Transforming Abstract Interpretations by Abstract Interpretation

New Challenges in Language-based Security

Roberto Giacobazzi and Isabella Mastroeni

Dipartimento di Informatica - Università di Verona - Verona, Italy
`(roberto.giacobazzi|isabella.mastroeni)@univr.it`

Abstract. In this paper we exploit abstract interpretation for transforming abstract domains and semantics. The driving force in both transformations is making domains and semantics, i.e. abstract interpretations themselves, complete, namely precise, for some given observation. We prove that a common geometric pattern is shared by all these transformations, both at the domain and semantic level. This pattern is based on the notion residuated closures, which in our case can be viewed as an instance of abstract interpretation. We consider these operations in the context of language-based security, and show how domain and semantic transformations model security policies and attackers, opening new perspectives in the model of information flow in programming languages.

1 Introduction

Abstract interpretation [6] is not only a theory for the approximation of the semantics of dynamic systems, but also a way of thinking information and computation. From this point of view a program can be seen as an abstraction transformer, generalising Dijkstra's predicate transformer semantics, by considering abstractions as the objects of the computation: The way a program transforms abstractions tells us a lot about the way information flows and is manipulated during the computation. Abstract non-interference [13] is an example of this use of abstract interpretation, capturing precisely the intuition that in order to understand *who* can attack the code and *what* information flows, we have to consider programs as abstraction transformers, attackers as abstract interpretations, and secrets as data properties, which are abstractions again. This view lets out new possibilities for abstract interpretation use, e.g. in security, code design and obfuscation, as well as posing problems concerning the methods according to which these transformations are studied. Even if clearly previewed in the early stages of abstract interpretation [8], this approach to the use of abstract interpretation is still relatively unexplored.

In this paper we show that the standard theory of abstract interpretation, based on the so called adjoint-framework of Galois connections, can be directly applied to reason about operators that transform abstract domains and semantics, yet providing new formal methodologies for the systematic design of abstract domain refinements and program transformations (in our case program deformations). We first show that most domain transformers can be viewed as suitable problems of completeness w.r.t. some given semantic feature of the considered programming language. This observation has indeed an intuitive justification: The goal of refining a domain is always that of improving its precision with respect to some basic semantic operation (e.g., arithmetic operations, unification in logic programs, data structure operations in simple and higher-order types, and temporal operators in model checking). Analogously, simplifying domains corresponds to the dual operation of reducing precision with respect to analogous semantic operations. We show that most well known domain transformers can be interpreted in this way and that the relation between refinement and simplification on domains is indeed an instance of the same abstract interpretation framework lifted to higher types, i.e. where the objects of abstraction/concretization are abstract domains or semantics rather then computational objects.

$$\begin{tikzcd}
\text{Core} \arrow[r, leftrightarrow, "+" near start, "-" near end] \arrow[d, leftrightarrow, "-" near start, "+" near end] & \text{Expander} \\
\text{Shell} \arrow[r, leftrightarrow, "-" near start, "+" near end] & \text{Compressor}
\end{tikzcd}$$

Residuated closures [2, 22] provide the instance of Galois connections on closure operators, i.e. on abstract domains. We prove that standard refinement/simplification (also called shell/core) and expansion/compression are residuated closures on abstract domains. In particular it is always possible to derive an operation which reduces a given domain by considering either the right (+) or left (−) adjoint of a domain refinement. We show that the meaning of these transformations are deeply different: while the left adjoint of a refinement is always a simplification which keeps completeness (core), the right one moves towards maximal incompleteness by reducing the domain (compression). Similarly we can always refine a domain by considering the same adjoints of a domain simplification. In this case the left adjoint always moves towards maximal incompleteness by refining the domain (expander) while the right one refines the domain yet keeping completeness (shell), as depicted above. We prove that this construction can be generalised to arbitrary semantic transformers, which in view of [10] may correspond to code transformations, where instead of transforming domains we transform semantics. The result is a unique geometric interpretation of abstract domain and semantic transformers where both notions can be systematically derived by considering standard abstract interpretation methods. We apply these transformations to the case of language-based security, by modelling security policies and attackers as suitable abstract domain and semantic transformations.

