A Formal Framework for Property-driven Obfuscation Strategies

Mila Dalla Preda, Isabella Mastroeni, and Roberto Giacobazzi

University of Verona, Italy
{mila.dallapreda, isabella.mastroeni, roberto.giacobazzi}@univr.it

Abstract. We study the existence and the characterization of function transformers that minimally or maximally modify a function in order to reveal or conceal a certain property. Based on this general formal framework we develop a strategy for the design of the maximal obfuscating transformation that conceals a given property while revealing the desired observational behaviour.
Keywords: Program transformation, abstract interpretation, semantics, code obfuscation.

1 Introduction

The last years have seen a considerable growth in the amount of software that is distributed over the Internet and in the amount of wireless devices that dominate our society. Common classes of web applications that are part of our daily lives include e-mail clients, e-banking, e-commerce, social shopping, social networks and e-voting. In this complex scenario users need to protect their devices against malicious software attacks (e.g., software viruses and internet worms), while software developers need to protect their products against malicious host attacks that usually aim at stealing, modifying or tampering with the code in order to obtain (economic) advantages over it. In this work we consider the today challenges in protecting software against malicious host attacks.

The security scenario. A key challenge in defending code that is 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. Indeed, software developers lose the control of their applications once they are distributed to a client machine. The most common malicious host attacks against proprietary programs are malicious reverse-engineering, software piracy and software tampering. Malicious reverse engineering refers to those techniques that aim at inspecting the inner workings of software applications and then to use the so extracted information for unlawful purposes. Both software tampering and software piracy need a preliminary reverse-engineering phase in order to understand the inner working of the program that they want to tamper with or to use unauthorized. Thus, the first defense against malicious host attacks consists in impeding reverse engineering as much as possible. In this work we focus on code obfuscation, one of the most promising software solutions for code protection. Code obfuscation [3] is a program transformation that aims at transforming programs in order to make them more difficult to analyze while revealing their functionality. Besides the negative result of Barak et al. [2], that states the impossibility of an "ideal" obfuscation that obfuscates every program by revealing only the properties that can be derived from the I/O semantics, in the last decades we have seen a big effort in developing and implementing new and efficient obfuscation strategies.

The problem. It is very important to deeply understanding *what* it is possible to obfuscate of a program and *when* it is possible to obfuscate it. We believe that the development of a systematic strategy for the design of an obfuscator parameterized with respect to the program properties, both to *conceal* and to *reveal*, would be an important advance in the state of the art. In particular, it would provide a better insight in the relation between the property revealed by an obfuscator, which usually is the I/O program behavior but which can be any observable property of the program, and the property concealed, e.g., the program dependencies, the control structure, and so on.

Our contribution. We propose a general framework of program transformations that focuses on the semantic properties that a transformation either reveals or conceals of the program semantics. In this context we study the existence and characterization of maximal program transformers that maximally transform a program semantics while keeping (*revelation transformer*) or losing (*concealment transformer*) a given property. We observe that the revelation transformer finds a concrete example in the program slicing transformation which transforms a program looking for the maximal subprogram *preserving* the I/O behavior on the criterion variables [16]. On the other hand, the concealment transformer does not correspond or model any real program (semantics) transformation since it adds anything that may confuse the property to conceal, potentially losing in this way also the original program behavior. Interestingly, the combination of these transformers provide a systematic strategy for the design of obfuscating transformations parametric on the program properties to conceal and reveal.

2 Motivating Scenario: Code obfuscation

Code obfuscation. Following the standard definition of Collberg et al. [3], a code obfuscation is a *potent* program transformation $\hat{t} : \mathbb{P} \to \mathbb{P}$ that preserves the I/O behavior of programs, where potent means that $\hat{t}$ makes programs more difficult to analyse. Indeed, code obfuscation aims at concealing some information, while preserving the observational behavior of programs (i.e., program denotational semantics) for keeping them usable. A typical example of code obfuscation is the insertion of fake branches through opaque predicates [3]. A true (resp. false) opaque predicate is a predicate that always evaluates to *true* (resp. *false*). Program functionality is preserved by inserting the intended behavior in the always taken branch and buggy code in the never executed branch. In both cases the constant value of the predicate has to be difficult to deduce for an external observers that sees both branches as possible.

Semantic code obfuscation. In [8] the informal definition of code obfuscation of Collberg et al. has been generalized and placed in a theoretical framework based on program semantics and abstract interpretation. The idea is to introduce a formal model of malicious host attacks and of code transformations that allows to rigorously specify in the abstract interpretation framework the amount of "obscurity" added by a transformation to program semantics.

