution requirement prerogative of the most general version of ANI and then quantifying the free variables.

In the general case, we can encapsulate into Abstract Non-Interference three confidentiality dimensions: *(what)*, the property of input values to protect; *(where)*, the input values for which non-interference should hold; and *(who)*, the observation capabilities (e.g., of an attacker) on output values. Formally: let η in $uco(\wp(\mathbb{I}))$ be an abstraction modeling the property of input values that we want to protect, namely whose variation has not to affect the output observation *(what)*; let ϕ in $uco(\wp(\mathbb{I}))$ be an abstraction fixing for which input values we aim at checking (abstract) non-interference *(where)*; and let ρ in $uco(\wp(\mathbb{O}))$ be an abstraction characterizing what is observable about output values *(who)*. The *(where)* abstraction can model declassification since we do not care whether the variation between values with the same ϕ abstraction yields interference in the output observation.

Being Abstract Non-Interference a 2-bounded hyperproperty [33], the execution requirement must characterize the pairs of executions of $f : \mathbb{I} \to \mathbb{O}$ that can be classified as non-interfering:

$$\pi_{\mathsf{ANI}} \triangleq \phi(x_1) = \phi(x_2) \wedge \eta(x_1) \neq \eta(x_2) \Rightarrow \rho f(x_1) = \rho f(x_2)$$

The above requirement has two free variables hence we can define different confidentiality properties starting from π_{ANI} by adding a quantification on its free variables x_1 and x_2 or by instantiating them with a value.

[8] It can also be defined, equivalently, in terms of the restricted semantic property $\forall_{\{v\}} x \,.\, \pi_c$.

First, we can fix two specific executions, having input values v_1 and v_2 in $\mathbb{I}$, obtaining the local confidentiality property $\pi_{\mathsf{ANI}}[{}^{\mathsf{v}_1,\mathsf{v}_2}/_{x_1,x_2}]$. This is not a very significant notion, simply requiring that two fixed computations are not exposing any (abstract) interference.

Then, we can fix one execution and quantify the other. An outer existential quantification would have no particular meaning. Indeed, given an input value v in $\mathbb{I}$, the confidentiality property $\exists x_1 \,.\, \pi_{\mathsf{ANI}}[{}^{\mathsf{v}}/_{x_2}]$, unfolded as $\exists x_1 \,.\, (\phi(x_1) = \phi(\mathsf{v}) \wedge \eta(x_1) \neq \eta(\mathsf{v}) \Rightarrow \rho f(x_1) = \rho f(\mathsf{v}))$, is satisfied by any f when $\phi \neq \top$ or $\eta \neq id$. Things are different when we adopt a universal outer quantification, i.e., by considering the confidentiality property $\forall x_1 \,.\, \pi_{\mathsf{ANI}}[{}^{\mathsf{v}}/_{x_2}]$, which indeed corresponds to the abstract version of an existing non-interference requirement, that is, single non-interference [16].

Definition 3.1 (Single Generalized Abstract Non-Interference). Let ϕ, η in $uco(\wp(\mathbb{I}))$ be abstractions on input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. Given an input value v in $\mathbb{I}$, a function $f : \mathbb{I} \to \mathbb{O}$ satisfies *Generalized Abstract Non-Interference on* v, w.r.t. ϕ, η in input and ρ in output, written $f \models \mathsf{GANI}_{(\mathsf{v})}\,[\langle \phi^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$, if:

$$\forall x \,.\, (\phi(x) = \phi(\mathsf{v}) \wedge \eta(x) \neq \eta(\mathsf{v}) \Rightarrow \rho f(x) = \rho f(\mathsf{v}))$$

This notion can be seen as a "controlled" notion of non-interference since it basically says that a specific execution is (abstract) non-interferent with any other execution.

Finally, we can quantify both free variables. Again, by definition of π_{ANI}, if one of the two variables is existentially quantified, then when $\phi \neq \top$ or $\eta \neq id$, the property would be trivially satisfied by any program, resulting in a not particular meaningful notion. Instead, if we universally quantify both variables, i.e., by considering the confidentiality property $\forall x_1 \,.\, \forall x_2 \,.\, \pi_{\mathsf{ANI}}$, we obtain a generalized notion of Abstract Non-Interference.

Definition 3.2 (Generalized Abstract Non-Intereference). Let ϕ, η in $uco(\wp(\mathbb{I}))$ be abstractions on input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. A function $f : \mathbb{I} \to \mathbb{O}$ satisfies *Generalized Abstract Non-Interference* (GANI in short) w.r.t. ϕ, η in input and ρ in output, written $f \models \mathsf{GANI}[\langle \phi^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$, if:

$$\forall x_1 \,.\, \forall x_2 \,.\, (\phi(x_1) = \phi(x_2) \wedge \eta(x_1) \neq \eta(x_2) \Rightarrow \rho f(x_1) = \rho f(x_2))$$

Let us show now that this notion precisely captures all the flavors of Abstract Non-Interference presented in the background. In particular, narrow (NANI) and declassified (DANI) Abstract Non-Interference can be obtained by checking Definition 3.2 for specific input observations: $\phi = \top$ in the case of DANI, not admitting any form of declassification; and $\eta = id$ in the case of NANI, not fixing any form of protection.

Proposition 3.3. *Let ϕ, η in $uco(\wp(\mathbb{I}))$ be abstractions on input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. The following hold for any function $f : \mathbb{I} \to \mathbb{O}$.*

- $f \models \mathsf{DANI}[\eta, \rho]$ *iff* $f \models \mathsf{GANI}[\langle \top^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$
- $f \models \mathsf{NANI}[\phi, \rho]$ *iff* $f \models \mathsf{GANI}[\langle \phi^{=}, id^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$

Proof. The proof directly follows from Definitions 3.2, 2.7, and 2.6. □

As far as Abstract Non-Interference (ANI) of Definition 2.4 is concerned, we can capture such a notion by using Definition 3.2 and changing the function under consideration: from the function f computing on concrete values (i.e., on $\mathbb{I}$) to the function $f \circ \eta$ computing on abstract values (i.e., on $\eta(\mathbb{I})$). This is necessary since ANI is defined as abstract program executions instead of concrete ones (as all other confidentiality notions).

Proposition 3.4. *Let ϕ, η in $uco(\wp(\mathbb{I}))$ be abstractions on input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. For any function $f : \mathbb{I} \to \mathbb{O}$ we have:*

$$f \models \mathsf{ANI}[\langle \phi, \eta \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}] \;\; \textit{iff} \;\; f \circ \eta \models \mathsf{GANI}[\langle \phi^{=}, id^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$$

Proof. The proof follows from Definitions 3.2 and 2.4. □

Example 3.5. Consider the following *Program 3* computing the remainder of the division by 2. That is, the program computes the function $f_5 : \mathbb{Z} \to \mathbb{Z}$ such that $f_5(x) = x \bmod 2$.

```
/*  Program 3  */
remainder := number;
if (number < 0) {
   remainder := -(remainder);
}
while (remainder >= 2) {
   remainder := remainder - 2;
}
out remainder;
```

Consider $\mathbb{I} = \mathbb{O} = \mathbb{Z}$ and the following abstractions $\phi, \eta, \rho \in uco(\wp(\mathbb{Z}))$: $\phi = \{\varnothing, \mathbb{Z}_{2k}, \mathbb{Z}_{2k+1}, \mathbb{Z}\}$ (Parity); $\eta = \{\varnothing, \mathbb{Z}^-, \mathbb{Z}^{+0}, \mathbb{Z}\}$ (Sign); and $\rho = \{\varnothing, \mathbb{Z}\} \cup \{\{z\} \mid z \in \mathbb{Z}\}$ (Constants). We have that $f_5 \models \mathsf{GANI}[\langle \phi^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$ holds. To see that, we have to check the pairs of inputs values with the same parity but with different sign, namely the pairs of the form:

a: $v_1 \in \mathbb{Z}_{2k} \cap \mathbb{Z}^-$ and $v_2 \in \mathbb{Z}_{2k} \cap \mathbb{Z}^{+0}$ e.g., $(-4, 0)$ or $(-2, 6)$
b: $v_1 \in \mathbb{Z}_{2k+1} \cap \mathbb{Z}^-$ and $v_2 \in \mathbb{Z}_{2k+1} \cap \mathbb{Z}^{+0}$ e.g., $(-3, 5)$

For the pairs of the form **a**, the function f_5 returns 0, e.g., $f_5(-4) = 0 = f_5(0)$, and $f_5(-2) = 0 = f_5(6)$; while for those of the form **b**, it returns 1, e.g., $f_5(-3) = 1 = f_5(5)$. When we observe constants in output, we cannot see any difference from the application of f_5 on any pair of the two forms. Indeed, for each pair we have that $f_5(v_1) = f_5(v_2)$, hence $\rho f_5(v_1) = \rho f_5(v_2)$ (in particular, it is either the constant $\{0\}$ or $\{1\}$). In this specific example, GANI holds for any η, thus, it is equivalent to NANI $f_5 \models \mathsf{NANI}[\phi, \rho]$ (Definition 2.6).

End Example.

3.2 Generalizing Opacity

To find common ground to compare confidentiality notions, specifically Abstract Non-Interference and Opacity, we now characterize (and indeed generalize) Opacity as a confidentiality property. Similarly to the previous subsection, we define the execution requirement imposed by Opacity (Definition 2.1). To be more generic, we define the requirement directly by adopting closure operators instead of equivalence relations[9]. Doing so, the property that has to be *opaque*, usually modeled as a binary predicate (Definition 2.1), may induce more than two partitions on values.

Similar to Abstract Non-Interference, the Opacity requirement stipulates a constraint on pairs of executions. Such a requirement intuitively says that a variation of the input values, up to an input abstraction η, should not result in a variation of the output values, up to an output abstraction ρ, when considering executions yielding from input values indistinguishable by another input abstraction ϕ. This is tantamount to saying that from the output observation ρ, we cannot deduce the input property η of values. Indeed, we always have the same output, up to ρ, stemming from input values having different abstractions in η. Formally:

$$\pi_{\mathrm{AOp}} \triangleq \phi(x_1) = \phi(x_2) \wedge \eta(x_1) \neq \eta(x_2) \wedge \rho f(x_1) = \rho f(x_2)$$

We can now fix two specific executions, instantiating x_1 and x_2 with two values. But, as it happens for Abstract Non-Interference, this would simply say that the two specific executions are (abstract) opaque. More interesting is when we fix one computation and quantify the other. In this case, the conjunctions in π_{AOp} make an outer universal quantification meaningless: the only function satisfying such a confidentiality property would be the one such that ρf is a constant, since it would

[9]Abstract domains strictly generalize equivalence relations, and therefore binary predicates [26].

be the case that $\phi = \top$ and $\eta = id$. Nevertheless, we obtain a meaningful confidentiality property when considering an outer existential quantification, i.e., when considering $\exists x_1 \,.\, \pi_{\mathsf{AOp}}[{}^{\mathsf{v}}/_{x_2}]$, for a fixed input value v in $\mathbb{I}$.