2 Abstract domains individually and collectively

We consider standard abstract domain definition as formalised in [6] and [8] in terms of Galois connections. It is well known that this is a restriction for abstract interpretation because relevant abstractions do not form Galois connections and Galois connections are not expressive enough for modelling dynamic fix-point approximation [9]. Formally, if $\langle C, \leq, \top, \bot, \vee, \wedge \rangle$ is a complete lattice, monotone functions $\alpha : C \xrightarrow{\text{m}} A$ and $\gamma : A \xrightarrow{\text{m}} C$ form an *adjunction* or a *Galois connection* if for any $x \in C$ and $y \in A$: $\alpha(x) \leq_A y \Leftrightarrow x \leq_C \gamma(y)$. α [resp. γ] is the *left-* [*right-*]*adjoint* to γ [α] and it is additive [co-additive], i.e. it preserves *lub*'s [*glb*] of all subsets of the domain (emptyset included). The right adjoint of a function α is $\alpha^+ \overset{\text{def}}{=} \lambda x.\ \bigvee \{\ y \,|\, \alpha(y) \leq x\ \}$. Conversely the left adjoint of γ is $\gamma^- \overset{\text{def}}{=} \lambda x.\ \bigwedge \{\ y \,|\, x \leq \gamma(y)\ \}$ [8]. Abstract domains can be also equivalently formalized as closure operators on the concrete domain. An *upper [lower] closure operator* $\rho : P \longrightarrow P$ on a poset P is monotone, idempotent, and extensive: $\forall x \in P.\ x \leq_P \rho(x)$ [reductive: $\forall x \in P.\ x \geq_P \rho(x)$]. Closures are uniquely determined by their fix-points $\rho(C)$. The set of all upper [lower] closure operators on P is denoted by $uco(P)$ [$lco(P)$]. The *lattice of abstract domains* of C, is therefore isomorphic to $uco(C)$, (cf. [6, Section 7] and [8, Section 8]). Recall that if C is a complete lattice, then $\langle uco(C), \sqsubseteq, \sqcup, \sqcap, \lambda x.\top, id\rangle$ is a complete lattice [24], where $id \overset{\text{def}}{=} \lambda x.x$ and for every $\rho, \eta \in uco(C)$, $\rho \sqsubseteq \eta$ iff $\forall y \in C.\ \rho(y) \leq \eta(y)$ iff $\eta(C) \subseteq \rho(C)$. A_1 is more precise than A_2 (i.e., A_2 is an abstraction of A_1) iff $A_1 \sqsubseteq A_2$ in $uco(C)$. Given $X \subseteq C$, the least abstract domain containing X is the least closure including X as fix-points, which is the *Moore-closure* $\mathcal{M}(X) \overset{\text{def}}{=} \{\bigwedge S \mid S \subseteq X\}$. Precision of an abstract interpretation typically relies upon the structure of the abstract domain [19]. Depending on where we compare the concrete and the abstract computations we obtain two different notions of completeness. If we compare the results in the abstract domain, we obtain what is called *backward completeness* ($\mathscr{B}$), while, if we compare the results in the concrete domain we obtain the so called *forward completeness* ($\mathscr{F}$) [8, 19, 15]. Formally, if $f : C \xrightarrow{\text{m}} C$ and $\rho \in uco(C)$, then ρ is $\mathscr{B}$-complete if $\rho \circ f \circ \rho = \rho \circ f$, while it is $\mathscr{F}$-complete if $\rho \circ f \circ \rho = f \circ \rho$. The problem of making abstract domains $\mathscr{B}$-complete has been solved in [19] and later generalised to $\mathscr{F}$-completeness in [15]. In a more general setting let $f : C_1 \longrightarrow C_2$ and $\rho \in uco(C_2)$ and $\eta \in uco(C_1)$. $\langle \rho, \eta\rangle$ is a pair of $\mathscr{B}$[$\mathscr{F}$]-complete abstractions for f if $\rho \circ f = \rho \circ f \circ \eta$ [$f \circ \eta = \rho \circ f \circ \eta$]. A pair of domain transformers has been associated with any completeness problem [11, 16], which are respectively a *domain refinement* and *simplification*. In [19] and [15], a constructive characterization of the most abstract refinement, called *complete shell*, and of the most concrete simplification, called *complete core*, of any domain, making it $\mathscr{F}$ or $\mathscr{B}$-complete, for a given continuous function f, is given as a solution of simple domain equations given by the following basic operators:

$$\boxed{\begin{array}{r|l} R_f^{\mathscr{F}} \overset{\text{def}}{=} \lambda X.\, \mathcal{M}(f(X)) & R_f^{\mathscr{B}} \overset{\text{def}}{=} \lambda X.\, \mathcal{M}(\bigcup_{y \in X} \max(f^{-1}(\downarrow y))) \\ C_f^{\mathscr{F}} \overset{\text{def}}{=} \lambda X.\ \{\ y \in L \,|\, f(y) \subseteq X\ \} & C_f^{\mathscr{B}} \overset{\text{def}}{=} \lambda X.\ \{\ y \in L \,|\, \max(f^{-1}(\downarrow y)) \subseteq X\ \} \end{array}}$$

