Math. Struct. in Comp. Science (2019), *vol.* 29, *pp.* 339–388. © Cambridge University Press 2018
doi:10.1017/S0960129518000038 First published online 15 May 2018

Semantics-based software watermarking by abstract interpretation

MILA DALLA PREDA[†] and MICHELE PASQUA

Department of Computer Science, University of Verona
Strada le Grazie 15, Verona 37134, Italy
Emails: mila.dallapreda@univr.it, michele.pasqua@univr.it

Received 10 November 2016; revised 22 January 2018

Software watermarking is a software protection technique used to defend the intellectual property of proprietary code. In particular, software watermarking aims at preventing software piracy by embedding a signature, i.e. an identifier reliably representing the owner, in the code. When an illegal copy is made, the owner can claim his/her identity by extracting the signature. It is important to hide the signature in the program in order to make it difficult for the attacker to detect, tamper or remove it. In this work, we present a formal framework for software watermarking, based on program semantics and abstract interpretation, where attackers are modelled as abstract interpreters. In this setting, we can prove that the ability to identify signatures can be modelled as a completeness property of the attackers in the abstract interpretation framework. Indeed, hiding a signature in the code corresponds to embed it as a semantic property that can be retrieved only by attackers that are complete for it. Any abstract interpreter that is not complete for the property specifying the signature cannot detect, tamper or remove it. We formalize in the proposed framework the major quality features of a software watermarking technique: secrecy, resilience, transparence and accuracy. This provides a unifying framework for interpreting both watermarking schemes and attacks, and it allows us to formally compare the quality of different watermarking techniques. Indeed, a large number of watermarking techniques exist in the literature and they are typically evaluated with respect to their secrecy, resilience, transparence and accuracy to attacks. Formally identifying the attacks for which a watermarking scheme is secret, resilient, transparent or accurate can be a complex and error-prone task, since attacks and watermarking schemes are typically defined in different settings and using different languages (e.g. program transformation vs. program analysis), complicating the task of comparing one against the others.

1. Introduction

A major issue in computer security is the protection of proprietary software against attacks that aim at stealing, modifying or tampering with the code in order to obtain (economic) advantages over it. Frontier Economics and the Business Software Alliance (BSA) estimated that the global value of software piracy in 2015 was around 24$ Billion, Frontier Economics also estimated that the projection of this value for 2022 is 42–95$ Billion (BSA 2016; Frontier-Economics 2016). Software developers are interested in

[†] This work was partly supported by the MIUR FIRB 2013 project FACE RBFR13AJFT.

protecting the intellectual property of their products against software piracy, namely to prevent the illegal (unlicensed) use of their code.

A key challenge in defending code running on an untrusted host is that there is no limit on the techniques that the host can use to extract sensitive data from the code and to violate its intellectual property and integrity. Code obfuscation, whose aim is to obstruct code comprehension, represents a preventive tool against software piracy: attackers cannot steal or tamper what they do not understand (Collberg *et al.* 1998). Once an attacker goes beyond this defence, software watermarking allows the owner of the violated code to prove the ownership of the pirated copies (Collberg and Thomborson 2002, 1999; Davidson and Myhrvold 1996; Moskowitz and Cooperman 1996). Software watermarking is a technique for embedding a signature, i.e. an identifier reliably representing the owner, in a cover program. This allows software developers to prove their ownership by extracting their signature from the pirated copies. In the last two decades, researchers have developed a variety of software watermarking techniques, e.g. (Collberg and Thomborson 2002, 1999; Dalla Preda *et al.* 2008; Nagra *et al.* 2002), that can be classified in three main categories according to their extraction process: static, dynamic and abstract watermarking. *Static watermarking* inserts signatures in the cover program either as data or code and then extracts them statically, namely without executing the code (Collberg and Thomborson 1999). Conversely, *dynamic watermarking* inserts signatures in the program execution state (i.e. in its semantics) and the extraction process requires the execution of the program, often on a special enabling input (Collberg and Thomborson 1999). *Abstract watermarking*, introduced in Cousot and Cousot (2004), encodes the signature in such a way that it could be extracted only by a suitable abstract execution of the program.

The efficiency of a watermarking scheme is typically evaluated according to the following features: *credibility* that measures how strongly it proves authorship, *secrecy* that deals with the complexity degree of signatures extraction by attackers, *transparency* that measures how difficult it is for an attacker to realize that a program is marked, *accuracy* that evaluates the observational equivalence of the marked and original program, *resilience* that measures how difficult it is for an attacker to compromise the correct extraction of the signature and *data-rate* that considers the amount of information that can be encoded by the considered watermarking scheme. When researchers propose a new watermarking technique they usually claim its efficiency in terms of the aforementioned features by discussing how the peculiar signature embedding and extraction methods are able to ensure good degrees of quality with respect to different attackers. However, existing embedding and extraction algorithms often work on different objects (control flow graph, variables, registers, etc) and attackers may use different program analysis techniques to compromise the embedded signature. This makes it difficult to formally compare the efficiency of different watermarking systems with respect to attacks and to discuss limits and potentialities of the watermarking schemes in order to decide which one is better to use in a given scenario. A unifying framework for software watermarking and attackers would help this evaluation.

These problems also derive from the lack of theoretical studies on software watermarking. Software watermarking has been formally defined in Barak *et al.* (2001), where the authors show that the existence of indistinguishability obfuscators implies that software watermarking cannot exist. Furthermore, the recent candidate construction of an indistinguishability obfuscator (Garg *et al.* 2013) lowers the hope of building meaningful watermarking scheme. Fortunately, this impossibility results rely on the fact that the signed program computes the same function as the original program. Indeed, in Barak *et al.* (2001) the authors suggest that if we relax this last constraint, i.e. we require that the watermarking process has only to preserve an ‘approximation’ of the original program’s functionality, then positive results might come. This naturally leads to reason about software watermarking at semantic level, as we do in the present work.

A first attempt to provide a formal semantics-based definition of software watermarking has been proposed in Giacobazzi (2008). Here, the author introduces the idea of viewing static and dynamic watermarking schemes as particular instances of a general abstract watermarking scheme. Intuitively, abstract watermarking is static because no execution is needed for signature extraction, and dynamic because the signature is hidden in the semantics of the code. The idea is to see static, dynamic and abstract watermarking techniques as particular instances of a common watermarking scheme based on program semantics and abstract interpretation. In this work, we follow this intuition and we transform the scheme proposed in Giacobazzi (2008) in a formal and consistent definition of a software watermarking system. The idea is to embed a signature s in a program by encoding it as a semantic property $\overline{\mathfrak{M}}(s)$, to be inserted in the semantics of the cover program. In this setting, the extraction process requires an analysis of the semantics of the marked code that has to be at least as precise as $\overline{\mathfrak{M}}(s)$. Interestingly, this notion of precision of the extraction corresponds to the notion of completeness of the analysis in abstract interpretation. This means that in order to extract the signature it is necessary to know how it is encoded. In this view, the semantic property for which the analysis has to be complete in order to extract the signature plays the role of an extraction key. The signature is hidden to any observer/analyzer of program’s semantics that is incomplete for $\overline{\mathfrak{M}}(s)$, namely to any observer/analyzer that does not know the ‘secret key’.

Based on these ideas, we provide a formal semantics-based definition of a watermarking system. Moreover, we provide a specification of the quality features of a watermarking system in terms of semantic program properties. For example, it turns out that a watermarking scheme is transparent *w.r.t.* an observer when the embedding process preserves the program properties in which the observer is interested. Moreover, the resilience of a watermarking scheme to collusive attacks, which attempt to remove the signature by comparing different marked programs, can be modelled as a property of abstract non-interference (Giacobazzi and Mastroeni 2004) among programs. Finally, we do a wider and more precise validation rather the one done in Giacobazzi (2008) (which is just sketched). We take into account five known watermarking techniques and we define them in our framework, with a comparison of their quality features. Our investigation and study in this direction has led to the following contributions.

— Specification of a formal framework based on program semantics and abstract interpretation for modelling software watermarking. The framework refines and extends the one proposed in Giacobazzi (2008).

- — Formalization of the quality features (resilience, secrecy, transparence and accuracy) used to measure the quality of a watermarking system in the framework.
- — Validation of the framework on five watermarking techniques, together with a qualitative comparison of their features.

The results presented in this work are a revised and extended version of Dalla Preda and Pasqua (2016).

2. Preliminaries

In this section, we present the background knowledge needed to read the rest of the paper. We start by recalling some standard mathematical notations (Section 2.1) and then the basic principles of abstract interpretation (Section 2.2). Next, in Section 2.3 we present the toy imperative programming language that we will use. We first present the syntax (Section 2.3.1) and the semantics (Section 2.3.2) of the toy language. In particular, in Section 2.3.2 we define the semantics for expressions and actions and the auxiliary functions, namely all the basic parts needed for the definition of the transitional semantics. From this latter, we define the trace semantics of programs, since we need the full history of programs executions in order to model the watermarking techniques. Then, we define the (traces) abstract semantics (Section 2.3.3) of programs written in the proposed toy language since we need abstract interpreters for modelling attackers and signature embedding. In Section 2.4, we introduce the notion of non-interference among programs, namely an high-order notion of non-interference since standard non-interference is defined among the variables of a program. We need this high-order notion of non-interference when modelling certain quality features of the watermarking scheme (as secrecy).

2.1. *Mathematical notation*

In the following, the symbol $\triangleq$ stands for 'is defined as'. Given two sets S and T, we denote with $\wp(S)$ the powerset of S, with $S \setminus T$ the set-difference between S and T, with $S \subset T$ strict inclusion and with $S \subseteq T$ inclusion. Let $S_\bot$ be set S augmented with the undefined value $\bot$, i.e. $S_\bot \triangleq S \cup \{\bot\}$. $\langle P, \leqslant \rangle$ denotes a poset P with ordering relation $\leqslant$, while a complete lattice P, with ordering $\leqslant$, least upper bound (lub) $\vee$, greatest lower bound (glb) $\wedge$, greatest element (top) $\top$ and least element (bottom) $\bot$ is denoted by $\langle P, \leqslant, \vee, \wedge, \top, \bot \rangle$. Given functions $f \in S \to T$ and $g \in T \to R$, their composition $g \circ f \in S \to R$ is $g \circ f \triangleq \lambda x . g(f(x))$. We denote with $\sqsubseteq$ the pointwise ordering between functions (on pesets). Given $f \in P \xrightarrow{c} Q$ on posets, f is (Scott)-continuous when it preserves lub of countable chains in P (we use the superscript ^{c} to denote a continuous function). Given $f \in P \xrightarrow{m} Q$ on posets, f is monotone if for any $p, p' \in P$ we have that $p \leqslant_P p'$ implies $f(p) \leqslant_Q f(p')$ (we use the superscript ^{m} to denote a monotone function). Given $f \in C \to D$ on complete lattices, f is additive [co-additive] when, for any $Y \subseteq C, f(\vee_C Y) = \vee_D f(Y)$ $[f(\wedge_C Y) = \wedge_D f(Y)]$. The right [left] adjoint of a function f is $f^+ \triangleq \lambda x. \bigvee\{y | f(y) \leqslant x\} [f^- \triangleq \lambda x. \bigwedge\{y | x \leqslant f(y)\}]$.

2.2. *Abstract interpretation*

Abstract interpretation is based on the idea that the behaviour of a program at different levels of abstraction is an approximation of its (concrete) semantics (Cousot and Cousot 1977, 1979). The concrete program semantics is computed on the concrete domain $\langle C, \leqslant_C \rangle$, while approximation is given by an abstract domain $\langle A, \leqslant_A \rangle$. In abstract interpretation, the abstraction is specified as a Galois connection, GC in short, (C, α, γ, A), namely as an abstraction map $\alpha \in C \rightarrow A$ and a concretization map $\gamma \in A \rightarrow C$ that are monotone and that form an adjunction: $\forall y \in A, x \in C \,.\, \alpha(x) \leqslant_A y \Rightarrow x \leqslant_C \gamma(y)$ (Cousot and Cousot 1977, 1979). α [resp. γ] is the left[right]-adjoint of γ [α] and it is additive [co-additive]. Abstract domains can be equivalently formalized as upper closure operators on the concrete domain (Cousot and Cousot 1979). The two approaches are equivalent, modulo isomorphic representations of the domain objects. An upper closure operator, or closure, on poset $\langle C, \leqslant \rangle$ is an operator $\varphi \in C \rightarrow C$ that is monotone, idempotent and extensive (i.e. $\forall c \in C \,.\, c \leqslant \varphi(c)$). Closures are uniquely determined by the set of their fixpoints $\varphi(C)$. The set of all closure on C is denoted by $uco(C)$. The lattice of abstract domains of C is therefore isomorphic to $uco(C)$ (Cousot and Cousot 1977, 1979). If C is a complete lattice, then $\langle uco(C), \sqsubseteq, \sqcup, \sqcap, \lambda x \,.\, \top, id \rangle$ is a complete lattice, where $id \triangleq \lambda x \,.\, x$ and for every $\rho, \eta \in uco(C)$, $\rho \sqsubseteq \eta$ *iff* $\forall y \in C \,.\, \rho(y) \leqslant \eta(y)$ *iff* $\eta(C) \subseteq \rho(C)$. The glb $\sqcap$ is isomorphic to the so called reduced product, i.e. $\bigsqcap_{i \in I} \rho_i$ is the most abstract common concretization of all ρ_i. Given $X \subseteq C$, the least abstract domain containing X is the least closure including X as fixpoints, which is the Moore-closure $\mathcal{M}(X) \triangleq \{\bigwedge S | S \subseteq X\}$. Note that $\bigsqcap_{i \in I} \rho_i = \mathcal{M}(\bigcup_{i \in I} \rho_i)$. If (C, α, γ, A) is a GC then $\varphi = \gamma \circ \alpha$ is the closure associated with A, such that $\varphi(C)$ is a complete lattice isomorphic to A.

Precision of an abstract interpretation is typically defined in terms of completeness. Depending on where we compare the concrete and the abstract computations, we obtain two different notions of completeness (Giacobazzi and Quintarelli 2001; Giacobazzi *et al.* 2000). If we compare the results in the abstract domain, we obtain what is called backward completeness ($\mathcal{B}$-completeness) while, if we compare the results in the concrete domain, we obtain the so called forward completeness ($\mathcal{F}$-completeness). Formally, if $f \in C \rightarrow C$ and $\rho \in uco(C)$, then ρ is $\mathcal{B}$-complete for f if $\rho \circ f = \rho \circ f \circ \rho$, while it is $\mathcal{F}$-complete for f if $f \circ \rho = \rho \circ f \circ \rho$. In a more general setting, if $f \in C \rightarrow C$ and $\rho, \eta \in uco(C)$, then $\langle \rho, \eta \rangle$ is a pair of $\mathcal{B}[\mathcal{F}]$-complete abstractions for f if $\rho \circ f = \rho \circ f \circ \eta$ [$f \circ \eta = \rho \circ f \circ \eta$] (equivalently, we say that f is $\mathcal{B}[\mathcal{F}]$-complete for $\langle \rho, \eta \rangle$). A complete over-approximation means that no false alarms are returned by the analysis, i.e. in $\mathcal{B}$-completeness the approximate semantics computed by manipulating abstract objects corresponds precisely to the abstraction of the concrete semantics, while in $\mathcal{F}$-completeness concrete semantics does not lose precision computing on abstract objects.

The least fixpoint (lfp) of an operator F on a poset $\langle P, \leqslant \rangle$, when it exists, is denoted by $lfp^{\leqslant} F$, or by $lfp F$ when $\leqslant$ is clear. Any continuous operator $F \in C \rightarrow C$ on a complete lattice $\langle C, \leqslant, \vee, \wedge, \top, \bot \rangle$ admits a lfp: $lfp^{\leqslant}_{\bot} F = \bigvee_{n \in \mathbb{N}} F^i(\bot)$, where for any $i \in \mathbb{N}$ and $x \in C$: $F^0(x) = x$ and $F^{i+1}(x) = F(F^i(x))$. Given an abstract domain $\langle A, \leqslant_A \rangle$ of $\langle C, \leqslant_C \rangle$, $F^\sharp \in A \rightarrow A$ is a correct (sound) approximation of $F \in C \rightarrow C$ when $\alpha(lfp^{\leqslant_C} F) \leqslant_A lfp^{\leqslant_A} F^\sharp$.

Inducing forward completeness. The problem of minimally modifying the abstract domains in order to gain completeness *w.r.t.* a given function has been solved in Giacobazzi and Quintarelli (2001); Giacobazzi *et al.* (2000). While in Giacobazzi and Mastroeni (2008) the authors addressed the dual problem, i.e. they show how to minimally modify a function in order to gain completeness *w.r.t.* the given abstract domains. So, it is always possible to minimally transform a given semantic function f in order to satisfy completeness. Minimally means to find the closest function, by reducing or increasing the images of f, *w.r.t.* a given property we want to hold for f (in this context, completeness). Here, we take into account only the case of increasing a given function, so we move upwards. According to Giacobazzi and Mastroeni (2008) given a monotone function $f \in C \xrightarrow{m} C$ and a pair of closures $\eta, \rho \in uco(C)$ we can define

$$\mathrm{F}_{\eta,\rho} \triangleq \lambda f \,.\, \lambda x \,.\begin{cases} \rho \circ f(x) & \text{if } x \in \eta(C) \\ f(x) & \text{otherwise} \end{cases}$$

We have that $\mathrm{F}_{\eta,\rho}(f) = \bigsqcap\{h \in C \to C | f \sqsubseteq h \wedge h \circ \eta = \rho \circ h \circ \eta\}$, namely $\mathrm{F}_{\eta,\rho}(f)$ is the smaller function greater than f that is $\mathcal{F}$-complete for $\langle \rho, \eta \rangle$. Unfortunately, $\mathrm{F}_{\eta,\rho}(f)$ may lack monotonicity. But any function can be transformed to the closest monotone function by considering the following basic transformer: $\mathrm{M} \triangleq \lambda f \,.\, \lambda x \,.\, \bigvee_C \{f(y) | y \leqslant_C x\}$ (Giacobazzi and Mastroeni 2008). So, we can define the forward completeness transformer as $\mathbb{F}_{\eta,\rho} \triangleq \mathrm{M} \circ \mathrm{F}_{\eta,\rho}$, such that $\mathbb{F}_{\eta,\rho}(f) = \bigsqcap\{h \in C \xrightarrow{m} C | f \sqsubseteq h \wedge h \circ \eta = \rho \circ h \circ \eta\}$.

2.3. *Programming language and semantics*

In this section, we introduce a simple imperative programming language, which is used in the rest of the work. It is a simple extension of the one introduced in Cousot and Cousot (2002). The main difference is the ability of programs to interact with the user, namely programs can receive input values.

2.3.1. *Syntax.* The syntax of our toy programming language is given by the following grammar:

$$\begin{array}{lll} E & ::= & n \mid X \mid E + E \mid E \cdot E \mid E - E \\ B & ::= & b \mid E < E \mid E = E \mid B \wedge B \mid B \vee B \mid \neg B \\ A & ::= & X := E \mid \texttt{input}\, X \mid \texttt{skip} \\ C & ::= & L : A \to L'; \mid L : B \to \{L_{\mathsf{tt}}, L_{\mathsf{ff}}\}; \mid L : \texttt{stop}, \end{array}$$

where the syntactic categories are described in Figure 1.

A program is defined as a set of commands, each one labelled with an unique identifier (its label). A command $L : A \to L'$; performs an action A and it passes the execution to the command labelled with L'. A command $L : B \to \{L_{\mathsf{tt}}, L_{\mathsf{ff}}\}$; passes the execution to the command at label L_{tt} if the guard B is evaluated to tt, and to the command at label L_{ff} otherwise. The command $L : \texttt{stop}$; stops the execution. Note that this latter can be simulated by $L : \texttt{skip} \to \bot$; and the command $L : \texttt{skip} \to L'$; can be simulated by $L : \mathsf{tt} \to \{L', \bot\}$; where $\bot$ is the undefined label. In the following, with a little abuse of notation, we use $\bot$ to refer to an undefined value of any type. The action $X := E$;

$n \in \texttt{Int}$	▷ integers
$b \in \texttt{Bool}$	▷ booleans
$X \in \texttt{Var}$	▷ program variables
$L \in \texttt{Lab}$	▷ program labels
$E \in \texttt{Exp}$	▷ arithmetic expressions
$B \in \texttt{Bexp}$	▷ boolean expressions
$A \in \texttt{Act}$	▷ program actions
$C \in \texttt{Com}$	▷ commands
$P \in \textsc{Imp} \triangleq \wp(\texttt{Com})$	▷ programs

Fig. 1. Syntactic categories.

$n \in \mathbb{Z}$	▷ integer numbers
$b \in \mathbb{B} \triangleq \{\mathsf{tt}, \mathsf{ff}\}$	▷ truth values
$\rho \in \texttt{Env} \triangleq \texttt{Var} \longrightarrow \mathbb{Z}_\perp$	▷ environments
$\imath \in \texttt{Sin} \triangleq \mathbb{Z}^\star = \bigcup_{n \in \mathbb{N}} \mathbb{Z}^n$	▷ standard inputs
$\zeta \in \texttt{Con} \triangleq \texttt{Env} \times \texttt{Sin}$	▷ contexts
$\varsigma \in \Sigma \triangleq \texttt{Com} \times \texttt{Con}$	▷ program states

Fig. 2. Value domains.

assigns the value of expression E to the variable X. The action $\texttt{input}\,X$ takes an input from the user and assigns its value to the variable X. The action $\texttt{skip}$ does not perform any operation.

2.3.2. *Semantics.* An environment $\rho \in \texttt{Env}$ maps each variable $X \in \texttt{dom}(\rho)$ to its value $\rho(X) \in \mathbb{Z}_\perp$. A context $\zeta \in \texttt{Con}$ is a pair binding an environment $\rho \in \texttt{Env}$ to a standard input $\imath \in \texttt{Sin}$, i.e. $\zeta = \langle \rho, \imath \rangle$. The value domains used in the definition of the semantics are reported in Figure 2.

In order to present the semantics, we need the following auxiliary functions.

— A function that returns the set of labels of a given command

$${}^{lab}\llbracket L : A \to L'; \rrbracket \triangleq \{L\}$$
$${}^{lab}\llbracket L : B \to \{L_{\mathsf{tt}}, L_{\mathsf{ff}}\}; \rrbracket \triangleq \{L\}$$
$${}^{lab}\llbracket L : \texttt{stop}; \rrbracket \triangleq \{L\}$$

and ${}^{lab}\llbracket P \rrbracket \triangleq \{{}^{lab}\llbracket C \rrbracket | C \in P\}$ denotes the set of all labels of a program P.

— A function that returns the set of variables of a given expression, action or command.

$${}^{var}\llbracket D \rrbracket \triangleq \{X \in \texttt{Var} | X \text{ is in } D\} \qquad \text{where } D \in \{E, B\}$$
$${}^{var}\llbracket X := E \rrbracket \triangleq \{X\} \cup {}^{var}\llbracket E \rrbracket$$
$${}^{var}\llbracket \texttt{input}\,X \rrbracket \triangleq \{X\}$$
$${}^{var}\llbracket \texttt{skip} \rrbracket \triangleq \varnothing$$
$${}^{var}\llbracket L : A \to L'; \rrbracket \triangleq {}^{var}\llbracket A \rrbracket$$
$${}^{var}\llbracket L : B \to \{L_{\mathsf{tt}}, L_{\mathsf{ff}}\}; \rrbracket \triangleq {}^{var}\llbracket B \rrbracket$$
$${}^{var}\llbracket L : \texttt{stop}; \rrbracket \triangleq \varnothing$$

and ${}^{var}\llbracket P \rrbracket \triangleq \bigcup_{C \in P} {}^{var}\llbracket C \rrbracket$ denotes the set of all variables of P.

— A function that returns the set of actions of a given command.

$$
\begin{aligned}
&{}^{act}[\![L : A \to L';]\!] \triangleq \{A\}\\
&{}^{act}[\![L : B \to \{L_{\mathsf{tt}}, L_{\mathsf{ff}}\};]\!] \triangleq \varnothing\\
&{}^{act}[\![L : \mathtt{stop};]\!] \triangleq \varnothing
\end{aligned}
$$

— A function that returns the set of successor labels of a given command.

$$
\begin{aligned}
&{}^{suc}[\![L : A \to L';]\!] \triangleq \{L'\}\\
&{}^{suc}[\![L : B \to \{L_{\mathsf{tt}}, L_{\mathsf{ff}}\};]\!] \triangleq \{L_{\mathsf{tt}}, L_{\mathsf{ff}}\}\\
&{}^{suc}[\![L : \mathtt{stop};]\!] \triangleq \{\bot\}
\end{aligned}
$$

— A function that returns the set of possible environments of a given set of variables.

$$
{}^{env}[\![\mathcal{X}]\!] \triangleq \{\rho \in \mathtt{Env} | \mathtt{dom}(\rho) = \mathcal{X}\} \qquad \text{where } \mathcal{X} \subseteq \mathtt{Var}
$$

and ${}^{env}[\![P]\!] \triangleq {}^{env}[\![{}^{var}[\![P]\!]]\!]$ denotes the set of all possible environments of P.

— A function that returns the set of possible contexts of a given environment.

$$
{}^{con}[\![\rho]\!] \triangleq \{\langle\rho, \imath\rangle \in \mathtt{Con} | \imath \in \mathtt{Sin}\} \qquad \text{where } \rho \in \mathtt{Env}
$$

and ${}^{con}[\![P]\!] \triangleq \{{}^{con}[\![\rho]\!] | \rho \in {}^{env}[\![P]\!]\}$ denotes the set of all possible contexts of P.