Definition 3.6 (Single Abstract Opacity). Let ϕ, η in $uco(\wp(\mathbb{I}))$ be abstractions on input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. Given an input value v in $\mathbb{I}$, the property η is *abstract opaque* for a function $f : \mathbb{I} \to \mathbb{O}$ *on* v w.r.t. ϕ in input and ρ in output, written $f \models \mathsf{AOp}^{(\mathsf{v})}[\eta : \langle\phi\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$, if:

$$\exists x \,.\, (\phi(x) = \phi(\mathsf{v}) \wedge \eta(x) \neq \eta(\mathsf{v}) \wedge \rho f(x) = \rho f(\mathsf{v}))$$

Going further, we can quantify both free variables. Again, since π_{AOp} is composed only of conjunctions, when both variables are universally quantified, we actually impose that $\phi = \top$ and $\eta = id$, implying that ρf is a constant, and resulting in a meaningless confidentiality property. A similar reasoning can be done if we existentially quantify x_1 and we universally quantify x_2. Instead, doing the opposite, namely universally quantifying x_1 and existentially quantifying x_2, we obtain the confidentiality property $\forall x_1 \,.\, \exists x_2 \,.\, \pi_{\mathsf{AOp}}$, that generalizes the notion of Opacity (Definition 2.1).

Definition 3.7 (Abstract Opacity). Let ϕ, η in $uco(\wp(\mathbb{I}))$ be abstractions input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. The property η is *abstract opaque* for a function $f : \mathbb{I} \to \mathbb{O}$ w.r.t. ϕ in input and ρ in output, written $f \models \mathsf{AOp}[\eta : \langle\phi\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$, if:

$$\forall x_1 \,.\, \exists x_2 \,.\, (\phi(x_1) = \phi(x_2) \wedge \eta(x_1) \neq \eta(x_2) \wedge \rho f(x_1) = \rho f(x_2))$$

The above definition indeed generalizes the classic notion of Opacity. Consider $\mathsf{P} \in \mathsf{Op}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$, where $\mathsf{P} \subseteq \mathbb{I}$ is a binary predicate on input values, while $\overset{\iota}{\sim}$ in $Eq(\mathbb{I})$ and $\overset{o}{\sim}$ in $Eq(\mathbb{O})$ are equivalence relations on input and output values, respectively. The corresponding partitioning closure for the binary predicate $\mathsf{P} \subseteq \mathbb{I}$ is $\eta_{\mathsf{P}} \triangleq \{\varnothing, \mathsf{P}, \overline{\mathsf{P}}, \mathbb{I}\}$. For the equivalence relations $\overset{\iota}{\sim}$ and $\overset{o}{\sim}$, the corresponding partitioning closures are $\phi_\iota \triangleq Clo^{\overset{\iota}{\sim}}$ and $\rho_o \triangleq Clo^{\overset{o}{\sim}}$, respectively. In the following, to simplify the notation, when we write ϕ_ι we mean the closure induced by the equivalence relation $\overset{\iota}{\sim}$ (the same applies for ρ_o with $\overset{o}{\sim}$).

Proposition 3.8. *Let $\overset{\iota}{\sim}$ in $Eq(\mathbb{I})$ and $\overset{o}{\sim}$ in $Eq(\mathbb{O})$ be equivalence relations on input and output values, respectively, and $\mathsf{P} \subseteq \mathbb{I}$ be a binary predicate on input values. Then, $\mathsf{P} \in \mathsf{SOp}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$ iff $f \models \mathsf{AOp}[\eta_{\mathsf{P}} : \langle\phi_\iota\rangle_{\mathbb{I}}, \langle\rho_o\rangle_{\mathbb{O}}]$.*

Proof. The proof trivially follows from observing that: by definition of η_{P}, we have $\eta_{\mathsf{P}}(x_1) \neq \eta_{\mathsf{P}}(x_2)$ iff $x_1 \in \mathsf{P}$ and $x_2 \in \overline{\mathsf{P}}$ (or vice versa); by definition of ϕ_ι, we have $\phi_\iota(x_1) = \phi_\iota(x_2)$ iff $x_1 \overset{\iota}{\sim} x_2$; by definition of ρ_o, we have $\rho_o f(x_1) = \rho_o f(x_2)$ iff $f(x_1) \overset{o}{\sim} f(x_2)$. □

When dealing with partitioning closures, what Abstract Opacity stipulates is that every equivalence class of a given equivalence relation is opaque, as per classic Opacity (Definition 2.1). Indeed, it is easy to see that when we consider a generic equivalence relation R in $Eq(\mathbb{O})$, we have that $f \models \mathsf{AOp}[\eta_{\mathsf{R}} : \langle\phi_\iota\rangle_{\mathbb{I}}, \langle\rho_o\rangle_{\mathbb{O}}]$ holds when $[\mathsf{v}]_{\mathsf{R}} \in \mathsf{Op}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$ for every input value v in $\mathbb{I}$. The particular case when the equivalence relation stems from a predicate $\mathsf{P} \subseteq \mathbb{I}$ is exactly the classic definition of Symmetric Opacity (Definition 2.1). Indeed, the equivalence classes of the equivalence relation induced by P are precisely P and $\overline{\mathsf{P}}$.

Example 3.9. Suppose to have a system that deals with the balance of bank accounts. Let $\mathbb{I} = \mathbb{O} = \mathbb{Z}$, where $\mathbb{Z}$ is the set of integer numbers. Consider the input observation $\phi_\iota \triangleq \{\varnothing, [-50, 50], \mathbb{Z} \setminus [-50, 50], \mathbb{Z}\}$ (where $\overset{\iota}{\sim} = \{[-50, 50], \mathbb{Z} \setminus [-50, 50]\}$ is the corresponding equivalence relation on $\mathbb{I}$) and the input property to protect $\eta_{\mathsf{R}} \triangleq \{\varnothing, [-\infty, -10], [-9, 9], [10, \infty], \mathbb{Z}\}$ (where the corresponding equivalence relation is $\mathsf{R} = \{[-\infty, -10], [-9, 9], [10, \infty]\}$). The property to protect may represent investment risk factors: $[10, \infty]$, for low risk; $[-9, 9]$, for normal risk; and $[-\infty, -10]$,

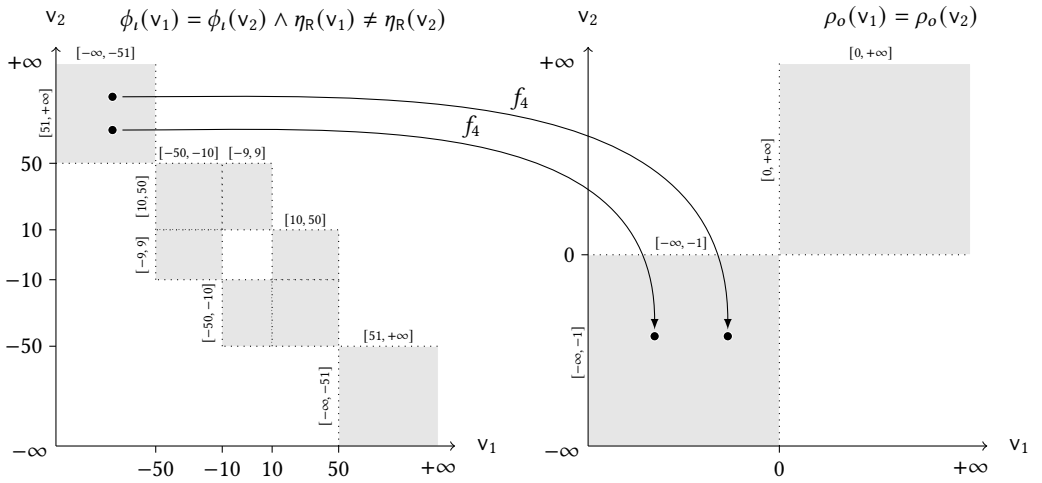

Fig. 2. Graphical explanation of Abstract Opacity for *Program 4*.

for high risk. Instead, the input observation can be used to profile users: $[-50, 50]$, for a common market profile; and $\mathbb{Z} \setminus [-50, 50]$, for corner case profiles (either too poor or too rich). Finally, let $\rho_o \triangleq \{\varnothing, \mathbb{Z}^-, \mathbb{Z}^{+0}, \mathbb{Z}\}$ (Sign domain) be the output observation (where $\overset{o}{\sim} = \{[-\infty, -1], [0, +\infty]\}$ is the corresponding equivalence relation). Now consider a *Program 4* that aims to hide low and high risk balances.

```
/*  Program 4  */
if (-9 <= balance and balance <= 9) { out balance; }
else {
   if (abs(balance) >= 50) {
      if (balance < 0) { out balance + 51; }
      else { out balance - 52; }
   } else {
      if (balance < 0) { out balance + 11; }
      else { out balance - 11; }
   }
}
```

The program computes the function $f_4 \in \mathbb{Z} \to \mathbb{Z}$ defined as

$$f_4(x) \triangleq \begin{cases} x + 51 & \text{if } x \leq -50 \\ x - 52 & \text{if } x \geq 50 \\ x + 11 & \text{if } -50 < x \leq -10 \\ x - 11 & \text{if } 10 \leq x < 50 \\ x & \text{otherwise} \end{cases}$$

The program protects the property η_R, since every equivalence class of R is made opaque by the program, namely $f_4 \models \mathsf{AOp}[\eta_R : \langle \phi_\iota \rangle_{\mathbb{I}}, \langle \rho_o \rangle_{\mathbb{O}}]$. Indeed, for each value (e.g., 0) we can find another value (e.g., 11) in the relation $\overset{\iota}{\sim}$ (i.e., $0 \overset{\iota}{\sim} 11$) but not in the relation R (i.e., $0 \not\mathrel{R} 11$) with the same result for $\rho_o f_4$ (i.e., $f_4(0) \overset{o}{\sim} f_4(11)$). This is summarized in Figure 2, where we report a graphical representation of Abstract Opacity for *Program 4*. On the left of the figure, we have the regions of values satisfying the input conjuncts of Abstract Opacity, that is $\phi_\iota(v_1) = \phi_\iota(v_2) \wedge \eta_R(v_1) \neq \eta_R(v_2)$, while on the right we have the regions of values satisfying the output conjunct of Abstract Opacity, that is $\rho_o(v_1) = \rho_o(v_2)$. *Program 4* (i.e., f_4) maps input regions to output regions, satisfying Abstract Opacity. Note that, if we do not apply the program (e.g., by considering f as the identity function)

we have that the equivalence class $[-9, 9]$ is not opaque. Thus, by observing the sign of output values, we can infer the investment risk factor of a given account.

End Example.

By rewriting Opacity in terms of abstractions, we can also state a correlation between the former and Generalized Abstract Non-Interference. Let us consider the single case first.

Theorem 3.10. *Let ϕ, η in $uco(\wp(\mathbb{I}))$ be abstractions on input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. Given an input value v in $\mathbb{I}$, if there exists another input value v' in $\mathbb{I}$ such that $\phi(\mathsf{v}) = \phi(\mathsf{v}')$ and $\eta(\mathsf{v}) \neq \eta(\mathsf{v}')$, then:*

$$f \models \mathsf{GANI}_{(\mathsf{v})}\,[\langle \phi^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}] \;\; \textit{implies} \;\; f \models \mathsf{AOp}_{(\mathsf{v})}\,[\eta : \langle \phi \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$$

Proof. Since, by assumption there exists v' such that $\phi(\mathsf{v}) = \phi(\mathsf{v}')$ and $\eta(\mathsf{v}) \neq \eta(\mathsf{v}')$, when GANI on v holds (by Definition 3.1) we have that $\rho f(\mathsf{v}) = \rho f(\mathsf{v}')$. Hence, we have that $\exists x\,.\,(\phi(x) = \phi(\mathsf{v}) \wedge \eta(x) \neq \eta(\mathsf{v}) \wedge \rho f(x) = \rho f(\mathsf{v}))$ holds by substituting x with v'. This implies (by Definition 3.6) $f \models \mathsf{AOp}_{(\mathsf{v})}\,[\eta : \langle \phi \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$. □

Such a correlation can be trivially extended to the non-single versions of Abstract Opacity and Generalized Abstract Non-Interference.