Let $\ell \in \{\mathscr{F}, \mathscr{B}\}$. In [19] the authors proved that the most concrete $\beta \sqsupseteq \rho$ such that $\langle \beta, \eta\rangle$ is ℓ-complete and the most abstract $\beta \sqsubseteq \eta$ such that $\langle \rho, \beta\rangle$ is ℓ-complete are respectively the ℓ-complete core and ℓ-complete shell, which are: $\mathcal{C}_f^{\ell,\eta}(\rho) \overset{\text{def}}{=} \rho \sqcup C_f^{\ell}(\eta)$ and $\mathcal{R}_f^{\ell,\rho}(\eta) \overset{\text{def}}{=} \eta \sqcap R_f^{\ell}(\rho)$. When $\eta = \rho$, then the fix-point iteration on abstract domains of the above function $\mathcal{R}_f^{\ell}(\rho) = gfp(\lambda X.\ \rho \sqcap R_f^{\ell}(X))$ is called the *absolute ℓ-complete shell*. By construction if f is additive then

$\mathcal{R}_f^{\mathscr{B}} = \mathcal{R}_{f+}^{\mathscr{F}}$ (analogously $\mathcal{C}_f^{\mathscr{B}} = \mathcal{C}_{f+}^{\mathscr{F}}$) [15]. This means that when we have to solve a problem of $\mathscr{B}$-completeness for an additive function then we can equivalently solve the corresponding $\mathscr{F}$-completeness problem for the right adjoint function.

Completeness can be used for modelling non-interference in language-based security [14, 1]. Abstract non-interference (ANI) [13] is a natural weakening of non-interference by abstract interpretation. Let $\eta, \rho \in uco(\mathbb{V}^{\mathtt{L}})$ and $\phi \in uco(\mathbb{V}^{\mathtt{H}})$, where $\mathbb{V}^{\mathtt{L}}$ and $\mathbb{V}^{\mathtt{H}}$ are the domains of public ($\mathtt{L}$) and private ($\mathtt{H}$) variables. η and ρ characterise the *attacker* to the policy. A policy characterises a weakening of the information that can flow. We consider $\phi \in uco(\mathbb{V}^{\mathtt{H}})$, which states what, of the private data, can indeed flow to the output observation, the so called *declassification* of ϕ. In the following if P is a program $[\![P]\!]$ denotes its denotational semantics. A program P satisfies declassified ANI if $\forall h_1, h_2 \in \mathbb{V}^{\mathtt{H}}, \forall l_1, l_2 \in \mathbb{V}^{\mathtt{L}}$:

$$\eta(l_1) = \eta(l_2) \;\wedge\; \phi(h_1) = \phi(h_2) \;\Rightarrow\; \rho([\![P]\!](h_1, \eta(l_1))^{\mathtt{L}}) = \rho([\![P]\!](h_2, \eta(l_2))^{\mathtt{L}}).$$

This notion says that, whenever the attacker is able to analyze the input property η and the ρ property of the output, then it can observe nothing more than the property ϕ of private input. Clearly, transforming abstractions corresponds here to transform attackers and policies. If $\mathcal{H}_\rho(\langle x^{\mathtt{H}}, x^{\mathtt{L}}\rangle) \stackrel{\text{def}}{=} \langle \mathbb{V}^{\mathtt{H}}, \rho(x^{\mathtt{L}})\rangle$, $\mathcal{H}_\eta^\phi(\langle x^{\mathtt{H}}, x^{\mathtt{L}}\rangle) \stackrel{\text{def}}{=} \langle \phi(x^{\mathtt{H}}), \eta(x^{\mathtt{L}})\rangle$, $[\![P]\!]_\eta \stackrel{\text{def}}{=} \lambda x.\, [\![P]\!](x^{\mathtt{H}}, \eta(x^{\mathtt{L}}))$ and the weakest liberal precondition semantics is $wlp_\eta^P \stackrel{\text{def}}{=} \lambda X.\ \{\ \langle h, \eta(l)\rangle \mid [\![P]\!](\langle h, \eta(l)\rangle) \subseteq X\ \}$ [5], then the equivalent $\mathscr{B}$- and $\mathscr{F}$-completeness equations modelling ANI above, with attacker η and ρ, and declassified by the partitioning abstraction[1] ϕ are [14]:

$$\mathcal{H}_\rho \circ [\![P]\!]_\eta \circ \mathcal{H}_\eta^\phi = \mathcal{H}_\rho \circ [\![P]\!] \iff \mathcal{H}_\eta^\phi \circ wlp_\eta^P \circ \mathcal{H}_\rho = wlp_\eta^P \circ \mathcal{H}_\rho \tag{1}$$

3 The geometry of abstract domain transformers

The notion of abstract domain refinement and simplification has been introduced in [11, 16] as a generalisation of most well-known operations for transforming abstract domains, e.g., those introduced in [8]. In this section we consider these notions as instances of a more general pattern where abstract domain transformers have the same structures of abstract domains. For the sake of simplicity we consider unary functions only, even if all the following results can be easily generalized to generic n-ary functions (e.g., see [16] for examples). If $\tau, \eta : uco(C) \longrightarrow uco(C)$, following [16], we distinguish between *domain refinements* which concretise domains, i.e. $X \subseteq \tau(X)$, and *domain simplifications* which simplify domains, i.e. $\eta(X) \subseteq X$. Monotone refinements and simplifications can be associated with closure operators: If τ [η] is a monotone refinement [simplification] then $\lambda X.\ gfp(\lambda Y.\ X \sqcap \tau(Y))$ [$\lambda X.\ lfp(\lambda Y.\ X \sqcup \eta(Y))$] is the corresponding idempotent refinement [simplification] [7]. Therefore, monotone refinements τ and simplifications η may have the same structure of abstract domains, as closure operators on $uco(C)$, resp. $\tau \in lco(uco(C))$ and

[1] An abstraction of a complete Boolean concrete domain C is *partitioning* if it is a complete Boolean sub-semilattice of C.

$\eta \in uco(uco(C))$. This observation will be the basis in order to lift standard abstract interpretation in higher types, i.e. from a theory for approximating computational objects, such as semantics, to a theory for abstract domain transformers. In this sense, standard Cousot and Cousot's Galois connection based abstract interpretation theory is perfectly adequate to develop also a theory of abstract domain transformers providing these transformations with the same calculational design techniques which are known for standard abstract interpretation [4]. In particular, Janowitz [2, 22], characterised the structure of residuated (adjoint) closure operators by the following basic result.

Theorem 1 ([22]). *Let $\langle \eta, \eta^+ \rangle$ and $\langle \eta^-, \eta \rangle$ be pairs of adjoint operators on C.*

$$(1)\quad \eta \in uco(C) \;\Leftrightarrow\; \eta^+ \in lco(C) \;\Leftrightarrow\; \begin{cases} \eta \circ \eta^+ = \eta^+ \\ \eta^+ \circ \eta = \eta \end{cases}$$

$$(2)\quad \eta \in uco(C) \;\Leftrightarrow\; \eta^- \in lco(C) \;\Leftrightarrow\; \begin{cases} \eta \circ \eta^- = \eta \\ \eta^- \circ \eta = \eta^- \end{cases}$$

Stated in terms of refinements, this result says that any (either right or left) adjoint of a refinement [simplification] is a simplification [refinement]. This means that for any refinement [simplification] we may have two possible simplifications [refinements] corresponding to either right or left adjoint, when they exist. Let $\tau \in lco(C)$ be a domain refinement. By Th. 1, if τ^- exists then $\tau^-(\tau(X)) = \tau(X)$ and $\tau(\tau^-(X)) = \tau^-(X)$. This means that τ^- is a simplification such that both τ and τ^- have the same sets of fix-points, namely τ^- reduces any abstract domain X until the reduced domain Y satisfies $\tau(Y) = Y$. Due to the analogy with completeness, in this case we call τ^- the *core* of τ and τ the *shell* of τ^-.