Modeling attackers. In this context, the typical attacker performs reverse-engineering on programs in order to steal or copy ideas. Automatic reverse-engineering techniques typically consist in static program analysis (e.g., data flow analysis, control flow analysis, alias analysis, program slicing) and dynamic program analysis (e.g., dynamic testing, profiling, program tracing). Hence, we consider two kind of attacks: one that executes the program, collects computational traces, and then analyses these traces looking for invariants, and the other that statically analyses the code. Thus, *dynamic* attacks can extract *properties* of the execution traces, while *static* attacks analyse the code looking for dynamic properties without executing the program. It is well known [5] that static analysis can be modeled in the context of abstract interpretation, where a **property** is extensionally represented as the set of all the data satisfying it and describes the abstraction of the corresponding data[1]. In particular, static analysis is performed as an *abstract execution* of programs, namely as the (fixpoint) semantics computation on the abstract data. Instead, dynamic analysis can be modeled as an approximated observation of the concrete execution since it describes partial knowledge of the real execution.
In the following, we model a property as the function φ mapping data in the minimal set satisfying it, hence φ is extensive (i.e., $\varphi(X) \supseteq X$) which means that to approximate means to add *noise*, it is idempotent since the whole approximation is added in one shot and finally it is monotone, preserving the approximation order. Namely it is an **upper closure operator** (uco) and the framework beneath is abstract interpretation [4, 5]. For instance, the property of signs of sets of integers numbers is represented by function $\varphi_{sign} : \wp(\mathbb{Z}) \to \wp(\mathbb{Z})$ that associated to every set $X \in \wp(\mathbb{Z})$ the set of integers with the same sign, for example $\varphi_{sign}(\{-3, -5\}) = \{-\infty \ldots 0\}$, while $\varphi_{sign}(\{-3, 5\}) = \mathbb{Z}$, namely we add all the noise in one shot. This formal framework, ensures that with any set of data we can always associate the *best* approximation, i.e., the minimal set satisfying the property and containing the original set of data. Hence, dynamic analysis can be seen as an instantiation of static analysis, namely as an approximated/abstracted observation of the (fixpont) semantic computation on concrete data. More formally, given the set of possible program states[2] Σ, we denote by $[\![P]\!] \in \wp(\Sigma^*)$ the (concrete) trace semantics of a program $P \in \mathbb{P}$. Thus, malicious host **attacks**, i.e., static and dynamic program analysers, are modeled as properties $\varphi \in uco(\wp(\Sigma^*))$ encoding the semantic features in which the attacker is interested. Simplifying, we can say that static attacks are modeled by abstracting the computation of the semantics on approximated inputs, i.e., $\varphi \circ [\![P]\!] \circ \varphi$[3], while dynamic attacks are modeled as (abstract) observations of concrete executions, i.e., $\varphi \circ [\![P]\!]$.

Syntactic vs semantic transformations. The formal definition of the relation between syntactic and semantic program transformations given by Cousot and Cousot [6] allows to reason on the effects that code transformations have on semantics. We consider $\mathbb{P}$ to be the domain of programs up to syntactic equivalence, where two programs P and Q are syntactically equivalent if $[\![P]\!] = [\![Q]\!]$, namely if they have the same semantics. In [6] programs are seen as abstractions of their semantics and this is formalized in

[1] For instance, the property of "being negative" is represented by the set of all negative numbers.
[2] A state specifies the content of memory and the continuation of the program.
[3] $\circ$ denotes function composition

the abstract interpretation framework. In particular, the semantic domain $\langle \wp(\Sigma^*), \subseteq \rangle$ is abstracted in the syntactic domain $\langle \mathbb{P}, \trianglelefteq \rangle$, where $\trianglelefteq$ is the order induced on programs, namely $P \trianglelefteq Q \overset{\text{def}}{=} [\![P]\!] \subseteq [\![Q]\!]$, the abstraction of $X \in \wp(\Sigma^*)$ is the semantics of the simplest program $\mathcal{P}([\![X]\!])$ (smallest number of instructions) that upper-approximates X. This means that, it is possible to associate to every syntactic program transformation $\hat{t} : \mathbb{P} \to \mathbb{P}$ its semantic counterpart $t : \wp(\Sigma^*) \to \wp(\Sigma^*)$ and vice versa: $t([\![P]\!]) = [\![\hat{t}(\mathcal{P}([\![P]\!]))]\!]$ and $\hat{t}(P) = \mathcal{P}(t([\![P]\!]))$. Equation $\hat{t}(P) = \mathcal{P}(t([\![P]\!]))$ expresses a syntactic transformation as an abstraction of the semantic transformation and it allows to derive a systematic methodology for the design of syntactic transformations from semantic ones [6]. When the semantic transformation t relies on results of undecidable problems, any effective algorithm $\hat{t}$ that tries to implement t would be an approximation of the ideal transformation $\mathcal{P} \circ t \circ [\![\cdot]\!]$, namely $t([\![P]\!]) \subseteq [\![\hat{t}(P)]\!]$ (equiv. $\mathcal{P}(t[\![P]\!]) \trianglelefteq \hat{t}(\mathcal{P}[\![P]\!])$).

In the context of code obfuscation, the formal framework of Cousot and Cousot allows to: (1) *model obfuscation potency:* reason on the effects that an obfuscation has on program semantics in order to deeply understand the semantic properties that are protected, i.e., concealed, by the obfuscation, (2) *property-driven obfuscation:* given the semantic properties to protect φ and to preserve δ, develop a semantic transformation that conceals φ and reveals δ and uses this semantic characterization as a "measure" of optimality for any syntactic transformation implementing the corresponding semantic code obfuscation. Based on the investigation of point (1) presented in [8] we address here point (2).
From now on we consider the semantic counterpart of code obfuscation, since at the semantic level we can formally understand what is concealed and what is revealed. Indeed, *studying obfuscation at the semantic level means studying its ideal behavior that would then be approximated during the implementation process.*

Modeling obfuscation potency. Every syntactic transformation $\hat{t}$ can be precisely mapped to a semantic transformation $t = [\![\cdot]\!] \circ \hat{t} \circ \mathcal{P}$, where $t[\![P]\!] = [\![\hat{t}(P)]\!]$ [6]. In [8] the authors characterize the obfuscating behavior of a program transformation by studying the effects that it has on program semantics.

Definition 1. *[8] A transformation $\hat{t} : \mathbb{P} \to \mathbb{P}$ is an obfuscation potent w.r.t. all those semantic properties $\varphi \in uco(\wp(\Sigma^*))$ that are* not preserved *by $\hat{t}$. A property is preserved by $\hat{t}$ iff $\forall P \in \mathbb{P} : \varphi([\![P]\!]) = \varphi([\![\hat{t}(P)]\!])$.*

Hence, the obfuscating behavior of a transformation $\hat{t} : \mathbb{P} \to \mathbb{P}$ can be characterized in terms of the most concrete property $\delta_{\hat{t}}$ *preserved* by $\hat{t}$ on all programs. It is possible to systematically derive $\delta_{\hat{t}}$ from $\hat{t}$ [8] and to characterize the properties concealed by $\hat{t}$ as:

$$\mathit{ObfuscatedBy}(\hat{t}) = \left\{ \varphi \in uco(\wp(\Sigma^*)) \,\middle|\, \exists X \in \wp(\Sigma^*),\ \delta_{\hat{t}}(X) \not\subseteq \varphi(X) \right\}$$