— Two functions $\mathtt{top} \in \mathtt{Sin} \to \mathbb{Z}_\bot$ and $\mathtt{next} \in \mathtt{Sin} \to \mathtt{Sin}$ that deal with standard inputs. The function top, given a standard input, returns the next value that will be passed to the program:

$$
\mathtt{top}(\imath) \triangleq \begin{cases} \bot & \text{if } \imath = \epsilon \\ z \in \mathbb{Z} & \text{if } \imath = z\imath' \end{cases}
$$

The function next, given a standard input, returns another standard input without the current value passed to the program:

$$
\mathtt{next}(\imath) \triangleq \begin{cases} \epsilon & \text{if } \imath = \epsilon \\ \imath' \in \mathtt{Sin} & \text{if } \imath = z\imath' \wedge z \in \mathbb{Z} \end{cases}
$$

Semantics of expressions and actions. In order to define the semantics of programs, we need to define the semantics of boolean/arithmetic expressions and the semantics of actions. Arithmetic expressions: ${}^{\mathcal{E}}[\![E]\!] \in \mathtt{Con} \to \mathbb{Z}_\bot$

$$
\begin{aligned}
&{}^{\mathcal{E}}[\![n]\!]\zeta \triangleq n\\
&{}^{\mathcal{E}}[\![X]\!]\langle\rho, \imath\rangle \triangleq \rho(X)\\
&{}^{\mathcal{E}}[\![E_1 \,\mathsf{op}\, E_2]\!]\zeta \triangleq \begin{cases} {}^{\mathcal{E}}[\![E_1]\!]\zeta \ \mathsf{op}\ {}^{\mathcal{E}}[\![E_2]\!]\zeta & \text{if } {}^{\mathcal{E}}[\![E_1]\!]\zeta \in \mathbb{Z} \text{ and } {}^{\mathcal{E}}[\![E_2]\!]\zeta \in \mathbb{Z} \\ \bot & \text{otherwise} \end{cases}
\end{aligned}
$$

Boolean expressions: ${}^{\mathcal{B}}[\![B]\!] \in \mathtt{Con} \to \mathbb{B}_\bot$

$$
\begin{aligned}
&{}^{\mathcal{B}}[\![\mathsf{tt}]\!]\zeta \triangleq \mathsf{tt}\\
&{}^{\mathcal{B}}[\![\mathsf{ff}]\!]\zeta \triangleq \mathsf{ff}\\
&{}^{\mathcal{B}}[\![E_1 < E_2]\!]\zeta \triangleq \begin{cases} {}^{\mathcal{E}}[\![E_1]\!]\zeta < {}^{\mathcal{E}}[\![E_2]\!]\zeta & \text{if } {}^{\mathcal{E}}[\![E_1]\!]\zeta \in \mathbb{Z} \text{ and } {}^{\mathcal{E}}[\![E_2]\!]\zeta \in \mathbb{Z} \\ \bot & \text{otherwise} \end{cases}\\
&{}^{\mathcal{B}}[\![E_1 = E_2]\!]\zeta \triangleq \begin{cases} {}^{\mathcal{E}}[\![E_1]\!]\zeta = {}^{\mathcal{E}}[\![E_2]\!]\zeta & \text{if } {}^{\mathcal{E}}[\![E_1]\!]\zeta \in \mathbb{Z} \text{ and } {}^{\mathcal{E}}[\![E_2]\!]\zeta \in \mathbb{Z} \\ \bot & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
{}^{\mathcal{B}}[\![\neg B]\!]\zeta &\triangleq \begin{cases} \neg^{\mathcal{B}}[\![B]\!]\zeta & \text{if } {}^{\mathcal{B}}[\![B]\!]\zeta \in \mathbb{B} \\ \bot & \text{otherwise} \end{cases} \\
{}^{\mathcal{B}}[\![B_1 \wedge B_2]\!]\zeta &\triangleq \begin{cases} {}^{\mathcal{B}}[\![B_1]\!]\zeta \wedge {}^{\mathcal{B}}[\![B_2]\!]\zeta & \text{if } {}^{\mathcal{B}}[\![B_1]\!]\zeta \in \mathbb{B} \text{ and } {}^{\mathcal{B}}[\![B_2]\!]\zeta \in \mathbb{B} \\ \bot & \text{otherwise} \end{cases} \\
{}^{\mathcal{B}}[\![B_1 \vee B_2]\!]\zeta &\triangleq \begin{cases} {}^{\mathcal{B}}[\![B_1]\!]\zeta \vee {}^{\mathcal{B}}[\![B_2]\!]\zeta & \text{if } {}^{\mathcal{B}}[\![B_1]\!]\zeta \in \mathbb{B} \text{ and } {}^{\mathcal{B}}[\![B_2]\!]\zeta \in \mathbb{B} \\ \bot & \text{otherwise} \end{cases}
\end{aligned}
$$

Actions: ${}^{\mathcal{A}}[\![A]\!] \in \texttt{Con} \to \texttt{Con}$

$$
\begin{aligned}
&{}^{\mathcal{A}}[\![X := E]\!]\langle \rho, \imath \rangle \triangleq \langle \rho[X \hookleftarrow {}^{\mathcal{E}}[\![E]\!]\langle \rho, \imath \rangle], \imath \rangle \\
&{}^{\mathcal{A}}[\![\texttt{input}\ X]\!]\langle \rho, \imath \rangle \triangleq \begin{cases} \langle \rho[X \hookleftarrow \texttt{top}(\imath)], \texttt{next}(\imath) \rangle & \text{if } \texttt{top}(\imath) \neq \bot \\ \langle \rho[X \hookleftarrow \bot], \imath \rangle & \text{otherwise} \end{cases} \\
&{}^{\mathcal{A}}[\![\texttt{skip}]\!]\zeta \triangleq \zeta
\end{aligned}
$$

Semantics of programs. A program state $\varsigma \in \Sigma$ is a pair $\varsigma = \langle C, \zeta \rangle$, where C is the command to be executed and ζ is the current context of execution. The transition relation $\mathbf{S} \in \Sigma \to \wp(\Sigma)$ specifies the successor states of a given state:

$$\mathbf{S}(\langle C, \zeta \rangle) \triangleq \left\{ \langle C', \zeta' \rangle | \zeta' = {}^{\mathcal{A}}[\![{}^{act}[\![C]\!]]\!]\zeta \wedge {}^{lab}[\![C']\!] \in {}^{suc}[\![C]\!] \right\}.$$

The set of states of a program is defined as: ${}^{sts}[\![P]\!] \triangleq P \times {}^{con}[\![P]\!]$. The transitional semantics ${}^{\mathbf{S}}[\![P]\!] \in^{sts} [\![P]\!] \to \wp({}^{sts}[\![P]\!])$ of a program P is

$${}^{\mathbf{S}}[\![P]\!]\langle C, \zeta \rangle \triangleq \{ \langle C', \zeta' \rangle \in \mathbf{S}(\langle C, \zeta \rangle) | \zeta, \zeta' \in {}^{con}[\![P]\!] \wedge C, C' \in P \}.$$

A trace $\sigma \in \Sigma$ is a sequence of states $\sigma_0, \ldots \sigma_{n-1}$ of length $|\sigma| = n > 0$ such that for all $i \in [1, n)$ we have $\sigma_i \in \mathbf{S}(\sigma_{i-1})$. With Σ^+ we indicate the set of all finite traces. If σ is a finite trace, we indicate with $\sigma_\vdash$ its first element, i.e. $\sigma_\vdash = \sigma_0$, and we indicate with $\sigma_\dashv$ its last element, i.e. $\sigma_\dashv = \sigma_{|\sigma|-1}$.

The *partial finite traces semantics* $(\!|P|\!)_\oplus \subseteq \Sigma^+$ of a program P is the set of all finite partial traces of P. This semantics can be computed as the least fixpoint of the so called transition function $F^\oplus_P \in \wp(\Sigma^+) \xrightarrow{m} \wp(\Sigma^+)$, defined as follows:

$$F^\oplus_P \triangleq \lambda S \,.\, {}^{sts}[\![P]\!] \cup \left\{ \sigma\varsigma\varsigma' | \varsigma' \in {}^{\mathbf{S}}[\![P]\!](\varsigma) \wedge \sigma\varsigma \in S \right\}.$$

So $(\!|P|\!)_\oplus = \mathit{lfp}^{\subseteq}_{\varnothing} F^\oplus_P$. If we are only interested in those executions of a program P starting from a given set $L \subseteq {}^{lab}[\![P]\!]$ of entry points, so that $I \triangleq \{\langle C, \zeta \rangle | \zeta \in {}^{con}[\![P]\!] \wedge C \in P \wedge {}^{lab}[\![C]\!] \in L\}$ is the set of initial states, we can consider the partial traces semantics $(\!|P\,[I]|\!)_\oplus \subseteq \Sigma^+$ of P which is the set of partial traces $\sigma \in \Sigma^+$ starting from an initial state $\sigma_0 \in I$. The partial traces semantics $(\!|P\,[I]|\!)_\oplus$ can be expressed in fixpoint form as $\mathit{lfp}^{\subseteq}_{\varnothing} F^\oplus_{P[I]}$ where $F^\oplus_{P[I]} \in \wp(\Sigma^+) \xrightarrow{m} \wp(\Sigma^+)$ is

$$F^\oplus_{P[I]} \triangleq \lambda S \,.\, I \cup \left\{ \sigma\varsigma\varsigma' | \sigma\varsigma \in S \wedge \varsigma' \in {}^{\mathbf{S}}[\![P]\!](\varsigma) \right\}.$$

So, we can define a function, called *partial input semantics*, $\{\!\!|P|\!\!\}_\oplus \in \wp(\Sigma) \to \wp(\Sigma^+)$ defined as follow:

$$\{\!\!|P|\!\!\}_\oplus \triangleq \lambda S \,.\, \mathit{lfp}^{\subseteq}_{\varnothing} F^\oplus_{P[S]} = \lambda S \,.\, (\!|P\,[S]|\!)_\oplus$$

A state ς is blocking (or final), *w.r.t.* a program P, if ${}^{\mathrm{S}}[\![P]\!](\varsigma) = \varnothing$. So the set of blocking states of the program P is $T^P \triangleq \{\langle C, \zeta \rangle \in {}^{sts}[\![P]\!] | {}^{suc}[\![C]\!] \nsubseteq {}^{lab}[\![P]\!]\}$. A maximal finite trace of a program P, is a trace $\sigma \in \Sigma^+$ of length n where the last state σ_{n-1} is blocking. $\llparenthesis P \rrparenthesis^n$ is the set of all finite traces of length n of the program P. The *maximal finite traces semantics* $\llparenthesis P \rrparenthesis_+$ of the program P is given by the union of all maximal finite traces of length $n > 0$, namely $\llparenthesis P \rrparenthesis_+ \triangleq \bigcup_{n>0} \{\sigma \in \llparenthesis P \rrparenthesis^n | \sigma_\dashv \in T^P\}$. This semantics can be expressed as the least fixpoint of the transition function $F_P^+ \in \wp(\Sigma^+) \xrightarrow{m} \wp(\Sigma^+)$ defined as follow:

$$F_P^+ \triangleq \lambda S \,.\, T^P \cup \left\{ \varsigma\varsigma'\sigma | \varsigma' \in {}^{\mathrm{S}}[\![P]\!](\varsigma) \wedge \varsigma'\sigma \in S \right\}.$$

Similarly, we can define a function, called *maximal input semantics*, $\Lbag P \Rbag_+ \in \wp(\Sigma) \to \wp(\Sigma^+)$ defined as follow:

$$\Lbag P \Rbag_+ \triangleq \lambda S \,.\, \{\sigma \in \llparenthesis P \rrparenthesis_+ | \sigma_\vdash \in S\}$$

Unfortunately, it seems that this semantic function cannot be expressed in fixpoint form in the lattice $\langle \wp(\Sigma^+), \subseteq, \cup, \cap, \Sigma^+, \varnothing \rangle$. A possible way for avoiding the problem is to calculate the function as the combination of partial input and maximal traces semantics, so as,

$$\Lbag P \Rbag_+ = \lambda S \,.\, \Lbag P \Rbag_\oplus(S) \cap \llparenthesis P \rrparenthesis_+.$$

But a better solution would be the definition of a specific semantic domain in which we are able to compute this function directly.

Prefix ordering. Let $\mathtt{pref} \in \Sigma^+ \to \wp(\Sigma^+)$ be a function that returns the set of prefixes of a given trace, so $\mathtt{pref}(\sigma) \triangleq \{\sigma' \in \Sigma^+ | \exists \sigma'' \in \Sigma^+ \cup \{\epsilon\} \,.\, \sigma = \sigma'\sigma''\}$. We can define the relation $\preccurlyeq \subseteq \wp(\Sigma^+) \times \wp(\Sigma^+)$ which is a partial order between sets of traces:

$$X \preccurlyeq Y \Leftrightarrow \begin{array}{l} \forall \sigma \in X \, \exists \sigma' \in Y \,.\, \sigma \in \mathtt{pref}(\sigma') \wedge \\ (\forall \sigma' \in Y \, \exists \sigma \in X \,.\, \sigma \in \mathtt{pref}(\sigma') \Rightarrow Y \subseteq X) \end{array}$$

Now, we have to define the elements that extend the poset $\langle \wp(\Sigma^+), \preccurlyeq \rangle$. The least upper bound $\biguplus$ is defined as follows:

$$\biguplus \mathcal{X} \triangleq \left\{ \sigma \in \bigcup_{X \in \mathcal{X}} X \,\middle|\, \forall \sigma' \in \bigcup_{X \in \mathcal{X}} X \,.\, \sigma \in \mathtt{pref}(\sigma') \Rightarrow \sigma = \sigma' \right\}$$

The bottom element is $\varnothing \in \wp(\Sigma^+)$, i.e. $\forall X \in \wp(\Sigma^+) \,.\, \varnothing \preccurlyeq X$ holds.

Proposition 2.1. $\langle \wp(\Sigma^+), \preccurlyeq, \uplus, \varnothing \rangle$ is a directed-complete partial order (DCPO).

Proof. See Appendix A.1, Page 379. □

Finally, let us define the maximal input semantic in this new domain. As for partial traces semantics, we can consider the maximal traces semantics $\llparenthesis P[I] \rrparenthesis_+ \subseteq \Sigma^+$ of P, which is the set of maximal traces $\sigma \in \Sigma^+$ starting from an initial state $\sigma_\vdash \in I$. Recall that the maximal finite traces semantics $\llparenthesis P \rrparenthesis_+$ is equal to $\bigcup_{n>0} \{\sigma \in \llparenthesis P \rrparenthesis^n | \sigma_\dashv \in T^P\}$. This latter, computed starting from states in I, is defined as $\llparenthesis P[I] \rrparenthesis_+ \triangleq \bigcup_{n>0} \{\sigma \in \llparenthesis P \rrparenthesis^n | \sigma_\vdash \in I \wedge \sigma_\dashv \in T^P\}$. In the following, we indicate with $\llparenthesis P \rrparenthesis_I^n$ the set $\{\sigma \in \llparenthesis P \rrparenthesis^n | \sigma_\vdash \in I\}$ and with $\llparenthesis P \rrparenthesis_I^{\bar{n}}$ the set $\{\sigma \in \llparenthesis P \rrparenthesis^n | \sigma_\vdash \in I \wedge \sigma_\dashv \in T^P\}$, so $\llparenthesis P[I] \rrparenthesis_+$ can be rewritten as $\bigcup_{n>0} \llparenthesis P \rrparenthesis_I^{\bar{n}}$. Now we can note that the following holds:

$$\bigcup_{n>0} (\!|P|\!)_I^{\bar{n}} = \biguplus_{n>0} (\!|P|\!)_I^{\bar{n}},$$

because for every n the traces in $(\!|P|\!)_I^{\bar{n}}$ are not prefixes of any trace in $(\!|P|\!)_I^{\overline{n+1}}$, due to the fact that they are maximal (and so the lub returns the union of this two sets).

So, the semantics can be expressed as $lfp_{\varnothing}^{\leqslant} F_{P[I]}^{+}$ of the Scott-continuous, and so monotone, function $F_{P[I]}^{+} \in \wp(\Sigma^{+}) \xrightarrow{c} \wp(\Sigma^{+})$:

$$F_{P[I]}^{+} \triangleq \lambda S\,.\,(\!|P|\!)_I^{1} \uplus \left\{\sigma\varsigma\varsigma' | \varsigma' \in {}^{\mathbf{S}}[\![P]\!](\varsigma) \wedge \sigma\varsigma \in S\right\}.$$

Proposition 2.2. $(\!|P[I]|\!)_{+} = lfp_{\varnothing}^{\leqslant} F_{P[I]}^{+}$.

Proof. See Appendix A.2, Page 380. □

2.3.3. *Abstract semantics.* If we are interested in the abstract semantics of a program, computed on a specific abstract domain, we need to compute the semantics using the *best correct approximation* of the transfer function on the abstract domain. So if a semantics is computed in a concrete domain C, as fixpoint of the transfer function F, then we can compute its abstract interpretation in an abstract domain $\rho \in uco(C)$ as the fixpoint of the function $\rho F \rho$. If the abstract domain is $\mathcal{B}$-complete for F then, we have that $lfp\ \rho F \rho = \rho(lfpF)$ (Giacobazzi *et al.* 2000).

For example, let $\langle \wp(\Sigma^{+}), \subseteq, \cup, \cap, \Sigma^{+}, \varnothing \rangle$ be the concrete domain and $(\!|P|\!)_{\oplus}$ the semantics computed as fixpoint of the transfer function $F_P^{\oplus}$. Then, the best correct approximation of P in $\rho \in uco(\wp(\Sigma^{+}))$ is as follows:

$$(\!|P|\!)_{\oplus}^{\rho} \triangleq lfp_{\varnothing}^{\subseteq}\ \rho \circ F_P^{\oplus} \circ \rho.$$

Let $\langle \wp(\Sigma^{+}), \leqslant, \uplus, \varnothing \rangle$ be the concrete domain and let $(\!|P[S]|\!)_{+}$ be the semantics computed as fixpoint of the transfer function $F_{P[S]}^{+}$. The best correct approximation of P in $\rho \in uco(\wp(\Sigma^{+}))$ is:

$$(\!|P[S]|\!)_{+}^{\rho} \triangleq lfp_{\varnothing}^{\leqslant}\ \rho \circ F_{P[S]}^{+} \circ \rho$$

So, we can define the abstract maximal input semantics of P in ρ as follows:

$$(\!|P|\!)_{+}^{\rho} \triangleq \lambda S\,.\,lfp_{\varnothing}^{\leqslant}\ \rho \circ F_{P[S]}^{+} \circ \rho$$

2.4. *Abstract non-interference*

In order to define some features of watermarking systems, as secrecy, we need to constrain the information flows occurring in the marked program, so we need a form of semantic interference. Abstract non-interference, ANI in short, Giacobazzi and Mastroeni (2004) is a weakening of non-interference by abstract interpretation. Let $\eta, \rho \in uco(\wp(\mathbb{Z}_{\perp}^{L}))$ and $\phi \in uco(\wp(\mathbb{Z}_{\perp}^{H}))$, where $\mathbb{Z}_{\perp}^{L}$ and $\mathbb{Z}_{\perp}^{H}$ are the domains of public (L) and private (H) variables. Here, η and ρ characterize the attacker, instead ϕ states what, of the private data, can flow to the output observation, the so called declassification of ϕ (Mastroeni

2005). A program P satisfies ANI, and we write $[\eta]P(\phi \Rightarrow \rho)$, if $\forall h_1, h_2 \in \mathbb{Z}_\perp^H$ and $\forall l_1, l_2 \in \mathbb{Z}_\perp^L$:

$$\eta(l_1) = \eta(l_2) \wedge \phi(h_1) = \phi(h_2) \Rightarrow \rho((\!|P|\!)_D(\langle h_1, l_1\rangle)^L) = \rho((\!|P|\!)_D(\langle h_2, l_2\rangle)^L).$$

Where with $(\!|P|\!)_D \in \wp(\Sigma) \to \wp(\Sigma)$ we denote the (angelic) denotational semantics of the program P (Cousot and Cousot 2002; Giacobazzi and Mastroeni 2002). This notion says that, whenever the attacker is able to observe the input property η and the property ρ of the output, then it can observe nothing more than the property ϕ of the private input. In order to model non-interference in code transformations, such as software watermarking, we consider an higher-order version of ANI, where the objects of observations are programs instead of values. Hence, we have a part of a program (semantics) that can change, and that is secret, and the environment that remains the same up to an observable property: these are the new private and public inputs. The function that, in some way, has to hide the change is now a program transformer, which takes the two parts of the program and provides a program as result. The output observation is the best correct approximation of the resulting program.

Here, the semantics we take into account is the maximal finite trace semantics. Let P be the set of cover programs, Q the set of secret programs and $\mathfrak{I} \in \textsc{Imp} \times \textsc{Imp} \to \textsc{Imp}$ an integration function. As usual, the attacker is modelled as a couple $\langle \eta, \rho\rangle$, with $\eta, \rho \in uco(\wp(\Sigma^+))$, that represents the input and output public observation power. Instead $\phi \in uco(\wp(\Sigma^+))$ is the property of the secret input.

Definition 2.1 (HOANI for maximal finite traces semantics). The integration program $\mathfrak{I}$, given $\eta, \phi, \rho \in uco(\wp(\Sigma^+))$, satisfies *higher-order abstract non-interference* (for maximal finite traces semantics) *w.r.t.* $\langle \eta, \phi, \rho\rangle$ and $\langle \mathsf{P}, \mathsf{Q}\rangle$ if

$$\forall P_1, P_2 \in \mathsf{P}\, \forall Q_1, Q_2 \in \mathsf{Q}\,.$$
$$(\!|P_1|\!)_+^\eta = (\!|P_2|\!)_+^\eta \wedge (\!|Q_1|\!)_+^\phi = (\!|Q_1|\!)_+^\phi \Rightarrow (\!|\mathfrak{I}(P_1, Q_1)|\!)_+^\rho = (\!|\mathfrak{I}(P_2, Q_2)|\!)_+^\rho.$$

We write $\underset{+}{\overset{\mathrm{H}}{}}[\eta]\mathfrak{I}(\phi \Rightarrow \rho)_{\mathfrak{bca}}$ to indicate that the program $\mathfrak{I}$ satisfies higher-order abstract non-interference (for maximal finite traces semantics) *w.r.t.* $\langle \eta, \phi, \rho\rangle$.

Deriving attackers. In this section, we introduce a method for defining attackers, via abstract interpretation, for which a program is safe. In this case, security refers to abstract non-interference. In particular, it is interesting to characterize the most concrete (i.e. the most precise) attacker for which a program is safe. In fact it can be shown that if $[\eta]P(\phi \Rightarrow \rho)$ then, for any β such that $\rho \sqsubseteq \beta$, it holds $[\eta]P(\phi \Rightarrow \beta)$ (Giacobazzi and Mastroeni 2004). That is, if abstract non-interference holds observing in output the property ρ, then it holds also observing in output any property more abstract than ρ. In Giacobazzi and Mastroeni (2004); Mastroeni (2005) it is shown how to derive the most concrete attacker for which ANI holds. Following these results, we can analogously derive the most concrete attacker for which HOANI holds. In this case, sets of values are replaced by sets of traces. Attackers are defined as pairs of abstract domains, therefore we characterize a domain transformer, parametric on the program to be analyzed, that transforms each non-secret output abstraction into the nearest abstraction for which

HOANI holds. We fix a public input property η and a private input property ϕ, with $\eta, \phi \in uco(\wp(\Sigma^+))$. We then consider an abstraction $\rho \in uco(\wp(\Sigma^+))$ such that HOANI does not hold *w.r.t.* a program $\mathfrak{I} \in$ Imp, i.e. $\underset{+}{\mathrm{H}}[\eta]\mathfrak{I}(\phi \Rightarrow \rho)_{\mathtt{bca}}$ does not hold. We can derive the most concrete $\hat{\rho}$ that is more abstract than ρ and such that $\underset{+}{\mathrm{H}}[\eta]\mathfrak{I}(\phi \Rightarrow \rho)_{\mathtt{bca}}$ holds, which is called *higher-order abstract secret kernel* for $\mathfrak{I}$.

Definition 2.2 (Secret kernel for HOANI). Let $\mathfrak{I} \in$ Imp and $\mathcal{K}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)} \in uco(\wp(\Sigma^+)) \to uco(\wp(\Sigma^+))$. Then,

$$\mathcal{K}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)} \triangleq \lambda\rho\,.\bigsqcap\left\{\beta \middle| \rho \sqsubseteq \beta \wedge \underset{+}{\mathrm{H}}[\eta]\mathfrak{I}(\phi \Rightarrow \beta)_{\mathtt{bca}}\right\}$$

is the higher-order abstract secret kernel trasformer for $\mathfrak{I}$.

To characterize this transformer we must characterize when a program property is safe. Program properties are collections of traces so, we have to characterize the sets of traces that can belong to the kernel in order to satisfy HOANI. To this end, we define a predicate on sets of traces identifying the elements that form the secret kernel. Clearly, these elements must ensure non-interference, namely they must abstract in the same element all objects that should be indistinguishable. To achieve this, we define two equivalence relations that group programs according to a property on public programs (η) and a property on private programs (ϕ). Let $\equiv_\eta, \equiv_\phi \subseteq$ Imp $\times$ Imp be such that:

$$\equiv_\eta \triangleq \{\langle P, P'\rangle | P, P' \in \mathsf{P} \wedge (\!|P|\!)^\eta_+ = (\!|P'|\!)^\eta_+\}$$
$$\equiv_\phi \triangleq \{\langle Q, Q'\rangle | Q, Q' \in \mathsf{Q} \wedge (\!|Q|\!)^\phi_+ = (\!|Q'|\!)^\phi_+\}.$$

So, we can define the set of indistinguishable elements for HOANI

$$\Upsilon^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(P,Q) = \{(\!|\mathfrak{I}(P',Q')|\!)_+ | P' \equiv_\eta P \wedge Q' \equiv_\phi Q\}.$$

These sets are collections of sets of traces that should be indistinguishable for each secure abstraction, i.e. they should be approximated in the same object. Similarly, we can define the predicate Secr for HOANI

$$\forall X \in \wp(\Sigma^+)\,.\,\mathtt{Secr}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(X) \Leftrightarrow \forall P \in \mathsf{P} \forall Q \in \mathsf{Q}\,.$$
$$(\exists Z \in \Upsilon^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(P,Q)\,.\,Z \subseteq X \Rightarrow \forall W \in \Upsilon^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(P,Q)\,.\,W \subseteq X).$$

So $\mathtt{Secr}(X)$ holds if X contains all the elements, i.e. all the sets of traces, that have to be indistinguishable or non of them. Indeed, Secr identifies all and only the sets which are in the secret kernel.

Theorem 2.1. Let $\eta, \phi \in \wp(\Sigma^+)$:

$$\mathcal{K}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(id) = \left\{X \in \wp(\Sigma^+) \middle| \mathtt{Secr}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(X)\right\}.$$

Proof. See Appendix A.3, Page 381. □

This means that the set of elements in $\wp(\Sigma^+)$ for which Secr holds corresponds to the secret kernel of *id*, namely it coincides with the most concrete domain for which HOANI holds. This abstraction is called *most powerful harmless attacker for HOANI* and it is precisely the most precise attacker for which the program is safe. Furthermore, we can characterize the secret kernel of a generic domain ρ.

Corollary 2.1. Let $\eta, \phi \in \wp(\Sigma^+)$:

$$\mathcal{K}^{\mathrm{H+}}_{\mathfrak{I},\eta,(\phi)}(\rho) = \left\{ X \in \rho \middle| \mathtt{Secr}^{\mathrm{H+}}_{\mathfrak{I},\eta,(\phi)}(X) \right\}$$

Proof. By definition $\mathcal{K}^{\mathrm{H+}}_{\mathfrak{I},\eta,(\phi)} = \lambda\rho\,.\,\mathcal{K}^{\mathrm{H+}}_{\mathfrak{I},\eta,(\phi)}(id) \sqcup \rho$, so we have to prove that this domain is exactly $\{X \in \rho | \mathtt{Secr}^{\mathrm{H+}}_{\mathfrak{I},\eta,(\phi)}(X)\}$. This means that this latter has to be equal to $\{X \in \wp(\Sigma^+) | \mathtt{Secr}^{\mathrm{H+}}_{\mathfrak{I},\eta,(\phi)}(X)\} \sqcup \rho$. Remember that two sets are the same if one is included in the other and vice versa. Consider an element $Y \in \{X \in \rho | \mathtt{Secr}^{\mathrm{H+}}_{\mathfrak{I},\eta,(\phi)}(X)\}$. Clearly for this element $Y \in \rho$ and $\mathtt{Secr}^{\mathrm{H+}}_{\mathfrak{I},\eta,(\phi)}(Y)$ hold, i.e. Y belongs to $\mathcal{K}^{\mathrm{H+}}_{\mathfrak{I},\eta,(\phi)}(id) \sqcup \rho$. Similarly the inverse inclusion holds. □

3. Semantics-based software watermarking

3.1. *The framework*

We follow the nomenclature introduced in Cousot and Cousot (2004) for describing the basic components of a watermarking technique for programs written in Imp and signatures $s \in \mathcal{S}$.

Stegomarker $\mathfrak{M} \in \mathcal{S} \to$ Imp, a function that generates a program which is the encoding of a given signature $s \in \mathcal{S}$, i.e. it generates the *stegomark* $\mathfrak{M}(s) \in$ Imp

Stegoembedder $\mathfrak{L} \in$ Imp × Imp → Imp, a function that generates a program which is the composition of a stegomark and a program, called *stegoprogram* $\mathfrak{L}(P, \mathfrak{M}(s)) \in$ Imp.

Stegoextractor $\mathfrak{F} \in$ Imp $\to \mathcal{S}$, a function that extracts the signature from a stegoprogram; for all $s \in \mathcal{S}$ it must be $s = \mathfrak{F}(\mathfrak{L}(P, \mathfrak{M}(s)))$.