Proposition 2. *Let $\tau \in lco(C)$ and $\eta \in uco(C)$. If $\langle \tau^-, \tau \rangle$ and $\langle \eta, \eta^+ \rangle$ are pairs of adjoint functions then we have $\tau^- = \lambda X.\ \bigwedge\{\tau(Y) | \tau(Y) \leq X\}$ and $\eta^+ = \lambda X.\ \bigvee\{\eta(Y) | X \leq \eta(Y)\}$.*

The interpretation of the second point of Th. 1 for refinements, i.e, of the right adjoint of a refinement τ, when it exists, is quite different. By Th. 1, we have that if τ^+ exists then $\tau^+(\tau(X)) = \tau^+(X)$ and $\tau(\tau^+(X)) = \tau(X)$. In this case $\tau^+(X)$ is not a fix-point of τ. Instead, it returns the most abstract domain whose precision can be lifted to that of X by refinement. τ^+ reduces any abstract domain X such that $\tau(X) = X$ towards the most abstract domain Y such that $\tau(Y) = X$. We call τ^+ the *compressor* of τ and τ the *expander* of τ^+.

Proposition 3. *Let $\tau \in lco(C)$ and $\eta \in uco(C)$. If $\langle \tau, \tau^+ \rangle$ and $\langle \eta^-, \eta \rangle$ are pairs of adjoint functions then we have $\tau^+ = \lambda X.\ \bigvee\{Y | \tau(Y) = \tau(X)\}$ and $\eta^- = \lambda X.\ \bigwedge\{Y | \eta(X) = \eta(Y)\}$.*

3.1 Shell vs core

Not all domain transformers admit adjoints, because not all closure are either additive or co-additive functions. However adjointness can be weakened by considering only those properties that make a transformer reversible, either as a pair shell-core or expander-compressor. In the following we describe the properties of invertible refinements since the properties of invertible simplifications can be derived by duality as shown above. By Prop. 2, the relation between the shell τ and the core τ^- is characterized by the fact that $\tau^-(X)$ isolates the most concrete domain which is contained in X and which is a fix-point of τ: $\tau^-(X) = \bigwedge \{ \tau(Y) \,|\, X \leq \tau(Y) \}$. While $\tau^- \circ \tau = \tau$ always holds for any $\tau \in \mathit{lco}(C)$, the key property which characterizes the pair shell-core, $\tau \circ \tau^- = \tau^-$, holds iff $\langle \tau^-, \tau \rangle$ is a pair of adjoint functions.

Theorem 4. *Let $\tau \in \mathit{lco}(C)$. $\tau \circ \tau^- = \tau^-$ holds iff τ is co-additive.*

This means that the relation between shell and core just holds in the standard adjoint framework. An example of the core/shell is in $\mathscr{F}$-completeness, where $\mathcal{C}_f^{\mathscr{F}}$ is a $\mathscr{F}$-completeness core for f iff $\mathcal{C}_f^{\mathscr{F}}(X) \sqsubseteq Y \Leftrightarrow X \sqsubseteq \mathcal{R}_f^{\mathscr{F}}(Y)$ [15].

3.2 Expander vs compressor

The notion of domain compression was introduced in [11, 16] and later developed for the case of *disjunctive completion* $\lambda X.\, \curlyvee(X)$, making a domain a complete join-subsemilattice of the concrete domain (viz. an additive closure), and *reduced product*, which is the *glb* in $\mathit{uco}(C)$, resp. in [17] and [3]. In view of Janowitz's results we review the theory of domain compression in [18], where the notion of *uniform closure* was introduced. $f : C \longrightarrow C$ is *join-uniform* [18] if for all $Y \subseteq C$, $(\exists \bar{x} \in Y.\, \forall y \in Y.\, f(y) = f(\bar{x})) \Rightarrow (\exists \bar{x} \in Y.\, f(\bigvee Y) = f(\bar{x}))$. Meet-uniformity is defined dually. By Prop. 3, the relation between the expander τ and its compressor τ^+ is characterized by the fact that $\tau^+(X)$ is the most abstract domain which allows us to reconstruct $\tau(X)$ by refinement: $\tau^+(X) = \bigsqcup \{ Y \,|\, \tau(X) = \tau(Y) \}$. While $\tau^+ \circ \tau = \tau^+$ always holds for any $\tau \in \mathit{lco}(C)$, the key property which characterizes the pair expander/compressor, is $\tau \circ \tau^+ = \tau$ and it holds iff τ is join-uniform. Join-uniformity captures precisely the intuitive insight of the pair expander/compressor. If τ is join-uniform, and $x \in C$, then there always exists a (unique) element $\bigvee Z$, such that $\tau(\bigvee Z) = \tau(x)$ where $Z = \{y \in C \mid \tau(x) = \tau(y)\}$. As observed in [18], τ^+ may fail monotonicity. In [18] the authors proved that τ^+ is monotone on a lifted order induced by τ. Let $\tau : C \xrightarrow{\mathrm{m}} C$, the lifted order $\leq_\tau \subseteq C \times C$ is defined as follows: $x \leq_\tau y \;\Leftrightarrow\; (\tau(x) \leq \tau(y)) \;\wedge\; (\tau(x) = \tau(y) \;\Rightarrow\; x \leq y)$. $\leq_\tau$ is such that $\leq \Rightarrow \leq_\tau$. Next theorem strengthen [18, Th. 5.10][2] proving the equivalence between reversibility and adjointness in $\leq_\tau$ for any refinement.