Indeed, the mapping of code transformations to the lattice of abstract interpretations allows to measure, reason and compare the potency and efficiency of different obfuscating transformations. The idea is that, the more abstract is the most concrete property preserved by a transformation, the more potent the transformation is, namely the bigger is the amount of obscurity added by the transformation. In the following, an obfuscation for a property $\varphi \in uco(\wp(\Sigma^*))$ is a semantic program transformation $t : \wp(\Sigma^*) \to \wp(\Sigma^*)$ concealing φ, i.e., such that $\exists P \in \mathbb{P} : \varphi([\![P]\!]) \neq \varphi(t([\![P]\!]))$.

```
original(){
    1. int c, nl = 0, nw = 0, nc = 0, in;
    2. in = false;
    3. while ((c = getchar()) != EOF){
        4. nc++;
        5. if(c == ' ' || c =='\n' || c == '\t') in = false;
        6. elseif(in == false) {in=true; nw++;}
        7. if(c == '\n')nl++;
    }
8. out(nl,nw,nc); }
```

```
obfuscated(){
    1. int c, nl = 0, nw = 0, nc = 0, in;
    2. in = false;
    3. while ((c = getchar()) != EOF){
        4. nc++;
        5. if(c == ' ' || c =='\n' || c == '\t') in = false;
        6. elseif(in == false) {in=true; nw++;}
        7. if(c == '\n'){if(nw <= nc)nl++};
        8. if(nl > nc) nw = nc+nl;
        9. elseif(nw > nc) nc = nw - nl;
    }
10. out(nl,nw,nc); }
```

Fig. 1. Slicing obfuscation example.

The challenge: property-driven obfuscations. The formal framework described above [8] allows to compare the potency of different obfuscation transformations and sometimes also their resilience[4] [7]. However, this theoretical investigation does not provide any insight in the design of an efficient obfuscation. Indeed, *what is still missing is a general strategy for designing an obfuscation given the specification of the property φ that it is important to protect, i.e., to conceal, and of the property δ that it is important to preserve, i.e., to reveal.* This is exactly the high level goal of our investigation. More specifically, we investigate a general framework of function transformers that aim at minimally or maximally transform a function in order to reveal or conceal a given semantic property. To this end, we first model and characterize the minimal transformations that preserve a certain property, later called *revelation*, since to preserve means to leave unchanged and therefore to *reveal* the property of the original program in the transformed/obfuscated program. Then we model and characterize the maximal transformation that loses a given property, later called *concealment* transformers, since to lose/hide a property of the original program means to change the property and, in this way, to *conceal* it. Next we show how the combination of revelation and concealment can be used for characterizing an obfuscating transformation from the specification of the property φ to be concealed and the property δ to be revealed. What we obtain is the characterization of the semantic transformation that exhibits the intended obfuscating behavior. This characterization could then be used to drive the design or to semantically analyse syntactic code obfuscations that implement the desired semantic behavior.

In the next example we describe the revealed and the concealed properties for a particular instance of code obfuscation, i.e., slicing obfuscation [15].

Example 1. Consider the word count program in Fig. 1 [15]. It takes in a block of text and outputs the number of lines (nl), words (nw) and characters (nc). The syntactic transformation $\hat{t}$ modifies line 7 by adding a true opaque predicate and adds lines 8 and 9 with false opaque predicates [15], i.e., $\hat{t}(\mathit{original}) = \mathit{obfuscated}$.

[4] The resilience measures how well a transformation holds up under attack from an automatic de-obfuscator [3]

Hence, the above transformation conceals the real data dependences of the program by adding fake dependences between program variables. Let $\mathcal{D}$ be the abstraction extracting only the (syntactic) dependencies among variables in a program (e.g, by means of program dependence graphs), then $\mathcal{D}([\![\textit{original}]\!]) \neq \mathcal{D}([\![\textit{obfuscated}]\!])$, since for instance line 8 adds the dependence of nw from nl. In this way, an external observer is not able to derive, from the analysis of the obfuscated program, the precise variables dependences of the original program, overstimating the information. Meanwhile, the proposed obfuscation reveals/preserves the input-output abstraction $\mathcal{I}$ of the program semantics, i.e., $\mathcal{I}([\![\textit{original}]\!]) = \mathcal{I}([\![\textit{obfuscated}]\!])$. Namely this information about the original program can be precisely derived by the observation of its obfuscated version. Hence, the obfuscation $\hat{t}$, while revealing $\mathcal{I}$ is potent w.r.t. $\mathcal{D}$.

3 Modeling Revelation

In this section, we study the characterization of function transformers that minimally modify a function in order to make it reveal a given property. To this end we consider the *complete lattice* $\langle L, \leq, \vee, \wedge, \top, \bot\rangle$[5] and the complete lattice of functions over L $\langle L \to L, \dot{\leq}, \dot{\vee}, \dot{\wedge}, \lambda x.\bot, \lambda x.\top\rangle$, where functions are ordered point-wise, i.e., $f \dot{\leq} g$ iff $\forall x \in L.\ f(x) \leq g(x)$. Given a function $f : L \to L$ and a property δ modeled as abstraction of L, i.e., $\delta \in uco(L)$[6], we define the two function transformers $\mathcal{R}^{\uparrow}_{\delta}$ and $\mathcal{R}^{\downarrow}_{\delta}$ that aim at computing the two functions closer to f in the domain $L \to L$, respectively from above and from below, and that reveal the property δ.