When $\mathfrak{L}$ and $\mathfrak{M}$ are clear from the context we denote the stegoprogram $\mathfrak{L}(P, \mathfrak{M}(s))$ as P_s. The stegoextractor takes a stegoprogram, analyzes it either statically or dynamically, and then it returns the signature encoded in the stegomark. It is well known (Cousot and Cousot 1979) that static analysis can be modelled in the context of abstract interpretation, where a property is extensionally represented as a closure operator representing the abstract domain of data satisfying it. In particular, static analysis is performed as an abstract execution of the program, namely as the (fixpoint) semantic computation on the abstract domain. Instead, dynamic analysis can be modelled as an approximated observation of a potentially abstract execution since it describes partial knowledge of the execution (only on certain inputs). This means that, in all cases, the encoded signature can be seen as a property of the stegomark's semantics and therefore of the stegoprogram's semantics. In this view, a stegoextractor is an abstract interpreter that executes the stegoprogram in the abstract domain $\beta \in uco(\wp(\Sigma^+))$ that allows to observe the hidden signature. In order to deal with dynamic watermarking, we need to model the enabling input that allows to extract the signature. Since in our model the residual input stream is part of the program state, the enabling input can be modelled as a state property

$\eta \in uco(\wp(\Sigma))$. We consider a set $\mathcal{P} \subseteq$ Imp of cover programs that do not already encode a signature[†] and we specify a watermarking system as a tuple $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$.

Definition 3.1 (Software Watermarking System). Given $\mathfrak{L} \in$ Imp × Imp → Imp, $\mathfrak{M} \in \mathcal{S} \rightarrow$ Imp and $\beta \in uco(\wp(\Sigma^+))$, the tuple $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$ is a *software watermarking system* for programs in $\mathcal{P}$ and signatures in $\mathcal{S}$ if $\mathfrak{M}$ is injective and there exists $\eta \in uco(\wp(\Sigma))$ such that $\forall P \in \mathcal{P}\, \forall s \in \mathcal{S}$:

$$\{\!|\mathfrak{L}(P,\mathfrak{M}(s))|\!\}^{\beta}_{+} = \lambda X\,.\begin{cases}\{\!|\mathfrak{M}(s)|\!\}^{\beta}_{+}(X) & \text{if } X \in \eta(\wp(\Sigma^+))\\ \{\!|P|\!\}^{\beta}_{+}(X) & \text{otherwise}\end{cases}$$

$$X \in \eta(\wp(\Sigma^+)) \Rightarrow \{\!|\mathfrak{M}(s)|\!\}^{\beta}_{+}(X) = (\!|\mathfrak{M}(s)|\!)^{\beta}_{+}.$$

This means that when computing the semantics in the abstract domain β, the stegoprogram $\mathfrak{L}(P,\mathfrak{M}(s))$ behaves like the stegomark $\mathfrak{M}(s)$ on the enabling inputs, and like the cover program P otherwise. Here, $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+}$ is precisely the information representing the watermark at the semantic level, namely the property of the stegomark that hides (encodes) the signature. In this setting, it is clearly possible to reduce the precise extraction of the signature to a completeness problem. To this end, we associate the stegomarker $\mathfrak{M}$ with its semantic counterpart $\overline{\mathfrak{M}} : \mathcal{S} \rightarrow uco(\wp(\Sigma^+))$, which encodes a signature in a semantic program property. In particular, given the watermarking system $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$ we define $\overline{\mathfrak{M}} \triangleq \lambda s.\{\varnothing, (\!|\mathfrak{M}(s)|\!)^{\beta}_{+}, \Sigma^+\}$, namely $\overline{\mathfrak{M}}$ takes a signature s and it returns the atomic closure of $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+}$. Indeed, $\overline{\mathfrak{M}}(s)$ provides a semantic representation of the signature s. Observe that, by construction, we have that $\forall s \in \mathcal{S}\,.\,\beta \sqsubseteq \overline{\mathfrak{M}}(s)$ and this ensures that β is precise enough for extracting the signature. Moreover, the abstract semantics of the stegoprogram computed on β reveals the watermark information $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+} \in \overline{\mathfrak{M}}(s)$ under the enabling input $X \in \eta(\wp(\Sigma))$ only if it is $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)$. This means that the semantics of the programs built by the stegoembedder can be fully understandable by the stegoextractor, namely this latter is able to extract the property representing the signature. Recalling the operator $\mathbb{F}$ inducing $\mathcal{F}$-completeness introduced in Section 2, we can say that if P_s is a stegoprogram then: $\{\!|P_s|\!\}^{\beta}_{+} = \mathbb{F}_{\eta,\overline{\mathfrak{M}}(s)}(\{\!|P_s|\!\}^{\beta}_{+})$, i.e. $\{\!|P_s|\!\}^{\beta}_{+}$ is $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)$.

Indeed, if $\{\!|P_s|\!\}^{\beta}_{+}$ is $\mathcal{F}$-complete then $\{\!|P_s|\!\}^{\beta}_{+} \circ \eta = \overline{\mathfrak{M}}(s) \circ \{\!|P_s|\!\}^{\beta}_{+} \circ \eta$ holds. When $\eta(\wp(\Sigma))$ contains X, we have that $\{\!|P_s|\!\}^{\beta}_{+}(X) = \overline{\mathfrak{M}}(s) \circ \{\!|P_s|\!\}^{\beta}_{+}(X)$ and consequently that $\{\!|P_s|\!\}^{\beta}_{+}(X) \in \overline{\mathfrak{M}}(s)$. This means that $\{\!|P_s|\!\}^{\beta}_{+}(X)$ is an element of $\overline{\mathfrak{M}}(s)$, more precisely it is exactly $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+}$ and it represents the signature s. If $\eta(\wp(\Sigma))$ does not contain X, the system should guarantee that the abstraction of the stegoprogram does not reveal the signature, so we have to chose β in such a way that $\{\!|P|\!\}^{\beta}_{+}(X) \notin \overline{\mathfrak{M}}(s)$, i.e. $\overline{\mathfrak{M}}(s)(\{\!|P|\!\}^{\beta}_{+}(X)) = \Sigma^+$ minimizes false positive. Note that, if the abstract semantics of the stegoprogram is complete, it may well happen that the concrete semantics of the stegoprogram is not complete, i.e. $\{\!|P_s|\!\}_{+}$ is not $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)$ (see Figure 3). This means that the knowledge of the

[†] This is an assumption usually made by software watermarking techniques, i.e. before the insertion the stegoembedder checks if the cover program is watermarkable.

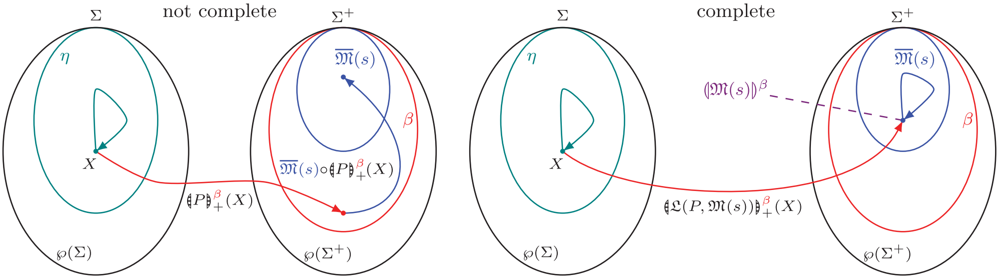

Σ
not complete
Σ^+
η
X
$\wp(\Sigma)$
$\overline{\mathfrak{M}}(s)$
β
$\overline{\mathfrak{M}}(s) \circ \Lbag P \Rbag_+^\beta(X)$
$\Lbag P \Rbag_+^\beta(X)$
$\wp(\Sigma^+)$
Σ
complete
Σ^+
η
X
$\wp(\Sigma)$
$\llparenthesis \mathfrak{M}(s) \rrparenthesis^\beta$
$\overline{\mathfrak{M}}(s)$
β
$\Lbag \mathfrak{L}(P, \mathfrak{M}(s)) \Rbag_+^\beta(X)$
$\wp(\Sigma^+)$

Fig. 3. $\mathcal{F}$-incompleteness[-completeness] for the cover program[stegoprogram].

stegomark may not be sufficient to extract the signature without knowing the semantic property used to encode it.

The different kinds of software watermarking techniques can be seen as instances of Definition 3.1.

— Static and abstract watermarking correspond to a watermarking system, where $\eta = id$ and β is decidable (i.e. implementable with static analysis). This captures the fact that the interpretation of the stegoprogram always reveals the stegomark, independently from the input.

— Dynamic watermarking corresponds to a watermarking system where $\eta \neq id$ and β is a generic (concrete) interpreter. In this case, the concrete semantics of the stegoprogram reveals the stegomark only when a particular input sequence is given.

Finally, since $\mathbb{F}$ is idempotent we can use it for tamper detection. In fact it holds that $\mathbb{F}_{\eta,\overline{\mathfrak{M}}(s)}(\Lbag P_s \Rbag_+^\beta) = \Lbag P_s \Rbag_+^\beta$. If we consider an attacker $\mathfrak{t} \in \textsc{Imp} \rightarrow \textsc{Imp}$ we have that $\mathbb{F}_{\eta,\overline{\mathfrak{M}}(s)}(\Lbag \mathfrak{t}(P_s) \Rbag_+^\beta) \neq \Lbag \mathfrak{t}(P_s) \Rbag_+^\beta$ and we can recognize that the stegoprogram has been tampered.

3.2. *Software watermarking features*

Given a software watermarking technique it is desirable to know if it is better or worse rather than other existing techniques, in some specific context of interest. In order to perform such a comparison we need to define some ‘features’ that allow us to measure the efficacy of a watermarking system. For this reason, in the following, we describe and formalize in the proposed framework the most significant features that a software watermarking system should have. The features that we introduce are the ones that can be formalized in terms of program semantics and abstract interpretation. There are features that cannot be formalized in this way as for example data-rate and credibility. Data-rate deals with the amount of information that can be embedded by the considered watermarking scheme and it strictly depends on the implementation of the stegomarker, so it is not an intrinsic property of the watermarking system. Credibility measures how strongly a watermarking scheme provides authorship and it typically requires statistical evaluations.

In the rest of this section, we refer to a watermarking system $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$, to a set of cover programs $\mathcal{P}$ and to a set of signatures $\mathcal{S}$. Moreover, we use β to denote the extraction domain and $\overline{\mathfrak{M}}(s)$ to denote the domain that semantically encodes the signature s.

3.2.1. *Resilience.* Resilience concerns the capacity of a software watermarking system to be immune to attacks. There are four major types of attacks (Collberg and Thomborson 1999).

Distortive attacks. The attacker modifies the stegoprogram in order to compromise the stegomark, i.e. the attacker applies syntactic or semantic transformations in order to make the signature no more recoverable by the stegoextractor.

Collusive attacks. The attacker compares different stegoprograms of the same cover program in order to obtain information on the stegomark. Doing so it could, for example, identify the location of the stegomark within the stegoprogram and then remove it.

Subtractive attacks. The attacker tries to eliminate the stegomark from the stegoprogram so that it is no longer possible to extract the signature (usually this attack needs a preliminary analysis for identifying the location of the stegomark).

Additive attacks. The attacker adds another stegomark to the stegoprogram so that the previous stegomark is ‘overwritten’ or so that the program contains more than one stegomark. In the latter case, it is not possible to establish which stegomark has been inserted first and therefore the legitimate owner cannot be determined.

A software watermarking system is resilient *w.r.t.* a particular kind of attack when it is immune to any attack of that type. We can classify attacks as *conservative* and *not conservative* according to the effects that they have on program semantics. Conservative attacks maintain the program denotational semantics (input/output) unmodified, while non-conservative attacks do not ensure this.

Observe that subtractive attacks and collusive attacks are related to the localization of the stegomark and hence the resilience to these attacks reduces to a secrecy problem (as explained below). In fact, following (Collberg and Thomborson 1999), we consider subtractive attacks to be those attacks that somehow locate the stegomark and then remove it. On the other hand, those attacks that eliminate the signature by creating a functionally equivalent unsigned program are considered to be distortive attacks in this work (they can be seen as distortive attacks that preserve the denotational semantics). Resilience to additive attacks is very difficult to obtain, in fact if an attacker adds another signature (with another technique) it is impossible to prove which stegomark was inserted first. For this reasons in the following, we focus on the resilience to distortive attacks.

A distortive attack can be seen as a program transformer $\mathfrak{t} \in \textsc{Imp} \to \textsc{Imp}$ that modifies programs preserving their denotational semantics, namely their input/output behaviour. Indeed, a distortive attack wants to modify the marked program as much as possible (while preserving its functionality) in order to compromise the stegomark. So, there are program's properties that the attack preserves and others that it does not preserve, namely there are abstractions $\psi \in uco(\wp(\Sigma^+))$ such that $\psi(\llparenthesis P \rrparenthesis_+) \neq \psi(\llparenthesis \mathfrak{t}(P) \rrparenthesis_+)$. According to Dalla Preda and Giacobazzi (2009), we denote with $\delta_{\mathfrak{t}} \in uco(\wp(\Sigma^+))$ the most concrete property preserved by the transformation $\mathfrak{t}$ on programs semantics, namely such that $\forall P \in \textsc{Imp} \,.\, \delta_{\mathfrak{t}}((\!|P|\!)_+) = \delta_{\mathfrak{t}}((\!|\mathfrak{t}(P)|\!)_+)$. This implies that every property ψ more abstract than $\delta_{\mathfrak{t}}$, i.e. $\delta_{\mathfrak{t}} \sqsubseteq \psi$, is preserved by $\mathfrak{t}$. Observe that when $\delta_{\mathfrak{t}} \sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) | s \in \mathcal{S}\}$ it means that the distortive attacker $\mathfrak{t}$ preserves the semantic encoding of all signatures and therefore the watermarking system is resilient *w.r.t.* $\mathfrak{t}$. When $\mathfrak{t}$ preserves the semantic encoding of a subset of possible signatures, those for which $\delta_{\mathfrak{t}} \sqsubseteq \overline{\mathfrak{M}}(s)$, we can identify the class of stegoprograms that resist to $\mathfrak{t}$. In the worst case, when $\forall s \in \mathcal{S} \,.\, \delta_{\mathfrak{t}} \not\sqsubseteq \overline{\mathfrak{M}}(s)$, the software watermarking system is not able to contrast in any way the attacker $\mathfrak{t}$. This leads to the definition of the following levels of resilience.

Definition 3.2 ($\mathfrak{t}$-resilience). A software watermarking system $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$ is:

- $\mathfrak{t}$-*resilient* when $\delta_{\mathfrak{t}} \sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) | s \in \mathcal{S}\}$.
- $\mathfrak{t}$-*vulnerable* when $\exists s \in \mathcal{S} \,.\, \delta_{\mathfrak{t}} \not\sqsubseteq \overline{\mathfrak{M}}(s)$.
- $\mathfrak{t}$-*ineffective* when $\forall s \in \mathcal{S} \,.\, \delta_{\mathfrak{t}} \not\sqsubseteq \overline{\mathfrak{M}}(s)$.

Furthermore, if an attacker $\mathfrak{t}$ is conservative it must preserve the denotational semantics, $\mathtt{DenSem} \in uco(\wp(\Sigma^+))$, of the original program so we have that $\delta_{\mathfrak{t}} \sqsubseteq \mathtt{DenSem}$. This domain is obtained from maximal finite traces semantics as $\mathtt{DenSem}(X) \triangleq \{\sigma \in \Sigma^+ | \exists \sigma' \in X \,.\, \sigma_{\vdash} = \sigma'_{\vdash} \wedge \sigma_{\dashv} = \sigma'_{\dashv}\}$. In this context, every property more abstract than $\mathtt{DenSem}$ is preserved.

Definition 3.3 (Resilience). A software watermarking system is resilient if

$$\mathtt{DenSem} \sqsubseteq \bigsqcap \left\{\overline{\mathfrak{M}}(s) | s \in \mathcal{S}\right\}$$

Basically, we say that a watermarking system is *resilient* when it is $\mathfrak{t}$-resilient to all those distortive attackers $\mathfrak{t}$ that preserve $\mathtt{DenSem}$. A software watermarking system that exhibits such behaviour has not been yet found and it is an open research topic to demonstrate its existence or not.[‡] If the attacker is not conservative, so it is willing to lose some original program functionalities in order to nullify the stegomark, then $\delta_{\mathfrak{t}} \not\sqsubseteq \mathtt{DenSem}$. In this case, we have a stronger notion of attackers and it is not possible to assert the resilience of a software watermarking system to distortive not conservative attacks.

This formalization of resilience allows us to compare two watermarking systems *w.r.t.* resilience. Given two watermarking systems $\mathfrak{A}_1 = \langle \mathfrak{L}_1, \mathfrak{M}_1, \beta_1 \rangle$ and $\mathfrak{A}_2 = \langle \mathfrak{L}_2, \mathfrak{M}_2, \beta_2 \rangle$, if it holds that $\bigsqcap\{\overline{\mathfrak{M}}_1(s) | s \in \mathcal{S}\} \sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}_2(s) | s \in \mathcal{S}\}$ then we have the following inclusion $\{\mathfrak{t} | \delta_{\mathfrak{t}} \sqsubseteq \{\overline{\mathfrak{M}}_1(s) | s \in \mathcal{S}\}\} \subseteq \{\mathfrak{t} | \delta_{\mathfrak{t}} \sqsubseteq \{\overline{\mathfrak{M}}_2(s) | s \in \mathcal{S}\}\}$. Therefore, $\mathfrak{A}_2$ is, in general, more resilient than $\mathfrak{A}_1$. In Section 4, we show what we can do when $\overline{\mathfrak{M}}_1$ and $\overline{\mathfrak{M}}_2$ are not comparable.

3.2.2. *Secrecy.* Secrecy concerns the difficulty of recovering the stegomark embedded in a stegoprogram. Ideally, a watermarking system is secret when it is impossible to extract the signature from a stegoprogram without knowing the stegoextractor. In practice, secrecy can be seen as the ability of the watermarking system to make indistinguishable to

[‡] The results in Barak *et al.* (2001) and recently in Garg *et al.* (2013) about impossibility of watermarking seem to lead to a negative answer.

an attacker the set of signatures embedded in a program. This clearly relates to the resilience to collusive attacks, which requires that an attacker is not able to distinguish between stegoprograms that embed different signatures in the same cover program. This notion can be formalized in terms of *higher-order abstract non-interference*, introduced in Subsection 2.4. The private input is the set of possible stegomarks $\mathsf{Q} = \{\mathfrak{M}(s) | s \in \mathcal{S}\}$, while the public input is the set of cover programs $\mathsf{P} = \mathcal{P}$. Let $\phi \in uco(\wp(\Sigma^+))$ be a property that represents some stegomarks, and indeed some signatures. We assume that the attacker does not have access to cover programs, so the abstraction of the public input is *id*.

Definition 3.4 (ϕ-secrecy). A software watermarking system $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$ is *ϕ-secret w.r.t.* an attacker ρ if $\underset{+}{\overset{\mathrm{H}}{}}[id]\mathfrak{L}(\phi \Rightarrow \rho)_{\mathfrak{bca}}$ holds, i.e. if $\forall P \in \mathsf{P}\, \forall Q_1, Q_2 \in \mathsf{Q}$ we have that

$$(\!|Q_1|\!)^{\phi}_{+} = (\!|Q_2|\!)^{\phi}_{+} \Rightarrow (\!|\mathfrak{L}(P, Q_1)|\!)^{\rho}_{+} = (\!|\mathfrak{L}(P, Q_2)|\!)^{\rho}_{+}.$$

This means that if we mark a cover program with two different signatures that are equivalent modulo ϕ, then the attacker ρ does not distinguish between the two generated stegoprograms. Thus, any signature with the same property ϕ can be used for generating stegoprograms resilient to collusive attacks made by the attacker ρ. We say that a system is *secret* when it is $\top$-secret, meaning that the set of indistinguishable signatures is $\mathcal{S}$. Given a property ϕ specifying a set of signatures, we can characterize the most concrete observer $\hat{\rho}$ for which $\underset{+}{\overset{\mathrm{H}}{}}[id]\mathfrak{L}(\phi \Rightarrow \hat{\rho})_{\mathfrak{bca}}$ holds, called *most powerful ϕ-secret attacker*. It can be characterized in terms of the secret kernel of higher-order abstract non-interference. Indeed, it corresponds to the most concrete domain $\hat{\rho}$ that is more abstract than *id* and such that $\underset{+}{\overset{\mathrm{H}}{}}[id]\mathfrak{L}(\top \Rightarrow \hat{\rho})_{\mathfrak{bca}}$ holds, i.e. $\hat{\rho} = \mathcal{K}^{\mathrm{H}+}_{\mathfrak{L},id,(\phi)}(id)$. The operator $\mathcal{K}_{\mathfrak{L},id,(\phi)}$ and the definition of the higher-order abstract non-interference can be found in Subsection 2.4. For example, the most powerful $\top$-secret attacker is $\mathcal{K}_{\mathfrak{L},id,(\top)}(id) = \{X \in \wp(\Sigma^+) | P \in \mathsf{P}, X = \bigcup_{Q \in \mathsf{Q}} (\!|\mathfrak{L}(P, Q)|\!)_{+}\} \cup \{\Sigma^+\}$ and it abstracts in the same object the traces of all possible stegoprograms related to the same cover program. Of course, any attacker with at least the same precision of the extractor β violates secrecy.

Thus, the secrecy level of a watermarking system is given by the most abstract property ϕ and by the most concrete observer $\hat{\rho}$ for which non-interference $\underset{+}{\overset{\mathrm{H}}{}}[id]\mathfrak{L}(\phi \Rightarrow \hat{\rho})_{\mathfrak{bca}}$ holds. The more ϕ is abstract and the more the system is secret. Vice versa, the more $\hat{\rho}$ is concrete and the more the system is secret. Observe that ϕ can range from *id* (all the signatures are distinguishable) to $\top$ (no signature is distinguishable). When the most powerful ϕ-secret attacker $\hat{\rho}$ is equal to $\top$ then every attacker is able to distinguish the signatures. Otherwise, the more $\hat{\rho}$ is concrete and the more the system is secret.

This formalization of secrecy allows us to compare two watermarking systems *w.r.t.* secrecy. Given two software watermarking systems $\mathfrak{A}_1 = \langle \mathfrak{L}_1, \mathfrak{M}_1, \beta_1 \rangle$ and $\mathfrak{A}_2 = \langle \mathfrak{L}_2, \mathfrak{M}_2, \beta_2 \rangle$ we consider their most powerful ϕ-secret attackers $\hat{\rho}_1$ and $\hat{\rho}_2$. If $\hat{\rho}_1 \sqsubseteq \hat{\rho}_2$ we have that $\mathfrak{A}_1$ is more secret than $\mathfrak{A}_2$ *w.r.t.* ϕ. Indeed, a stronger attacker is necessary in order to violate ϕ-secrecy for $\mathfrak{A}_1$, rather than $\mathfrak{A}_2$. In Section 4, we show what we can do when $\hat{\rho}_1$ and $\hat{\rho}_2$ are not comparable.

3.2.3. *Transparence.* Transparence concerns the ability to make it hard to realize that a program is a stegoprogram, namely if it contains a signature. A watermarking system is transparent *w.r.t.* an observer if the latter is not able to distinguish a cover program from every stegoprogram generated starting from it.

Definition 3.5 (Transparence). A software watermarking system $\langle \mathfrak{L}, \mathfrak{M}, \beta \rangle$ is transparent *w.r.t.* an attacker $\rho \in uco(\wp(\Sigma^+))$ if $\forall P \in \mathcal{P}\, \forall s \in \mathcal{S}\, .\, (\!|P|\!)^{\rho}_{+} = (\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!)^{\rho}_{+}$.

The greatest is the set of observers for which the system is transparent and the greatest is the level of transparence of the watermarking system. So, the characterization of the most concrete observer $\tilde{\rho}$ for which the system is transparent is a good measure of the transparence of the software watermarking system. This observer $\tilde{\rho}$ is called the *most powerful transparent attacker*. This attacker can be characterized following the same approach used for the most powerful $\top$-secret attacker. In fact a system, in order to be transparent *w.r.t.* an attacker has clearly to be $\top$-secret *w.r.t.* that attacker. So, recalling the set of indistinguishable elements for HOANI $\Upsilon^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(P, Q)$ introduced in Section 2.4, we can define its counterpart for transparence, namely the set of all elements that have to be indistinguishable for ensuring transparence as follows:

$$\Upsilon^{\mathrm{H}+}_{\mathfrak{L},id,(\top)}(P, \mathfrak{M}(s)) = \left\{ (\!|\mathfrak{L}(P', \mathfrak{M}(s'))|\!)_{+} | P' \equiv_{id} P \wedge \mathfrak{M}(s') \equiv_{\top} \mathfrak{M}(s) \right\} \cup \left\{ (\!|P|\!)_{+} \right\}.$$

These sets are collections of sets of traces that a secure abstraction should not distinguish, i.e. that a secure abstraction should approximate in the same object. Then, we can continue the construction of the secret kernel as done in Section 2.4. Clearly, the analysis is useful for any observer most concrete, or as concrete as, β. In fact the system cannot be transparent for attackers that are at least as precise as the extractor.

Similarly to what we have done for secrecy, given two software watermarking systems $\mathfrak{A}_1 = \langle \mathfrak{L}_1, \mathfrak{M}_1, \beta_1 \rangle$ and $\mathfrak{A}_2 = \langle \mathfrak{L}_2, \mathfrak{M}_2, \beta_2 \rangle$, we consider their most powerful transparent attackers $\tilde{\rho}_1$ and $\tilde{\rho}_2$: if $\tilde{\rho}_1 \sqsubseteq \tilde{\rho}_2$ we have that $\mathfrak{A}_1$ is more transparent than $\mathfrak{A}_2$. In Section 4, we show what we can do when $\tilde{\rho}_1$ and $\tilde{\rho}_2$ are not comparable.

3.2.4. *Accuracy.* A watermarking system is accurate if it preserves the functionality of the cover program, i.e. the cover program and the stegoprogram have to exhibit the same *observable behaviour*. This concept can be defined as ‘behaviour as experienced by the user’ (Collberg *et al.* 1997). This means that the stegoprogram can do something that the cover program does not do, as long as these *side-effects* are not visible to the user. Clearly, this definition is very loose and it depends on what the user is able to observe of program’s execution. We formalize this concept by requiring that the stegoprogram and the original program have the same *observable* denotational semantics. This means that, fixed what the user is able to observe, the stegoprogram and the cover program must exhibit the same input/output behaviour *w.r.t.* the fixed observation level.

Formally, we define an observational abstraction $\alpha_{\mathcal{O}}$ that characterizes what is interesting to observe about the denotational semantics of programs. Then, accuracy requires that the cover program P and its stegoprogram P_s are indistinguishable *w.r.t.* $\alpha_{\mathcal{O}}$, i.e. $\alpha_{\mathcal{O}}((\!|P|\!)_D) = \alpha_{\mathcal{O}}((\!|P_s|\!)_D)$ (Cousot and Cousot 2002). Here, $(\!|P|\!)_D$ denotes the (angelic) denotational semantics defined in Cousot (2002), which is an abstraction of the maximal trace semantics. Indeed, $(\!|P|\!)_D$ is isomorphic to $\texttt{DenSem}((\!|P|\!)_+)$. Then let $\langle D_{\mathcal{O}}, \leqslant_{\mathcal{O}}\rangle$ be a poset and $\alpha_{\mathcal{O}} \in \wp(\Sigma) \to D_{\mathcal{O}}$ be a function such that $(\langle \wp(\Sigma), \subseteq\rangle, \alpha_{\mathcal{O}}, \alpha_{\mathcal{O}}^{+}, \langle D_{\mathcal{O}}, \leqslant_{\mathcal{O}}\rangle)$ is a Galois connection. We say that two programs $P, Q \in$ Imp are $\alpha_{\mathcal{O}}$-*observationally equivalent iff* $\alpha_{\mathcal{O}}((\!|P|\!)_D) = \alpha_{\mathcal{O}}((\!|Q|\!)_D)$.