Corollary 3.11. *Let ϕ, η in $uco(\wp(\mathbb{I}))$ be abstractions on input and ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. If for each input value v in $\mathbb{I}$ there exists another input value v' in $\mathbb{I}$ such that $\phi(\mathsf{v}) = \phi(\mathsf{v}')$ and $\eta(\mathsf{v}) \neq \eta(\mathsf{v}')$, then:*

$$f \models \mathsf{GANI}[\langle \phi^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}] \;\; \textit{implies} \;\; f \models \mathsf{AOp}[\eta : \langle \phi \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$$

Proof. The proof follows from Theorem 3.10 and from observing that: $f \models \mathsf{GANI}[\langle \phi^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$ iff for every v in $\mathbb{I}$ we have $f \models \mathsf{GANI}_{(\mathsf{v})}\,[\langle \phi^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$; and $f \models \mathsf{AOp}[\eta : \langle \phi \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$ iff for every v in $\mathbb{I}$ we have $f \models \mathsf{AOp}_{(\mathsf{v})}\,[\eta : \langle \phi \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$. □

3.3 Generalizing Final Opacity

We can also define final opacity [6] in our formalism, acting on the output of program semantics. Recall that the global version of opacity [6] is obtained by changing the denotation of program results, i.e., $\mathbb{O}$. For instance, we can consider traces of values instead of single (output) values. We directly provide a generalization, introducing an input observation, as it has been done for (initial) Opacity. Note that, even if the definitions of opacity and final opacity appear quite different in [6], they are the same concept instantiated on a different predicate to make opaque. Indeed, given a predicate $\mathsf{P} \subseteq \mathbb{O}$ on output values, we can define a corresponding predicate $\mathsf{P}^{-f} \subseteq \mathbb{I}$ on input values, parametric on a given function $f : \mathbb{I} \to \mathbb{O}$. In particular, we define $\mathsf{P}^{-f} \triangleq \{\mathsf{v} \mid f(\mathsf{v}) \in \mathsf{P}\}$ as the set of input values whose image under f satisfies P. To simplify the notation, we denote with $\mathsf{P}^{\smile}$ the predicate P^{-f} when f is clear from the context. Then, final opacity [6] for P is simply Opacity for $\mathsf{P}^{\smile}$.

Definition 3.12 (Final Opacity). Let $\overset{\iota}{\sim}$ in $Eq(\mathbb{I})$ and $\overset{o}{\sim}$ in $Eq(\mathbb{O})$ be equivalence relations on input and output, respectively. A binary predicate $\mathsf{P} \subseteq \mathbb{O}$ is *finally opaque* for a function $f : \mathbb{I} \to \mathbb{O}$ w.r.t. $\overset{\iota}{\sim}$ in input and $\overset{o}{\sim}$ in output, denoted $\mathsf{P} \in \mathsf{FOp}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$, if:

$$\forall_{\mathsf{P}^{\smile}}\, x_1\,.\,\exists_{\overline{\mathsf{P}}^{\smile}}\, x_2\,.\,(x_1 \overset{\iota}{\sim} x_2 \wedge f(x_1) \overset{o}{\sim} f(x_2))$$

When $\mathsf{P}, \overline{\mathsf{P}} \in \mathsf{FOp}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$, P is *symmetrically and finally opaque*, written $\mathsf{P} \in \mathsf{SFOp}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$.

It is easy to note that $\mathsf{P} \in \mathsf{FOp}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$ if and only if $\mathsf{P}^{\smile} \in \mathsf{Op}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$. Note that, when $\overset{o}{\sim}$ is the identity, we have that $f(\mathsf{P}^{\smile}) \cap f(\overline{\mathsf{P}^{\smile}}) = \varnothing$, and therefore P cannot be finally opaque.

Similar to what we have done in the previous subsection for Opacity, we can generalize Final Opacity by adopting generic properties in place of binary predicates and by identifying the execution

requirement to define it as a confidentiality property. Final Opacity (Definition 3.12) requires that executions starting from input values indistinguishable by the input abstraction ϕ yield a variation of the output values, up to an output abstraction η, that cannot be distinguished by another output observation ρ. That is, changes in the output observation η are not revealed by observing the computation output up to ρ. Formally:

$$\pi_{\mathsf{AFOp}} \triangleq \phi(x_1) = \phi(x_2) \wedge \eta f(x_1) \neq \eta f(x_2) \wedge \rho f(x_1) = \rho f(x_2)$$

Analogously to what we have seen for Abstract Non-Interference and Opacity, we can define confidentiality properties for Final Opacity by quantifying and/or instantiating the free variables of π_{AFOp}. As we observed for Opacity, given the logical structure of the execution requirement, the only meaningful case is the one where we instantiate the variable x_2 and we existentially quantify x_1, that is $\exists x_1 \,.\, \pi_{\mathsf{AFop}}[{}^{\mathsf{v}}/_{x_2}]$, given an input value v in $\mathbb{I}$.

Definition 3.13 (Single Abstract Final Opacity). Let ϕ in $uco(\wp(\mathbb{I}))$ be an abstraction on input and η, ρ in $uco(\wp(\mathbb{O}))$ be abstractions on output. Given an input value v in $\mathbb{I}$, the property η is *finally abstract opaque* for a function $f : \mathbb{I} \longrightarrow \mathbb{O}$ *on* v w.r.t. ϕ in input and ρ in output, written $f \models \mathsf{AFOp}(\mathsf{v})\,[\eta : \langle\phi\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$, if:

$$\exists x \,.\, \phi(x) = \phi(\mathsf{v}) \wedge \eta f(x) \neq \eta f(\mathsf{v}) \wedge \rho f(x) = \rho f(\mathsf{v})$$

When quantifying on both variables, with a similar reasoning as for Opacity, we consider as meaningful the case where we universally quantify on x_1, and we existentially quantify on x_2, that is $\forall x_1 \,.\, \exists x_2 \,.\, \pi_{\mathsf{AFop}}$, obtaining so far an abstract notion of Final Opacity.

Definition 3.14 (Abstract Final Opacity). Let ϕ in $uco(\wp(\mathbb{I}))$ be an abstraction on input and η, ρ in $uco(\wp(\mathbb{O}))$ be abstractions on output. The property η is *finally abstract opaque* for a function $f : \mathbb{I} \longrightarrow \mathbb{O}$ w.r.t. ϕ in input and ρ in output, written $f \models \mathsf{AFOp}[\eta : \langle\phi\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$, if:

$$\forall x_1 \,.\, \exists x_2 \,.\, \phi(x_1) = \phi(x_2) \wedge \eta f(x_1) \neq \eta f(x_2) \wedge \rho f(x_1) = \rho f(x_2)$$

The above definition indeed generalizes the classic notion of Final Opacity. Consider $\mathsf{P} \in \mathsf{FOp}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$, where $\mathsf{P} \subseteq \mathbb{O}$ is a binary predicate on output values, while $\overset{\iota}{\sim}$ in $Eq(\mathbb{I})$ and $\overset{o}{\sim}$ in $Eq(\mathbb{O})$ are equivalence relations on input and out values, respectively. The corresponding partitioning closure for the binary predicate $\mathsf{P} \subseteq \mathbb{I}$ is $\eta_{\mathsf{P}} \triangleq \{\varnothing, \mathsf{P}, \overline{\mathsf{P}}, \mathbb{O}\}$. For the equivalence relations $\overset{\iota}{\sim}$ and $\overset{o}{\sim}$, the corresponding partitioning closures are $\phi_\iota \triangleq Clo^{\overset{\iota}{\sim}}$ and $\rho_o \triangleq Clo^{\overset{o}{\sim}}$ respectively.

Proposition 3.15. *Let $\overset{\iota}{\sim}$ in $Eq(\mathbb{I})$ and $\overset{o}{\sim}$ in $Eq(\mathbb{O})$ be equivalence relations on input and output, respectively, and $\mathsf{P} \subseteq \mathbb{O}$ a binary predicate on output values. Then, $\mathsf{P} \in \mathsf{FOp}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$ iff $f \models \mathsf{AFOp}[\eta_{\mathsf{P}} : \langle\phi_\iota\rangle_{\mathbb{I}}, \langle\rho_o\rangle_{\mathbb{O}}]$.*

Proof. The proof trivially follows from observing that: by definition of η_{P}, we have $\eta_{\mathsf{P}} f(x_1) \neq \eta_{\mathsf{P}} f(x_2)$ iff $f(x_1) \in \mathsf{P}$ and $f(x_2) \in \overline{\mathsf{P}}$; by definition of $\mathsf{P}^{\backsim}$, we have $f(x_1) \in \mathsf{P}$ iff $x_1 \in \mathsf{P}^{\backsim}$ and $f(x_2) \in \overline{\mathsf{P}}$ iff $x_2 \in \overline{\mathsf{P}^{\backsim}}$; by definition of ϕ_ι, we have $\phi_\iota(x_1) = \phi_\iota(x_2)$ iff $x_1 \overset{\iota}{\sim} x_2$; by definition of ρ_o, we have $\rho_o f(x_1) = \rho_o f(x_2)$ iff $f(x_1) \overset{o}{\sim} f(x_2)$. □

4 Confidentiality Notions for Code Protection

As noted in the previous section, Final Opacity can be seen as Opacity w.r.t. the inverse predicate/property to protect. However, in this section, we will provide a different viewpoint of Final Opacity, proving that it can be used to precisely define a novel confidentiality property useful in code protection, precisely in code *obfuscation* [10].

Indeed, by using both Final Opacity and Generalized Abstract Non-Interference, we can characterize a confidentiality property determining whether a program (already) obfuscates a desired

semantic property, in such a case we may avoid to transform (i.e., obfuscate) the program. Precisely as it happens in language-based security, where we have a confidentiality property to satisfy (e.g., Abstract Non-Interference or Opacity) and the corresponding enforcing techniques (e.g., security type systems), also for code protection, we can consider the existing obfuscation techniques as the code transformations enforcing a desired *code protection property*.

In Reference [15], the authors described how obfuscation strategies can be modeled regarding complete program transformations. In particular, obfuscation should achieve two simultaneous goals: *conceal*, i.e., hide as much as possible, a given aspect of the original program; and *preserve*, i.e., do not distort, another given aspect of the original program. In Reference [15], the authors proved that concealment corresponds to making incomplete a program analysis, aiming at retrieving the program aspect that obfuscation is supposed to hide. Recall that, a function $f : \mathbb{I} \longrightarrow \mathbb{O}$ is *complete* [24] for an abstraction ϕ on input values and an abstraction η on *output* values when the following semantic property holds: $\forall x \,.\, (\eta f(x) = \eta f \phi(x))$. Moreover, recall that, as in the rest of the article, the *output* domain $\mathbb{O}$ of the semantic function f is not necessarily the program output, but is in general the denotation modeling the program meaning (e.g., output data, concrete execution trace log, etc.).

We can formally define code obfuscation as considered in Reference [15] in terms of program (in-)completeness and Generalized Abstract Non-Interference (the latter for the preservation part). In the following definition, ϕ represents the observation capability on input values.