Definition 2 (Minimal revelation). *Let $\delta \in uco(L)$ with L complete lattice. $\mathcal{R}^{\uparrow}_{\delta}, \mathcal{R}^{\downarrow}_{\delta} : (L \to L) \to (L \to L)$ are the minimal revelations from above and from below for δ.*

$$\mathcal{R}^{\uparrow}_{\delta}(f) \overset{\text{def}}{=} \dot{\bigwedge} \left\{ g : L \to L \,\middle|\, \forall x \in L.\ \delta(x) = \delta(g(x)), f \dot{\leq} g \right\}$$
$$\mathcal{R}^{\downarrow}_{\delta}(f) \overset{\text{def}}{=} \dot{\bigvee} \left\{ g : L \to L \,\middle|\, \forall x \in L.\ \delta(x) = \delta(g(x)), g \dot{\leq} f \right\}$$

It is interesting to study $\mathcal{R}^{\uparrow}_{\delta}(f)$ and $\mathcal{R}^{\downarrow}_{\delta}(f)$ when: (*) $\mathcal{R}^{\uparrow}_{\delta}(f)$ and $\mathcal{R}^{\downarrow}_{\delta}(f)$ do *not trivially* transform f, i.e., they do not transform f in the top or the bottom of the functional domain; (**) $\mathcal{R}^{\uparrow}_{\delta}(f)$ and $\mathcal{R}^{\downarrow}_{\delta}(f)$ reveal the property δ. In order to guarantee that the transformer always characterizes the *minimal* obfuscation, $\mathcal{R}^{\uparrow}_{\delta}$ has to be monotone and extensive (approximating from above), and it has to be idempotent. Hence, $\mathcal{R}^{\uparrow}_{\delta}$ has to be an uco of the lattice $L \to L$, and dually $\mathcal{R}^{\downarrow}_{\delta}$ has to be a **lower closure operator** (*lco*[7]). The following result identifies the conditions on the relation between f and δ that guarantees the condition (*) for the minimal revelation (from above and from below).

Proposition 1. *Let L be a complete lattice, $f : L \to L$ and $\delta \in uco(L)$, we have that*

1. $\mathcal{R}^{\uparrow}_{\delta}(f) \neq \lambda x.\top$ *iff* $\forall x \in L.\ \delta(f(x)) \leq \delta(x)$ *iff* $\exists y \geq f(x).\ \delta(y) = \delta(x)$*;*
2. $\mathcal{R}^{\downarrow}_{\delta}(f) \neq \lambda x.\bot$ *iff* $\exists y \leq f(x).\ \delta(y) = \delta(x)$*;*

[5] A **complete lattice** is a partial ordered set with least upper bound and greatest lower bound existing for each set of elements.

[6] The domain $uco(L)$ is the complete lattice of all abstractions of L, modeling properties on L.

[7] A function is an lco if it is monotone, reductive ($f(x) \leq x$) and idempotent.

Thus, we can find a non trivial simplification of f revealing δ, iff function f "loses" something of the property δ of the original element. Analogous for the refinement of f. The following result proves that $\mathcal{R}^{\uparrow}_{\delta}(f)$ and $\mathcal{R}^{\downarrow}_{\delta}(f)$ are precisely the minimal transformers inducing the revelation of the property δ, namely they satisfy (**).

Theorem 1. *Let L be a complete lattice, $f : L \rightarrow L$ and $\delta \in uco(L)$. (1) If δ meet-uniform*[8] *[11], then $\mathcal{R}^{\uparrow}_{\delta} \in uco(L \rightarrow L)$; (2) $\mathcal{R}^{\downarrow}_{\delta} \in lco(L \rightarrow L)$.*

In this theorem we introduce the notion of uniformity, in particular of meet uniformity which means that the greatest lower bound (glb) operation *preserves* the property δ, namely the glb of elements with same property δ has the same property δ. This precisely models the fact that we can find the best approximation of x from below sharing the same property δ of x. Thus, given a function $f : L \rightarrow L$, we have that $\mathcal{R}^{\uparrow}_{\delta}(f)$ returns the closest function that is greater than f and that reveals the property δ. For this reason, we refer to $\mathcal{R}^{\uparrow}_{\delta}(f)$ as the *minimal revelation from above* of f w.r.t. δ. Dually, we have that given a function $f : L \rightarrow L$, then $\mathcal{R}^{\downarrow}_{\delta}(f)$ returns the closest function that is smaller than f and that reveals the property δ. Analogously, we refer to $\mathcal{R}^{\downarrow}_{\delta}(f)$ as the *minimal revelation from below* of f w.r.t. δ. Given a property $\delta \in uco(L)$ we define the *kernel* of δ with respect to an element $x \in L$ as $\mathtt{K}_{\delta}(x) \overset{\text{def}}{=} \{\, y \mid \delta(x) = \delta(y) \,\}$. Then we use the shorthand $\mathtt{K}^{\wedge}_{\delta}(x) \overset{\text{def}}{=} \wedge \{\, y \mid \delta(x) = \delta(y) \,\}$, and $\mathtt{K}^{\vee}_{\delta}(x) = \vee \{\, y \mid \delta(x) = \delta(y) \,\}$. Note that $\mathtt{K}^{\vee}_{\delta}(x) = \delta(x)$ being δ an uco.

Theorem 2. *Let L be a complete lattice, $f : L \rightarrow L$ and $\delta \in uco(L)$. In the hypotheses of Prop.* 1 *we have that:*

1. *If δ meet-uniform then $\mathcal{R}^{\uparrow}_{\delta}(f) = \lambda x.\, \mathtt{K}^{\wedge}_{\delta}(x) \vee f(x)$;*
2. *$\mathcal{R}^{\downarrow}_{\delta}(f) = \lambda x.\, \delta(x) \wedge f(x)$.*

The above characterization says that the minimal revelation from above w.r.t. δ of f is the function that associates with each x the least upper bound between $f(x)$ and the smallest element that preserves δ on x. Indeed, this corresponds to adding the minimal amount of information to $f(x)$ in order to make it preserve the property δ on x. An analogous reasoning holds for the minimal revelation from below.