Hence, a software watermarking system is accurate for an observational abstraction $\alpha_{\mathcal{O}}$, if for every program $P \in \mathcal{P}$ and for every signature $s \in \mathcal{S}$ it holds that P is $\alpha_{\mathcal{O}}$-observationally equivalent to P_s.

Definition 3.6 (Accuracy). Given a poset $\langle D_{\mathcal{O}}, \leqslant_{\mathcal{O}}\rangle$ and an observational abstraction $\alpha_{\mathcal{O}} \in \wp(\Sigma) \to D_{\mathcal{O}}$ such that $(\langle \wp(\Sigma), \subseteq\rangle, \alpha_{\mathcal{O}}, \alpha_{\mathcal{O}}^{+}, \langle D_{\mathcal{O}}, \leqslant_{\mathcal{O}}\rangle)$ is a GC, we have that a watermarking system $\langle \mathfrak{L}, \mathfrak{M}, \beta\rangle$ is *accurate*, *w.r.t.* $\alpha_{\mathcal{O}}$, if for each program $P \in \mathcal{P}$ and for each signature $s \in \mathcal{S}$ it holds that $\alpha_{\mathcal{O}}((\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!)_{\mathrm{D}}) = \alpha_{\mathcal{O}}((\!|P|\!)_{\mathrm{D}})$.

Accuracy states that given an observational abstraction $\alpha_{\mathcal{O}}$, every cover program P is $\alpha_{\mathcal{O}}$-*observationally equivalent* to any stegoprogram P_s embedding a signature s.

For example, a reasonable observational abstraction could be the output given to the user. In the language, Imp there is a command for catching the values inserted by the user, but there are no commands for showing the results provided to the user. This can be simulated by restricting a subset of program's variables to store the values that are shown to the user. We denote with $\texttt{Var}_{out}(P) \subseteq {}^{var}[\![P]\!]$ this set of output variables. The user who wants to observe the output of the program P has to check the values of the variables in this set. Let us recall that the environment function $\rho \in \texttt{Env}$ defines the binding between variables and their current values. We denote with $\texttt{dom}_{out}(\rho) \subseteq \texttt{dom}(\rho)$ the set of output variables of environment ρ. The abstraction that catches the output given to the user has to observe only the values of those variables, so $\alpha_{\mathcal{O}} \triangleq \alpha_{\mathcal{O}}^{out} \in \wp(\Sigma) \to \wp(\texttt{Env})$ can be defined in the following way:

$$\alpha_{\mathcal{O}}^{out}(X) \triangleq \left\{ \rho' \in \texttt{Env} \;\middle|\; \exists\varsigma \in X \,.\, \begin{array}{c} \varsigma = \langle C, \langle \rho, \iota\rangle\rangle \wedge \\ \texttt{dom}(\rho') = \texttt{dom}_{out}(\rho) \wedge \\ \forall y \in \texttt{dom}_{out}(\rho) \,.\, \rho'(y) = \rho(y) \end{array} \right\}$$

In this case, $\alpha_{\mathcal{O}}^{out}((\!|P|\!)_D)$ catches the input/output behaviour of P, specified as a relation between the inputs inserted by the user and the outputs provided to the user during the execution of P. Clearly, this approach is not applicable to programs that do not interact with the user. In this case, it is necessary to chose another type of observable abstraction, more suitable for the context.

If we apply $\alpha_{\mathcal{O}}^{out}$ to Definition 3.6, we obtain that a software watermarking system is accurate, *w.r.t.* $\alpha_{\mathcal{O}}^{out}$, if

$$\forall P \in \mathcal{P}\, \forall s \in \mathcal{S} \,.\, \alpha_{\mathcal{O}}^{out}((\!|P|\!)_D) = \alpha_{\mathcal{O}}^{out}((\!|P_s|\!)_D),$$

As regarding accuracy, this is a property that is not directly comparable among different watermarking techniques since it is defined *w.r.t.* the observational abstraction of interest. However, the proposed formal framework provides the right setting for formally proving the accuracy of a watermarking system *w.r.t.* an observational property.

4. Validation

In order to validate our model, we have formalized five known watermarking techniques in our framework and we have compared them *w.r.t.* resilience, secrecy, transparence and accuracy. In the following, we indicate with $\bigsqcap\overline{\mathfrak{M}}$ the reduced product of all the abstract stegomark $\overline{\mathfrak{M}}(s)$, i.e. $\bigsqcap\overline{\mathfrak{M}} \triangleq \bigsqcap\{\overline{\mathfrak{M}}(s) | s \in \mathcal{S}\}$ (see Subsection 2.2).

4.1. *Watermarking techniques*

In order to cover a wide range of watermarking systems and to prove the generality of our framework, we took in consideration an heterogeneous set of techniques. In the following, we report a brief description of the considered watermarking techniques.

Block-reordering watermarking. Static technique, introduced in the patent (Davidson and Myhrvold 1996). The signature is a natural number and it is codified as a permutation of the basic blocks of the cover program by modifying the program's *Control Flow Graph*, shortly CFG, (direct jumps are inserted to preserve program semantics). The embedding ensures that the semantics of the cover program remains unmodified.

Static graph-based watermarking. Static technique, introduced in Venkatesan *et al.* (2001). The signature is a natural number and it is codified as a graph which is added to the CFG of the cover program while preserving its semantics. In particular, a program whose CFG has the same shape of the graph generated starting from the signature is derived and then added to cover program's CFG. The embedding ensures that the semantics of the cover program remains unmodified. The nodes of the graph encoding the signature are marked before embedding in order to be identifiable at extraction time.

Dynamic graph-based watermarking. Dynamic technique, introduced in Collberg and Thomborson (1999). The signature is a natural number and it is codified as a graph allocated in the dynamic memory (in the *heap*) of the program, during a particular execution. This execution is generated by a particular sequence of enabling input values. Given a signature, a graph that encodes the signature and the code that builds this graph are generated. The embedder adds the code that generates the signature graph to the cover program in such a way that this code is executed only with the enabling input. Furthermore, in order to facilitate the extraction process, the code that builds the root of the signature graph is the last one to be executed, in this way the root of the signature graph is the last node inserted in the heap.

Path-based watermarking. Dynamic technique, introduced in Collberg *et al.* (2004). The signature is a natural number and it is codified as a sequence of choices (true/false) made at conditional statements during a particular execution of the program. This execution is generated by a particular sequence of enabling input values. The embedder takes the program code and it adds bogus branches in order to generate the desired true/false sequence when executed on the enabling input.

Abstract constant propagation watermarking. This is the only known abstract watermarking technique and it was introduced in Cousot and Cousot (2004). The signature is a natural number and it is inserted into a particular variable `w` which, although being

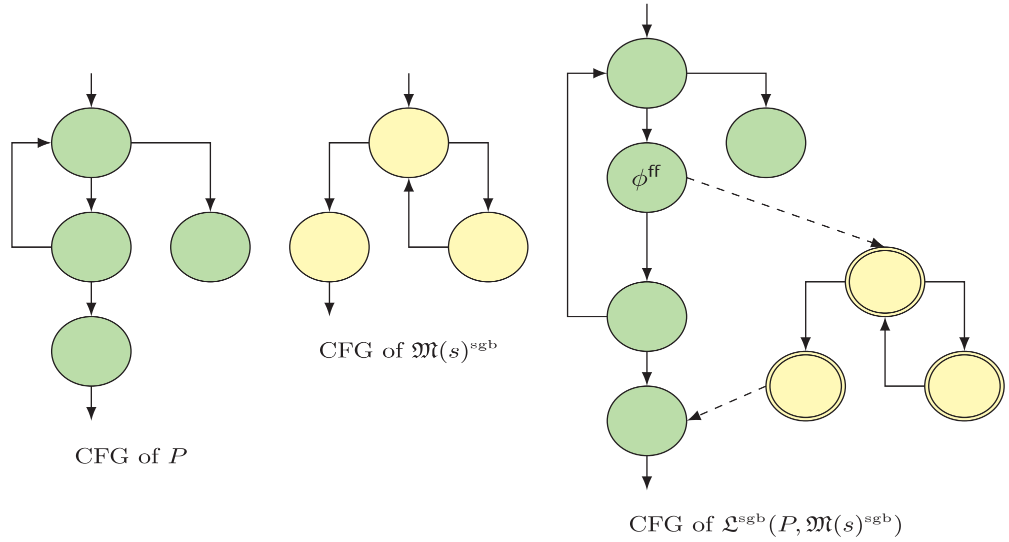

ϕ^{ff}
CFG of P
CFG of $\mathfrak{M}(s)^{\mathrm{sgb}}$
CFG of $\mathfrak{L}^{\mathrm{sgb}}(P, \mathfrak{M}(s)^{\mathrm{sgb}})$

Fig. 4. Static graph-based watermarking.

> modified during program execution, remains constant modulo an integer n. This means that `w` is constant only in the domain of congruences modulo n, while other domains consider `w` to have stochastic behaviour. Thus, only the abstract interpretation of the program on a domain which is able to represent precisely congruences modulo n can extract the signature. In order to embed arbitrarily long signatures the technique refers to the Chinese remainder theorem. The insertion of the signature does not alter the semantics of the cover program.

We provide the formalization for static, dynamic and abstract watermarking techniques. Doing so, we want to emphasize our main claim, i.e. that static and dynamic watermarking are instances of abstract watermarking (for this latter the encoding in the framework is intuitively more natural).

4.1.1. *Static techniques.* We present block-reordering watermarking and static graph-based watermarking together because they are very similar. In the following, we use the apex $^{\mathrm{br}}$ to refer to block-reordering watermarking and the apex $^{\mathrm{sgb}}$ to refer to static graph-based watermarking.

In block-reordering watermarking the signature is codified as a permutation of the basic blocks of the cover program's CFG, where direct jumps are inserted to preserve the cover program's semantics. Instead, in static graph-based watermarking a program whose CFG encodes the signature is added to the cover program in a way that the semantics of this latter remains unmodified, as showed in Figure 4. The nodes of the added graph are marked before the embedding.

Let $\mathtt{graph} \in \mathbb{N} \to \mathbb{G}$ be a function that codifies a signature in a graph. Let $\mathtt{cfg} \in \Sigma^+ \to \mathbb{G}$ be a function that, given a trace σ, returns the CFG of σ and let $\mathtt{mark} \in \Sigma^+ \to \mathbb{G}$ be a function that, given a trace σ, outputs the marked subgraph of the CFG of σ, for a certain marking criterion (building the CFG and locating its marked nodes are both task easily implementable analyzing a program trace).

The semantics $\llparenthesis P \rrparenthesis_{+}^{\beta_{\mathrm{br}}}$ extracts the CFG of P, for block-reordering, and the semantics $\llparenthesis P \rrparenthesis_{+}^{\beta_{\mathrm{sgb}}}$ extracts the marked subgraph of the CFG of P, for static graph-based. So the extraction domains β_{br} and β_{sgb} are as follows:

$$\beta_{\mathrm{br}} \triangleq \{X \in \wp(\Sigma^{+}) | \exists g \in \mathbb{G} \,.\, X = \{\sigma \in \Sigma^{+} | \mathtt{cfg}(\sigma) = g\}\} \cup \{\varnothing, \Sigma^{+}\},$$
$$\beta_{\mathrm{sgb}} \triangleq \{X \in \wp(\Sigma^{+}) | \exists g \in \mathbb{G} \,.\, X = \{\sigma \in \Sigma^{+} | \mathtt{mark}(\sigma) = g\}\} \cup \{\varnothing, \Sigma^{+}\}.$$

In β_{br} there are all the sets of traces with the same CFG, instead, in β_{sgb} there are all the sets of traces whose CFG contains the same marked graph. We indicate with $\mathcal{W}_s^{\mathrm{br}} \triangleq \{\sigma \in \Sigma^{+} | \mathtt{graph}(s) = \mathtt{cfg}(\sigma)\}$ the set of traces whose CFG codifies the signature s, and with $\mathcal{W}_s^{\mathrm{sgb}} \triangleq \{\sigma \in \Sigma^{+} | \mathtt{graph}(s) = \mathtt{mark}(\sigma)\}$ the set of traces whose CFG contains the marked graph that codifies the signature s. Both are static techniques so $\eta = \eta_{\mathrm{br}} = \eta_{\mathrm{sgb}} = id$. Clearly, $\llparenthesis \mathfrak{M}(s) \rrparenthesis_{+}^{\beta_{\mathrm{br}}} = \mathcal{W}_s^{\mathrm{br}}$ and so $\overline{\mathfrak{M}}(s)^{\mathrm{br}} = \{\varnothing, \mathcal{W}_s^{\mathrm{br}}, \Sigma^{+}\}$. Analogously, $\llparenthesis \mathfrak{M}(s) \rrparenthesis_{+}^{\beta_{\mathrm{sgb}}} = \mathcal{W}_s^{\mathrm{sgb}}$ and so $\overline{\mathfrak{M}}(s)^{\mathrm{sgb}} = \{\varnothing, \mathcal{W}_s^{\mathrm{sgb}}, \Sigma^{+}\}$.

Let $\overline{\mathbb{G}} \triangleq \{\bot, \mathbb{G}\} \cup \{\mathbb{G}\}$. The domains β_{br}, β_{sgb} can be defined as $\beta_{\mathrm{br}} \triangleq \beta_{\mathrm{br}\gamma} \circ \beta_{\mathrm{br}\alpha}$, $\beta_{\mathrm{sgb}} \triangleq \beta_{\mathrm{sgb}\gamma} \circ \beta_{\mathrm{sgb}\alpha}$, where $\beta_{\mathrm{br}\alpha}, \beta_{\mathrm{sgb}\alpha} \in \wp(\Sigma^{+}) \rightarrow \overline{\mathbb{G}}$ and $\beta_{\mathrm{br}\gamma}, \beta_{\mathrm{sgb}\gamma} \in \overline{\mathbb{G}} \rightarrow \wp(\Sigma^{+})$ are as follows:

$$\beta_{\mathrm{br}_\alpha} \triangleq \lambda X \,.\, \begin{cases} \bot & \text{if } X = \varnothing \\ g & \text{if } \forall \sigma \in X \,.\, g = \mathtt{cfg}(\sigma) \\ \mathbb{G} & \text{otherwise} \end{cases} \quad \beta_{\mathrm{br}_\gamma} \triangleq \lambda g \,.\, \begin{cases} \varnothing & \text{if } g = \bot \\ \{\sigma \in \Sigma^{+} | g = \mathtt{cfg}(\sigma)\} & \text{if } g \in \mathbb{G} \\ \Sigma^{+} & \text{otherwise} \end{cases}$$

$$\beta_{\mathrm{sgb}_\alpha} \triangleq \lambda X \,.\, \begin{cases} \bot & \text{if } X = \varnothing \\ g & \text{if } \forall \sigma \in X \,.\, g = \mathtt{mark}(\sigma) \\ \mathbb{G} & \text{otherwise} \end{cases} \quad \beta_{\mathrm{sgb}_\gamma} \triangleq \lambda g \,.\, \begin{cases} \varnothing & \text{if } g = \bot \\ \{\sigma \in \Sigma^{+} | g = \mathtt{mark}(\sigma)\} & \text{if } g \in \mathbb{G} \\ \Sigma^{+} & \text{otherwise} \end{cases}$$

Instead $\overline{\mathfrak{M}}(s)^{\mathrm{br}}$ and $\overline{\mathfrak{M}}(s)^{\mathrm{sgb}}$ can be defined as $\overline{\mathfrak{M}}(s)^{\mathrm{br}} \triangleq \overline{\mathfrak{M}}(s)_{\gamma}^{\mathrm{br}} \circ \overline{\mathfrak{M}}(s)_{\alpha}^{\mathrm{br}}$ and $\overline{\mathfrak{M}}(s)^{\mathrm{sgb}} \triangleq \overline{\mathfrak{M}}(s)_{\gamma}^{\mathrm{sgb}} \circ \overline{\mathfrak{M}}(s)_{\alpha}^{\mathrm{sgb}}$, where $\overline{\mathfrak{M}}(s)_{\gamma}^{\mathrm{br}}, \overline{\mathfrak{M}}(s)_{\gamma}^{\mathrm{sgb}} \in \wp(\Sigma^{+}) \rightarrow \wp(\Sigma^{+})$ are $\overline{\mathfrak{M}}(s)_{\gamma}^{\mathrm{br}} = \overline{\mathfrak{M}}(s)_{\gamma}^{\mathrm{sgb}} \triangleq id$ and $\overline{\mathfrak{M}}(s)_{\alpha}^{\mathrm{br}}, \overline{\mathfrak{M}}(s)_{\alpha}^{\mathrm{sgb}} \in \wp(\Sigma^{+}) \rightarrow \wp(\Sigma^{+})$ are as follows:

$$\overline{\mathfrak{M}}(s)_{\alpha}^{\mathrm{br}} \triangleq \lambda X \,.\, \begin{cases} \varnothing & \text{if } X = \varnothing \\ \mathcal{W}_s^{\mathrm{br}} & \text{if } X \subseteq \mathcal{W}_s^{\mathrm{br}} \\ \Sigma^{+} & \text{otherwise} \end{cases} \quad \overline{\mathfrak{M}}(s)_{\alpha}^{\mathrm{sgb}} \triangleq \lambda X \,.\, \begin{cases} \varnothing & \text{if } X = \varnothing \\ \mathcal{W}_s^{\mathrm{sgb}} & \text{if } X \subseteq \mathcal{W}_s^{\mathrm{sgb}} \\ \Sigma^{+} & \text{otherwise} \end{cases}$$

It is simple to see that for all signatures s we have $\overline{\mathfrak{M}}(s)^{\mathrm{br}}(\wp(\Sigma^{+})) \subseteq \beta_{\mathrm{br}}(\wp(\Sigma^{+}))$ and $\overline{\mathfrak{M}}(s)^{\mathrm{sgb}}(\wp(\Sigma^{+})) \subseteq \beta_{\mathrm{sgb}}(\wp(\Sigma^{+}))$, hence $\beta_{\mathrm{br}} \sqsubseteq \overline{\mathfrak{M}}(s)^{\mathrm{br}}$ and $\beta_{\mathrm{sgb}} \sqsubseteq \overline{\mathfrak{M}}(s)^{\mathrm{sgb}}$. The input domain is *id* so there is not an enabling input or, equivalently, all the inputs reveal the watermark.

Clearly, for every possible set of initial states, the CFG of the stegoprogram is the same, i.e. it exists $g \in \mathbb{G}$ such that $\forall \sigma \in \llparenthesis \mathfrak{L}^{\mathrm{br}}(P, \mathfrak{M}(s)^{\mathrm{br}}) \rrparenthesis_{+}$ we have that $g = \mathtt{cfg}(\sigma)$ and it exists $g' \in \mathbb{G}$ such that $\forall \sigma \in \llparenthesis \mathfrak{L}^{\mathrm{sgb}}(P, \mathfrak{M}(s)^{\mathrm{sgb}}) \rrparenthesis_{+}$ we have that $g' = \mathtt{cfg}(\sigma)$. Due to the techniques' definitions, the graph g is equal to $\mathtt{graph}(s)$ and in g' there is a marked subgraph equal to $\mathtt{graph}(s)$. So, we have that $\{\!|\mathfrak{L}^{\mathrm{br}}(P, \mathfrak{M}(s)^{\mathrm{br}})|\!\}_{+}^{\beta_{\mathrm{br}}}(X) = \mathcal{W}_s^{\mathrm{br}}$ and $\{\!|\mathfrak{L}^{\mathrm{sgb}}(P, \mathfrak{M}(s)^{\mathrm{sgb}})|\!\}_{+}^{\beta_{\mathrm{sgb}}}(X) = \mathcal{W}_s^{\mathrm{sgb}}$, for every possible set of initial states.

Now, we can note that the CFG of $\mathfrak{M}(s)^{\text{br}}$ is $\texttt{graph}(s)$, so $\{\!|\mathfrak{M}(s)^{\text{br}}|\!\}_{+}^{\beta_{\text{br}}}(X) = \mathcal{W}_s^{\text{br}}$ for every possible set of initial states. Therefore, we have $\forall X \in \wp(\Sigma) \,.\, \{\!|\mathfrak{L}^{\text{br}}(P, \mathfrak{M}(s)^{\text{br}})|\!\}_{+}^{\beta_{\text{br}}}(X) = \{\!|\mathfrak{M}(s)^{\text{br}}|\!\}_{+}^{\beta_{\text{br}}}(X) = (\!|\mathfrak{M}(s)^{\text{br}}|\!)_{+}^{\beta_{\text{br}}}$. Analogously, the CFG of $\mathfrak{M}(s)^{\text{sgb}}$ is $\texttt{graph}(s)$ and it is marked by design, so $\{\!|\mathfrak{M}(s)^{\text{sgb}}|\!\}_{+}^{\beta_{\text{sgb}}}(X) = \mathcal{W}_s^{\text{sgb}}$ for every possible set of initial states. Hence, for every set of initial states X, we have that both $\{\!|\mathfrak{L}^{\text{sgb}}(P, \mathfrak{M}(s)^{\text{sgb}})|\!\}_{+}^{\beta_{\text{sgb}}}(X)$ and $\{\!|\mathfrak{M}(s)^{\text{sgb}}|\!\}_{+}^{\beta_{\text{sgb}}}(X)$ are equal to $\mathcal{W}_s^{\text{sgb}}$, which represents the signature s.

So we can note that, as expected, for every signature s, $\{\!|\mathfrak{L}^{\text{br}}(P, \mathfrak{M}(s)^{\text{br}})|\!\}_{+}^{\beta_{\text{br}}}$ is $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)^{\text{br}}$ and $\{\!|\mathfrak{L}^{\text{sgb}}(P, \mathfrak{M}(s)^{\text{sgb}})|\!\}_{+}^{\beta_{\text{sgb}}}$ is $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)^{\text{sgb}}$.

Let us briefly discuss the features of these techniques.

Resilience. The systems are not resilient, because they are not fully immune to distortive attacks, i.e. $\texttt{DenSem} \not\sqsubseteq \bigsqcap\{\overline{\mathfrak{M}(s)}^{\text{br}} | s \in \mathcal{S}\}$ and $\texttt{DenSem} \not\sqsubseteq \bigsqcap\{\overline{\mathfrak{M}(s)}^{\text{sgb}} | s \in \mathcal{S}\}$. In fact, suppose that $\texttt{DenSem} \sqsubseteq \bigsqcap \overline{\mathfrak{M}}^{\text{br}}$ and $\texttt{DenSem} \sqsubseteq \bigsqcap \overline{\mathfrak{M}}^{\text{sgb}}$, so $\forall X \in \wp(\Sigma^{+})$ it holds that $\texttt{DenSem}(X) \subseteq \bigsqcap \overline{\mathfrak{M}}^{\text{br}}(X)$ and $\forall Y \in \wp(\Sigma^{+})$ it holds that $\texttt{DenSem}(Y) \subseteq \bigsqcap \overline{\mathfrak{M}}^{\text{sgb}}(Y)$. Let $X = \mathcal{W}_s^{\text{br}}$ and $Y = \mathcal{W}_s^{\text{sgb}}$, for a generic signature s, so $\bigsqcap \overline{\mathfrak{M}}^{\text{br}}(X) = \mathcal{W}_s^{\text{br}}$ and $\bigsqcap \overline{\mathfrak{M}}^{\text{sgb}}(Y) = \mathcal{W}_s^{\text{sgb}}$. But $\mathcal{W}_s^{\text{br}} \subsetneq \texttt{DenSem}(\mathcal{W}_s^{\text{br}})$, because there is at least a trace with the same initial and final state of a trace in $\mathcal{W}_s^{\text{br}}$ with $\texttt{cfg}(\sigma) \neq \texttt{graph}(s)$. For example, take a program equal to $\mathfrak{M}(s)^{\text{br}}$ in which we insert an opaque predicate.

Clearly its traces are in $\texttt{DenSem}(\mathcal{W}_s^{\text{br}})$ but they are not in $\mathcal{W}_s^{\text{br}}$, because this traces have a different CFG. Analogously, $\mathcal{W}_s^{\text{sgb}} \subsetneq \texttt{DenSem}(\mathcal{W}_s^{\text{sgb}})$, because there is at least a trace with the same initial and final state of a trace in $\mathcal{W}_s^{\text{sgb}}$ with $\texttt{mark}(\sigma) \neq \texttt{graph}(s)$. For example, take a program equal to $\mathfrak{L}^{\text{sgb}}(P, \mathfrak{M}(s)^{\text{sgb}})$ in which all the nodes of its CFG are unmarked. Clearly its traces are in $\texttt{DenSem}(\mathcal{W}_s^{\text{sgb}})$ but they are not in $\mathcal{W}_s^{\text{sgb}}$, because these traces do not have a marked subgraph.

So there are X and Y such that $\texttt{DenSem}(X) \nsubseteq \bigsqcap \overline{\mathfrak{M}}^{\text{br}}(X)$ and $\texttt{DenSem}(Y) \nsubseteq \bigsqcap \overline{\mathfrak{M}}^{\text{sgb}}(Y)$. Hence $\texttt{DenSem} \not\sqsubseteq \bigsqcap \overline{\mathfrak{M}}^{\text{br}}$ and $\texttt{DenSem} \not\sqsubseteq \bigsqcap \overline{\mathfrak{M}}^{\text{sgb}}$. Indeed, the systems are vulnerable to control flow obfuscation techniques (basically the ones which modify the CFG). For example, a CFG flattening attack is able to damage the stegomark.

Secrecy. As regarding secrecy, the most powerful $\top$-secret attackers for block-reordering watermarking and static graph-based watermarking are as follows:

$$\mathcal{K}_{\mathfrak{L}^{\text{br}},id,(\top)}(id) = \{X \in \wp(\Sigma^{+}) | P \in \mathsf{P}, X = \textstyle\bigcup_{Q \in \mathsf{Q}} (\!|\mathfrak{L}^{\text{br}}(P,Q)|\!)_{+}\} \cup \{\Sigma^{+}\},$$
$$\mathcal{K}_{\mathfrak{L}^{\text{sgb}},id,(\top)}(id) = \{X \in \wp(\Sigma^{+}) | P \in \mathsf{P}, X = \textstyle\bigcup_{Q \in \mathsf{Q}} (\!|\mathfrak{L}^{\text{sgb}}(P,Q)|\!)_{+}\} \cup \{\Sigma^{+}\}.$$

They abstract in the same object the traces of all possible stegoprograms related to the same cover program.

Accuracy. Finally, the systems are $\alpha_{\mathcal{O}}^{out}$ accurate, where $\alpha_{\mathcal{O}}^{out}$ is an abstraction that observes only the output values showed to the user, since this property of the denotational semantics is preserved. Clearly, for every program P and for every signature s it holds that $\alpha_{\mathcal{O}}^{out}((\!|P|\!)_D) = \alpha_{\mathcal{O}}^{out}((\!|\mathfrak{L}^{\text{br}}(P, \mathfrak{M}(s)^{\text{sgb}})|\!)_D)$ and $\alpha_{\mathcal{O}}^{out}((\!|P|\!)_D) = \alpha_{\mathcal{O}}^{out}((\!|\mathfrak{L}^{\text{sgb}}(P, \mathfrak{M}(s)^{\text{sgb}})|\!)_D)$. In fact, the reordering of nodes of the CFG for block-reordering does not affect the variables of the cover program (there are new variables in the stegoprogram but, by design, these do not interfere with the variables of the cover program) and the same holds for the embedding algorithm of static graph-based, which guarantees that the added code does not affect the variables of the cover program. So, with the same input, the cover program and the stegoprogram give the same output, in both cases. Other features of these techniques are presented in Tables 1–3.