Definition 4.1 (Weak Abstract Obfuscation). Let ϕ in $uco(\wp(\mathbb{I}))$ be an abstraction on input and η, ρ in $uco(\wp(\mathbb{O}))$ be abstractions on output. A function $f : \mathbb{I} \longrightarrow \mathbb{O}$ satisfies *Weak Abstract Obfuscation* under ϕ in input, w.r.t. η to *conceal* and ρ to *preserve*, written $f \models \mathsf{WAObf}[\langle\phi\rangle_{\mathbb{I}}, \langle\eta^{\mathsf{C}}, \rho^{\mathsf{P}}\rangle_{\mathbb{O}}]$, if:

`(weak-conceal)` f weak conceals η under ϕ, that is, f is incomplete for η and ϕ; and
`(preserve)` f preserves ρ under ϕ, that is, $f \models \mathsf{GANI}[\langle\phi^{=}, id^{\neq}\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$.

In de-obfuscation by static analysis, the abstractions η and ϕ usually coincide. Indeed, $\eta f \eta$ represents a static analysis of (the program) f aiming at retrieving the property η of f hidden by the obfuscation transformation. When `(weak-conceal)` holds the analysis is incomplete, i.e., there exists an input v such that $\eta f(\mathsf{v}) \neq \eta f \eta(\mathsf{v})$, indicating a de-obfuscation failure, at least on that input. The preservation part given by GANI indicates that the obfuscation transformation has not distorted the property ρ.

In Reference [36], it has been proven that Completeness and a formulation of Abstract Non-Interference corresponding to GANI of Definition 3.2 are equivalent. That is, given a function $f : \mathbb{I} \longrightarrow \mathbb{O}$, an abstraction ϕ on input values and an abstraction η on output values, we have [36]:

$$\forall x \,.\, (\eta f(x) = \eta f \phi(x)) \text{ iff } f \models \mathsf{GANI}[\langle\phi^{=}, id^{\neq}\rangle_{\mathbb{I}}, \langle\eta\rangle_{\mathbb{O}}]$$

Due to this correspondence, the `(weak-conceal)` requirement of Definition 4.1 is equivalent to the fact that $f \models \mathsf{GANI}[\langle\phi^{=}, id^{\neq}\rangle_{\mathbb{I}}, \langle\eta\rangle_{\mathbb{O}}]$ does not hold, thus modeling (weak) obfuscation in terms of Generalized Abstract Non-Interference only. Under this different viewpoint, we can note that Definition 4.1 is a too permissive notion, potentially failing to adequately model concealment. Indeed, the negation of GANI says that it should exist a pair of inputs, indistinguishable by ϕ, such that f conceals (i.e., protect) η. Formally: $\exists x_1 \,.\, \exists x_2 \,.\, (\phi(x_1) = \phi(x_2) \wedge \eta f(x_1) \neq \eta f(x_2))$, by definition of GANI. Furthermore, Definition 4.1 says that f must preserve ρ for any possible pair of inputs indistinguishable by ϕ. Formally: $\forall x_1 \,.\, \forall x_2 \,.\, (\phi(x_1) = \phi(x_2) \implies \rho f(x_1) = \rho f(x_2))$, again, by definition of GANI. We deemed this notion of obfuscation as weak since the concealment part is indeed too permissive. The problem is that being Completeness (and, indeed, GANI) a universal property, its negation only requires the existence of some (potentially few) concealing

executions (for all the other input values the program aspect may not be concealed). Instead, the preservation part is a universal property, hence (correctly) requiring that all executions are preserving.

Example 4.2. Consider the following program $P_{\bigcirc}$ computing the 2nd power of a given number. When the number is odd, the program also inverts the sign of the result.

```
/* Original Program */
if (num % 2 == 0) {
   res := num * num;
} else {
   res := -(num * num);
}
out res;
```

Consider the classic parity Par and sign Sgn abstractions on numeric values reported in Figure 1 (left-center). Suppose that the parity of values is observed in input (i.e., $\phi = \mathsf{Par}$); the parity of values should be *preserved* on output (i.e., $\rho = \mathsf{Par}$); and the sign of values should be *concealed* on the semantic observation (i.e., $\eta = \mathsf{Sgn}$). Consider an analysis $f^\sharp = [\![P_{\bigcirc}]\!]^{\mathsf{ParSgn}}$ on the abstraction ParSgn reported in Figure 1 (right), in this case, the observation is the final abstract value obtained by the analysis computed on the ParSgn domain. Specifically, the analysis can retrieve the combined information about the parity and sign of numeric values.

The program $P_{\bigcirc}$ is not obfuscated w.r.t. the considered abstractions, since $f^\sharp \not\models \mathsf{WAObf}[\langle \mathsf{Par} \rangle_{\mathbb{I}}, \langle \mathsf{Sgn}^{\mathsf{C}}, \mathsf{Par}^{\mathsf{P}} \rangle_{\mathbb{O}}]$ does not hold. In this case, Par is preserved, but Sgn is not concealed. Indeed, we have completeness for $f^\sharp$. Each pair of even numbers, independently from their sign, is mapped to $\mathbb{Z}^{+0}_{2k}$ by $f^\sharp$, hence indistinguishable by both Par and Sgn, meaning that both are preserved (none is concealed). Indeed, $\rho(\mathbb{Z}^{+0}_{2k}) = \mathsf{Par}(\mathbb{Z}^{+0}_{2k}) = \mathbb{Z}_{2k}$. Similarly, each pair of odd numbers, independently from the numbers' sign, is mapped to $\mathbb{Z}^{-}_{2k+1}$ by $f^\sharp$, hence, again, indistinguishable by both Par and Sgn. Indeed, $\eta(\mathbb{Z}^{-}_{2k+1}) = \mathsf{Sgn}(\mathbb{Z}^{-}_{2k+1}) = \mathbb{Z}^{-}$.

Therefore, by Definition 4.1, we realize that the *program must be obfuscated*. Consider the following obfuscation $P_{\textcircled{\scriptsize\textbullet}}$ of $P_{\bigcirc}$, where multiplication is implemented by using iterative sums [9, 18, 19]. Since iterative sums are slower than direct multiplication, we may perform obfuscation on even numbers only, for performance needs. The semantics of the original program is not affected by the obfuscation, namely $[\![P_{\bigcirc}]\!] = [\![P_{\textcircled{\scriptsize\textbullet}}]\!]$.

```
/* Obfuscation 1 */
if (num % 2 == 0) {
   if (num < 0) { t := -num; }
   else { t := num; }
   res := 0;
   while (t != 0) {
      res := res + num;
      t := t - 1;
   }
   if (num < 0) { res := -res; }
} else {
   res = -(num * num);
}
out res;
```

The program $P_{\textcircled{\scriptsize\textbullet}}$ is weakly obfuscated, since we have that $f^\sharp \models \mathsf{WAObf}[\langle \mathsf{Par} \rangle_{\mathbb{I}}, \langle \mathsf{Sgn}^{\mathsf{C}}, \mathsf{Par}^{\mathsf{P}} \rangle_{\mathbb{O}}]$ holds. In this case, Par is preserved, and Sgn is only weakly concealed. Indeed, we have incompleteness for $f^\sharp$. Specifically, positive even numbers are mapped to $\mathbb{Z}^{+0}_{2k}$ by $f^\sharp$, while negative even numbers are mapped to $\mathbb{Z}_{2k}$ by $f^\sharp$. In the latter case, when computing the abstract value of res, the first iteration `0 + num` of the loop is abstracted into $\mathbb{Z}^{+0}_{2k} + \mathbb{Z}^{-}_{2k} = \mathbb{Z}_{2k}$, so the final abstract value for res is also $\mathbb{Z}_{2k}$. Hence, they are indistinguishable by Par (preserved)

but not by Sgn (concealed). Indeed, $\rho(\mathbb{Z}^{+0}_{2k}) = \mathsf{Par}(\mathbb{Z}^{+0}_{2k}) = \mathbb{Z}_{2k} = \mathsf{Par}(\mathbb{Z}_{2k}) = \rho(\mathbb{Z}_{2k})$, while $\eta(\mathbb{Z}^{+0}_{2k}) = \mathsf{Sgn}(\mathbb{Z}^{+0}_{2k}) = \mathbb{Z}^{+0} \neq \mathbb{Z} = \mathsf{Sgn}(\mathbb{Z}_{2k}) = \eta(\mathbb{Z}_{2k})$. However, each pair of odd numbers, independently from the sign, is mapped to $\mathbb{Z}^{-}_{2k+1}$ by $f^\sharp$, hence indistinguishable by both Par (preserved) and Sgn (not concealed). Hence, *not all inputs are obfuscated.*

End Example.

We can make the definition of obfuscation stronger by considering Final Opacity in place of GANI (and, thus, Completeness) for the concealment part. Consider an (input) property η being finally abstract opaque for f w.r.t. the input abstraction ϕ and the output abstraction $\top$. Then, we can introduce a stronger notion of concealment:

(conceal) f conceals η under ϕ, that is, $f \models \mathsf{AFOp}[\eta : \langle\phi\rangle_{\mathbb{I}}, \langle\top\rangle_{\mathbb{O}}]$

This version of concealment says that for every possible input value, it must exist another input value, indistinguishable by ϕ, such that f conceals (i.e., protect) η. Formally: $\forall x_1 \,.\, \exists x_2 \,.\, \phi(x_1) = \phi(x_2) \wedge \eta f(x_1) \neq \eta f(x_2)$, by definition of Abstract Final Opacity. It is trivial to observe that (conceal) is stronger than (weak-conceal). Indeed, it stipulates that for *any* input value we have to conceal the program aspect to protect. We can then define obfuscation as a combination of Abstract Final Opacity and Generalized Abstract Non-Interference.

Definition 4.3 (Abstract Obfuscation). Let ϕ in $uco(\wp(\mathbb{I}))$ be an abstraction on input and η, ρ in $uco(\wp(\mathbb{O}))$ be an abstraction on output. A function $f : \mathbb{I} \to \mathbb{O}$ satisfies *Abstract Obfuscation* under ϕ in input, w.r.t. η to conceal and ρ to preserve, written $f \models \mathsf{AObf}[\langle\phi\rangle_{\mathbb{I}}, \langle\eta^{\mathsf{C}}, \rho^{\mathsf{P}}\rangle_{\mathbb{O}}]$, if:

(conceal) f conceals η under ϕ, that is, $f \models \mathsf{AFOp}[\eta : \langle\phi\rangle_{\mathbb{I}}, \langle\top\rangle_{\mathbb{O}}]$; and
(preserve) f preserves ρ under ϕ, that is, $f \models \mathsf{GANI}[\langle\phi^{=}, id^{\neq}\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$.

In the context of de-obfuscation by static analysis, the abstractions η and ϕ usually coincide. But, by replacing (weak-conceal) with (conceal), we have that the static analysis of (the program) f is incomplete on *all* inputs, thus making de-obfuscation infeasible. The preservation part given by GANI still indicates that the obfuscation transformation has not distorted the property ρ.