4 Modeling Concealment

In this section, we investigate the function transformers that maximally modify a function in order to hide/conceal a given property. The idea is to find the farthest functions from f, on the lattice $L \rightarrow L$, that lose a certain property φ, and these are characterized as the farthest functions having the same revelation transformer of f w.r.t φ. These transformers are precisely the adjoints [14] of the revelation ones, which while transforming, in order not to reach the top, keep the strong bind with the original function consisting in having the same revelation transformer.

[8] δ is **meet uniform on** x iff $\delta(\wedge \{\, y \mid \delta(x) = \delta(y) \,\}) = \delta(x)$, δ is meet uniform if $\forall x \in L$ we have δ **meet uniform** on x.

Definition 3 (Maximal concealment). *Let L be a complete lattice, $f : L \to L$ and $\varphi \in uco(L)$. We define $\mathcal{C}^{\downarrow}_{\varphi}, \mathcal{C}^{\uparrow}_{\varphi} : (L \to L) \to (L \to L)$ as:*

$$\mathcal{C}^{\downarrow}_{\varphi}(f) \overset{def}{=} \dot{\bigwedge} \left\{ g : L \to L \,\middle|\, \mathcal{R}^{\uparrow}_{\varphi}(f) = \mathcal{R}^{\uparrow}_{\varphi}(g) \right\}$$
$$\mathcal{C}^{\uparrow}_{\varphi}(f) \overset{def}{=} \dot{\bigvee} \left\{ g : L \to L \,\middle|\, \mathcal{R}^{\downarrow}_{\varphi}(f) = \mathcal{R}^{\downarrow}_{\varphi}(g) \right\}$$

These transformers are interesting when $\mathcal{C}^{\downarrow}_{\varphi}(f)$ and $\mathcal{C}^{\uparrow}_{\varphi}(f)$ have the same revelation (from above or from below) of f, and this clearly is not always true. In particular, the maximal concealments are defined as the adjoints of the revelation transformers and these adjoint transformers do not always exists. Recall that an upper closure admits adjoint, which is a lower closure [14], if it is meet-uniform [9], while, dually, a lower closure admits adjoint, which is an upper closure, if it is join-uniform[9]. It is worth noting, that uniformity is a *local* property, namely a function g may be uniform on a particular input x, i.e., for meet-uniformity $g(x) = g(\wedge \{ y \,|\, g(x) = g(y) \})$, and fail uniformity on other inputs. In this case we say that g is (meet)-uniform on x and w.r.t. x we can find the adjoint, i.e., $\bigwedge \{ y \,|\, g(x) = g(y) \}$. Hence, we have that if $\mathcal{R}^{\uparrow}_{\varphi}$ on a function f is meet-uniform then $\mathcal{C}^{\downarrow}_{\varphi}(f)$ is the minimum, namely it is an uco and behaves as adjoint of $\mathcal{R}^{\uparrow}_{\varphi}$, and dually if $\mathcal{R}^{\downarrow}_{\varphi}$ is join-uniform on f then $\mathcal{C}^{\uparrow}_{\varphi}(f)$ is the maximum.

Theorem 3. *Let L be a complete Boolean algebra*[10]*, $\varphi \in uco(L)$. For each $f : L \to L$ satisfying the hypotheses of Prop.* 1 *we have that:*

1. *If φ is meet-uniform then $\mathcal{R}^{\uparrow}_{\varphi} \in uco(L \to L)$ is meet-uniform on f;*
2. *$\mathcal{R}^{\downarrow}_{\varphi} \in lco(L \to L)$ is join-uniform on f.*

Observe that if function f does not satisfy the hypotheses of Prop. 1 then, for instance, $\mathcal{R}^{\uparrow}_{\varphi}(f) = \lambda x. \top$. In this case, $\mathcal{R}^{\uparrow}_{\varphi}$ is not meet-uniform on f since the glb of the functions g_i having the same revelation $\mathcal{R}^{\uparrow}_{\varphi}(g_i) = \lambda x. \top$ may have a different revelation.

Example 2. Suppose $L = \wp(\mathbb{N})$, $f(X) = \{ x+1 \,|\, x \in X \}$, and consider the set of functions $\{ g_i \,|\, g_i = \lambda X. \{ x + i \,|\, x \in X \}, \ i \text{ odd} \}$, and suppose $\delta = \mathit{Par}$ characterizing the parity of integers, for instance $\mathit{Par}(X) = \text{Even}$ iff $\forall x \in X.\ x$ even. Then it is trivial to note that for each i, and for each X of even numbers, we have that for all Y such that $\mathit{Par}(Y) = \mathit{Par}(X) = \text{Even}$, $Y \not\subseteq g_i(X)$ since Y must contain only even numbers, while $g_i(X)$ contains only odd numbers. On the other hand, $\bigcap_i g_i(X) = \varnothing \subseteq \mathit{Par}(X)$ since, for instance, $g_i(\{2,4\}) \cap g_i(\{6,7\}) = \varnothing$.

The following result provides a characterization of the maximal concealment transformers in terms of the kernel of the property to hide.

Proposition 2. *Let L be a complete Boolean algebra, $\varphi \in uco(L)$. For each $f : L \to L$ satisfying the hypotheses of Prop.* 1 *we have that:*

1. *If φ is meet-uniform then $\mathcal{C}^{\downarrow}_{\varphi}(f) = \lambda x. \bigvee \{ z \,|\, z \leq f(x),\ z \wedge \mathtt{K}^{\wedge}_{\varphi}(x) = \bot \}$;*
2. *$\mathcal{C}^{\uparrow}_{\varphi}(f) = \lambda x. \bigwedge \{ z \,|\, z \geq f(x), z \vee \varphi(x) = \top \}$;*

[9] Join uniformity is the dual notion of meet uniformity.
[10] A complete Boolean algebra is a complete complemented and distributive lattice [13].

Hence, the maximal concealment from below w.r.t. property φ of f is the function that associates with each element x the greatest element, smaller than $f(x)$, that loses any information about the property φ on x, namely the greatest element that is the complement w.r.t. $\bot$ of the smallest element that preserves φ on x (dually for $\mathfrak{C}^{\uparrow}_{\varphi}(f)$). These characterizations turn out to be particularly meaningful when interpreted on a powerset domain ordered by set inclusion.