4.1.2. *Dynamic techniques.* These techniques need a notion of enabling input, namely a sequence $I = I_0 I_1 \dots I_k$ of input values which 'activates' the watermark. In our framework, we model the enabling input as a state property $\eta \in uco(\Sigma)$. In a dynamic technique, $\eta = \wp(\Upsilon) \cup \{\Sigma\}$, where Υ represents the set of states enabling the watermark, i.e.

$$\Upsilon \triangleq \left\{ \varsigma \in \Sigma \;\middle|\; \begin{array}{l} \varsigma = \langle C, \langle \rho, \imath \rangle\rangle \wedge |\imath| = |I| \wedge \\ \forall j \in [0, |I|) \,.\, \mathtt{top}(\mathtt{next}(\imath)^j) = I_j \end{array} \right\}.$$

The domain η can be defined as $\eta \triangleq \eta_\gamma \circ \eta_\alpha$ where $\eta_\alpha, \eta_\gamma \in \wp(\Sigma) \to \wp(\Sigma)$ are

$$\eta_\alpha \triangleq \lambda X \,.\, \begin{cases} X & \text{if } X \subseteq \Upsilon \\ \Sigma & \text{otherwise} \end{cases} \qquad \eta_\gamma \triangleq id.$$

The first dynamic technique, we present is dynamic graph-based watermarking. The signature, a natural number, is encoded by a graph allocated in the dynamic memory (in the *heap*), during a particular execution of the program. As said before, this execution is generated by a particular sequence of input values $I = I_0 I_1 \dots I_k$ called enabling sequence. Given a signature, a graph that encodes the signature and the code that builds this graph are generated. The embedder takes the program and it adds the code that generates the graph in some locations of the cover program and this code is executed only on the enabling input.

Let $\mathtt{graph} \in \mathbb{N} \to \mathbb{G}$ be a function that codifies a signature in a graph, as the one used for static techniques, and $\mathtt{heap} \in \mathbb{H} \to \wp(\mathbb{G})$ be a function that extracts the graphs memorized in an heap. For the formalization of this technique in the framework, we have to extend the information contained in the states of execution traces. We insert in the state the heap of the program at the current execution step, i.e. $\Sigma = \langle \mathtt{Com} \times \mathtt{Con} \times \mathbb{H} \rangle$. So $\sigma = \langle C, \zeta, \mathcal{H} \rangle$, where $\mathcal{H} \in \mathbb{H}$ is an heap. Let $\mathcal{G} \in \mathbb{G} \to \wp(\Sigma^+)$ the function

$$\mathcal{G} \triangleq \lambda g \,.\, \left\{ \sigma \in \Sigma^+ \;\middle|\; \begin{array}{l} |\sigma| = n + 1 \wedge \sigma_n = \langle C_n, \langle \rho_n, \imath_n \rangle, \mathcal{H}_n \rangle \wedge \\ \mathtt{top}(\imath_n) = \epsilon \wedge g \in \mathtt{heap}(\mathcal{H}_n) \wedge \mathtt{root}(g) \in \mathtt{heap}(\mathcal{H}_n) \wedge \\ \forall j \in [0, n) \,.\, \mathtt{root}(g) \notin \mathtt{heap}(\mathcal{H}_j) \wedge \mathtt{top}(\imath_n) = \epsilon \end{array} \right\}.$$

The semantics $(\!|P|\!)^\beta_+$ extracts the graph (with the root inserted at last) memorized in the heap of the program P when all the input values are consumed, so the domain β is

$$\beta \triangleq \{X \in \wp(\Sigma^+) | g \in \mathbb{G}, X = \mathcal{G}(g)\} \cup \{\varnothing, \Sigma^+\},$$

and it contains all the sets of traces that have the same graph (with the root inserted at last) memorized in the heap, when all the input values are consumed. With $\mathcal{W}_s \triangleq \mathcal{G}(\mathtt{graph}(s))$ we indicate the set of traces for which, when all the input values are consumed, the heap contains the encoding of the signature s. Clearly, $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+} = \mathcal{W}_s$ and so $\overline{\mathfrak{M}}(s) = \{\varnothing, \mathcal{W}_s, \Sigma^+\}$. Let again $\overline{\mathbb{G}} \triangleq \{\bot, \mathbb{G}\} \cup \{\mathbb{G}\}$. The domain β can be defined as $\beta \triangleq \beta_\gamma \circ \beta_\alpha$ where $\beta_\alpha \in \wp(\Sigma^+) \to \overline{\mathbb{G}}$ and $\beta_\gamma \in \overline{\mathbb{G}} \to \wp(\Sigma^+)$ are

$$\beta_\alpha \triangleq \lambda X\,.\begin{cases} \bot & \text{if } X = \varnothing \\ g \in \mathbb{G} & \text{if } \forall \sigma \in X\,.\ \begin{array}{l} |\sigma| = n+1 \wedge \sigma_n = \langle C_n, \langle \rho_n, \iota_n\rangle, \mathcal{H}_n\rangle \wedge \\ g \in \mathtt{heap}(\mathcal{H}_n) \wedge \mathtt{root}(g) \in \mathtt{heap}(\mathcal{H}_n) \wedge \\ \forall j \in [0,n)\,.\,\mathtt{root}(g) \notin \mathtt{heap}(\mathcal{H}_j) \wedge \mathtt{top}(\iota_n) = \epsilon \end{array} \\ \mathbb{G} & \text{otherwise} \end{cases}$$

$$\beta_\gamma \triangleq \lambda g\,.\begin{cases} \varnothing & \text{if } g = \bot \\ \mathcal{G}(g) & \text{if } g \in \mathbb{G} \\ \Sigma^+ & \text{otherwise} \end{cases}$$

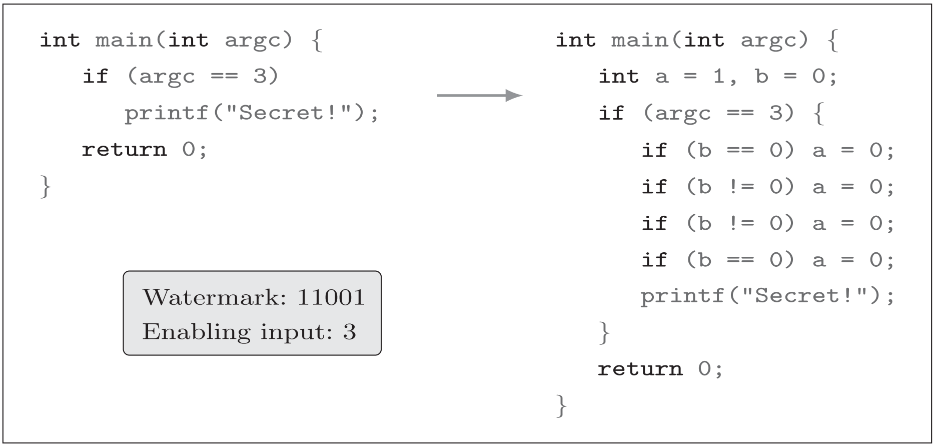
Instead $\overline{\mathfrak{M}}(s)$ can be defined as $\overline{\mathfrak{M}}(s) \triangleq \overline{\mathfrak{M}}(s)_\gamma \circ \overline{\mathfrak{M}}(s)_\alpha$ where $\overline{\mathfrak{M}}(s)_\alpha, \overline{\mathfrak{M}}(s)_\gamma \in \wp(\Sigma^+) \to \wp(\Sigma^+)$ are

$$\overline{\mathfrak{M}}(s)_\alpha \triangleq \lambda X\,.\begin{cases} \varnothing & \text{if } X = \varnothing \\ \mathcal{W}_s & \text{if } X \subseteq \mathcal{W}_s \\ \Sigma^+ & \text{otherwise} \end{cases} \qquad \overline{\mathfrak{M}}(s)_\gamma \triangleq \mathit{id}$$

It is simple to see that for any signature s we have $\overline{\mathfrak{M}}(s)(\wp(\Sigma^+)) \subseteq \beta(\wp(\Sigma^+))$ and so $\beta \sqsubseteq \overline{\mathfrak{M}}(s)$. If $X \in \eta(\Sigma)$ then $X \subseteq \Upsilon$. All these sets X contain states that encode the enabling input, so $\mathfrak{L}(P, \mathfrak{M}(s))$ executes the code which builds the graph $\mathtt{graph}(s)$ in the heap. Indeed, we have that $\{\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!\}^{\beta}_{+}(X) = \mathcal{W}_s$ for every possible set of initial states contained in X. Now, the same reasoning can be done for $\mathfrak{M}(s)$, because it codifies the signature by design (starting from the sets of input states which encode the enabling input). So $\{\!|\mathfrak{M}(s)|\!\}^{\beta}_{+}(X) = \mathcal{W}_s$ for every $X \in \eta(\Sigma)$. If $X \notin \eta(\Sigma)$ then $X \nsubseteq \Upsilon$. All these X do not contain states which encode the enabling input, so $\mathfrak{L}(P, \mathfrak{M}(s))$ does not execute the code which builds $\mathtt{graph}(s)$ in the heap. So, when the set of initial states X encodes the enabling input, we have that both $\{\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!\}^{\beta}_{+}(X)$ and $\{\!|\mathfrak{M}(s)|\!\}^{\beta}_{+}(X)$ are equal to $\mathcal{W}_s$, which represents the signature s. We can note that, as expected, for every signature s, we have that $\{\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!\}^{\beta}_{+}$ is $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)$.

Let us briefly discuss the features of this technique.

Resilience. The system is not resilient, because it is not fully immune to distortive attacks, i.e. $\mathtt{DenSem} \not\sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) | s \in \mathcal{S}\}$. In fact, suppose that $\mathtt{DenSem} \sqsubseteq \bigsqcap\overline{\mathfrak{M}}$, so $\forall X \in \wp(\Sigma^+)$ it holds that $\mathtt{DenSem}(X) \subseteq \bigsqcap\overline{\mathfrak{M}}(X)$. Let $X = \mathcal{W}_s$, for a generic signature s, so $\bigsqcap\overline{\mathfrak{M}}(X) = \mathcal{W}_s$. But $\mathcal{W}_s \subsetneq \mathtt{DenSem}(\mathcal{W}_s)$, because there is at least a trace with the same initial and final state of a trace in $\mathcal{W}_s$ without $\mathtt{graph}(s)$ among the objects memorized in the heap. For example, take a program equal to $\mathfrak{M}(s)$ in which the code that inserts the root node is duplicated.



```
int main(int argc) {
   if (argc == 3)
      printf("Secret!");
   return 0;
}
```

```
int main(int argc) {
   int a = 1, b = 0;
   if (argc == 3) {
      if (b == 0) a = 0;
      if (b != 0) a = 0;
      if (b != 0) a = 0;
      if (b == 0) a = 0;
      printf("Secret!");
   }
   return 0;
}
```

Fig. 5. Path-based watermarking.

Clearly, the traces of this program are in $\texttt{DenSem}(\mathcal{W}_s)$ but they are not in $\mathcal{W}_s$, because in this traces the last heap is not the only one containing the root of the graph. So there is a X such that $\texttt{DenSem}(X) \not\subseteq \sqcap\overline{\mathfrak{M}}(X)$ and hence $\texttt{DenSem} \not\sqsubseteq \sqcap\overline{\mathfrak{M}}$. Indeed, the system is vulnerable to attacks that modify the structure of the runtime objects created. For example, a node-splitting attack is able to damage the stegomark (if it modifies the structure of the root node of the graph then the extractor is not able to recognize the stegomark).

Secrecy. As regarding secrecy, the most powerful $\top$-secret attacker for dynamic graph-based watermarking is $\mathcal{K}_{\mathfrak{L},id,(\top)}(id) = \{X \in \wp(\Sigma^+) | P \in \mathsf{P}, X = \bigcup_{Q\in\mathsf{Q}} (\!|\mathfrak{L}(P,Q)|\!)_+\} \cup \{\Sigma^+\}$ and it abstracts in the same object the traces of all possible stegoprograms related to the same cover program.

Accuracy. Finally, the system is $\alpha^{out}_{\mathcal{O}}$ accurate. Indeed, the behaviour of the cover program is preserved *w.r.t.* this observation. Clearly, for every program P and for every signature s it holds that $\alpha^{out}_{\mathcal{O}}((\!|P|\!)_D) = \alpha^{out}_{\mathcal{O}}((\!|P_s|\!)_D)$. In fact the embedding algorithm guarantees that the inserted code does not affect the variables of the cover program. So, with the same input, the cover program and the stegoprogram give the same output. Other features of this technique are presented in Tables 1–3.

The second dynamic technique, we present is path-based watermarking. Let again $I = I_0 I_1 \ldots I_k$ be the enabling input, i.e. the sequence of input values which activates the watermark. The embedder takes the program and it adds bogus branches in a way that the sequence of choices at conditional statements during the execution on the enabling input is equal to the binary encoding of the signature (see Figure 5).

Let $\texttt{bin} \in \mathbb{N} \rightarrow \{0,1\}^\star$ be a function that returns the binary encoding of a natural number and $\texttt{branch} \in \Sigma^+ \rightarrow \{0,1\}^\star$ be a function that extracts the sequence of choices at conditional statements in a trace. For example, we could encode a 1 whenever the guard of an instruction is evaluated to tt, and a 0 whenever the guard is evaluated to ff. Let

$\mathcal{N} \in \mathbb{N} \to \wp(\Sigma^+)$ be the function

$$\mathcal{N} \triangleq \lambda k\,.\left\{\sigma \in \Sigma^+ \;\middle|\; \begin{array}{l} |\sigma| = n+1 \wedge \mathtt{branch}(\sigma) = \mathtt{bin}(k) \wedge \\ \sigma_n = \langle C, \langle \rho, \imath\rangle\rangle \wedge \mathtt{top}(\imath) = \epsilon \end{array}\right\}$$

The semantics $(\!|P|\!)^{\beta}_{+}$ has to extract the sequence of choices at conditional statements for the program P, so the domain β is $\beta \triangleq \{X \in \wp(\Sigma^+) | k \in \mathbb{N} \wedge X = \mathcal{N}(k)\} \cup \{\varnothing, \Sigma^+\}$ and it contains all the sets of traces that have done the same choices, when all the input values are consumed. With $\mathcal{W}_s \triangleq \mathcal{N}(s)$ we indicate the set of traces for which, when all the input values are consumed, the sequence of choices at conditional statements codify the signature s.

Clearly $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+} = \mathcal{W}_s$ and so $\overline{\mathfrak{M}}(s) = \{\varnothing, \mathcal{W}_s, \Sigma^+\}$. Let $\overline{\mathbb{N}} \triangleq \{\bot, \mathbb{N}\} \cup \mathbb{N}$. The domain $\overline{\mathfrak{M}}(s)$ can be defined as $\overline{\mathfrak{M}}(s) \triangleq \overline{\mathfrak{M}}(s)_{\gamma} \circ \overline{\mathfrak{M}}(s)_{\alpha}$ where $\overline{\mathfrak{M}}(s)_{\alpha}, \overline{\mathfrak{M}}(s)_{\gamma} \in \wp(\Sigma^+) \to \wp(\Sigma^+)$ are

$$\overline{\mathfrak{M}}(s)_{\alpha} \triangleq \lambda X\,.\begin{cases} \varnothing & \text{if } X = \varnothing \\ \mathcal{W}_s & \text{if } X \subseteq \mathcal{W}_s \\ \Sigma^+ & \text{otherwise} \end{cases} \qquad \overline{\mathfrak{M}}(s)_{\gamma} \triangleq \lambda id$$

Instead β can be defined as $\beta \triangleq \beta_{\gamma} \circ \beta_{\alpha}$ where $\beta_{\alpha} \in \wp(\Sigma^+) \to \overline{\mathbb{N}}$, $\beta_{\gamma} \in \overline{\mathbb{N}} \to \wp(\Sigma^+)$ are

$$\beta_{\alpha} \triangleq \lambda X\,.\begin{cases} \bot & \text{if } X = \varnothing \\ k \in \mathbb{N} & \text{if } \forall \sigma \in X \;:\; \begin{array}{l} |\sigma| = n+1 \wedge \\ \sigma_n = \langle C, \langle \rho, \imath\rangle\rangle \wedge \\ \mathtt{top}(\imath) = \epsilon \wedge \\ \mathtt{branch}(\sigma) = \mathtt{bin}(k) \end{array} \\ \mathbb{N} & \text{otherwise} \end{cases} \qquad \beta_{\gamma} \triangleq \lambda k\,.\begin{cases} \varnothing & \text{if } k = \bot \\ \mathcal{N}(k) & \text{if } k \in \mathbb{N} \\ \Sigma^+ & \text{otherwise} \end{cases}$$

It is simple to see that for all signatures s we have $\overline{\mathfrak{M}}(s)(\wp(\Sigma^+)) \subseteq \beta(\wp(\Sigma^+))$ and so $\beta \sqsubseteq \overline{\mathfrak{M}}(s)$. If $X \in \eta(\wp(\Sigma))$ then $X \subseteq \Upsilon$, therefore the choices at conditional statements made by $\mathfrak{L}(P, \mathfrak{M}(s))$ starting from states in X are equal to $\mathtt{bin}(s)$, i.e. $\{\!\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!\!\}^{\beta}_{+}(X) = \mathcal{W}_s$. The same reasoning can be done for $\mathfrak{M}(s)$, because it codifies the signature by design (starting from the sets of input states which encode the enabling input) and therefore $\{\!\!|\mathfrak{M}(s)|\!\!\}^{\beta}_{+}(X) = \mathcal{W}_s$ for every $X \in \eta(\wp(\Sigma))$. If $X \notin \eta(\Sigma)$ then $X \nsubseteq \Upsilon$. All these X do not contain states which encode the enabling input, so the choices at conditional statements made by $\mathfrak{L}(P, \mathfrak{M}(s))$ starting from states in X are not equal to $\mathtt{bin}(s)$. Hence, when the set of initial states X encodes the enabling input, we have that both $\{\!\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!\!\}^{\beta}_{+}(X)$ and $\{\!\!|\mathfrak{M}(s)|\!\!\}^{\beta}_{+}(X)$ are equal to $\mathcal{W}_s$, which represents the signature s. So, we can note that, as expected, for every signature s, $\{\!\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!\!\}^{\beta}_{+}$ is $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)$.

Let us briefly discuss the features of this technique.

Resilience. The system is not resilient since it is not immune to distortive attacks that preserve the denotational semantics, i.e. $\mathtt{DenSem} \not\sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) | s \in \mathcal{S}\}$. In fact, suppose that $\mathtt{DenSem} \sqsubseteq \bigsqcap \overline{\mathfrak{M}}$, so $\forall X \in \wp(\Sigma^+)$ it holds that $\mathtt{DenSem}(X) \subseteq \bigsqcap \overline{\mathfrak{M}}(X)$. Let $X = \mathcal{W}_s$, for a generic signature s, so $\bigsqcap \overline{\mathfrak{M}}(X) = \mathcal{W}_s$. But $\mathcal{W}_s \subsetneq \texttt{DenSem}(\mathcal{W}_s)$, because there is at least a trace with the same initial and final state of a trace in $\mathcal{W}_s$ with $\texttt{branch}(\sigma) \neq \texttt{bin}(s)$. For example, take a program equal to $\mathfrak{M}(s)$ in which we insert an opaque predicate. Clearly, its traces are in $\texttt{DenSem}(\mathcal{W}_s)$ but they are not in $\mathcal{W}_s$, because this traces have a different number of conditional statements. So, there is a X such that $\texttt{DenSem}(X) \not\subseteq \bigsqcap \overline{\mathfrak{M}}(X)$ and hence $\texttt{DenSem} \not\sqsubseteq \bigsqcap \overline{\mathfrak{M}}$. Indeed, the system is vulnerable to control flow obfuscation techniques. For example, both edge-flipping and opaque predicate insertion attacks are able to damage the stegomark.

Secrecy. As regarding secrecy, the most powerful $\top$-secret attacker for path-based watermarking is $\mathcal{K}_{\mathfrak{L},id,(\top)}(id) = \{X \in \wp(\Sigma^+) | P \in \mathsf{P}, X = \bigcup_{s \in \mathcal{S}} (\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!)_+\} \cup \{\Sigma^+\}$. It abstracts in the same object the traces of all possible stegoprograms related to the same cover program.

Accuracy. Finally, the system is $\alpha_{\mathcal{O}}^{out}$ accurate, where $\alpha_{\mathcal{O}}^{out}$ is an abstraction that observes only the output values showed to the user, since this property of denotational semantics is preserved. Clearly, for every program P and for every signature s it holds that $\alpha_{\mathcal{O}}^{out}((\!|P|\!)_D) = \alpha_{\mathcal{O}}^{out}((\!|P_s)|\!)_D)$. In fact the embedding algorithm guarantees that the insertion and modification of the conditional statements do not affect the variables of the cover program. Thus, for every input, the cover program and the stegoprogram return the same output to the user. Other features of this technique are presented in Tables 1–3.

4.1.3. *Abstract watermarking*. There is only one known abstract watermarking technique, namely abstract constant propagation watermarking (Cousot and Cousot 2004). The signature is a natural number and it is inserted in a particular variable that, although being modified during program execution, remains constant modulo an integer n. The signature is inserted ensuring no modifications to the semantics of the original program. This method allows us to insert a signature that is arbitrarily large, since its embedding in the cover program relies on the decomposition of the signature into k parts using the Chinese remainder theorem. In this case the inserted mark is given by the composition of k partial marks that encode numbers smaller than the limit imposed by the considered programming language. Without loss of generality, in the following, in order to keep the presentation simple, we assume that such limit does not exist and the signature is inserted without decomposition.

The embedder modifies the original program in the following way. First, a new variable `w` that hides the signature value s is declared. Next, two integer-valued polynomials `init` and `iter` are chosen for the initialization and the modification of the value of `w`. The instruction `w` := `init(1)` (initialization) is inserted in a random point of the program that is always executed. The instruction `w` := `iter(w)` (iteration) is inserted in a random point of the program (after initialization), possibly inside a loop. The polynomials must satisfy the following conditions[§]: $\texttt{init(1)} \equiv_n s$ and $\texttt{iter(w)} \equiv_n s$. Therefore, once initialized, `w` remains constant modulo n, even if its value changes in $\mathbb{Z}$ at each iteration.

[§] For the generation of such polynomials one can take advantage of the Horner method (Cousot and Cousot 2004).

For the formalization of this technique in the framework, we have to extend the information contained in the states of execution traces. So, assuming to have an enumeration of the programs in Imp, we insert in the state the identifier of the program which contains it. We denote with σ^i the identifier contained in the states of trace σ. Now, we can define the relation $\approx \subseteq \Sigma^+ \times \Sigma^+$ such that for all $\sigma, \overline{\sigma}$ we have that $\sigma \approx \overline{\sigma}$ *iff* $\sigma^i = \overline{\sigma}^i$. It is straightforward to note that $\approx$ is an equivalence relation. So, given a set of traces X we denote with $X/_{\approx}$ its quotient set, i.e. the set of its equivalence class induced by $\approx$. Let $\mathbb{Z}_n$ be the (quotient) ring of integers modulo n and $\overline{\mathbb{Z}}_n \triangleq \{\bot, \mathbb{Z}_n\} \cup \mathbb{Z}_n$. We assume to have a function $\mathtt{IsConst_n} \in \wp(\Sigma^+) \times \mathtt{Lab} \to (\mathtt{Var} \to \overline{\mathbb{Z}}_n)$ such that: given a set of traces X and a label l, $\mathtt{IsConst}_n(X,l)(y)$ returns the value of y modulo n if the variable is constant in $\mathbb{Z}_n$ into the set of traces X at label l. If the variable is undefined it returns $\bot$ and $\mathbb{Z}_n$ if the variable is not constant modulo n (this function implements a constant propagation analysis). Finally, $\mathtt{const}_n \in \wp(\Sigma^+) \to \wp(\mathbb{Z}_n)$ is defined as follows:

$$\mathtt{const}_n \triangleq \lambda X\,.\bigcup_{y\in\mathtt{Var}} \{\mathtt{IsConst}_n(X,l)(y) | \exists l \in \mathtt{Lab}\,.\,\mathtt{IsConst}_n(X,l)(y) \in \mathbb{Z}_n\}\,.$$

In short, this function returns all the values in $\mathbb{Z}_n$ of the variables that are constant modulo n into a given set of traces (at some label). The semantics $(\!|P|\!)^{\beta}_{+}$ performs constant propagation modulo n for the program P, so

$$\beta = \left\{ X \in \wp(\Sigma^+) \;\middle|\; \begin{array}{l} \exists N \subseteq \mathbb{Z}_n\,.\,N \neq \varnothing \wedge \\ X = \bigcup_{id\in\mathbb{N}} \left\{ Y \in \wp(\Sigma^+) \;\middle|\; \begin{array}{l} \forall \sigma \in Y\,.\,\sigma^i = id \wedge \\ \mathtt{const}_n(Y) = N \end{array} \right\} \end{array} \right\} \cup \{\varnothing, \Sigma^+\}$$

The domain β contains all the sets of traces with the same values of the variables constant modulo n. With $\mathcal{W}_s \triangleq \bigcup_{id\in\mathbb{N}} \{X \in \wp(\Sigma^+) | \forall \sigma \in X\,.\,\sigma^i = id \wedge \mathtt{const}_n(X) = \{s \bmod n\}\}$ we indicate the set of traces that have a constant variable encoding of the signature s. Clearly, $(\!|\mathfrak{M}(s)|\!)^{\beta}_{+} = \mathcal{W}_s$ and so $\overline{\mathfrak{M}}(s) = \{\varnothing, \mathcal{W}_s, \Sigma^+\}$. The domain β can be defined as $\beta \triangleq \beta_\gamma \circ \beta_\alpha$, where $\beta_\alpha \in \wp(\Sigma^+) \to \wp(\mathbb{Z}_n)$ and $\beta_\gamma \in \wp(\mathbb{Z}_n) \to \wp(\Sigma^+)$ are

$$\beta_\alpha \triangleq \lambda X\,.\begin{cases} \varnothing & \text{if } X = \varnothing \\ N & \text{if } N \subsetneq \mathbb{Z}_n \wedge N \neq \varnothing \wedge \forall X_j \in X/_{\approx}\,.\,\mathtt{const}_n(X_j) = N \\ \mathbb{Z}_n & \text{otherwise} \end{cases}$$

$$\beta_\gamma \triangleq \lambda N\,.\begin{cases} \varnothing & \text{if } N = \varnothing \\ \bigcup_{id\in\mathbb{N}} \left\{ X \in \wp(\Sigma^+) \;\middle|\; \begin{array}{l} \forall \sigma \in X\,.\,\sigma^i = id \wedge \\ \mathtt{const}_n(X) = N \end{array} \right\} & \text{if } N \notin \{\varnothing, \mathbb{Z}_n\} \\ \Sigma^+ & \text{otherwise} \end{cases}$$

Instead $\overline{\mathfrak{M}}(s)$ can be defined as $\overline{\mathfrak{M}}(s) \triangleq \overline{\mathfrak{M}}(s)_\gamma \circ \overline{\mathfrak{M}}(s)_\alpha$ where $\overline{\mathfrak{M}}(s)_\alpha, \overline{\mathfrak{M}}(s)_\gamma \in \wp(\Sigma^+) \to \wp(\Sigma^+)$ are

$$\overline{\mathfrak{M}}(s)_\alpha \triangleq \lambda X\,.\begin{cases} \varnothing & \text{if } X = \varnothing \\ \mathcal{W}_s & \text{if } X \subseteq \mathcal{W}_s \\ \Sigma^+ & \text{otherwise} \end{cases} \qquad \overline{\mathfrak{M}}(s)_\gamma \triangleq id$$

It is simple to see that for all signatures s we have $\overline{\mathfrak{M}}(s)(\wp(\Sigma^+)) \subseteq \beta(\wp(\Sigma^+))$ and so $\beta \sqsubseteq \overline{\mathfrak{M}}(s)$. The input domain is *id* so there is no an enabling input, or equivalently, all the inputs reveal the watermark. For every possible set of initial states, the constant propagation modulo n of $\mathfrak{L}(P, \mathfrak{M}(s))$ is the same, i.e. it exists $N \subsetneq \mathbb{Z}_n$ not empty such that $\texttt{const}_n(\llparenthesis \mathfrak{L}(P, \mathfrak{M}(s)) \rrparenthesis_+) = N$. Due to the definition of this watermarking technique, N is the set $[s]_{\equiv_n}$, namely it contains all the values that are equivalent modulo n to s. So, we have that $\{\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!\}^{\beta}_{+}(X) = \mathcal{W}_s$ for every possible set of initial states. Now, the constant propagation modulo n of $\mathfrak{M}(s)$ is exactly $[s]_{\equiv_n}$, so $\{\!|\mathfrak{M}(s)|\!\}^{\beta}_{+}(X) = \mathcal{W}_s$ for every possible set of initial states. Hence, for every set of initial states X, we have that both $\{\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!\}^{\beta}_{+}(X)$ and $\{\!|\mathfrak{M}(s)|\!\}^{\beta}_{+}(X)$ are equal to $\mathcal{W}_s$, which represents the signature s. So we can note that, as expected, for every signature s, $\{\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!\}^{\beta}_{+}$ is $\mathcal{F}$-complete for η and $\overline{\mathfrak{M}}(s)$.