Theorem 5. *Let $\tau \in \mathit{lco}(L)$ and $\tau^+ = \lambda x.\, \bigvee \{ y \,|\, \tau(y) = \tau(x) \}$. The following facts are equivalent:*

1. *$\tau \circ \tau^+ = \tau$;*
2. *τ is join-uniform on $\leq$;*
3. *τ is additive on $\leq_\tau$ and the right adjoint of τ on $\leq_\tau$ is τ^+.*

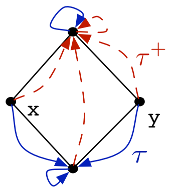
[2] In [18, Th. 5.10] the authors proved only that 1. $\Leftrightarrow$ 2. $\Rightarrow$ 3.

The relation between join-uniformity and meet-uniformity is preserved by the relation of adjointness.

Proposition 6. *Let $\tau \in lco(L)$ be a join-uniform operator on $\leq$. Then we have $\tau^+(x) = \bigvee\{y | \tau(x) = \tau(y)\}$ is meet-uniform on $\leq$.*

Unfortunately, the inverse implication of Prop. 6 does not hold in general.

Example 7. In the picture on the right, we provide an example where the map τ^+ (denoted with dashed arrows) is meet-uniform, while τ is not join-uniform. Indeed, note that $\tau^+ = \lambda X. \top$, which is clearly meet-uniform, while τ is not join-uniform since, for instance, $\tau(x) = \tau(y) \neq \top$, but $\tau(x \vee y) = \tau(\top) = \top$.



Examples of join-uniform refinements include disjunctive completion and reduced product , where the corresponding compressors are the *disjunctive base* [17] and *complementation* [3]. Both are compressors associated with completeness refinements, the first being $\mathscr{F}$-completeness w.r.t. disjunction and the second being $\mathscr{B}$-completeness w.r.t. conjunction, i.e. $\mathscr{F}$-completeness w.r.t. implication [21, 20]. The problem of studying whether a generic completeness refinement admits a compressor has been investigated with the aim of finding a characterization of all the functions f such that the corresponding completeness refinement has the compressor. The only known characterisation is in [12], for the case of $\mathscr{F}$-completeness. In this case the authors provide an algorithmic construction which is based on the the notion of f-reducible element, i.e. those elements that can be generated by others by means of the function f or by Moore closure. If all the f-irreducible elements form an abstract domain, then this is called the *complete base* and the compressor locally (i.e., for the particular abstract domain to which we apply the algorithm) exists.

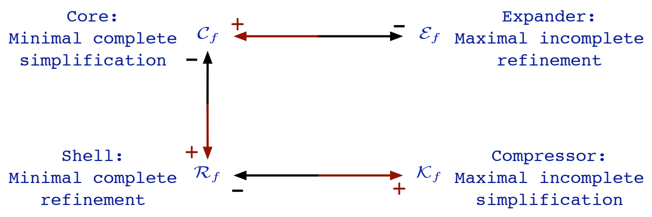


Fig. 1. Basic abstract domain transformers.

3.3 Transforming abstractions for transforming policies

By transforming abstractions in abstract non interference (Eq. 1), we transform the corresponding non-interference policy. In particular, if we transform the input abstraction we transform the declassification policy, while when we transform the output abstraction we transform the attacker policy [14]. Let us see the meaning of the shell/core and expander/compressor transformations in these cases.