Example 4.4. Consider the obfuscation $P_{\mathbf{O}}$ of the original program $P_{\bigcirc}$ from Example 4.2. Now suppose to apply data-obfuscation on odd numbers, by inserting semantics-preserving operations into expressions (e.g., adding zero into mathematical expressions) and yielding the following obfuscation $P_{\bullet}$. Again, the semantics of the original program is not affected by the obfuscation, namely $[\![P_{\bigcirc}]\!] = [\![P_{\bullet}]\!]$.

```
/* Obfuscation 2 */
if (num % 2 == 0) {
       if (num < 0) { t := -num; }
   else { t := num; }
   res := 0;
   while (t != 0) {
          res := res + num;
          t := t - 1;
   }
   if (num < 0) { res := -res; }
} else {
       res := -(num + 0)*(num + 0);
}
out res;
```

The program $P_{\bullet}$ is (fully) obfuscated, as per Definition 4.3, since we have that $f^\sharp \models \mathsf{AObf}[\langle\mathsf{Par}\rangle_{\mathbb{I}}, \langle\mathsf{Sgn}^{\mathsf{C}}, \mathsf{Par}^{\mathsf{P}}\rangle_{\mathbb{O}}]$ holds when considering the analysis on ParSgn for the obfuscated program, i.e., $f^\sharp = [\![P_{\bullet}]\!]^{\mathsf{ParSgn}}$. In this case, Par is preserved, and Sgn is concealed. Indeed, positive even numbers are mapped to $\mathbb{Z}^{+0}_{2k}$ by $f^\sharp$, while negative even numbers are mapped to $\mathbb{Z}_{2k}$ by $f^\sharp$. Hence, they are indistinguishable by Par (preserved) but not by Sgn (concealed). Similarly, positive

odd numbers are mapped to $\mathbb{Z}^{-}_{2k+1}$ by $f^\sharp$, while negative odd numbers are mapped to $\mathbb{Z}_{2k+1}$ by $f^\sharp$. In the latter case, when computing the abstract value of `res`, the expression `num + 0` is abstracted into $\mathbb{Z}^{-}_{2k+1} + \mathbb{Z}^{+0}_{2k} = \mathbb{Z}_{2k+1}$, so the final abstract value for `res` is also $\mathbb{Z}_{2k+1}$. Hence, they are indistinguishable by Par (preserved) but not by Sgn (concealed). Indeed, $\rho(\mathbb{Z}^{-}_{2k+1}) = \mathsf{Par}(\mathbb{Z}^{-}_{2k+1}) = \mathbb{Z}_{2k+1} = \mathsf{Par}(\mathbb{Z}_{2k+1}) = \rho(\mathbb{Z}_{2k+1})$, while $\eta(\mathbb{Z}^{-}_{2k+1}) = \mathsf{Sgn}(\mathbb{Z}^{-}_{2k+1}) = \mathbb{Z}^{-} \neq \mathbb{Z} = \mathsf{Sgn}(\mathbb{Z}_{2k+1}) = \eta(\mathbb{Z}_{2k+1})$. Hence, for each input, there exists another input (one with the opposite sign and the same parity), allowing us to conceal Sgn.

Therefore, by Definition 4.3, we can realize that the *program does not need to be obfuscated*, as the program aspects of interest are already preserved or concealed, respectively.

End Example.

5 Verifying Confidentiality Notions

In this section, we show how the verification mechanism for non-interference proposed in Reference [34] can be generalized to other, more permissive, confidentiality notions, such as Generalized Abstract Non-Interference and Opacity.

5.1 A Verification Mechanism for Generalized Abstract Non-Interference

Let us start with Generalized Abstract Non-Interference (GANI). Being a hyperproperty, this confidentiality notion requires comparing multiple program executions. As explained in Section 2, this means that GANI is defined on collections of sets of program executions, namely it is an element of $\wp(\wp(\mathbb{I} \times \mathbb{O}))$. We recall that here we consider as program executions the program input-output pairs (see Section 2.6 for details).

Note that, the notion of Abstract Non-Interference (ANI) introduced in Section 2 cannot be defined as a hyperproperty of (concrete) program executions, that is, as a hyperproperty of computations over data values. This is not surprising since ANI already considers abstract computations instead of concrete ones: the computation starts from abstract values in η and not from concrete values in $\mathbb{I}$. In other words, ANI specifies a relation between the input observation (ϕ) and the output observation (ρ) of *abstract* computations ($f\eta$), namely computations executed on abstract values. So, we cannot define ANI, in general, on the same denotation domain of concrete executions. Things are different when the input abstraction η is the identity, meaning that no abstraction is indeed applied. In that case, ANI with $\eta = id$ specifies a relation between the input observation (ϕ) and the output one (ρ) of *concrete* computations (f). The former reasoning applies to GANI as well, since no abstraction is performed in input to concrete computations, by definition. Thus, GANI can be expressed as a hyperproperty of computations over data values.

By a straightforward application of Definition 3.2, we can formulate GANI as a set of sets of the program *concrete* executions, i.e., as a hyperproperty, as follows:

$$\mathrm{HP}^{\mathsf{GANI}} \triangleq \left\{ X \subseteq \mathbb{I} \times \mathbb{O} \;\middle|\; \begin{array}{l} \forall \left(\mathsf{v}_1, \mathsf{v}'_1\right), \left(\mathsf{v}_2, \mathsf{v}'_2\right) \in X. \\ \phi(\mathsf{v}_1) = \phi(\mathsf{v}_2) \wedge \ \eta(\mathsf{v}_1) \neq \eta(\mathsf{v}_2) \ \Rightarrow \ \rho\left(\mathsf{v}'_1\right) = \rho\left(\mathsf{v}'_2\right) \end{array} \right\}$$

We now show how it is possible to define an abstract interpretation-based verification mechanism for $\mathrm{HP}^{\mathsf{GANI}}$, and, thus, for Generalized Abstract Non-Interference. Recalling Sction 2.6, a program P satisfies GANI, w.r.t. ϕ, η and ρ, namely $f_P \models \mathsf{GANI}[\langle \phi^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$, iff $\{[\![P]\!]\} \subseteq \mathrm{HP}^{\mathsf{GANI}}$, where here $[\![P]\!] \subseteq \mathbb{I} \times \mathbb{O}$ is the input-output semantics of P. This check is undecidable, so we can exploit the abstract interpretation framework [12] in order to make the verification feasible, similarly to what has been done for non-interference in [34]. GANI is a 2-bounded hyperproperty [33], which means that we can check it on the subsets of the semantics with cardinality 2 (i.e., on pairs of executions). Hence, given $(\!|P|\!) \triangleq \{X \subseteq [\![P]\!] \mid |X| = 2\}$, we have that $f_P \models \mathsf{GANI}[\langle \phi^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$ iff

$(\!|P|\!) \subseteq \mathrm{HP}^{\mathsf{GANI}}$. Furthermore, $(\!|P|\!)$ can be partitioned as follows:

$$(\!|P|\!)_{\phi,\eta} \triangleq \left\{\{(\mathsf{v}_1, \mathsf{v}_1'), (\mathsf{v}_2, \mathsf{v}_2')\} \in (\!|P|\!) \;\middle|\; \phi(\mathsf{v}_1) = \phi(\mathsf{v}_2) \;\wedge\; \eta(\mathsf{v}_1) \neq \eta(\mathsf{v}_2)\right\}$$
$$(\!|P|\!)_{\overline{\phi,\eta}} \triangleq \left\{\{(\mathsf{v}_1, \mathsf{v}_1'), (\mathsf{v}_2, \mathsf{v}_2')\} \in (\!|P|\!) \;\middle|\; \phi(\mathsf{v}_1) \neq \phi(\mathsf{v}_2) \;\vee\; \eta(\mathsf{v}_1) = \eta(\mathsf{v}_2)\right\}$$

Since $(\!|P|\!)_{\overline{\phi,\eta}} \subseteq \mathrm{HP}^{\mathsf{GANI}}$ is trivially always true, we just have to check whether $(\!|P|\!)_{\phi,\eta} \subseteq \mathrm{HP}^{\mathsf{GANI}}$. Following [34], we can simplify even further the verification process considering the Galois connection[10]:

$$\langle \wp(\wp(\mathbb{D} \times \mathbb{D})), \subseteq \rangle \underset{\alpha_{nr}}{\overset{\alpha_{nr}^{-}}{\leftrightarrows}} \langle \wp(\wp(\mathbb{D})) \times \wp(\wp(\mathbb{D})), \subseteq \rangle$$

where, without loss of generality, we assume that input and output domains coincide, i.e., $\mathbb{D} \triangleq \mathbb{O} = \mathbb{I}$. The non-relational abstraction α_{nr} [34] first forgets the relation between the input and the output of the computation (i.e., we only keep the relation between the whole set of inputs and the whole set of outputs), then it forgets the relation between sets of executions (i.e., we only keep the relation between the set of all possible input sets of and the set of all output sets).

At this point, we have that $f_P \models \mathsf{GANI}[\langle \phi^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}]$ iff $\alpha_{nr}((\!|P|\!)_{\phi,\eta}) \subseteq \alpha_{nr}(\mathrm{HP}^{\mathsf{GANI}})$. Then, it is easy to note that $\alpha_{nr}((\!|P|\!)_{\phi,\eta}) \subseteq \alpha_{nr}(\mathrm{HP}^{\mathsf{GANI}})$ iff[11] $\alpha_{nr}((\!|P|\!)_{\phi,\eta})_{\dashv}$ contains only sets whose elements all agree on ρ. We denote with equiv_ρ the set $\{X \subseteq \mathbb{D} \mid \forall \mathsf{v}_1, \mathsf{v}_2 \in X.\ \rho(\mathsf{v}_1) = \rho(\mathsf{v}_2)\}$ of all sets of elements in $\mathbb{D}$ that agree on ρ. We have the following equivalence:

$$f_P \models \mathsf{GANI}[\langle \phi^{=}, \eta^{\neq} \rangle_{\mathbb{I}}, \langle \rho \rangle_{\mathbb{O}}] \;\text{ iff }\; \alpha_{nr}((\!|P|\!)_{\phi,\eta})_{\dashv} \subseteq \mathsf{equiv}_\rho. \tag{3}$$

Example 5.1. Consider the abstractions and the *Program 3* defined in Example 3.5. We have that $(\!|P_3|\!) = \{\{(\mathsf{v}_1, \mathsf{v}_1'), (\mathsf{v}_2, \mathsf{v}_2')\} \mid \mathsf{v}_1' = \mathsf{v}_1 \bmod 2 \text{ and } \mathsf{v}_2' = \mathsf{v}_2 \bmod 2\}$. Furthermore, $(\!|P_3|\!)_{\phi,\eta}$ is:

$$\left\{\{(\mathsf{v}_1, \mathsf{v}_1'), (\mathsf{v}_2, \mathsf{v}_2')\} \in (\!|P_3|\!) \;\middle|\; \mathsf{v}_1 \in \mathbb{Z}_{2k} \cap \mathbb{Z}^{-} \text{ and } \mathsf{v}_2 \in \mathbb{Z}_{2k} \cap \mathbb{Z}^{+0}\right\} \cup \qquad \textit{set A}$$
$$\left\{\{(\mathsf{v}_1, \mathsf{v}_1'), (\mathsf{v}_2, \mathsf{v}_2')\} \in (\!|P_3|\!) \;\middle|\; \mathsf{v}_1 \in \mathbb{Z}_{2k+1} \cap \mathbb{Z}^{-} \text{ and } \mathsf{v}_2 \in \mathbb{Z}_{2k+1} \cap \mathbb{Z}^{+0}\right\} \qquad \textit{set B}$$

For all elements $\{(\mathsf{v}_1, \mathsf{v}_1'), (\mathsf{v}_2, \mathsf{v}_2')\}$ in the set A, by definition of P_3, we have that $\mathsf{v}_1' = \mathsf{v}_2' = 0$. Instead, for all elements $\{(\mathsf{v}_1, \mathsf{v}_1'), (\mathsf{v}_2, \mathsf{v}_2')\}$ in the set B, by definition of P_3, we have that $\mathsf{v}_1' = \mathsf{v}_2' = 1$. So, we have that $\alpha_{nr}((\!|P|\!)_{\phi,\eta})_{\dashv} = \{\{0\}, \{1\}\}$, which is a subset of $\mathsf{equiv}_\rho = \{\{n\} \mid n \in \mathbb{Z}\}$, as expected.