Corollary 1. *Let D be a complete lattice, $L = \wp(D)$ and $\varphi \in uco(L)$. For each $f : L \leftarrow L$ satisfying the hypotheses of Prop.* 1

1. *If φ is meet-uniform then $\mathfrak{C}^{\downarrow}_{\varphi}(f) = \lambda X.\, f(X) \smallsetminus \mathtt{K}^{\cap}_{\varphi}(X)$;*
2. *$\mathfrak{C}^{\uparrow}_{\varphi}(f) = \lambda X.\, f(X) \cup (D \smallsetminus \varphi(X))$.*

Indeed, for any $X \in \wp(D)$, $\mathfrak{C}^{\downarrow}_{\varphi}(f)(X)$ can be obtained by erasing from $f(X)$ the smallest set having the same property φ than X, namely the minimal information that leads to property $\varphi(X)$. On the other side, $\mathfrak{C}^{\uparrow}_{\varphi}(f)(X)$ can be obtained by adding to $f(X)$ the biggest element in D that does not share the same property $\varphi(X)$, namely by adding the maximal amount of obscurity to $f(X)$ w.r.t. φ.

5 Characterizing property-driven obfuscations

In this section we discuss how by combining revelation and concealment transformers we can characterize an obfuscation starting from the specification of the property $\delta \in uco(\wp(\Sigma^*))$ to reveal and of the property $\varphi \in uco(\wp(\Sigma^*))$ to conceal. The concealment transformer provides an important understanding of what we have to add in order to *obfuscate* a given property: it characterizes the set of all the possible computations that we have to add in order to gain confusion on the observation of φ. At this point the revelation transformer allows us to refine this information by *avoiding* all the computations that do not preserve a property δ.

Proposition 3. *The* maximal *property-driven obfuscation strategy, concealing φ meet-uniform and revealing δ, is $\mathfrak{O}^{\delta}_{\varphi}(f) = \lambda X.\, X \cup (\delta(X) \cap (\Sigma^* \smallsetminus \mathtt{K}^{\cap}_{\varphi}(X)))$ for f in the hypotheses of Prop.* 1,

Starting from the identity, we first lose the information concerning $\varphi(X)$ by taking those traces with a different φ property w.r.t. X, then we guarantee the preservation of $\delta(X)$ by selecting only those traces which have the same δ property of X. Finally, we add all the original traces in order to guarantee that the original semantics is preserved. In order to guarantee the existence of the *maximal* property-driven obfuscation we have the *meet-uniformity* hypothesis on the property to conceal. In general, properties may also fail meet-uniformity, in this case the obfuscation strategy we obtain loses maximality and also unicity. Indeed, if φ is not meet-uniform, then it does not exist the minimal set in $\mathtt{K}_{\varphi}(X)$. However this is not a problem since we can choose any element on this set, for instance $\varphi(X)$ itself, which in general may not be minimal, but still allows to obtain an obfuscation strategy. In particular, if we choose $Y \in \mathtt{K}_{\varphi}(X)$ then $\Sigma^* \smallsetminus Y$ is not maximal, but it still adds confusion on the property φ. In this case, we denote this choice inside $\mathtt{K}_{\varphi}(X)$ as $\hat{\mathtt{K}}_{\varphi}(X)$ and the corresponding concealment $\hat{\mathfrak{C}}$. Note that, this

weakening is important also for computational issues, indeed the $\mathtt{K}_\varphi(X)$ can be computed only for finite domains, while in general (as it happens for the real obfuscations) it is easier to generate one element of this set, namely $\hat{\mathtt{K}}_\varphi(X)$.
Moreover, we can observe that the definition of $\mathfrak{O}^\delta_\varphi$ provided in Proposition 3 is quite strong in the context of code obfuscation not only for the meet-uniformity requirement (that can be weakened) but also because it adds the whole semantics $\llbracket P \rrbracket$ to the semantics of the obfuscated program. This implies that the original semantics has to be *contained* in the obfuscated program, which is a strong requirement since it is sufficient to contain the *abstract* semantics $\delta(\llbracket P \rrbracket)$. This observation is important also to partially fill the gap between the proposed strategy and existing code obfuscations which *transform* also traces of P, namely obfuscations such that $\llbracket \hat{t}(P) \rrbracket \not\supseteq \llbracket P \rrbracket$. The obfuscating component of $\mathfrak{O}^\delta_\varphi$ is the set we add to X, namely $\delta(X) \cap (\Sigma^* \smallsetminus \mathtt{K}^\cap_\varphi(X))$, while the preservation condition forces any obfuscated version of X to stay inside $\mathtt{K}_\delta(X)$. Hence, we can provide the following weakened characterization of code obfuscations.

Proposition 4. *Let* $\varphi, \delta \in uco(\wp(\Sigma^*))$. $\hat{\mathfrak{O}}^\delta_\varphi(X)$ *is a property-driven obfuscation of* X *iff* $\hat{\mathfrak{O}}^\delta_\varphi(X) \in \mathtt{K}_\delta(X) \cap (\wp(\Sigma^*) \smallsetminus \mathtt{K}_\varphi(X))$ *iff* $\hat{\mathfrak{O}}^\delta_\varphi(X) \in \mathtt{K}_\delta(X) \wedge \hat{\mathfrak{O}}^\delta_\varphi(X) \subseteq \Sigma^* \smallsetminus \hat{\mathtt{K}}_\varphi(X)$. *If* φ *is meet-uniform then we take as* $\hat{\mathtt{K}}_\varphi(X)$ *precisely* $\mathtt{K}^\cap_\varphi(X)$.

Hence, in this case the *maximal* property-driven obfuscation is the maximal subset of $\mathtt{K}_\delta(X)$ contained in $\Sigma^* \smallsetminus \hat{\mathtt{K}}_\varphi(X)$. Finally, the next result shows the relation between the existence condition of property-driven obfuscations and the semantic code obfuscation characterization provided in [8] (see Section 2). In particular, coherently with [8], a property-driven obfuscator, concealing φ and revealing δ, exists iff δ does not imply φ.