Let us briefly discuss the features of this technique.

Resilience. The system is not resilient, since it is not immune to distortive attacks that preserve the denotational semantics, i.e. $\texttt{DenSem} \not\sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) | s \in \mathcal{S}\}$. In fact, suppose that $\texttt{DenSem} \sqsubseteq \bigsqcap \overline{\mathfrak{M}}$, so $\forall X \in \wp(\Sigma^+)$ it holds that $\texttt{DenSem}(X) \subseteq \bigsqcap \overline{\mathfrak{M}}(X)$. Let $X = \mathcal{W}_s$, for a generic signature s, so $\bigsqcap \overline{\mathfrak{M}}(X) = \mathcal{W}_s$. But $\mathcal{W}_s \subsetneq \texttt{DenSem}(\mathcal{W}_s)$, because there is at least a trace with the same initial and final state of a trace in $\mathcal{W}_s$ which belongs to a program that does not have a constant variable modulo n equal to s. So there is a X such that $\texttt{DenSem}(X) \not\subseteq \bigsqcap \overline{\mathfrak{M}}(X)$ and hence $\texttt{DenSem} \not\sqsubseteq \bigsqcap \overline{\mathfrak{M}}$. Indeed the system is vulnerable to data obfuscation techniques: if we alter the representation of variables then the stegomark is lost.

Secrecy. The most powerful $\top$-secret attacker for abstract constant propagation watermarking is $\mathcal{K}_{\mathfrak{L},id,(\top)}(id) = \{X \in \wp(\Sigma^+) | P \in \mathsf{P}, X = \bigcup_{Q \in \mathsf{Q}} \llparenthesis \mathfrak{L}(P, Q) \rrparenthesis_+\} \cup \{\Sigma^+\}$. It abstracts in the same object the traces of all possible stegoprograms related to the same cover program.

Accuracy. Finally, the system is $\alpha^{out}_{\mathcal{O}}$ accurate, where $\alpha^{out}_{\mathcal{O}}$ is an abstraction that observes only the output values showed to the user, since this property of the denotational semantics is preserved. In fact, the embedding algorithm guarantees that the added code do not affect the variables of the cover program. So, with the same input, the cover program and the stegoprogram give the same output. Other features of this technique are presented in Tables 1–3.

4.2. *Comparison: resilience*

Until now we have formalized and validated the framework. The next step is clearly to show how to use the work done so far for comparing different software watermarking systems, in order to be able to chose a technique rather than another according to the needs. In Section 3, we have discussed how different watermarking systems $\mathfrak{A}_1 = \langle \mathfrak{L}_1, \mathfrak{M}_1, \beta_1 \rangle$ and $\mathfrak{A}_2 = \langle \mathfrak{L}_2, \mathfrak{M}_2, \beta_2 \rangle$ could be compared *w.r.t.* resilience by comparing the degree of abstraction of $\bigsqcap\{\overline{\mathfrak{M}_1}(s) | s \in \mathcal{S}\}$ and $\bigsqcap\{\overline{\mathfrak{M}_2}(s) | s \in \mathcal{S}\}$. Of course it may happen that these two abstractions are not comparable. In this case, what we can do is to compare their resilience *w.r.t.* a specific distortive attack.

Distortive attacks are semantics-preserving program transformations that try to compromise the extraction of the watermark. There are program transformations specifically designed for this purpose like code obfuscation techniques, and others that may corrupt the watermark as a side effect, like code optimization. Given the semantics-based formalization of software watermarking and its features, we model distortive attacks as semantic program transformations as done in Dalla Preda and Giacobazzi (2005), considering that their syntactic counterpart can always be derived (Cousot and Cousot 2002). Indeed, any syntactic program transformer $\mathfrak{t}$, altering the code of P and returning a new program P', induces a corresponding semantic transformer $\overline{\mathfrak{t}}$ turning $(\!|P|\!)_+$ into $(\!|P'|\!)_+$. In the following, we consider some well known obfuscations and optimization techniques that can be used as distortive attacks.

4.2.1. *Edge-flipping.* The edge-flipping obfuscation exchanges the code executed in the true branch with the code executed in the false branch of every conditional statement. In order to preserve the program semantics every branch condition is replaced with its negation.

We semantically model the edge-flipping attacker $\mathfrak{t}^{\mathfrak{ef}} \in \textsc{Imp} \to \textsc{Imp}$ as the semantic transformation $\overline{\mathfrak{t}^{\mathfrak{ef}}}(X) \triangleq \{\overline{\mathfrak{t}^{\mathfrak{ef}}}(\sigma) | \sigma \in X\}$ with

$$\overline{\mathfrak{t}^{\mathfrak{ef}}}(\langle C, \zeta\rangle \sigma) \triangleq \begin{cases} \langle C, \zeta\rangle \overline{\mathfrak{t}^{\mathfrak{ef}}}(\sigma) & \text{if } \begin{array}{l} C = L : \texttt{stop}; \vee \\ C = L : A \to L'; \end{array} \\ \langle L : \neg B \to \{L_F, L_T\};, \zeta\rangle \overline{\mathfrak{t}^{\mathfrak{ef}}}(\sigma) & \text{if } C = L : B \to \{L_T, L_F\}. \end{cases}$$

The most concrete preserved property is

$$\delta_{\mathfrak{t}^{\mathfrak{ef}}} = \bigsqcup_{P \in \textsc{Imp}} \{X \subseteq \Sigma^+ | \texttt{Pres}_{P, \overline{\mathfrak{t}^{\mathfrak{ef}}}}(X)\},$$

where $\texttt{Pres}_{P, \overline{\mathfrak{t}^{\mathfrak{ef}}}}(X)$ if and only if

$$\forall Y \subseteq (\!|P|\!)_+ . \, Y \subseteq X \Rightarrow \bigcup \{Z \subseteq \Sigma^+ | Z = \overline{\mathfrak{t}^{\mathfrak{ef}}}(Y)\} \subseteq X.$$

This means that a set of traces X is preserved by the edge-flipping transformation if it contains all the traces that can be obtained from traces in X by inverting every conditional branch and negating the related guard.

4.2.2. *Dead code elimination.* This is an optimization technique and it consists in the elimination of those parts of the program that are never executed or which do not affect the semantics of the program. Let $\mathcal{I} \in \textsc{Imp} \to \wp(\texttt{Lab})$ be the result of a preliminary static analysis that returns the subset of program labels corresponding to dead code commands. Usually, the preliminary static analysis consists of dead/faint variable analysis. Given a program P, we assume to know the set $\mathcal{I}^P \subseteq {}^{lab}[\![P]\!]$ of labels that the preliminary static analysis has classified as dead code. We assume that conditional commands cannot be classified as dead code. We denote with $\texttt{slab}(\varsigma)$ the label of the command contained in the state ς, i.e. if $\varsigma = \langle C, \zeta\rangle$ then $\texttt{slab}(\varsigma) = {}^{lab}[\![C]\!]$.

We semantically model the dead code elimination attacker $\mathfrak{t}^{\mathfrak{dce}} \in \textsc{Imp} \rightarrow \textsc{Imp}$ as the semantic transformation $\overline{\mathfrak{t}^{\mathfrak{dce}}} \in \wp(\Sigma^+) \times \wp(\mathtt{Lab}) \rightarrow \wp(\Sigma^+)$ with

$$\overline{\mathfrak{t}^{\mathfrak{dce}}}(X,\mathcal{I}^P) \triangleq \{\overline{\mathfrak{t}^{\mathfrak{dce}}}(\sigma,\mathcal{I}^P)|\sigma \in X\}$$

$$\overline{\mathfrak{t}^{\mathfrak{dce}}}(\sigma,\mathcal{I}^P) \triangleq \textsc{Elimination}(\sigma,\mathcal{I}^P)$$

See Appendix B for the algorithm Elimination, Algorithm 1 - Page 383. The most concrete preserved property is

$$\delta_{\mathfrak{t}^{\mathfrak{dce}}} = \bigsqcup_{P \in \textsc{Imp}} \left\{ X \subseteq \Sigma^+ \middle| \mathtt{Pres}_{P,\overline{\mathfrak{t}^{\mathfrak{dce}}}}(X) \right\},$$

where $\mathtt{Pres}_{P,\overline{\mathfrak{t}^{\mathfrak{dce}}}}(X)$ if and only if

$$\forall Y \subseteq (\!|P|\!)_+ . Y \subseteq X \Rightarrow \bigcup \left\{ Z \subseteq \Sigma^+ \middle| Z = \overline{\mathfrak{t}^{\mathfrak{dce}}}(Y,\mathcal{I}^P) \right\} \subseteq X.$$

This means that a set of traces X is preserved by the dead code elimination transformation if it contains all the traces that can be obtained from traces in X by eliminating every command indicated by $\mathcal{I}^P$.

4.2.3. *Loop-unrolling.* This is an optimization technique but it is often used in distortive attacks. It consists in the unfolding of the loop a certain number of times, i.e. the body of the loop is replicated for a given factor, the so called unrolling factor. Next, the number of iterations is divided by the loop unrolling factor. The easiest looping constructs to unroll are for-loops. Whenever a program P includes a for-loop F, we write $F \in \mathtt{fors}(P)$. More formally, $F \in \mathtt{fors}(P)$ *iff* $F \triangleq \{G,I\} \cup H$ and $F \subseteq P$. The command $G \triangleq l^G : X < E \rightarrow \{l^H, l^O\};$, with $l^G \neq l^H$ and $l^G \neq l^O$, implements a branching named *guard*. As F always starts with the evaluation of its guard, we have that l^G is the entrypoint of F, ${}^{lab}[\![F]\!] \cap {}^{lab}[\![P \setminus F]\!] = \varnothing$ and ${}^{suc}[\![P \setminus F]\!] \cap {}^{lab}[\![F]\!] \subseteq \{l^G\}$. The guard is satisfied as long as $X \in \mathtt{Var}$ is less than $E \in \mathtt{Exp}$ (for short, we ignore similar kinds of for-loops which use $>$, $\leqslant$ or $\geqslant$ as comparison operator). If the guard is not satisfied, the for-loop ends transferring the control flow at entrypoint $l^O \notin {}^{lab}[\![F]\!]$. Otherwise, the execution goes on through H, a set of commands named *body*, and eventually through an increment command $I \triangleq l^I : X := X + E' \rightarrow l^G;$, with $l^I \neq l^G$ and $l^I = l^H \vee l^I \in {}^{suc}[\![H]\!]$ (notice that I makes the control flow return to the guard again). We formally define H as the collection of all the commands of P that are reachable from G without going through I, i.e. $H \triangleq \mathit{lfp}^{\subseteq}\mathtt{flow}(P)$, where $\mathtt{flow}(P)(Q) \triangleq \{C \in P \setminus \{I\} | {}^{lab}[\![C]\!] = {}^{suc}[\![G]\!] \vee \exists C' \in Q \,.\, {}^{lab}[\![C]\!] = {}^{suc}[\![C']\!]\}$. We require $l^G, l^I \notin {}^{lab}[\![H]\!]$. We expect both X and the variables in E and E' not to be assigned inside H. We require X not to be used in E or E'.

The finite partial trace $\langle G,\zeta\rangle\eta\langle I,\zeta'\rangle \in (\!|F|\!)_{\oplus}$ is an iteration of the for-loop F, where $\eta \in (\!|H|\!)_{\oplus}$ (if $H = \varnothing$ then $\eta = \epsilon$). A maximal trace $\sigma \in (\!|F|\!)_{\oplus}$ is a sequence of 'terminating' iterations followed by a state with the command at label l^O. Along the trace, the values of E and E' do not change, while the value of X, which is constant throughout each iteration, increases by E' from one iteration to another. Thus, if ζ is a context in a state of $\sigma \in (\!|F|\!)_{\oplus}$, we can predict how many increments X still has to undergo, i.e. the number of the iterations from ζ till the end of σ (actually we only need the environment ρ contained in ζ). We just need to define $\alpha_F \in {}^{con}[\![F]\!] \to \mathbb{N}$ such that

$$\alpha_F(\zeta) \triangleq \begin{cases} \left\lfloor \frac{({}^{\mathcal{E}}[\![E]\!]\zeta - {}^{\mathcal{E}}[\![X]\!]\zeta)+({}^{\mathcal{E}}[\![E']\!]\zeta - {}^{\mathcal{E}}[\![X]\!]\zeta)}{{}^{\mathcal{E}}[\![E']\!]\zeta} \right\rfloor & \text{if } {}^{\mathcal{E}}[\![E]\!]\zeta \geqslant {}^{\mathcal{E}}[\![X]\!]\zeta \\ 0 & \text{otherwise} \end{cases}$$

We let τ be the total number of iterations of σ. Along $\sigma \in (\!|F|\!)_{\oplus}$, the iterations are naturally unfolded, i.e. they come sequentially one after another. In the original program F they fold because any command $C \in F$, although occurring in many different iterations, always appears with the same entrypoint ${}^{lab}[\![C]\!]$.

Loop-unrolling changes the labels in the following way: given the so-called unrolling factor $u \in \mathbb{N}$, it makes all and only the occurrences of C at iterations $k \texttt{ mod } u$ have the same label (with $0 \leqslant k < \tau$), thus partitioning the iterations of σ into u classes. Only iterations from the same class fold together. So, the code of the unrolled loop is u times longer than F and each of its iterations sequentially executes the task of u native iterations. In this work, we consider only the case in which the total number of iterations is known (it is constant), and so we can set $u = \tau$. This is also the standard optimizing behaviour of most compilers, like `gcc`. So, we assume that $\texttt{fors}(P)$ contains only the for-loops that have a constant number of iterations (we need a preliminary analysis of program P) and that $\texttt{iters}(F)$ returns the number of iterations of $F \in \texttt{fors}(P)$. Let $\mathcal{I} \in \textsc{Imp} \to \wp(\texttt{Lab} \times \texttt{Lab})$ be the result of a preliminary static analysis that given a program returns the for-loops that can be unrolled. It represents the loop by mean of a pair of labels which identify the guard and the increment of the loop. Given a program P, we assume to know the set $\mathcal{I}^P \subseteq {}^{lab}[\![P]\!] \times {}^{lab}[\![P]\!]$ of pairs of labels that the preliminary static analysis has classified as representative of loops that can be unrolled.

We semantically model the loop unrolling attacker $\mathfrak{t}^{\mathfrak{lu}} \in \textsc{Imp} \to \textsc{Imp}$ as the semantic transformation $\overline{\mathfrak{t}^{\mathfrak{lu}}} \in \wp(\Sigma^+) \times \wp(\texttt{Lab} \times \texttt{Lab}) \to \wp(\Sigma^+)$ with

$$\overline{\mathfrak{t}^{\mathfrak{lu}}}(X, \mathcal{I}^P) \triangleq \{\overline{\mathfrak{t}^{\mathfrak{lu}}}(\sigma, \mathcal{I}^P) | \sigma \in X\}$$
$$\overline{\mathfrak{t}^{\mathfrak{lu}}}(\sigma, \mathcal{I}^P) \triangleq \textsc{Unroll}(\sigma, \mathcal{I}^P)$$

See Appendix B for the algorithm Unroll – Algorithm 2, Page 384. The most concrete preserved property is

$$\delta_{\mathfrak{t}^{\mathfrak{lu}}} = \bigsqcup_{P \in \textsc{Imp}} \left\{ X \subseteq \Sigma^+ \;\middle|\; \texttt{Pres}_{P,\overline{\mathfrak{t}^{\mathfrak{lu}}}}(X) \right\},$$

where $\texttt{Pres}_{P,\overline{\mathfrak{t}^{\mathfrak{lu}}}}(X)$ if and only if

$$\forall Y \subseteq (\!|P|\!)_{+} . \, Y \subseteq X \Rightarrow \bigcup \left\{ Z \subseteq \Sigma^+ \;\middle|\; Z = \overline{\mathfrak{t}^{\mathfrak{lu}}}(Y, \mathcal{I}^P) \right\} \subseteq X.$$

This means that a set of traces X is preserved by the loop-unrolling transformation if it contains all the traces that can be obtained from traces in X by substituting the loops at program points indicated by $\mathcal{I}^P$ with their sequences of iterations.

4.2.4. *Opaque predicate insertion.* It is a well known obfuscation technique that obfuscates the control flow of the program by inserting opaque predicates. A predicate is opaque if it is evaluated to a constant value. Let $\mathcal{I} \in \textsc{Imp} \to \wp(\texttt{Lab})$ be the result of a preliminary static analysis that returns the set of program labels where it is possible to insert opaque predicates. Usually, the preliminary static analysis consists of a liveness analysis. Given a program P, we assume to know the set $\mathcal{I}^P \subseteq {}^{lab}[\![P]\!]$ of labels that the preliminary static analysis has classified as candidates for opaque predicate insertion. Let P^{t} be a true opaque predicate, i.e. a Boolean expression that is always evaluated to tt, let $\hat{L}$ be a label not in P, i.e. $\hat{L} \notin {}^{lab}[\![P]\!]$, and let $\tilde{L}$ be a random label of P. Again, we denote with $\mathtt{slab}(\varsigma)$ the label of the command contained in the state ς, i.e. if $\varsigma = \langle C, \zeta \rangle$ then $\mathtt{slab}(\varsigma) = {}^{lab}[\![C]\!]$.

We semantically model the opaque predicate insertion attacker $\mathfrak{t}^{\mathfrak{opi}} \in \textsc{Imp} \rightarrow \textsc{Imp}$ as the semantic transformation $\overline{\mathfrak{t}^{\mathfrak{opi}}} \in \wp(\Sigma^+) \times \wp(\mathtt{Lab}) \rightarrow \wp(\Sigma^+)$ with $\overline{\mathfrak{t}^{\mathfrak{opi}}}(X, \mathcal{I}^P) \triangleq \{\overline{\mathfrak{t}^{\mathfrak{opi}}}(\sigma, \mathcal{I}^P) | \sigma \in X\}$, where $\overline{\mathfrak{t}^{\mathfrak{opi}}}(\langle C, \zeta \rangle \sigma, \mathcal{I}^P) \triangleq$

$$\begin{cases} \langle C, \zeta \rangle \overline{\mathfrak{t}^{\mathfrak{opi}}}(\sigma, \mathcal{I}^P) & \text{if } \mathtt{slab}(\langle C, \zeta \rangle) \notin \mathcal{I}^P \\ \langle L : \mathrm{P}^{\mathrm{t}} \rightarrow \{\hat{L}, \tilde{L}\};, \zeta \rangle \langle \hat{L} : A \rightarrow L';, \zeta \rangle \overline{\mathfrak{t}^{\mathfrak{opi}}}(\sigma, \mathcal{I}^P) & \text{if } \begin{array}{l} \mathtt{slab}(\langle C, \zeta \rangle) \in \mathcal{I}^P \wedge \\ C = L : A \rightarrow L'; \end{array} \\ \langle L : \mathrm{P}^{\mathrm{t}} \rightarrow \{\hat{L}, \tilde{L}\};, \zeta \rangle \langle \hat{L} : B \rightarrow \{L_T, L_F\};, \zeta \rangle \overline{\mathfrak{t}^{\mathfrak{opi}}}(\sigma, \mathcal{I}^P) & \text{if } \begin{array}{l} \mathtt{slab}(\langle C, \zeta \rangle) \in \mathcal{I}^P \wedge \\ C = L : B \rightarrow \{L_T, L_F\}; \end{array} \\ \langle L : \mathrm{P}^{\mathrm{t}} \rightarrow \{\hat{L}, \tilde{L}\};, \zeta \rangle \langle \hat{L} : \mathtt{stop};, \zeta \rangle \overline{\mathfrak{t}^{\mathfrak{opi}}}(\sigma, \mathcal{I}^P) & \text{if } \begin{array}{l} \mathtt{slab}(\langle C, \zeta \rangle) \in \mathcal{I}^P \wedge \\ C = L : \mathtt{stop}; \end{array} \end{cases}$$

The most concrete preserved property is

$$\delta_{\mathfrak{t}^{\mathfrak{opi}}} = \bigsqcup_{P \in \textsc{Imp}} \left\{ X \subseteq \Sigma^+ \middle| \mathtt{Pres}_{P, \overline{\mathfrak{t}^{\mathfrak{opi}}}}(X) \right\}$$

where $\mathtt{Pres}_{P, \overline{\mathfrak{t}^{\mathfrak{opi}}}}(X)$ if and only if

$$\forall Y \subseteq (\!|P|\!)_+ . \, Y \subseteq X \Rightarrow \bigcup \left\{ Z \subseteq \Sigma^+ \middle| Z = \overline{\mathfrak{t}^{\mathfrak{opi}}}(Y, \mathcal{I}^P) \right\} \subseteq X$$

This means that a set of traces X is preserved by the opaque predicate insertion transformation if it contains all the traces that can be obtained from traces in X by inserting the opaque predicate P^{t} at program points indicated by $\mathcal{I}^P$.

4.2.5. *Loop-invariant code motion.* This is an optimization technique and it consists in moving outside the loops the code which is loop-invariant, i.e. those instructions that can be moved outside the body of a loop without affecting the semantics of the program. In this context, we assume that only assignments can be moved outside loops. Let $\mathcal{I} : \textsc{Imp} \rightarrow \wp(\mathtt{Lab})$ be the result of a preliminary static analysis that returns the set of program labels of those commands that can be moved without affecting the behaviour of the program. Usually, the preliminary static analysis consists of a reaching definitions analysis. Given a program P, we assume to know the set $\mathcal{I}^P \subseteq {}^{lab}[\![P]\!]$ of labels that the preliminary static analysis has classified as loop-invariant. For every $l \in \mathcal{I}^P$ it is trivial to retrieve the for-loop $F \in \mathtt{fors}(P)$ (function defined formally at Page 372) that contains the command at label l. This can be done observing a program trace. So, we assume to have a function $\mathtt{entry}$ that retrieves the label of the loop's guard related to the loop which contains the command with label l, namely $l^G = \mathtt{entry}(l)$ *iff* $F = \{G, I\} \cup H$, $l \in {}^{lab}[\![H \cup I]\!]$ and $l^G = {}^{lab}[\![G]\!]$. We assume also to have a function $\mathtt{exit}$ that retrieves the label of the command just next to the loop which contains the command with label l, namely $l^O = \texttt{exit}(l)$ *iff* $F = \{G, I\} \cup H$, $l \in {}^{lab}[\![H \cup I]\!]$ and $G = l^G : B \to \{l^H, l^O\}$. Let $\hat{L}$ be a label not in P, i.e. $\hat{L} \notin {}^{lab}[\![P]\!]$. Again we denote with $\texttt{slab}(\varsigma)$ the label of the command contained in the state ς, i.e. if $\varsigma = \langle C, \zeta \rangle$ then $\texttt{slab}(\varsigma) = {}^{lab}[\![C]\!]$.

We semantically model the loop-invariant code motion attacker $\mathfrak{t}^{\mathfrak{licm}} \in \textsc{Imp} \to \textsc{Imp}$ as the semantic transformation $\overline{\mathfrak{t}^{\mathfrak{licm}}} \in \wp(\Sigma^+) \times \wp(\texttt{Lab}) \to \wp(\Sigma^+)$ with

$$\overline{\mathfrak{t}^{\mathfrak{licm}}}(X, \mathcal{I}^P) \triangleq \{\overline{\mathfrak{t}^{\mathfrak{licm}}}(\sigma, \mathcal{I}^P) | \sigma \in X\}$$
$$\overline{\mathfrak{t}^{\mathfrak{licm}}}(\sigma, \mathcal{I}^P) \triangleq \textsc{Motion}(\sigma, \mathcal{I}^P).$$

See Appendix B for the algorithm Motion – Algorithm 3, Page 385. The most concrete preserved property is

$$\delta_{\mathfrak{t}^{\mathfrak{licm}}} = \bigsqcup_{P \in \textsc{Imp}} \left\{ X \subseteq \Sigma^+ \,\middle|\, \texttt{Pres}_{P, \overline{\mathfrak{t}^{\mathfrak{licm}}}}(X), \right\}$$

where $\texttt{Pres}_{P, \overline{\mathfrak{t}^{\mathfrak{licm}}}}(X)$ if and only if

$$\forall Y \subseteq (\!|P|\!)_+ . \, Y \subseteq X \Rightarrow \bigcup \left\{ Z \subseteq \Sigma^+ \,\middle|\, Z = \overline{\mathfrak{t}^{\mathfrak{licm}}}(Y, \mathcal{I}^P) \right\} \subseteq X.$$

This means that a set of traces X is preserved by the loop-invariant code motion transformation if it contains all the traces that can be obtained from traces in X by moving each command, at program points indicated by $\mathcal{I}^P$, just before (outside) the loop that contains it.

4.2.6. *Data-obfuscation.* This is a class of obfuscation techniques that obscure the data structures used by programs. These transformations convert the representation of variables to a representation that is harder to analyze: variables are encoded in an unnatural way (Collberg *et al.* 1997), either changing variables types or modifying their values. Hence, these transformations affect how data is stored and the methods used to interpret stored data. So, there is an *data-encoding* function $\texttt{Enc} \in T \to T'$ that gives the new representation of a variable, and a *data-decoding* function $\texttt{Dec} \in T' \to T$ that gives back the original representation. Of course, operations on variables need to be changed too. Hence, for every operation $\mathsf{op} \in T \times T \to T$ we need a new operation $\mathsf{op}' \in T' \times T' \to T'$. For example, a simple data-encoding for integers variables is the one that increments values by one. In our language, the data-encoding of a value is $\texttt{Enc}(n) = n + 1$ and the data-decoding is $\texttt{Dec}(n) = n - 1$. The new arithmetic operations are as follows:

$$E_1 +' E_2 = \texttt{Enc}(E_1) + \texttt{Enc}(E_2) - 1 \qquad E_1 -' E_2 = \texttt{Enc}(E_1) - \texttt{Enc}(E_2)$$
$$E_1 \cdot' E_2 = \texttt{Enc}(E_1) \cdot \texttt{Enc}(E_2) - \texttt{Enc}(E_1) - \texttt{Enc}(E_2) + 2 \qquad \text{with } \texttt{Enc}(X) = X.$$

The actions which need modifications are only assignments and inputs. For the first, we need to replace $X := E$ with $X := \texttt{Enc}(E)$, for the second we need to add after an input $\texttt{input}\, X$ an assignment $X := X + 1$. We semantically model the data-obfuscation attacker $\mathfrak{t}^{\mathfrak{do}} \in \textsc{Imp} \to \textsc{Imp}$ as the semantic transformation $\overline{\mathfrak{t}^{\mathfrak{do}}} \in \wp(\Sigma^+) \to \wp(\Sigma^+)$ with

$$\overline{\mathfrak{t}^{\mathfrak{do}}}(X) \triangleq \{\overline{\mathfrak{t}^{\mathfrak{do}}}(\sigma) | \sigma \in X\}$$
$$\overline{\mathfrak{t}^{\mathfrak{do}}}(\sigma) \triangleq \textsc{Enc-Dec}(\sigma)$$

See Appendix B for the algorithm Enc-Dec – Algorithm 4, Page 386. The most concrete preserved property is

$$\delta_{\mathfrak{t}\mathfrak{do}} = \bigsqcup_{P \in \textsc{Imp}} \{X \subseteq \Sigma^{+} | \mathtt{Pres}_{P,\overline{\mathfrak{t}\mathfrak{do}}}(X)\},$$

where $\mathtt{Pres}_{P,\overline{\mathfrak{t}\mathfrak{do}}}(X)$ if and only if

$$\forall Y \subseteq (\!|P|\!)_{+} \,.\, Y \subseteq X \Rightarrow \bigcup \{Z \subseteq \Sigma^{+} | Z = \overline{\mathfrak{t}^{\mathfrak{do}}}(Y)\} \subseteq X.$$

This means that a set of traces X is preserved by the data-obfuscation transformation if it contains all the traces that can be obtained from traces in X by data-encoding/data-decoding all variables with the functions `Enc` and `Dec`.