End Example.

Now, let $\mathbb{D}_2 \triangleq \{\{\mathsf{v}_1, \mathsf{v}_2\} \subseteq \mathbb{D} \mid \phi(\mathsf{v}_1) = \phi(\mathsf{v}_2) \;\wedge\; \eta(\mathsf{v}_1) \neq \eta(\mathsf{v}_2)\}$ and $\{\!|P|\!\} : \wp(\mathbb{D}) \to \wp(\mathbb{D})$ be the collecting semantics of P. It is easy to note that $\alpha_{nr}((\!|P|\!)_{\phi,\eta})_{\dashv}$ coincides with the set $\{\{\!|P|\!\}(X) \mid X \in \mathbb{D}_2\}$. By defining a correct collecting hypersemantics $\{\!|P|\!\}_h : \wp(\wp(\mathbb{D})) \to \wp(\wp(\mathbb{D}))$ for P (see again Subsection 2.6 for details), we have that $\{\{\!|P|\!\}(X) \mid X \in \mathbb{D}_2\} \subseteq \{\!|P|\!\}_h(\mathbb{D}_2)$. Hence, the verification of GANI boils down to check $\{\!|P|\!\}_h(\mathbb{D}_2) \subseteq \mathsf{equiv}_\rho$, which implies $\alpha_{nr}((\!|P|\!)_{\phi,\eta})_{\dashv} \subseteq \mathsf{equiv}_\rho$.

This check is undecidable since it still involves the computation of the concrete semantics. To perform verification, we need to abstract the computation of the concrete collecting hypersemantics. We can instantiate the *hyperlevel constants* domain of Reference [33] on $\wp(\mathbb{D})$, since ρ is an abstract domain of $\mathbb{D}$ (i.e., it is a closure on $\wp(\mathbb{D})$). In particular, following the construction of [33], we have that $\rho^{\flat\mathsf{c}} = \wp(\mathit{Atom}(\rho)) \cup \{\rho\}$. Applying the lifting transformer of [33] we obtain $\mathcal{L}(\rho) \triangleq \lambda\mathcal{X}.\ \{\rho(X) \mid X \in \mathcal{X}\}$. Composing the two, we obtain $\alpha_{gani} \triangleq \rho^{\flat\mathsf{c}} \circ \mathcal{L}(\rho)$, that forms, together with its left adjoint $\gamma_{gani} = \alpha_{ani}^{-}$, the Galois connection [33]:

$$\langle \wp(\wp(\mathbb{D})), \subseteq \rangle \underset{\alpha_{gani}}{\overset{\gamma_{gani}}{\leftrightarrows}} \langle \rho^{\flat\mathsf{c}}, \subseteq \rangle$$

[10] We abuse notation denoting with $\subseteq$ both set-inclusion and its component-wise lift to pairs, i.e., $(\mathcal{X}, \mathcal{Y}) \subseteq (\mathcal{T}, \mathcal{Z}) \triangleq (\mathcal{X} \subseteq \mathcal{T} \wedge \mathcal{Y} \subseteq \mathcal{Z})$.

[11] Given a generic pair (a, b), we denote with $(a, b)_{\dashv}$ its projection on the second element.

By construction, $Atom(\rho) = \mathsf{equiv}_\rho$, which belongs to $\rho^{\mathfrak{bc}}$. We finally have that if $\alpha_{gani}(\{\!|P|\!\}_h(\mathbb{D}_2)) \subseteq \mathsf{equiv}_\rho$ then $\{\!|P|\!\}_h(\mathbb{D}_2) \subseteq \mathsf{equiv}_\rho$. In turn, this implies that $\{[\![P]\!]\} \subseteq \mathrm{HP}^{\mathsf{GANI}}$. Hence, we can soundly approximate the verification of GANI by computing the approximated (abstract) hypersemantics on the hyper domain $\rho^{\mathfrak{bc}}$, checking whether all pairs of computations have constant value in ρ. With a sound abstract hypersemantics $\{\!|P|\!\}_h^\rho$, namely a semantics such that for every $\mathcal{X}$ in $\wp(\wp(\mathbb{D}))$ it holds $\alpha_{gani}(\{\!|P|\!\}_h(\mathcal{X})) \subseteq \{\!|P|\!\}_h^\rho(\alpha_{gani}(\mathcal{X}))$, we can move the verification completely on the abstract domain.

Theorem 5.2 (GANI Verification). *Given $\mathbb{D}_2^\rho$ in $\rho^{\mathfrak{bc}}$ such that $\mathbb{D}_2 \subseteq \gamma_{gani}(\mathbb{D}_2^\rho)$, if $\{\!|P|\!\}_h^\rho(\mathbb{D}_2^\rho) \subseteq \mathsf{equiv}_\rho$ then P satisfies* GANI*, that is, $f_P \models \mathsf{GANI}[\langle\phi^{=}, \eta^{\neq}\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$.*

Proof. The proof is a trivial generalization of the proof of Theorem 4.6 provided in [34]. □

Note on implementation. GANI is indeed a collection of hyperproperties, since it is parametric on the abstractions ϕ, η and ρ. The abstract hypersemantics is strictly related to these abstractions. Hence, an actual abstract hypersemantics for GANI cannot be systematically defined. Once the abstractions are fixed, an abstract hypersemantics can be defined, for instance, following the approach used for non-interference described in [34].

5.2 A Verification Mechanism for Opacity

Being also Opacity a hyperproperty, its verification requires comparing different program executions. Again, we recall that here we consider as program executions just the program input-output pairs (see Section 2.6 for details). Let us observe that Symmetric Opacity (of Definition 2.1 and of [41]) can be formulated as the hyperproperty $\mathrm{HP}^{\mathsf{SOP}}(\mathsf{P}) \triangleq \mathrm{HP}^{\mathsf{OP}}(\mathsf{P}) \cap \mathrm{HP}^{\mathsf{OP}}(\overline{\mathsf{P}})$, where:

$$\mathrm{HP}^{\mathsf{OP}}(\mathsf{P}) \triangleq \left\{ X \subseteq \mathbb{I} \times \mathbb{O} \;\middle|\; \forall (\mathsf{v}_1, \mathsf{v}_1') \in X\ (\mathsf{v}_1 \in \mathsf{P})\ \exists (\mathsf{v}_2, \mathsf{v}_2') \in X\ (\mathsf{v}_2 \notin \mathsf{P}).\ \mathsf{v}_1 \overset{\iota}{\sim} \mathsf{v}_2,\ \wedge\ \mathsf{v}_1' \overset{o}{\sim} \mathsf{v}_2' \right\}$$

It is easy to note that $\mathrm{HP}^{\mathsf{SOP}}(\mathsf{P})$ is not subset-closed[12] [7, 32]. Hence, it is quite a general hyperproperty that is hard to verify using static analysis. Nevertheless, we can exploit the connection between GANI and Abstract Opacity, as introduced in Definition 3.7, to verify Symmetric Opacity.

Indeed, we have already probed the equivalence between Abstract Opacity and Symmetric Opacity, and that GANI implies Abstract Opacity under some assumptions.

Theorem 5.3. *Let $\overset{\iota}{\sim}$ in $Eq(\mathbb{I})$ and $\overset{o}{\sim}$ in $Eq(\mathbb{O})$ be equivalence relations on input and output values, respectively, and $\mathsf{P} \subseteq \mathbb{I}$ be a binary predicate on input values. If for each input value v in $\mathbb{I}$ there exists another input value v' in $\mathbb{I}$ such that $\phi_\iota(\mathsf{v}) = \phi_\iota(\mathsf{v}')$ and $\rho_o(\mathsf{v}) \neq \rho_o(\mathsf{v}')$, then:*

$$f \models \mathsf{GANI}[\langle\phi_\iota^{=}, \eta_{\mathsf{P}}^{\neq}\rangle_{\mathbb{I}}, \langle\rho_o\rangle_{\mathbb{O}}] \quad \textit{implies} \quad \mathsf{P} \in \mathsf{SOp}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$$

Proof. The proof trivially follows from Proposition 3.8 and Corollary 3.11. □

Example 5.4. Consider the abstractions and the *Program 3* defined in Example 3.5. Now, let $\overset{\iota}{\sim} = Rel^{\phi} = \{(z_1, z_2) \mid z_1, z_2 \in \mathbb{Z}_{2k} \text{ or } z_1, z_2 \in \mathbb{Z}_{2k+1}\}$. Note that the abstraction η from Example 3.5 is the partitioning closure associated with the binary predicate $\mathsf{P} \triangleq \{z \mid z \in \mathbb{Z}^{+0}\}$. The premises of Theorem 5.3 are trivially fulfilled, namely for each v_1 in $\mathbb{I}$ there exists v_2 in $\mathbb{I}$ such that $\phi(\mathsf{v}_1) = \phi(\mathsf{v}_2)$, or, equivalently, $\mathsf{v}_1 \overset{\iota}{\sim} \mathsf{v}_2$ (i.e., having the same parity) and $\eta(\mathsf{v}_1) \neq \eta(\mathsf{v}_2)$ (i.e., having different sign). For instance, if v_1 is even and greater than or equal to zero, then we can take another even number lower than zero. We have already seen that $f \models \mathsf{GANI}[\langle\phi^{=}, \eta^{\neq}\rangle_{\mathbb{I}}, \langle\rho\rangle_{\mathbb{O}}]$ holds. By the same case analysis of Example 3.5, we also have that $\rho f(\mathsf{v}_1) = \rho f(\mathsf{v}_2)$ (i.e., the same constant is observed

[12] Subsets of opaque semantics may not be opaque due to the existential quantifier in the Opacity definition.

in output). Hence, given $\overset{o}{\sim} = Rel^{\rho} = \{(z,z) \mid z \in \mathbb{Z}\}$, we have that $f(\mathsf{v}_1) \overset{o}{\sim} f(\mathsf{v}_2)$. Then, we can conclude that $\mathsf{P} \in \mathsf{SOp}(f, \overset{\iota}{\sim}, \overset{o}{\sim})$.

End Example.