Corollary 2. *A property-driven obfuscator* $\hat{\mathfrak{O}}^\delta_\varphi(X)$ *exists, by Proposition* 4*, iff we have* $\mathtt{K}_\delta(X) \cap (\wp(\Sigma^*) \smallsetminus \mathtt{K}_\varphi(X)) \neq \varnothing$ *iff* $\exists Y \in \wp(\Sigma^*).\ \delta(Y) \not\subseteq \varphi(Y)$.

Example 3. Let $\mathit{Sign}(\wp(\mathbb{Z})) = \{\top, 0+, 0, 0-, +, -, \varnothing\}$ be the property of signs. This property is not meet-uniform, for instance $\mathit{Sign}(\{1,2\}) = + = \mathit{Sign}(\{3,4\})$ and the intersection is such that $\mathit{Sign}(\varnothing) = \varnothing$. Analogously, we can prove that its lift on $\wp(\Sigma^*)$[11] is also not meet-uniform. Hence, in this case, we have to use $\hat{\mathtt{K}}$, and given $X \in \wp(\Sigma^*)$, for simplicity as $\hat{\mathtt{K}}_{\mathit{Sign}}(X)$ we take $\mathit{Sign}(X) \in \mathtt{K}_{\mathit{Sign}}(X)$ obtaining so far a weakened version $\hat{\mathfrak{C}}^\uparrow_{\mathit{Sign}}$ of $\mathfrak{C}^\uparrow_{\mathit{Sign}}$. Let $\mathfrak{I}$ be the I/O property[12]. Let us consider, $\varphi = \mathit{Sign}$, $\delta = \mathfrak{I}$ and $X = \llbracket P \rrbracket \in \wp(\Sigma^*)$, where P is:

```
P :
  1. input x;
  2. y := 2;
  3. while x > 0 do
       4. y := y + 2;
       5. x := x − 1
     endw
  6. output y;
  7. end
```

```
t̂(P) :
  1. input x;
  2. y := −2;
  3. while x > 0 do
       4. y := y + 2;
       5. x := x − 1
     endw
  6′. if x = 0 then y := y + 4;
  6. output y;
  7. end
```

[11] Given $X \in \wp(\Sigma^*)$ we define $\mathit{Sign}(X) \overset{\text{def}}{=} \cup \{\sigma \in \Sigma^* \mid \forall i.\ \sigma_i \in \mathit{Sign}(\{\ \sigma'_i \mid \sigma' \in X\ \})\}$, where σ_i is the i-th state of the trace σ.

[12] $\mathfrak{I}(X) = \{\ \sigma \mid \exists \sigma' \in X.\ \sigma_\dashv = \sigma'_\dashv,\ \sigma_\vdash = \sigma'_\vdash\ \}$, where given $\sigma \in \Sigma^*$ we denote $\sigma_\vdash$ the first state of σ and with $\sigma_\dashv$ the last one, equal to the undefined value $\perp$ if σ is infinite

In sake of simplicity, suppose $x \geq 0$. Then we have that

$$\begin{aligned}
\textit{Sign}(\llbracket P \rrbracket) &= \left\{ \sigma \,\middle|\, \exists n \geq 0.\, \sigma \in \langle 0+, \bot \rangle \to \langle 0+, + \rangle \to \langle +, + \rangle^n \to \langle 0, + \rangle \right\} \\
\mathcal{I}(\llbracket P \rrbracket) &= \left\{ \sigma \,\middle|\, \exists \sigma' \in \llbracket P \rrbracket.\, \sigma_{\dashv} = \sigma'_{\dashv},\ \sigma_{\vdash} = \sigma'_{\vdash} \right\} \\
\mathtt{K}_{\mathcal{I}}(\llbracket P \rrbracket) &= \left\{ Y \,\middle|\, \mathcal{I}(Y) = \mathcal{I}(\llbracket P \rrbracket) \right\} \\
\hat{\mathfrak{O}}^{\mathcal{I}}_{\textit{Sign}}(\llbracket P \rrbracket) &= \llbracket P \rrbracket \cup (\mathcal{I}(\llbracket P \rrbracket) \cap \left\{ \sigma \,\middle|\, \sigma \notin \textit{Sign}(\llbracket P \rrbracket) \right\})
\end{aligned}$$

Then a property-driven obfuscation $\hat{\mathfrak{O}}^{\mathcal{I}}_{\textit{Sign}}(\llbracket P \rrbracket)$ has to be such that $\hat{\mathfrak{O}}^{\mathcal{I}}_{\textit{Sign}}(\llbracket P \rrbracket) \in \mathtt{K}_{\mathcal{I}}(\llbracket P \rrbracket)$ and $\hat{\mathfrak{O}}^{\mathcal{I}}_{\textit{Sign}}(\llbracket P \rrbracket) \subseteq \Sigma^* \smallsetminus \textit{Sign}(X)$. Hence, the semantics of the obfuscation must be a set of traces such that the I/O behavior is precisely that of P, but which sign property changes. Trivially we can prove that such a set always exists and therefore that the hypothesis of Proposition 4 are satisfied. This means that we can make a program transformation that during the computation changes the sign of the variable y, for instance, without changing the final value. At this point any program whose semantics satisfies these conditions may be considered as an obfuscation of P. For instance, consider $\hat{t}(P)$ above on the right, we have $\mathcal{I}(\llbracket P \rrbracket) = \mathcal{I}(\llbracket \hat{t}(P) \rrbracket)$ and

$$\textit{Sign}(\llbracket \hat{t}(P) \rrbracket) = \left\{ \sigma \,\middle|\, \exists n.\, \sigma \in \langle 0+, \bot \rangle \to \langle 0+, - \rangle \to \langle +, 0+ \rangle \to \langle 0, + \rangle \right\} \neq \textit{Sign}(\llbracket P \rrbracket).$$