4.2.7. *Comparison results.* The formalization of both the watermarking techniques and the distortive attacks in the semantic setting has allowed us to formally prove the resilience of the considered watermarking systems, *w.r.t.* the distortive attacks described above.

Proposition 4.1. Static graph-based watermarking is $\mathfrak{t}^{\mathfrak{opi}}$-ineffective. This means that $\forall s \in \mathcal{S} \,.\, \delta_{\mathfrak{t}\mathfrak{opi}} \not\sqsubseteq \overline{\mathfrak{M}}(s)$.

Proof. We prove this by showing that its negation leads to an absurd. So, suppose that $\exists s \in \mathcal{S} \,.\, \delta_{\mathfrak{t}\mathfrak{opi}} \sqsubseteq \overline{\mathfrak{M}}(s)$. Indeed, for every set of traces X we have $\delta_{\mathfrak{t}\mathfrak{opi}}(X) \subseteq \overline{\mathfrak{M}}(s)(X)$. Take $X = \mathcal{W}_s$. In this case, $\overline{\mathfrak{M}}(s)(X) = \mathcal{W}_s$ and, due to extensivity, $\mathcal{W}_s \subseteq \delta_{\mathfrak{t}\mathfrak{opi}}(\mathcal{W}_s)$. So $\mathcal{W}_s = \delta_{\mathfrak{t}\mathfrak{opi}}(\mathcal{W}_s)$ must necessarily hold. Note that $\delta_{\mathfrak{t}\mathfrak{opi}}(\mathcal{W}_s) \in \delta_{\mathfrak{t}\mathfrak{opi}}(\wp(\Sigma^{+}))$ and that $\mathtt{Pres}_{P,\overline{\mathfrak{t}\mathfrak{opi}}}(\mathcal{W}_s)$ does not hold. This latter implies $\mathcal{W}_s \notin \delta_{\mathfrak{t}\mathfrak{opi}}(\wp(\Sigma^{+}))$. But this is absurd because $\mathcal{W}_s = \delta_{\mathfrak{t}\mathfrak{opi}}(\mathcal{W}_s) \in \delta_{\mathfrak{t}\mathfrak{opi}}(\wp(\Sigma^{+}))$. □

Proposition 4.2. Static graph-based watermarking is $\mathfrak{t}^{\mathfrak{ef}}$-resilient. This means that $\delta_{\mathfrak{t}\mathfrak{ef}} \sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) | s \in \mathcal{S}\}$.

Proof. We prove that the negation of this proposition leads to an absurd. So, suppose that $\delta_{\mathfrak{t}\mathfrak{ef}} \not\sqsubseteq \bigsqcap\{\overline{\mathfrak{M}}(s) | s \in \mathcal{S}\} = \bigsqcap \overline{\mathfrak{M}}$, which is equivalent to say that for every set of traces X we have $\bigsqcap \overline{\mathfrak{M}}(X) \subsetneq \delta_{\mathfrak{t}\mathfrak{ef}}(X)$. Recall that, for all $s \in \mathcal{S}$, $\mathcal{W}_s = (\!|\mathfrak{M}(s)|\!)^{\beta}_{+} \in \bigsqcap \overline{\mathfrak{M}}$. Take $X = \mathcal{W}_s$, for some signature s. In this case, $\bigsqcap \overline{\mathfrak{M}}(X) = \mathcal{W}_s$ and so $\mathcal{W}_s \subsetneq \delta_{\mathfrak{t}\mathfrak{ef}}(\mathcal{W}_s)$. Note that $\mathtt{Pres}_{P,\overline{\mathfrak{t}\mathfrak{ef}}}(\mathcal{W}_s)$ holds hence $\mathcal{W}_s \in \delta_{\mathfrak{t}\mathfrak{ef}}(\wp(\Sigma^{+}))$. Indeed we have that $\mathcal{W}_s \subsetneq \delta_{\mathfrak{t}\mathfrak{ef}}(\mathcal{W}_s) = \mathcal{W}_s$, which is absurd. □

It is possible to reason about the resilience of the other watermarking techniques in a similar manner. Table 1 summarizes our results by showing which watermarking system is resilient (✓), *w.r.t.* an attacker, and which one is ineffective (×). We can observe that path-base watermarking is not resilient *w.r.t.* edge-flipping but it is resilient *w.r.t.* dead-code elimination. Abstract watermarking is resilient *w.r.t.* control-flow obfuscations and code-optimization techniques but it is not resilient *w.r.t.* data obfuscations, even the simplest. Indeed, the attacker $\mathfrak{t}^{\mathfrak{do}}$ maintains the signature variable `w` constant modulo n, but it changes its value: after the attack the signature is $s + 1$ instead of s. More sophisticated data obfuscations can be applied. For instance, it is possible to encode variables in a way that they are congruent modulo an arbitrary value, so letting the stegoextractor unable to retrieve any signature. Interestingly, the dynamic graph-based watermarking is resilient

Table 1. *Resilience.*

	$\mathfrak{t}^{\mathfrak{cf}}$	$\mathfrak{t}^{\mathfrak{dce}}$	$\mathfrak{t}^{\mathfrak{lu}}$	$\mathfrak{t}^{\mathfrak{opi}}$	$\mathfrak{t}^{\mathfrak{licm}}$	$\mathfrak{t}^{\mathfrak{do}}$
Block-reordering	✓	✓	×	×	✓	✓
Static graph-based	✓	✓	×	×	✓	✓
Dynamic graph-based	✓	✓	✓	✓	✓	✓
Path-based	×	✓	×	×	✓	✓
Abstract constant propagation	✓	✓	✓	✓	✓	×

w.r.t. every attack, this means that it embeds the signature in an abstract property that is preserved by all the considered attackers. Thus, if we want to develop a watermarking system resilient to common obfuscation techniques we should encode the signature in an abstract property implied by the denotational semantics.

We can also note that some optimization techniques, like dead-code elimination and loop-invariant code motion, do not interfere with all the watermarking schemes analyzed, so they can be applied safely. Instead, loop unrolling must be applied with more attention. Indeed we cannot use this optimization technique with block-reordering or path-based watermarking because it damages the stegomark.

4.3. *Comparison: secrecy and transparence*

In Section 3, we have discussed how different watermarking systems could be compared *w.r.t.* secrecy and transparence based on the comparison of their most powerful secret and transparent attackers, respectively. Also in this case, it may happen that the two abstractions are not comparable. In this case, we compare secrecy and transparence *w.r.t.* particular observations.

4.3.1. *Observations.* Let us denote with $\mathfrak{O}^{\mathfrak{cfg}}$ the abstract interpreter that computes the control flow graph of programs, with $\mathfrak{O}^{\mathfrak{acp}}$ the abstract interpreter that performs the analysis of constant propagation modulo n ($n \in \mathbb{N}$), with $\mathfrak{O}^{\mathfrak{hg}}$ the abstract interpreter that extracts the graph allocated in the heap, and with $\mathfrak{O}^{\mathfrak{ccs}}$ the abstract interpreter that retrieves the choices made at conditional statements. Note that this abstract interpreters implement the stegoextractors of the watermarking techniques described in Section 4.1: $\mathfrak{O}^{\mathfrak{cfg}}$ for block-reordering watermarking, $\mathfrak{O}^{\mathfrak{acp}}$ for abstract constant propagation watermarking, $\mathfrak{O}^{\mathfrak{hg}}$ for dynamic graph-based watermarking and $\mathfrak{O}^{\mathfrak{ccs}}$ for path-based watermarking.

4.3.2. *Comparison results.* Given the semantic formalization of the considered watermarking systems and of the four observers introduced above, we provide a formal proof of the secrecy and transparence of these watermarking systems *w.r.t.* $\mathfrak{O}^{\mathfrak{cfg}}$, $\mathfrak{O}^{\mathfrak{acp}}$, $\mathfrak{O}^{\mathfrak{hg}}$, and $\mathfrak{O}^{\mathfrak{ccs}}$.

Proposition 4.3. Static graph-based watermarking is not $\top$-secret w.r.t $\mathfrak{O}^{\mathfrak{cfg}}$.

Table 2. *Secrecy.*

	$\mathfrak{O}^{\mathfrak{cfg}}$		$\mathfrak{O}^{\mathfrak{acp}}$		$\mathfrak{O}^{\mathfrak{hg}}$		$\mathfrak{O}^{\mathfrak{ccs}}$	
Block-reordering	–	–	ϕ	$\top$	ϕ	$\top$	–	–
Static graph-based	–	–	ϕ	$\top$	ϕ	$\top$	ϕ	$\top$
Dynamic graph-based	ϕ	$\top$	ϕ	$\top$	–	–	ϕ	$\top$
Path-based	–	–	ϕ	$\top$	ϕ	$\top$	–	–
Abstract constant propagation	ϕ	$\top$	–	–	ϕ	$\top$	ϕ	$\top$

Table 3. *Transparence.*

	$\mathfrak{O}^{\mathfrak{cfg}}$	$\mathfrak{O}^{\mathfrak{acp}}$	$\mathfrak{O}^{\mathfrak{hg}}$	$\mathfrak{O}^{\mathfrak{ccs}}$
Block-reordering	×	✓	✓	×
Static graph-based	×	✓	✓	×
Dynamic graph-based	✓	✓	×	✓
Path-based	×	✓	✓	×
Abstract constant propagation	✓	×	✓	✓

Proof. Let $\rho = \overline{\mathfrak{O}^{\mathfrak{cfg}}}$, i.e. the semantic counterpart of $\mathfrak{O}^{\mathfrak{cfg}}$. $\forall P \in \mathcal{P}\, \forall s, s' \in \mathcal{S}$, we have that $(\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!)^{\rho}_{+} \neq (\!|\mathfrak{L}(P, \mathfrak{M}(s'))|\!)^{\rho}_{+}$. In fact, ρ is more concrete than the extraction domain and this makes it able to see differences between any stegoprogram. In fact the extraction domain β distinguish programs looking at the marked subgraph of CFGs while ρ can distinguish CFGs in a more precise way. □

Proposition 4.4. Block-reordering watermarking is transparent *w.r.t.* $\mathfrak{O}^{\mathfrak{acp}}$.

Proof. Let $\rho = \overline{\mathfrak{O}^{\mathfrak{acp}}}$, i.e. the semantic counterpart of $\mathfrak{O}^{\mathfrak{acp}}$. $\forall P \in \mathcal{P}\, \forall s \in \mathcal{S}$ we have that $(\!|P|\!)^{\rho}_{+} = (\!|\mathfrak{L}(P, \mathfrak{M}(s))|\!)^{\rho}_{+}$. In fact, the reordering of the basic block of P does not alter the values of the program's variables. So, the constant propagation modulo n computed on P and every possible stegoprogram returns the same results. □

It is possible to reason about secrecy and the transparence of the other watermarking techniques in a similar manner. Tables 2 and 3 summarize our results. For example, block-reordering watermarking is not $\top$-secret and it is not ϕ-secret, for any possible ϕ, *w.r.t.* $\mathfrak{O}^{\mathfrak{cfg}}$. Instead, it is $\top$-secret, and so ϕ-secret for all possible ϕ, *w.r.t.* an observer which looks at constants ($\mathfrak{O}^{\mathfrak{acp}}$). Moreover, abstract constant propagation watermarking is invisible *w.r.t.* $\mathfrak{O}^{\mathfrak{cfg}}$, while it is not invisible *w.r.t.* $\mathfrak{O}^{\mathfrak{acp}}$. So, as we can see, all these techniques are not invisible only *w.r.t.* the observers which are more precise than $\sqcap\{\overline{\mathfrak{M}}(s) | s \in \mathcal{S}\}$.

Finally, the dynamic graph-based watermarking is secret and transparent *w.r.t.* all attackers, except $\mathfrak{O}^{\mathfrak{hg}}$. Note that this latter retrieves the same information of the dynamic graph-based watermarking stegoextractor, hence the technique cannot be secret/transparent *w.r.t.* this observer. The same holds for abstract constant propagation watermarking. Hence, even if we cannot compare directly the watermarking schemes, we can say that the dynamic graph-based watermarking and abstract constant propagation watermarking are the more safe techniques to use, *w.r.t.* the given set of attackers.

5. Conclusion

In this paper, we have introduced a semantics-based definition of software watermarking and its qualifying features. This definition is general enough to allow the specification of static, abstract and dynamic watermarking techniques. Indeed, all these techniques can be seen as the exploitation of a completeness hole for the insertion of the signature in an efficient way. Only attackers that are complete *w.r.t.* the semantic encoding of the signature are able to observe the signature and potentially tamper with it. This means that the abstract domain used for the semantic encoding of the signature $\overline{\mathfrak{M}}(s)$ acts like a secret key that allows to disclose the signature only to those attackers that are complete *w.r.t.* $\overline{\mathfrak{M}}(s)$.

Regarding the features of a watermarking scheme, our general framework provides a formal setting for proving the efficiency of a watermarking scheme *w.r.t.* resilience, secrecy, transparence and accuracy. To validate our theory we have proved the efficiency of five known watermarking systems. Thus, we provide a general theory where researchers can build a formal evidence of the quality of the watermarking system that they propose. We believe that this is an important contribution that can be considered as the first step towards a formal theory for software watermarking, where new and existing techniques can be certified *w.r.t.* their efficiency.

We would like to thank Roberto Giacobazzi for the initial discussions on this work and Isabella Mastroeni for the discussions on higher order abstract non-interference. We also would like to thank the anonymous reviewers for the useful suggestions and comments, helping us in improving the presentation of our work.

Appendix A. Proofs

A.1. *Proof of Proposition 2.1*

First, we have to prove that $\langle \wp(\Sigma^+), \leqslant \rangle$ is a partially ordered set.

Reflexivity. For all $X \in \wp(\Sigma^+)$, $X \leqslant X$ holds, in fact $\forall \sigma \in X \, \exists \sigma' \in X \,.\, \sigma \in \mathtt{pref}(\sigma')$ holds for $\sigma' = \sigma$ and hence $X \subseteq X$ is trivially true.

Antisymmetry. For all $X, Y \in \wp(\Sigma^+)$:

a) from $X \leqslant Y$ it follows that $\forall \sigma \in X \, \exists \sigma' \in Y \,.\, \sigma \in \mathtt{pref}(\sigma')$ and so from $Y \leqslant X$ (the second part of the conjunction) it follows that $X \subseteq Y$.

b) from $Y \leqslant X$ it follows that $\forall \sigma' \in Y \, \exists \sigma \in X \,.\, \sigma' \in \mathtt{pref}(\sigma)$ and so from $X \leqslant Y$ (the second part of the conjunction) it follows that $Y \subseteq X$.

From **a** and **b** it follows that $X = Y$.

Transitivity. For all $X, Y, Z \in \wp(\Sigma^+)$

c) from $X \leqslant Y$ it follows that $\forall \sigma^x \in X\, \exists \sigma^y \in Y\, .\, \sigma^x \in \mathtt{pref}(\sigma^y)$.
d) from $Y \leqslant Z$ it follows that $\forall \sigma^{y'} \in Y\, \exists \sigma^z \in Z\, .\, \sigma^{y'} \in \mathtt{pref}(\sigma^z)$.

Considering **c** and **d** it is easy to find for all $\sigma^x \in X$ a $\sigma^z \in Z$ such that $\sigma^x \in \mathtt{pref}(\sigma^z)$, in fact with $\sigma^y = \sigma^{y'}$ the relation is satisfied. In the end from $Z \subseteq Y$ and $Y \subseteq X$, it follows that $Z \subseteq X$. So $X \leqslant Z$ holds.

So $\leqslant$ is a partial order on $\wp(\Sigma^+)$.

Second, for all $X \in \mathcal{X} \subseteq \wp(\Sigma^+)$ we have that $X \leqslant \biguplus \mathcal{X}$, namely $\biguplus \mathcal{X}$ is an upper bound of $\mathcal{X}$, but we have to prove that it is the least. In order to prove that $\forall \mathcal{X} \subseteq \wp(\Sigma^+)\, \forall \varpi \in \wp(\Sigma^+)\, .\, (\forall X \in \mathcal{X}\, .\, X \leqslant \varpi) \Rightarrow \biguplus \mathcal{X} \leqslant \varpi$ we show that its negate is false, i.e. we prove that it does not exist $\varpi \in \wp(\Sigma^+)$ such that $(\forall X \in \mathcal{X}\, .\, X \leqslant \varpi) \Rightarrow (\varpi \leqslant \biguplus \mathcal{X} \wedge \varpi \neq \biguplus \mathcal{X})$. $\forall X \in \mathcal{X}\, .\, X \leqslant \varpi$ means that $\forall \sigma \in \bigcup_{X \in \mathcal{X}} X\, \exists \sigma' \in \varpi\, .\, \sigma \in \mathtt{pref}(\sigma')$ and $\varpi \leqslant \biguplus \mathcal{X}$ means that $\forall \sigma' \in \varpi\, \exists \sigma'' \in \biguplus \mathcal{X}\, .\, \sigma' \in \mathtt{pref}(\sigma'')$. By definition, $\biguplus \mathcal{X} \subseteq \bigcup_{X \in \mathcal{X}} X$ so $\forall \sigma' \in \varpi\, \exists \sigma'' \in \bigcup_{X \in \mathcal{X}} X\, .\, \sigma' \in \mathtt{pref}(\sigma'')$. From the last proposition and from $\forall \sigma \in \bigcup_{X \in \mathcal{X}} X\, \exists \sigma' \in \varpi\, .\, \sigma \in \mathtt{pref}(\sigma')$ it follows that $\varpi = \bigcup_{X \in \mathcal{X}} X$ (and so $\biguplus \mathcal{X} \subseteq \varpi$). By the fact that $\biguplus \mathcal{X} \neq \varpi$ we have that $\exists \sigma'' \in \biguplus \mathcal{X}\, \forall \sigma' \in \varpi\, .\, \sigma'' \notin \mathtt{pref}(\sigma')$ which is absurd because $\biguplus \mathcal{X} \subseteq \varpi$.

Third, the bottom element is $\varnothing \in \wp(\Sigma^+)$, i.e. $\forall X \in \wp(\Sigma^+)\, .\, \varnothing \leqslant X$ holds. In fact, with no traces, the conditions of the relation are vacuously true and, if $X = \varnothing$, $\varnothing \leqslant \varnothing$ trivially holds.

Finally, for all $\mathcal{X} \subseteq \wp(\Sigma^+)$ it is easy to note that $\biguplus \mathcal{X}$ exists and it is a set of finite traces, so $\biguplus \mathcal{X} \in \wp(\Sigma^+)$. So, by the fact that $\langle \wp(\Sigma^+), \leqslant \rangle$ has minimum (bottom), in addition to the fact that for each subset of $\wp(\Sigma^+)$ there is a least upper bound in $\wp(\Sigma^+)$, we get that $\langle \wp(\Sigma^+), \leqslant, \uplus, \varnothing \rangle$ is a join semi-lattice and hence a DCPO. □

A.2. *Proof of Proposition 2.2*

The first iterates of $F^+_{P[I]}$ for $lfp^{\leqslant}_{\varnothing} F^+_{P[I]}$ are as follows:

$$X^0 = \varnothing$$

$$X^1 = F^+_{P[I]}(X^0) = \llparenthesis P \rrparenthesis^1_I \uplus \{\sigma\varsigma\varsigma' | \sigma\varsigma \in \varnothing \wedge \varsigma' \in {}^{\mathbf{S}}[\![P]\!](\varsigma)\} = I = \llparenthesis P \rrparenthesis^1_I$$

$$X^2 = F^+_{P[I]}(X^1) = \llparenthesis P \rrparenthesis^1_I \uplus \{\sigma\varsigma\varsigma' | \sigma\varsigma \in X^1 \wedge \varsigma' \in {}^{\mathbf{S}}[\![P]\!](\varsigma)\} =$$

$$= \llparenthesis P \rrparenthesis^1_I \uplus (\llparenthesis P \rrparenthesis^{\bar{1}}_I \uplus \llparenthesis P \rrparenthesis^2_I) = \llparenthesis P \rrparenthesis^{\bar{1}}_I \uplus \llparenthesis P \rrparenthesis^2_I$$

$$X^3 = F^+_{P[I]}(X^2) = \llparenthesis P \rrparenthesis^1_I \uplus \{\sigma\varsigma\varsigma' | \sigma\varsigma \in X^2 \wedge \varsigma' \in {}^{\mathbf{S}}[\![P]\!](\varsigma)\} =$$

$$= \llparenthesis P \rrparenthesis^1_I \uplus (\llparenthesis P \rrparenthesis^{\bar{1}}_I \uplus \llparenthesis P \rrparenthesis^{\bar{2}}_I \uplus \llparenthesis P \rrparenthesis^3_I) = \llparenthesis P \rrparenthesis^{\bar{1}}_I \uplus \llparenthesis P \rrparenthesis^{\bar{2}}_I \uplus \llparenthesis P \rrparenthesis^3_I$$

By recurrence the nth iterate of $F^+_{P[I]}$ is

$$X^n = \biguplus_{i=1}^{n-1} \llparenthesis P \rrparenthesis^{\bar{i}}_I \uplus \llparenthesis P \rrparenthesis^n_I.$$

In fact

$$\begin{aligned} F^{+}_{P[I]}(X^n) &= \llparenthesis P \rrparenthesis^{1}_{I} \uplus \{\sigma\varsigma\varsigma' | \sigma\varsigma \in X^n \wedge \varsigma' \in {}^{\mathbb{S}}[\![P]\!](\varsigma)\} = \\ &= \llparenthesis P \rrparenthesis^{1}_{I} \uplus (\biguplus_{i=1}^{n} \llparenthesis P \rrparenthesis^{\bar{i}}_{I} \uplus \llparenthesis P \rrparenthesis^{n+1}_{I}) = \\ &= \biguplus_{i=1}^{n} \llparenthesis P \rrparenthesis^{\bar{i}}_{I} \uplus \llparenthesis P \rrparenthesis^{n+1}_{I} = X^{n+1} \end{aligned}$$

Due to the fact that $F^{+}_{P[I]}$ is Scott-continuous and that it is defined over a DCPO, $F^{+}_{P[I]}$ admits a fixpoint, i.e. exists $k \in \mathbb{N}$ such that $F^{+}_{P[I]}(X^k) = X^k$, and it is exactly the least upper bound of the Kleene chain of $F^{+}_{P[I]}$ starting from $\varnothing$. Due to this considerations, we get that

$$F^{+}_{P[I]}(X^k) = \biguplus_{i=1}^{k} \llparenthesis P \rrparenthesis^{\bar{i}}_{I} \uplus \llparenthesis P \rrparenthesis^{k+1}_{I} = \biguplus_{i=1}^{k-1} \llparenthesis P \rrparenthesis^{\bar{i}}_{I} \uplus \llparenthesis P \rrparenthesis^{k}_{I} = X^k$$

and so that

$$\begin{aligned} \llparenthesis P \rrparenthesis^{\bar{k}}_{I} \cup \llparenthesis P \rrparenthesis^{k+1}_{I} &= \llparenthesis P \rrparenthesis^{k}_{I} \\ \llparenthesis P \rrparenthesis^{\bar{k}}_{I} \cup \llparenthesis P \rrparenthesis^{k+1}_{I} &= \llparenthesis P \rrparenthesis^{\bar{k}}_{I} \cup \{\sigma \in \llparenthesis P \rrparenthesis^{k}_{I} | \sigma_{\dashv} \notin T^P\} \\ \llparenthesis P \rrparenthesis^{k+1}_{I} &= \{\sigma \in \llparenthesis P \rrparenthesis^{k}_{I} | \sigma_{\dashv} \notin T^P\} \end{aligned}$$

The only way to make the last equation true is to have $\llparenthesis P \rrparenthesis^{k+1}_{I} = \varnothing$ and $\{\sigma \in \llparenthesis P \rrparenthesis^{k}_{I} | \sigma_{\dashv} \notin T^P\} = \varnothing$, due to the fact that $\{\sigma \in \llparenthesis P \rrparenthesis^{k}_{I} | \sigma_{\dashv} \notin T^P\}$ and $\llparenthesis P \rrparenthesis^{k+1}_{I}$ do not share any element. So finally, we have that it exists $k \in \mathbb{N}$ such that ${F^{+}_{P[I]}}^{k}(\varnothing) = \biguplus_{i=1}^{k} \llparenthesis P \rrparenthesis^{\bar{i}}_{I} = lfp^{\leqslant}_{\varnothing} F^{+}_{P[I]}$. □

A.3. *Proof of Theorem 2.1*

First, it is necessary to prove that this set is an upper closure operator and, after that, we have to prove that is the most concrete closure for which the program is safe. Upper closure operators are isomorphic to Moore families, so we prove that $\{X \in \wp(\Sigma^+) | \mathtt{Secr}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(X)\}$ is a Moore family. For doing so, we have only to prove that this set contains the intersection of all its elements. Consider $X, Y \in \{Z \in \wp(\Sigma^+) | \mathtt{Secr}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(Z)\}$. For hypothesis we have that:

$$\begin{aligned} &\forall P \in \mathsf{P}\, \forall Q \in \mathsf{Q}\,. \\ &\quad (\exists Z \in \Upsilon^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(P,Q)\,.\, Z \subseteq X \Rightarrow \forall W \in \Upsilon^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(P,Q)\,.\, W \subseteq X) \wedge \\ &\quad (\exists Z \in \Upsilon^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(P,Q)\,.\, Z \subseteq Y \Rightarrow \forall W \in \Upsilon^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(P,Q)\,.\, W \subseteq Y). \end{aligned}$$

So, we have to prove that the same condition holds for $X \cap Y$. Suppose that $\forall P \in \mathsf{P}\, \forall Q \in \mathsf{Q}$ it exists Z in $\Upsilon^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(P,Q)$ such that $Z \subseteq X \cap Y$ (indeed $Z \subseteq X$ and $Z \subseteq Y$). For hypothesis $\mathtt{Secr}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(X)$ and $\mathtt{Secr}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(Y)$ hold, so $\forall W \in \Upsilon^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(P,Q)$ we have that W is contained in X and W is contained in Y, namely $W \subseteq X \cap Y$. Therefore $\mathtt{Secr}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(X \cap Y)$ holds. This result can be easily extended to a generic intersection, so $\{X \in \wp(\Sigma^+) | \mathtt{Secr}^{\mathrm{H}+}_{\mathfrak{I},\eta,(\phi)}(X)\}$ is a Moore family.