This means that we can reuse enforcement and verification mechanisms developed for GANI in Subsection 5.1 to verify Opacity. Indeed, due to Theorem 5.3, Symmetric Opacity $\mathsf{P} \in \mathsf{SOp}(f_P, \overset{\iota}{\sim}, \overset{o}{\sim})$ is implied by $f_P \models \mathsf{GANI}[\langle \phi_\iota^{=}, \eta_{\mathsf{P}}^{\neq} \rangle_{\mathbb{I}}, \langle \rho_o \rangle_{\mathbb{O}}]$ where $\phi_\iota = Clo^{\overset{\iota}{\sim}}$, $\rho_o = Clo^{\overset{o}{\sim}}$, and $\eta_{\mathsf{P}} = \{\varnothing, \mathsf{P}, \overline{\mathsf{P}}, \mathbb{I}\}$; and assuming that for all v_1 in P there exists v_2 in $\overline{\mathsf{P}}$ such that $\phi_\iota(\mathsf{v}_1) = \phi_\iota(\mathsf{v}_2)$. The latter check can be easily performed by looking at the structure of ϕ_ι. Therefore, to show that a predicate P is symmetrically opaque for f_P, we may just check GANI for P. In particular, the check boils down to $\alpha_{nr}(\langle\!|P|\!\rangle_{\phi_\iota,\eta_{\mathsf{P}}})_{\dashv} \subseteq \mathsf{equiv}_{\rho_o}$, where

$$\mathsf{equiv}_{\rho_o} \triangleq \{X \subseteq \mathbb{D} \mid \forall \mathsf{v}_1, \mathsf{v}_2 \in X.\ \rho_o(\mathsf{v}_1) = \rho_o(\mathsf{v}_2)\}$$

is the set of all sets of elements in $\mathbb{D}$ that agree on ρ_o, and

$$\langle\!|P|\!\rangle_{\phi_\iota,\eta_{\mathsf{P}}} \triangleq \left\{\{(\mathsf{v}_1,\mathsf{v}_1'),(\mathsf{v}_2,\mathsf{v}_2')\} \in \langle\!|P|\!\rangle \;\middle|\; \phi_\iota(\mathsf{v}_1) = \phi_\iota(\mathsf{v}_2) \text{ and } \eta_{\mathsf{P}}(\mathsf{v}_1) \neq \eta_{\mathsf{P}}(\mathsf{v}_2)\right\}$$

Similarly to the GANI case, if $\alpha_{nr}(\langle\!|P|\!\rangle_{\phi_\iota,\eta_{\mathsf{P}}})_{\dashv} \subseteq \mathsf{equiv}_{\rho_o}$ holds, then $\mathsf{P} \in \mathsf{SOp}(f_P, \overset{\iota}{\sim}, \overset{o}{\sim})$. Again, it is easy to note that $\alpha_{nr}(\langle\!|P|\!\rangle_{\phi_\iota,\eta_{\mathsf{P}}})_{\dashv}$ coincides with the set $\{\{\!|P|\!\}(X) \mid X \in \mathbb{D}_2\}$, where $\mathbb{D}_2 \triangleq \{\{\mathsf{v}_1,\mathsf{v}_2\} \subseteq \mathbb{I} \mid \phi_\iota(\mathsf{v}_1) = \phi_\iota(\mathsf{v}_2) \text{ and } \eta_{\mathsf{P}}(\mathsf{v}_1) \neq \eta_{\mathsf{P}}(\mathsf{v}_2)\}$. By definition of correct hypersemantics, we have that $\{\{\!|P|\!\}(X) \mid X \in \mathbb{D}_2\} \subseteq \{\!|P|\!\}_h(\mathbb{D}_2)$. So, we just need to check $\{\!|P|\!\}_h(\mathbb{D}_2) \subseteq \mathsf{equiv}_{\rho_o}$, in turn implying $\alpha_{nr}(\langle\!|P|\!\rangle_{\phi_\iota,\eta_{\mathsf{P}}})_{\dashv} \subseteq \mathsf{equiv}_{\rho_o}$.

This check is also undecidable. We can then again exploit the hyperlevel constants domain (parametric in ρ_o this time) $\rho_o^{\mathfrak{bc}} \triangleq \wp(Atom(\rho_o)) \cup \{\rho_o\}$ to compute an abstract collecting hypersemantics approximating the check. The Galois connection is given by the abstraction function $\alpha_{sop} \triangleq \rho_o^{\mathfrak{bc}} \circ \mathcal{L}(\rho_o)$ and its corresponding concretization $\gamma_{sop} = \alpha_{sop}^{-}$. By construction, $Atom(\rho_o) = \mathsf{equiv}_{\rho_o}$, which belongs to $\rho_o^{\mathfrak{bc}}$. We then have that $\alpha_{sop}(\{\!|P|\!\}_h(\mathbb{D}_2)) \subseteq \mathsf{equiv}_{\rho_o}$ implies $\{\!|P|\!\}_h(\mathbb{D}_2) \subseteq \mathsf{equiv}_{\rho_o}$. Finally, with a sound abstract hypersemantics $\{\!|P|\!\}_h^{\rho_o}$, we can move the verification of Symmetric Opacity totally on the abstract domain.

Theorem 5.5 (Opacity Verification). *Given $\mathbb{D}_2^{\rho_o}$ in $\rho_o^{\mathfrak{bc}}$ such that $\mathbb{D}_2 \subseteq \gamma_{sop}(\mathbb{D}_2^{\rho_o})$, if $\{\!|P|\!\}_h^{\rho_o}(\mathbb{D}_2^{\rho_o}) \subseteq \mathsf{equiv}_{\rho_o}$ then P satisfies Symmetric Opacity, that is, $\mathsf{P} \in \mathsf{SOp}(f_P, \overset{\iota}{\sim}, \overset{o}{\sim})$.*

Proof. The proof follows from Theorems 5.3 and 5.2. □

The parametricity of Theorem 5.5 in the predicate P to check is given by $\mathbb{D}_2$ (and, hence, by $\mathbb{D}_2^{\rho_o}$), that is defined in terms of P. Moreover, as it happens for GANI, Opacity is also a collection of hyperproperties, parametric on the equivalence relations $\overset{\iota}{\sim}$ and $\overset{o}{\sim}$, and the predicate P to check. Hence, again, an actual abstract hypersemantics for Opacity can be defined only when the abstractions are fixed.

Note on precision. Being Opacity a hyperproperty containing a $\forall\exists$ quantifier alternation, it is theoretically very challenging to statically verify, and even to dynamically test [11]. The proposed verification analysis for Opacity trades soundness for an inevitable precision loss. We indeed had to resort to approximating Opacity verification with ANI verification, which is theoretically simpler. Thus, our analysis can lose precision when used to verify Opacity. Nevertheless, while possibly yielding false positives, our analysis is sound and practically fast, being the abstract domain adopted limited in size.

6 Conclusion

In this article, we investigated the relation between two fundamental confidentiality notions, i.e., opacity [39, 41] and abstract non-interference [21], showing how they represent a different

perspective on the same security issue. To do so, we provided a formal framework for describing the different flavors of abstract non-interference and (abstract) opacity in the same setting, allowing us to compare the two notions formally. In particular, we proved that abstract non-interference is stronger than opacity under specific assumptions. It should be mentioned that the proposed formalism is not specific to abstract non-interference and opacity considered in the article, but it can be used to model generic confidentiality notions.

Another outcome of understanding the connection between opacity and abstract non-interference has been the characterization of a novel confidentiality notion modeling code obfuscation, which can be useful to check whether a system really needs to be obfuscated. This represents a more precise formalization than the one proposed in References [15, 18, 19, 22] based on the incompleteness of program analyses. Indeed, being an existential property (i.e., its refutation requires a single counterexample), incompleteness is too weak for being realistic in the context of code obfuscation.

The proven relation between opacity and abstract non-interference also led to effective verification of opacity by extending what has been done for non-interference in Reference [34]. Specifically, adapting the abstract interpretation-based verification mechanism of Reference [34] to abstract non-interference resulted in effective verification of opacity by exploiting the connection between the two notions discovered in the current article.

Additionally, by leveraging the introduced formalism, we were able to model in the same setting also (abstract) final opacity [6], checking opacity on outputs instead of inputs. Due to its "opposite" nature, the verification mechanism existing for non-interference cannot be seamlessly extended to cope with final opacity. As future work, we plan to investigate this issue in order to make feasible the verification of confidentiality notions focusing on outputs rather than inputs.

From a practical point of view, as future work, we aim at extending the implementation for non-interference [34] to concretely provide a static analysis for verifying the proposed confidentiality properties when abstractions have been specified. From a theoretical point of view, we can observe that from the proposed confidentiality properties, we can derive their dual versions. Intuitively, a dual confidentiality property checks whether a program *does not* satisfy the execution requirement (as per Section 3) of the original confidentiality property. For instance, the dual of Abstract Non-Interference (Definition 2.4) describes a form of *abstract dependency* between input and output data-properties. In contrast, the dual of Abstract Opacity (Definition 4.3) describes a form of *abstract transparency* of an input data-property. Since the notion of abstract dependency is not new in the literature [31, 37], we believe that a deeper understanding of such notions, in particular with respect to the existing literature, deserves further investigation.

Another line of future work could be to consider *active attackers* in our framework. Active attackers introduce the capability of an attacker to alter code, rather than simply observing code execution. A path to follow when considering such an attack model is to check program *robustness*, in addition to confidentiality compliance. In this context, robustness, as introduced in Reference [42], asserts that an active attacker, who can modify program code in some fixed points (holes), is not able to disclose more private information than a passive attacker, who merely observes public data. The problem of checking abstract non-interference under robustness assumptions (i.e., considering active attackers) has been tackled in [2] by exploiting a weakest precondition semantics. In particular, this semantics simulates the analysis an attacker can perform backward, i.e., from the public output towards the private input. However, such a work has been developed on the specific notion of ANI originally proposed for security (allowing a public/private partition of data only), and not on GANI, which is more general. Hence, extending GANI to deal with active attackers within the definition itself is surely an interesting future work. A possibility here could be to introduce "holes" in programs as done in [2, 42], and quantify over the code that can fill such holes. Such an

extension will impact our verification mechanism. The analysis proposed in Section 5 builds on an abstract interpretation-based hyperproperty check, which is inductively defined on the syntax of a *single program*. To the best of our knowledge, there are no abstract interpretation-based static analyses considering *multiple programs* at the same time, and it could be interesting to consider this problem as well. A possibility here could be to take inspiration from recent developments in program synthesis [27], where simple program properties are verified for infinite sets of programs by exploiting a Hoare logic-like analysis.

6.1 Related Work

The original notion of opacity [6] defined for Labeled Transition Systems (LTSs) only considers an observation on the output of the computation. More recently, the same notion defined for programs [41] has been extended by considering also an input observation. In Reference [41], it is shown how this notion is crucial in several privacy scenarios, seeking to answer fundamental questions on understanding and enforcing opacity. Moreover, it is shown that a program satisfies non-interference if it is opaque for the possible input predicates. Unfortunately, while adding the input abstraction, in Reference [41], the relation between the observed input data property and the predicate to protect has not been investigated (it should be clear that if the abstraction implies the predicate, then it cannot be opaque, independently of the analyzed computation). In our approach, where the observation capability and the data property to protect are modeled in the same way, this relation becomes clear.

Concerning opacity verification, Balun and Masopust [3] provide an algorithmic approach to verify specific flavors of *state-based opacity*, that is, weak and strong k-step opacity, on Discrete Event Systems (DESs). Here, k-step opacity is an opacity notion considered in the context of DESs that characterizes a system's ability to maintain confidential behavior for a bounded time window under passive attackers. In essence, a system is k-step opaque if a passive attacker, who only sees a limited set of events, cannot definitively determine that the system was in a secret state at any time instant within the last k observable steps leading up to the current observation. The weak version of k-step opacity says that the attacker cannot determine the exact time when the system was in a secret state within the last k observable steps leading up to the current observation. Bourouis et al. [4] consider the same flavors of opacity on finite LTSs. Then, they present efficient algorithms for verifying opacity in all these forms with an on-the-fly, dynamic approach. Similarly, Falcone and Marchand [17] show how to model-check, verify, and enforce k-step opacity at runtime on LTSs. Klein et al. [28] provide an algorithmic approach to verify current-state opacity, corresponding to k-opacity with no time window (i.e., $k = 0$) on timed automata. Saadaoui et al. [40] discuss the formalization and verification of *language-based opacity* when considering DESs represented by partially observable Petri nets. The difference between state-based and language-based opacity is that the latter considers the whole computation, while the former considers only the current system state. Specifically, they identify two opacity notions, called *consistency* and *non-secrecy*, and exploit the mathematical characterization of a Petri net system to separately check each notion. The combined verification allows for the ultimately verification of language-based opacity.