We believe that it is not difficult to provide an intuition of how we can interpret well-known code obfuscations as property-driven obfuscations whose detailed development deserve further work. Consider for instance the following examples.
Consider again the example of CFG and opaque predicates introduced before. In this case, by Proposition 4 a property-driven obfuscator is any semantics whose I/O abstraction, $\mathcal{I}$, is the same as the one of the original program, but corresponding to a different CFG, namely different $\mathcal{G}$ abstraction w.r.t. the original program. Namely, $\hat{t}(P)$ is a property-driven obfuscator if $\llbracket \hat{t}(P) \rrbracket \in \mathtt{K}_{\mathcal{I}}(\llbracket P \rrbracket)$ and $\llbracket \hat{t}(P) \rrbracket \subseteq \Sigma^* \smallsetminus \hat{\mathtt{K}}_{\mathcal{G}}(\llbracket P \rrbracket)$ (it is easy to believe that $\mathcal{G}$ is not meet-uniform). It is clear that the precise obfuscation algorithm will choose a particular program $\hat{t}(P)$ whose abstract I/O semantics is the same as P but with a different CFG.
Consider Ex. 1 about *slicing obfuscation*. Then, $\llbracket \textit{obfuscated} \rrbracket = \hat{\mathfrak{O}}^{\mathcal{I}}_{\mathcal{D}}(\llbracket \textit{original} \rrbracket)$, and we observe that $\hat{\mathfrak{O}}^{\mathcal{I}}_{\mathcal{D}}(\llbracket \textit{original} \rrbracket) \in \mathtt{K}_{\mathcal{I}}(\llbracket \textit{original} \rrbracket)$ and that $\hat{\mathfrak{O}}^{\mathcal{I}}_{\mathcal{D}}(\llbracket \textit{original} \rrbracket) \subseteq \Sigma^* \smallsetminus \hat{\mathtt{K}}_{\mathcal{D}}(\llbracket \textit{original} \rrbracket)$. Namely, as explained before, it is a program whose semantics has the same I/O abstraction of the original program, but whose program variable dependencies are changed.

6 Future Work

We have proposed a property-driven characterization of code obfuscation obtained by composing revelation and concealment transformers. By instantiating this characterization to a specific program P we obtain a specification $\mathcal{R}^{\downarrow}_{\delta}(\mathcal{C}^{\uparrow}_{\varphi}(\llbracket P \rrbracket))$ of the semantics of the obfuscation of P. We plan to investigate the existence of this obfuscated program in terms of the interplay between the two considered properties φ and δ in the semantics of P. Our intuition is that it is possible to obfuscate P revealing φ while concealing δ only if these properties are somehow independent in the semantics of P.

There exists another semantics-based notion of obfuscation [10] that specifies transformation potency on the abstract program semantics, instead of on the abstraction of the concrete semantics has we have done. We plan to study revelation and concealment transformers also on this more general notion of semantic obfuscation.

From the theoretical point of view, this can be related with other property-driven function transformers, such as the complete shells and cores [12], the incomplete transformers [10], the transformers towards additivity [1], obtaining so far a framework for property-driven transformers parametric on the property to guarantee.

References

1. H. Andéka, R. J. Greechie, and G. E. Strecker. On residuated approximations. In *Categorical Methods in Computer Science*, volume 393 of *LNCS*, pages 333 – 339, 1989.
2. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO ’01: 21st Internat. Cryptology Conference on Advances in Cryptology*, pages 1–18. Springer-Verlag, 2001.
3. C. Collberg, C. D. Thomborson, and D. Low. Manufactoring cheap, resilient, and stealthy opaque constructs. In *ACM Symp. on Principles of Programming Languages* (*POPL ’98*), pages 184–196. ACM Press, 1998.
4. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *ACM Symp. on Principles of Programming Languages* (*POPL ’77*), pages 238–252. ACM Press, 1977.
5. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *ACM Symp. Principles of Programming Languages* (*POPL ’79*), pages 269–282. ACM Press, 1979.
6. P. Cousot and R. Cousot. Systematic design of program transformation frameworks by abstract interpretation. In *ACM Symp. on Principles of Programming Languages*, pages 178–190. ACM Press, 2002.
7. M. Dalla Preda and R. Giacobazzi. Control code obfuscation by abstract interpretation. In *Software Engineering and Formal Methods (SCAM’05)*, pages 301–310. IEEE Computer Society, 2005.
8. M. Dalla Preda and R. Giacobazzi. Semantics-based code obfuscation by abstract interpretation. *Journal of Computer Security*, 17(6):855–908, 2009.
9. R. Giacobazzi and I. Mastroeni. Transforming abstract interpretations by abstract interpretation. In *International Static Analysis Symposium, SAS’08*, volume 5079 of *LNCS*, pages 1–17. Springer-Verlag, 2008.
10. R. Giacobazzi and I. Mastroeni. Making abstract interpretation incomplete - modeling the potency of obfuscation. In *19th International Static Analysis Symp. (SAS ’12)*, volume 7460 of *LNCS*, pages 129 – 145, 2012.
11. R. Giacobazzi and F. Ranzato. Uniform closures: order-theoretically reconstructing logic program semantics and abstract domain refinements. *Inform. and Comput.*, 145(2):153–190, 1998.
12. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretation complete. *Journal of the ACM*, 47(2):361–416, March 2000.
13. G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer-Verlag, 1980.
14. M. F. Janowitz. Residuated closure operators. *Portug. Math.*, 26(2):221–252, 1967.
15. A. Majumdar, S. J. Drape, and C. D. Thomborson. Slicing obfuscations: design, correctness, and evaluation. In *ACM workshop on Digital Rights Management*, pages 70–81. ACM, 2007.
16. M. Weiser. Program slicing. In *ICSE ’81: Proceedings of the 5th international conference on Software engineering*, pages 439–449. IEEE Press, 1981.