Second, we have to prove non-interference, i.e. for $\hat{\rho} \triangleq \{X \in \wp(\Sigma^{+}) | \mathtt{Secr}^{\mathrm{H}+}_{\Im,\eta,(\phi)}(X)\}$ it is true $\underset{+}{\mathrm{H}}[\eta]\Im(\phi \Rightarrow \hat{\rho})_{\mathtt{bca}}$. Suppose that it exists $P_1, P_2 \in \mathsf{P}$ and $Q_1, Q_2 \in \mathsf{Q}$ such that $\llparenthesis P_1 \rrparenthesis^{\eta}_{+} = \llparenthesis P_2 \rrparenthesis^{\eta}_{+} \wedge \llparenthesis Q_1 \rrparenthesis^{\phi}_{+} = \llparenthesis Q_2 \rrparenthesis^{\phi}_{+}$, with $\llparenthesis \Im(P_1, Q_1) \rrparenthesis^{\hat{\rho}}_{+} \neq \llparenthesis \Im(P_2, Q_2) \rrparenthesis^{\hat{\rho}}_{+}$. We can note that $\llparenthesis \Im(P_1, Q_1) \rrparenthesis_{+}$ and $\llparenthesis \Im(P_2, Q_2) \rrparenthesis_{+}$ are in $\Upsilon^{\mathrm{H}+}_{\Im,\eta,(\phi)}(P_1, Q_1)$, in fact $P_1 \equiv_{\eta} P_2$ and $Q_1 \equiv_{\phi} Q_2$. Let $X_1, X_2 \in \wp(\Sigma^{+})$ defined as $X_1 \triangleq \llparenthesis \Im(P_1, Q_1) \rrparenthesis^{\hat{\rho}}_{+}$ and $X_2 \triangleq \llparenthesis \Im(P_2, Q_2) \rrparenthesis^{\hat{\rho}}_{+}$. For hypothesis, we have that $X_1 \neq X_2$ but, due to the fact that $X_1, X_2 \in \hat{\rho}$, we have $\mathtt{Secr}^{\mathrm{H}+}_{\Im,\eta,(\phi)}(X_1)$ and $\mathtt{Secr}^{\mathrm{H}+}_{\Im,\eta,(\phi)}(X_2)$. Now, if $X_2 \nsubseteq X_1$, we have $\llparenthesis \Im(P_1, Q_1) \rrparenthesis_{+} \in \Upsilon^{\mathrm{H}+}_{\Im,\eta,(\phi)}(P_1, Q_1)$ such that $\llparenthesis \Im(P_1, Q_1) \rrparenthesis_{+} \subseteq X_1$, whilst $\llparenthesis \Im(P_2, Q_2) \rrparenthesis_{+} \in \Upsilon^{\mathrm{H}+}_{\Im,\eta,(\phi)}(P_1, Q_1)$ such that $\llparenthesis \Im(P_1, Q_1) \rrparenthesis_{+} \nsubseteq X_1$: absurd (for the hypothesis on X_1). Similarly, reasoning on $X_2 \subseteq X_1$ we obtain the same contradiction on X_2. Therefore, $\llparenthesis P_1 \rrparenthesis^{\eta}_{+} = \llparenthesis P_2 \rrparenthesis^{\eta}_{+} \wedge \llparenthesis Q_1 \rrparenthesis^{\phi}_{+} = \llparenthesis Q_2 \rrparenthesis^{\phi}_{+}$ implies that $\llparenthesis \Im(P_1, Q_1) \rrparenthesis^{\hat{\rho}}_{+}$ is equal to $\llparenthesis \Im(P_2, Q_2) \rrparenthesis^{\hat{\rho}}_{+}$.

Finally we have to prove that the closure is the most concrete. Suppose the contrary, i.e. it exists a domain ρ such that $\hat{\rho} \not\sqsubseteq \rho$ and $\underset{+}{\mathrm{H}}[\eta]\Im(\phi \Rightarrow \rho)_{\mathtt{b}ca}$ holds. Remember that $\hat{\rho} \sqsubseteq \rho$ *iff* $\rho \subseteq \hat{\rho}$. Take into account the case in which $\hat{\rho} \subsetneq \rho$. At this point we can note that it exists $X \in \rho$ such that $X \notin \hat{\rho}$, i.e. for which $\mathtt{Secr}^{\mathrm{H}+}_{\Im,\eta,(\phi)}(X)$ does not hold. But this means that it exists $P \in \mathsf{P}$ and $Q \in \mathsf{Q}$ such that $\exists Z \in \Upsilon^{\mathrm{H}+}_{\Im,\eta,(\phi)}(P, Q).\, Z \subseteq X$ and $\exists W \in \Upsilon^{\mathrm{H}+}_{\Im,\eta,(\phi)}(P, Q).\, W \nsubseteq X$. So $Z = \llparenthesis \Im(P_1, Q_1) \rrparenthesis_{+}$, for some $P_1 \in \mathsf{P}$, $Q_1 \in \mathsf{Q}$ and $W = \llparenthesis \Im(P_2, Q_2) \rrparenthesis_{+}$, for some $P_2 \in \mathsf{P}$, $Q_2 \in \mathsf{Q}$, with $P_1 \equiv_{\eta} P_2 \equiv_{\eta} P$ and $Q_2 \equiv_{\phi} Q_2 \equiv_{\phi} Q$ due to the fact that both the sets belong to $\Upsilon^{\mathrm{H}+}_{\Im,\eta,(\phi)}(P, Q)$. All this implies that $\llparenthesis \Im(P_1, Q_1) \rrparenthesis^{\rho}_{+} \subseteq X$ due to $\llparenthesis \Im(P_1, Q_1) \rrparenthesis_{+} \in X$, whilst $\llparenthesis \Im(P_2, Q_2) \rrparenthesis^{\rho}_{+} \nsubseteq X$ due to $\llparenthesis \Im(P_2, Q_2) \rrparenthesis_{+} \notin X$. Indeed $\llparenthesis \Im(P_1, Q_1) \rrparenthesis^{\rho}_{+} \neq \llparenthesis \Im(P_2, Q_2) \rrparenthesis^{\rho}_{+}$, but this is impossible, in fact we supposed that $\underset{+}{\mathrm{H}}[\eta]\Im(\phi \Rightarrow \rho)_{\mathtt{b}ca}$. So a domain like ρ doesn't exist. □

Appendix B. Algorithms

```
Algorithm 1 ELIMINATION
Require: $\sigma, \mathcal{I}$
for all $l \in \mathcal{I}$ do
    Let $\sigma_i$ such that $\texttt{slab}(\sigma_i) = l$
    $l' \leftarrow \texttt{slab}(\sigma_{i+1})$
    $j \leftarrow 0$
    while $j < |\sigma|$ do                          ▷ $|\sigma|$ is updated at every cycle
        Let $\sigma_j = \langle C_j, \zeta_j \rangle$
        if $\texttt{slab}(\sigma_j) = l$ then
            Let $\sigma_{j+1} = \langle C_{j+1}, \zeta_{j+1} \rangle$
            $\sigma_{j+1} \leftarrow \langle C_{j+1}, \zeta_j \rangle$
            $\sigma_j \leftarrow \varnothing$
            $j \leftarrow j - 1$
        else
            if $C_j = L : A \to l;$ then
                $\sigma_j \leftarrow \langle L : A \to l';, \zeta_j \rangle$
            if $C_j = L : B \to \{l, L_F\};$ then
                $\sigma_j \leftarrow \langle L : B \to \{l', L_F\};, \zeta_j \rangle$
            if $C_j = L : B \to \{L_T, l\};$ then
                $\sigma_j \leftarrow \langle L : B \to \{L_T, l'\};, \zeta_j \rangle$
        $j \leftarrow j + 1$
Ensure: $\sigma$
```

The syntactic encoding (for arithmetic expressions) $\texttt{Enc} \in E \to E$ is inductively defined as follows:

$$\begin{aligned}
\texttt{Enc}(n) &= (n+1)\\
\texttt{Enc}(X) &= X\\
\texttt{Enc}(E_1 + E_2) &= (\texttt{Enc}(E_1) + \texttt{Enc}(E_2))\\
\texttt{Enc}(E_1 \cdot E_2) &= (\texttt{Enc}(E_1) \cdot \texttt{Enc}(E_2))\\
\texttt{Enc}(E_1 - E_2) &= (\texttt{Enc}(E_1) - \texttt{Enc}(E_2)).
\end{aligned}$$

The syntactic encoding (for boolean expressions) $\texttt{Enc} \in B \to B$ is inductively defined as follows:

$$\begin{aligned}
\texttt{Enc}(b) &= b\\
\texttt{Enc}(\neg B) &= \neg\texttt{Enc}(B)\\
\texttt{Enc}(B_1 \wedge B_2) &= \texttt{Enc}(B_1) \wedge \texttt{Enc}(B_2)\\
\texttt{Enc}(B_1 \vee B_2) &= \texttt{Enc}(B_1) \vee \texttt{Enc}(B_2)\\
\texttt{Enc}(E_1 < E_2) &= \texttt{Enc}(E_1) < \texttt{Enc}(E_2)\\
\texttt{Enc}(E_1 = E_2) &= \texttt{Enc}(E_1) = \texttt{Enc}(E_2)
\end{aligned}$$

Algorithm 2 UNROLL

```
Require: $\sigma, \mathcal{I}$
for all $\langle l^G, l^I \rangle \in \mathcal{I}$ do
    Let $\sigma_i = \langle C_i, \zeta_i \rangle$ such that $\texttt{slab}(\sigma_i) = l^G$
    Let $\sigma_j = \langle C_j, \zeta_j \rangle$ such that $\texttt{slab}(\sigma_j) = l^I$
    Let $X$ the variable of the guard in $C_i$, i.e. $C_i = l^G : X < E \rightarrow \{l^H, l^O\};$
    Let $X$ the variable in the increment $C_j$, i.e. $C_j = l^I : X := X + \dot{E} \rightarrow l^G;$
    Let $\dot{e}$ be the value of expression $\dot{E}$ computed in context $\zeta_j$
    $\texttt{Itrs} \leftarrow$ ordered list of pairs $\langle i, j \rangle$ with, $i < j$, such that:
                $\texttt{slab}(\sigma_i) = l^G \wedge \texttt{slab}(\sigma_j) = l^O \wedge \forall k \in (i, j) . \texttt{slab}(\sigma_k) \neq l^O$
    for all $\langle i, j \rangle \in \texttt{Itrs}$ do                                    ▷ in list order
        $m \leftarrow 0$
        for $k = i$ to $j - 1$ do
            Let $\sigma_k = \langle C_k, \langle \rho_k, \iota_k \rangle \rangle$
            if $\texttt{slab}(\sigma_k) = l^G$ then
                $m \leftarrow m + 1$
                if $m > 1$ then
                    $L \leftarrow l^{G^{m-1}}$
                    $\rho_k \leftarrow \rho_k[X \mapsfrom \rho_k(X) - m\dot{e}]$
                else
                    $L \leftarrow l^G$
                $\sigma_k \leftarrow \langle L : \texttt{skip} \rightarrow l^{H^m}; \langle \rho_k, \iota_k \rangle \rangle$
            if $\texttt{slab}(\sigma_k) = l^I$ then
                $\sigma_k \leftarrow \langle l^{I^m} : \texttt{skip} \rightarrow l^{G^m};, \langle \rho_k[X \mapsfrom \rho_k(X) - m\dot{e}], \iota_k \rangle \rangle$
            if $\texttt{slab}(\sigma_k) \neq l^G \wedge \texttt{slab}(\sigma_k) \neq l^I$ then          ▷ so $C_k \in H$
                if $C_k = L : B \rightarrow \{L_T, L_F\};$ then
                    $B' \leftarrow B[X \mapsfrom X + (m-1)\dot{e}]$
                    $C_k \leftarrow L^m : B' \rightarrow \{L_T^m, L_F^m\};$
                if $C_k = L_1 : A \rightarrow L_2;$ then
                    $A' \leftarrow A[X \mapsfrom X + (m-1)\dot{e}]$
                    $C_k \leftarrow L_1^m : A' \rightarrow L_2^m;$
                $\sigma_k \leftarrow \langle C_k, \langle \rho_k[X \mapsfrom \rho_k(X) - m\dot{e}], \iota_k \rangle \rangle$
        Let $\sigma_{j-1} = \langle C_{j-1}, \zeta_{j-1} \rangle$
        $\sigma' \leftarrow \sigma_0 \ldots \sigma_i \ldots \sigma_{j-1}$
        $\sigma' \leftarrow \sigma' \langle l^{H^m} : X := X + (m-1)\dot{e} \rightarrow l^O;, \zeta_{j-1} \rangle$
        $\sigma \leftarrow \sigma' \sigma_j \ldots \sigma_{|\sigma|-1}$
Ensure: $\sigma$
```

Algorithm 3 MOTION

```
Require: $\sigma, \mathcal{I}$
for all $l \in \mathcal{I}$ do
    Let $\sigma_k = \langle C_l, \zeta_k \rangle$ such that $\mathtt{slab}(\sigma_k) = l$
    Let $X$ be the variable assigned in $C_l$, i.e. $C_l = l : X := E_l \rightarrow l';$
    Let $e$ be the value of expression $E_l$ computed in context $\zeta_j$
    $l^G \leftarrow \mathtt{entry}(l)$
    $l^O \leftarrow \mathtt{exit}(l)$
    $\mathtt{Itrs} \leftarrow$ ordered list of pairs $\langle i, j \rangle$, with $i < j$, such that:
                $\mathtt{slab}(\sigma_i) = l^G \wedge \mathtt{slab}(\sigma_j) = l^O \wedge \forall k \in (i, j) \,.\, \mathtt{slab}(\sigma_k) \neq l^O$
    Let $\langle i, j \rangle$ the first element of $\mathtt{Itrs}$
    $w \leftarrow 0$
    for all $\langle i, j \rangle \in \mathtt{Itrs}$ do                                ▷ in list order
        $\sigma' \leftarrow \sigma_w \dots \sigma_{i-2}$
        Let $\sigma_{i-1} = \langle C_{i-1}, \zeta_{i-1} \rangle$
        if $C_{i-1} = L : A \rightarrow l^G;$ then
            $\sigma' \leftarrow \sigma' \langle L : A \rightarrow \hat{L}; \zeta_{i-1} \rangle \langle \hat{L} : X := E_l \rightarrow l^G, \zeta_{i-1} \rangle$
        if $C_{i-1} = L : B \rightarrow \{l^G, L_F\};$ then
            $\sigma' \leftarrow \sigma' \langle L : B \rightarrow \{\hat{L}, L_F\}; \zeta_{i-1} \rangle \langle \hat{L} : X := E_l \rightarrow l^G, \zeta_{i-1} \rangle$
        if $C_{i-1} = L : B \rightarrow \{L_T, l^G\};$ then
            $\sigma' \leftarrow \sigma' \langle L : B \rightarrow \{L_T, \hat{L}\}; \zeta_{i-1} \rangle \langle \hat{L} : X := E_l \rightarrow l^G, \zeta_{i-1} \rangle$
        for all $k \in [i, j)$ do
            Let $\sigma_k = \langle C_k, \langle \rho_k, \imath_k \rangle \rangle$
            $\sigma_k \leftarrow \langle C_k, \langle \rho_k[X \hookleftarrow e], \imath_k \rangle \rangle$
            if $\mathtt{slab}(\sigma_k) \neq l$ then
                if $C_k = L : A \rightarrow l$ then
                    $\sigma_k \leftarrow \langle L : A \rightarrow l';, \langle \rho_k, \imath_k \rangle \rangle$
                if $C_k = L : B \rightarrow \{l, L_F\};$ then
                    $\sigma_k \leftarrow \langle L : B \rightarrow \{l', L_F\};, \langle \rho_k, \imath_k \rangle \rangle$
                if $C_k = L : B \rightarrow \{L_T, l\};$ then
                    $\sigma_k \leftarrow \langle L : B \rightarrow \{L_T, l'\};, \langle \rho_k, \imath_k \rangle \rangle$
            $\sigma' \leftarrow \sigma' \sigma_k$
        $w \leftarrow j$
    Let $\langle i, j \rangle$ the last element of $\mathtt{Itrs}$
    $\sigma' \leftarrow \sigma' \sigma_j \dots \sigma_{|\sigma|-1}$
    $\sigma \leftarrow \sigma'$
$i \leftarrow 0$
for $j = 0$ to $|\sigma| - 1$ do
    if $\mathtt{slab}(\sigma_j) \notin \mathcal{I}$ then
        $\sigma'_i \leftarrow \sigma_j$
        $i \leftarrow i + 1$
Ensure: $\sigma'$
```

Algorithm 4 Enc-Dec

```
Require: $\sigma$
Itrs $\leftarrow$ list of variables of $\sigma$
Labs $\leftarrow$ set of labels of $\sigma$
Let $\sigma_0 = \langle C^0, \zeta^0 \rangle$
$\sigma' \leftarrow \langle L : \mathtt{Itrs}_0 := \mathtt{Itrs}_0 + 1 \rightarrow L';, \zeta^0 \rangle$ such that $L, L' \notin \mathtt{Labs}$
$k \leftarrow 1$
while $k < |\mathtt{Itrs}|$ do
    Let $\sigma'_{k-1} = \langle L : A \rightarrow L';, \langle \rho, \iota \rangle \rangle$
    $\sigma' \leftarrow$ $\sigma' \langle L' : \mathtt{Itrs}_k := \mathtt{Itrs}_k + 1 \rightarrow L'';, \langle \rho[\mathtt{Itrs}_{k-1} \mapsfrom \rho(\mathtt{Itrs}_{k-1}) + 1], \iota \rangle \rangle$ such that $L'' \notin \mathtt{Labs}$
    $k \leftarrow k + 1$
Let $\sigma'_{k-1} = \langle L : A \rightarrow L';, \langle \rho, \iota \rangle \rangle$
$\sigma'' \leftarrow \sigma'_0 \dots \sigma'_{k-2}$
$\sigma' \leftarrow \sigma'' \langle L : A \rightarrow {}^{lab}[\![C^0]\!], \langle \rho, \iota \rangle \rangle \langle C^0, \langle \rho[\mathtt{Itrs}_{k-1} \mapsfrom \rho(\mathtt{Itrs}_{k-1}) + 1], \iota \rangle \rangle$
$\sigma' \leftarrow \sigma' \sigma_1 \dots \sigma_{|\sigma|-1}$
while $k < |\sigma'|$ do
    Let $\sigma'_k = \langle C, \langle \rho, \iota \rangle \rangle$
    $\rho' \leftarrow \rho$
    for all $j \in [0, |\mathtt{Itrs}|)$ do
        $\rho' \leftarrow \rho'[\mathtt{Itrs}_j \mapsfrom \rho'(\mathtt{Itrs}_j) + 1]$
    if $C = L : B \rightarrow \{L_{\mathsf{tt}}, L_{\mathsf{ff}}\};$ then
        $\sigma' \leftarrow \sigma'_0 \dots \sigma'_{k-1} \langle L : \mathtt{Enc}(B) \rightarrow \{L_{\mathsf{tt}}, L_{\mathsf{ff}}\};, \langle \rho', \iota \rangle \rangle \sigma'_{k+1} \dots \sigma'_{|\sigma'|-1}$
    else if $C = L : X := E \rightarrow L';$ then
        $\sigma' \leftarrow \sigma'_0 \dots \sigma'_{k-1} \langle L : X := \mathtt{Enc}(E) \rightarrow L';, \langle \rho', \iota \rangle \rangle \sigma'_{k+1} \dots \sigma'_{|\sigma'|-1}$
    else if $C = L : \mathtt{input}\ X \rightarrow L';$ then
        $\sigma'' \leftarrow \sigma'_0 \dots \sigma'_{k-1} \langle L : \mathtt{input}\ X \rightarrow L''; \langle \rho, \iota \rangle \rangle$ such that $L'' \notin \mathtt{Labs}$
        Let $\sigma'_{k+1} = \langle C^{k+1}, \langle \rho^{k+1}, \iota^{k+1} \rangle \rangle$
        $\sigma'' \leftarrow \sigma'' \langle L'' : X := X + 1 \rightarrow L';, \langle \rho^{k+1}, \iota^{k+1} \rangle \rangle$
        $\rho'' \leftarrow \rho'[X \mapsfrom \rho^{k+1}(X) + 1]$
        $\sigma' \leftarrow \sigma'' \langle C^{k+1}, \langle \rho'', \iota^{k+1} \rangle \rangle \sigma'_{k+2} \dots \sigma'_{|\sigma'|-1}$
    else
        $\sigma' \leftarrow \sigma'_0 \dots \sigma'_{k-1} \langle C, \langle \rho', \iota \rangle \rangle \sigma'_{k+1} \dots \sigma'_{|\sigma'|-1}$
    $k \leftarrow k + 1$
Let $\sigma'_{k-1} = \langle L : \mathtt{stop}, \zeta \rangle$
$\sigma' \leftarrow \sigma'_0 \dots \sigma'_{k-2} \langle L : \mathtt{Itrs}_0 := \mathtt{Itrs}_0 - 1 \rightarrow L';, \zeta \rangle$ such that $L' \notin \mathtt{Labs}$
$j \leftarrow 1$
while $j < |\mathtt{Itrs}|$ do
    Let $\sigma'_{k-1} = \langle L : A \rightarrow L';, \langle \rho, \iota \rangle \rangle$
    $\sigma' \leftarrow$ $\sigma' \langle L' : \mathtt{Itrs}_j := \mathtt{Itrs}_j - 1 \rightarrow L''; \langle \rho[\mathtt{Itrs}_{j-1} \mapsfrom \rho(\mathtt{Itrs}_{j-1}) - 1], \iota \rangle \rangle$ such that $L'' \notin \mathtt{Labs}$
    $k \leftarrow k + 1$
    $j \leftarrow j + 1$
Let $\sigma'_{k-1} = \langle L : A \rightarrow L';, \zeta \rangle$
$\sigma' \leftarrow \sigma' \langle L' : \mathtt{stop}; \zeta \rangle$
Ensure: $\sigma'$
```

References

Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P. and Yang, K. (2001). On the (im)possibility of obfuscating programs. In: *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, Springer-Verlag 1–18.

BSA (2016). Global Software Survey: Seizing Opportunity Through License Compliance, Online. . Available at http://globalstudy.bsa.org/2016/.

Collberg, C., Carter, E., Debray, S., Huntwork, A., Kececioglu, J., Linn, C. and Stepp, M. (2004). Dynamic path-based software watermarking. *SIGPLAN Not.* **39** (6) 107–118.

Collberg, C. and Thomborson, C. (2002). Watermarking, tamper-proofing, and obfuscation-tools for software protection. *IEEE Transactions Software Engineering* **28** 735–746.

Collberg, C. and Thomborson, C.D. (1999). Software watermarking: Models and dynamic embeddings. In: *POPL '99: Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ACM 311–324.

Collberg, C., Thomborson, C.D. and Low, D. (1997). A taxonomy of obfuscating transformations; Technical Report 148; Department of Computer Science, The University of Auckland.

Collberg, C., Thomborson, C.D. and Low, D. (1998). Manufactoring cheap, resilient, and stealthy opaque constructs. In: *Proceedings of Conference Record of the 25st ACM Symp osium on Principles of Programming Languages (POPL'98)*, ACM Press 184–196.

Cousot, P. (2002). Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theoretical Computer Science* **277** (1–2) 47–103.

Cousot, P. and Cousot, R. (1977). Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: *Conference Record of the 4th ACM Symposium on Principles of Programming Languages (POPL'77)*, ACM Press 238–252.

Cousot, P. and Cousot, R. (1979). Systematic design of program analysis frameworks. In: *Conference Record of the 6th ACM Symposium on Principles of Programming Languages (POPL'79)*, ACM Press 269–282.

Cousot, P. and Cousot, R. (2002). Systematic design of program transformation frameworks by abstract interpretation. In: *Conference Record of the 29th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ACM Press 178–190.

Cousot, P. and Cousot, R. (2004). An abstract interpretation-based framework for software watermarking. In: *Conference Record of the 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ACM Press, New York, NY, USA 173–185.

Dalla Preda, M. and Giacobazzi, R. (2005). Semantic-based code obfuscation by abstract interpretation. In: *Proceeding of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, Lecture Notes in Computer Science, vol. 3580, Springer-Verlag 1325–1336.

Dalla Preda, M. and Giacobazzi, R. (2009). Semantic-based code obfuscation by abstract interpretation. *Journal of Computer Security* **17** (6) 855–908.

Dalla Preda, M., Giacobazzi, R. and Visentini, E. (2008). Hiding software watermarks in loop structures. In: *Proceedings of the Static Analysis, 15th International Symposium, SAS 2008*'; , Valencia, Spain, July 16–18, 2008, Lecture Notes in Computer Science, vol. 5079 174–188.

Dalla Preda, M. and Pasqua, M. (2016). Software watermarking: A semantics-based approach. In: *Proceeding of the 6th Workshop on Numerical and Symbolic Abstract Domains (NSAD 2016)*, Edinburgh, Scotland, September 11, 2016 Elsevier – Electronic Notes in Theoretical Computer Science, 71–85. `https://doi.org/10.1016/j.entcs.2017.02.005`

Davidson, R.L. and Myhrvold, N. (1996). Method and system for generating and auditing a signature for a computer program. US Patent number 5,559,884.

Frontier-Economics (2016). The economic impacts of counterfeiting and piracy – report prepared for bascap and inta. online. Available at: https://iccwbo.org/publication/economic-impacts-counterfeiting-piracy-report-prepared-bascap-inta/.

Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A. and Waters, B. (2013). Candidate indistinguishability obfuscation and functional encryption for all circuits. In: *IACR Cryptology ePrint Archive*, 451.

Giacobazzi, R. (2008). Hiding information in completeness holes – new perspectives in code obfuscation and watermarking. In: *Proceedings of The 6th IEEE International Conferences on Software Engineering and Formal Methods (SEFM'08)*, IEEE Press. 7–20.

Giacobazzi, R. and Mastroeni, I. (2002). Compositionality in the puzzle of semantics. In: *Proceedings of the 2002 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM '02)*, Portland, Oregon, USA, January 14–15 87–97.

Giacobazzi, R. and Mastroeni, I. (2004). Abstract non-interference: Parameterizing non-interference by abstract interpretation. In: *Proceedings of the 31st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'04)*, ACM-Press 186–197.

Giacobazzi, R. and Mastroeni, I. (2008). Transforming abstract interpretations by abstract interpretation. In: Alpuente, M. (ed.), *Proceedings of The 15th International Static Analysis Symposium, SAS'08*, Lecture Notes in Computer Science; vol. 5079, Springer-Verlag, 1–17.

Giacobazzi, R. and Quintarelli, E. (2001). Incompleteness, counterexamples and refinements in abstract model-checking. In: Cousot, P. (ed.), *Proceedings of the 8th Internat. Static Analysis Symposium (SAS'01)*, Lecture Notes in Computer Science, vol. 2126, Springer-Verlag, 356–373.

Giacobazzi, R., Ranzato, F. and Scozzari, F. (2000). Making abstract interpretation complete. *Journal of the ACM* **47** (2) 361–416.

Mastroeni, I. (2005). Abstract Non-Interference - An Abstract Interpretation-based Approach to Secure Information Flow; PhD thesis; University of Verona – Dep. of Computer Science; Strada le Grazie 15, 37134, Verona (Italy).

Moskowitz, S.A. and Cooperman, M. (1996). Method for stega-cipher protection of computer code; US patent 5.745.569; Assignee: The Dice Company.

Nagra, J., Thomborson, C.D. and Collberg, C. (2002). A functional taxonomy for software watermarking. *Australian Computer Science Communications* **24** (1) 177–186.

Venkatesan, R., Vazirani, V. and Sinha, S. (2001). A graph theoretic approach to software watermarking. In: Moskowitz, I. (ed.), *Information Hiding,* Lecture Notes in Computer Science, vol. 2137, Springer, Berlin/Heidelberg, 157–168.