Studying the correlation between the previously mentioned flavors of opacity and the classic notions of opacity considered in the current article would be interesting, but we have not yet investigated this aspect. Since our formalization can be instantiated with different denotations of program executions, we believe that modeling k-step opacity and similar notions as instantiations of Abstract (Final) Opacity would be possible. From the point of view of verification, all previous approaches consider DESs, while ours considers actual programs. Hence, the applicability of our approach is more direct, since it does not require a program to DSE encoding.

Finally, Liu et al. [29] introduce a quantifiable measure of current-state opacity that considers the likelihood of satisfying opacity for stochastic control systems modeled as general Markov Decision Processes (gMDPs). They also propose verification methods tailored to this novel quantitative notion of opacity for finite gMDPs by using value iteration techniques.

Our formalization of opacity considers abstractions on program execution denotations, modeled in terms of data properties. Thus, our approximation of the system behavior is qualitative and orthogonal to the quantitative approximation considered by Liu et al. Nevertheless, it would be interesting to study the correlation between these two apparently different notions of opacity in a formal setting.

An initial investigation of the relation between opacity and abstract non-interference has been carried out in our previous work [35], for which the current article represents a follow-up work. In the current article, we propose a formalism and a methodology to reason about confidentiality properties in general terms, for which opacity and abstract non-interference are particular cases. By exploiting the formal framework proposed in the current article, we were able to generalize and extend the results from [35]. Indeed, differently from [35], the relation proved in the current article does not require any assumption on the abstractions adopted. Moreover, by incrementally reasoning on confidentiality notions, starting from execution requirements and then adding execution quantifiers, in the current article, we were able to draw simpler connections between opacity and abstract non-interference, and to characterize confidentiality properties on single executions. In addition, we investigated the connection between opacity and abstract non-interference with an apparently unrelated concept like code obfuscation. This investigation led us to provide a way for modeling more realistic obfuscation properties, enabling the identification of programs for which obfuscation is not necessary since the data-property to protect is already concealed. As far as verification is concerned, the approach provided in the current article is a simple adaptation of the one proposed in Reference [35] to the new formalism introduced.

References

[1] M. Assaf, D. A. Naumann, J. Signoles, E. Totel, and F. Tronel. 2017. Hypercollecting semantics and its application to static analysis of information flow. In *Proc. of POPL*. 874–887.

[2] M. Balliu and I. Mastroeni. 2010. A weakest precondition approach to robustness. *LNCS Transactions on Computational Science* 10 (2010), 261–297. DOI: https://doi.org/10.1007/978-3-642-17499-5

[3] J. Balun and T. Masopust. 2023. Verifying weak and strong k-step opacity in discrete-event systems. *Automatica* 155 (2023), 111153. DOI: https://doi.org/10.1016/j.automatica.2023.111153

[4] Amina Bourouis, Kais Klai, Yamen El Touati, and Nejib Ben Hadj-Alouane. 2015. Checking opacity of vulnerable critical systems on-the-fly. *Int. J. Inf. Technol. Web Eng.* 10, 1 (Jan. 2015), 1–30. DOI: https://doi.org/10.4018/ijitwe.2015010101

[5] R. Bruni, R. Giacobazzi, R. Gori, and F. Ranzato. 2021. A logic for locally complete abstract interpretations. In *Symposium on Logic in Computer Science, LICS*. IEEE, New York City, US, 1–13.

[6] J. W. Bryans, M. Koutny, L. Mazaré, and P. Y. A. Ryan. 2008. Opacity generalised to transition systems. *Int. J. Inf. Sec.* 7, 6 (2008), 421–435.

[7] M. R. Clarkson and F. B. Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.

[8] E. Cohen. 1977. Information transmission in computational systems. *SIGOPS Oper. Syst. Rev.* 11, 5 (1977), 133–139.

[9] C. Collberg and J. Nagra. 2009. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley Professional.

[10] C. Collberg, C. Thomborson, and D. Low. 1997. *A Taxonomy of Obfuscating Transformations*. Technical Report 148. Department of Computer Sciences, The University of Auckland. Retrieved from http://www.cs.auckland.ac.nz/$\sim$collberg/Research/Publications/CollbergThomborsonLow97a/index.html

[11] A. Correnson, T. Nießen, B. Finkbeiner, and G. Weissenbacher. 2024. Finding ∀∃ hyperbugs using symbolic execution. *Proc. ACM Program. Lang.* 8, OOPSLA2, Article 321 (Oct. 2024), 26 pages. DOI: https://doi.org/10.1145/3689761

[12] P. Cousot and R. Cousot. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL*. 238–252.

[13] P. Cousot and R. Cousot. 1979. Systematic design of program analysis frameworks. In *Proc. of Conf. Record of the 6th ACM Symp. on Principles of Programming Languages (POPL'79)*. ACM Press, New York, 269–282.

[14] M. Dalla Preda and Roberto Giacobazzi. 2005. Semantic-based code obfuscation by abstract interpretation. In *Proc. of ICALP*. 1325–1336.
[15] M. Dalla Preda and I. Mastroeni. 2018. Characterizing a property-driven obfuscation strategy. *J. Comput. Secur.* 26, 1 (2018), 31–69.
[16] D. Devriese and F. Piessens. 2010. Noninterference through secure multi-execution. In *2010 IEEE Symposium on Security and Privacy*. 109–124. DOI: https://doi.org/10.1109/SP.2010.15
[17] Y. Falcone and H. Marchand. 2015. Enforcement and validation (at runtime) of various notions of opacity. *Discrete Event Dynamic Systems* 25, 4 (Dec. 2015), 531–570. DOI: https://doi.org/10.1007/s10626-014-0196-4
[18] R. Giacobazzi. 2008. Hiding information in completeness holes—new perspectives in code obfuscation and watermarking. In *Proc. of The 6th IEEE International Conferences on Software Engineering and Formal Methods (SEFM'08)*. IEEE Press, 7–20.
[19] R. Giacobazzi, N. D. Jones, and I. Mastroeni. 2012. Obfuscation by partial evaluation of distorted interpreters. In *Proc. of the ACM SIGPLAN Symp. on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'12)*, O. Kiselyov and S. Thompson (Eds.). ACM Press, 63–72.
[20] R. Giacobazzi and I. Mastroeni. 2010. Adjoining classified and unclassified information by abstract interpretation. *Journal of Computer Security* 18, 5 (2010), 751–797.
[21] R. Giacobazzi and I. Mastroeni. 2018. Abstract non-interference: A unifying framework for weakening information-flow. *ACM Trans. Priv. Secur.* 21, 2 (2018), 1–31.
[22] Roberto Giacobazzi, Isabella Mastroeni, and Mila Dalla Preda. 2017. Maximal incompleteness as obfuscation potency. *Formal Aspects Comput.* 29, 1 (2017), 3–31.
[23] R. Giacobazzi, F. Ranzato, and F. Scozzari. 2000. Making abstract interpretation complete. *Journal of the ACM* 47, 2 (2000), 361–416.
[24] Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. 2000. Making abstract interpretations complete. *J. ACM* 47, 2 (2000), 361–416. DOI: https://doi.org/10.1145/333979.333989
[25] D. J. D. Hughes and V. Shmatikov. 2004. Information hiding, anonymity and privacy: A modular approach. *J. Comput. Secur.* 12, 1 (2004), 3–36.
[26] S. Hunt and I. Mastroeni. 2005. The PER model of abstract non-interference. In *Proc. of SAS*. 171–185.
[27] Jinwoo Kim, Shaan Nagy, Thomas Reps, and Loris D'Antoni. 2025. Semantics of sets of programs. *Proc. ACM Program. Lang.* 9, OOPSLA1, Article 110 (April 2025), 27 pages. DOI: https://doi.org/10.1145/3720515
[28] Julian Klein, Paul Kogel, and Sabine Glesner. 2024. Verifying opacity of discrete-timed automata. In *Proceedings of the 2024 IEEE/ACM 12th International Conference on Formal Methods in Software Engineering (FormaliSE) (FormaliSE'24)*. Association for Computing Machinery, New York, NY, USA, 55–65. DOI: https://doi.org/10.1145/3644033.3644376
[29] Siyuan Liu, Xiang Yin, Dimos V. Dimarogonas, and Majid Zamani. 2025. On approximate opacity of stochastic control systems. *IEEE Trans. Automat. Control* 70, 6 (2025), 3846–3861. DOI: https://doi.org/10.1109/TAC.2024.3516202
[30] I. Mastroeni and A. Banerjee. 2011. Modelling declassification policies using abstract domain completeness. *Mathematical Structures in Computer Science* 21, 6 (2011), 1253–1299.
[31] I. Mastroeni and D. Nikolic. 2010. An abstract unified framework for (abstract) program slicing. In *12th International Conference on Formal Engineering Methods, ICFEM 201 (Lecture Notes in Computer Science)*. Spinger-Verlag, 452–467.
[32] I. Mastroeni and M. Pasqua. 2017. Hyperhierarchy of semantics - A formal framework for hyperproperties verification. In *Proc. of SAS*. 232–252.
[33] I. Mastroeni and M. Pasqua. 2018. Verifying bounded subset-closed hyperproperties. In *Proc. of SAS*. 1–20.
[34] I. Mastroeni and M. Pasqua. 2019. Statically analyzing information flows: An abstract interpretation-based hyper-analysis for non-interference. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019*. 2215–2223. DOI: https://doi.org/10.1145/3297280.3297498
[35] I. Mastroeni and M. Pasqua. 2022. Verifying opacity by abstract interpretation. In *SAC'22: The 37th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, April 25 - 29, 2022*, Jiman Hong, Miroslav Bures, Juw Won Park, and Tomás Cerný (Eds.). ACM, 1817–1826. DOI: https://doi.org/10.1145/3477314.3507119
[36] I. Mastroeni and M. Pasqua. 2023. Domain precision in galois connection-less abstract interpretation. In *Static Analysis*, Manuel V. Hermenegildo and José F. Morales (Eds.). Springer Nature Switzerland, Cham, 434–459.
[37] I. Mastroeni and D. Zanardini. 2017. Abstract program slicing: An abstract interpretation-based approach to program slicing. *ACM Trans. Comput. Logic* 18, 1, Article 7 (feb 2017), 58 pages. DOI: https://doi.org/10.1145/3029052
[38] F. Ranzato and F. Tapparo. 2004. Strong preservation as completeness in abstract interpretation. In *Proc. of ESOP*. 18–32.
[39] P.Y.A. Ryan and T. Peacock. 2006. Opacity - Further Insights on an Information Flow Property. (2006). Technical Report CS-TR-958, University of Newcastle upon Tyne.

[40] I. Saadaoui, A. Labed, Z. Li, A. M. El-Sherbeeny, and H. Du. 2023. Language-based opacity verification in partially observed petri nets through linear constraints. *Mathematics* 11, 18 (2023). DOI: https://doi.org/10.3390/math11183880
[41] D. Schoepe and A. Sabelfeld. 2015. Understanding and enforcing opacity. In *Proc. of CSF*. 539–553.
[42] S. Zdancewic and A. C. Myers. 2001. Robust declassification. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations (CSFW'01)*. IEEE Computer Society, USA, 5.

Received 22 April 2025; revised 26 September 2025; accepted 1 December 2025