Shell: The maximal released information by an attacker. The shell minimally refines the domain $\mathcal{H}_\eta^\phi$ in order to satisfy Eq. 1 [14]. When the equation does not hold, it means that the given attacker is able to disclose more information than what is modelled by ϕ about the private input. In non-interference, disclosing means observing variations in the ρ abstraction of the output due to input variations not modelled by ϕ. In other words, there are at least two private inputs h_1 and h_2 with the same property ϕ which generate a different ρ property in output. Therefore in order to characterise the closest declassification policy satisfied by the program, namely the maximal information disclosed by the attacker, we have to refine the policy ϕ, and in particular we have to distinguish by ϕ the values h_1 and h_2, above. This is what the shell does, by modelling the minimal amount of distinguishable values that the attacker is able to observe. We denote by $\mathcal{R}_{[\![P]\!]}^{\mathcal{H}_\rho}(\mathcal{H}_\eta^\phi)$ this transformation.

Example 8. Consider the program fragment: $P \overset{\text{def}}{=} l := l * h^2$, with $l : \mathtt{L}$ and $h : \mathtt{H}$. We want to find the shell in order to make the input/outpur pair of abstract domains $\langle \mathcal{H}, \mathcal{H}_{Par} \rangle$, where $Par \overset{\text{def}}{=} \{\mathbb{Z}, 2\mathbb{Z}+1, 2\mathbb{Z}, \varnothing\}$, complete for $[\![P]\!]$. Let $\mathcal{H} \overset{\text{def}}{=} \mathcal{H}_{id}^{id}$.

$$\mathcal{R}_{[\![P]\!]}^{\mathcal{H}_{Par}}(\mathcal{H}) = \mathcal{H} \sqcap \left(\left\{ \begin{array}{l} \langle \mathbb{Z}, \mathbb{Z} \rangle, \langle \mathbb{Z}, 2\mathbb{Z} \rangle \cup \langle 2\mathbb{Z}, 2\mathbb{Z}+1 \rangle, \langle 2\mathbb{Z}+1, 2\mathbb{Z}+1 \rangle, \\ \langle 2\mathbb{Z}+1, 2\mathbb{Z} \rangle, \varnothing \end{array} \right\} \right)$$

Note that $\langle 2\mathbb{Z}, 2\mathbb{Z}+1 \rangle, \langle 2\mathbb{Z}, l \rangle \in \mathcal{R}_{[\![P]\!]}^{\mathcal{H}_{Par}}(\mathcal{H})$ for each $l \in 2\mathbb{Z}+1$. For instance, in this case $\mathcal{H}_{Par}([\![P]\!](\mathcal{R}_{[\![P]\!]}^{\mathcal{H}_{Par}}(\mathcal{H})(\langle 2, 3 \rangle))) = \mathcal{H}_{Par}([\![P]\!](\langle 2\mathbb{Z}, 3 \rangle)) = \langle \mathbb{Z}, 2\mathbb{Z} \rangle = \mathcal{H}_{Par}([\![P]\!](\langle 2, 3 \rangle))$.

Core: The most powerful attacker for a declassification policy. The core minimally transform the domain $\mathcal{H}_\rho$ in order to satisfy Eq. 1 [14]. Exactly as before, when the equation does not hold, it means that the attacker, observing ρ, is able to disclose more information than what is modelled by ϕ about the private input. In this case we simplify the model of the attacker. If there are at least two private inputs h_1 and h_2 having the same property ϕ which generate a different ρ property in output, then we decide to simplify the attacker making the corresponding output values indistinguishable. The core of $\mathcal{H}_\rho$ collapses all the outputs generated by private inputs with a different ϕ property. In this way, we are able to characterize, given a declassification policy ϕ, the most powerful attacker, weaker than ρ, which is harmless, namely unable to disclose any information about ϕ of private data. We denote by $\mathcal{C}_{[\![P]\!]}^{\mathcal{H}_\eta^\phi}(\mathcal{H}_\rho)$ this tranformation.

Example 9. Let the program $P \overset{\text{def}}{=}$ **while** $h \neq 0$ **do** $l := 2l; h := 0$ **endw**, with $l : \mathtt{L}$ and $h : \mathtt{H}$. $\mathcal{C}_{[\![P]\!]}^{\mathcal{H}}(\mathcal{H}_{id}) = \{\langle \mathbb{Z}, L \rangle \mid \forall l \in \mathbb{V}^{\mathtt{L}} \,.\, l \in L \;\Leftrightarrow\; 2l \in L\}$ makes $\langle \mathcal{H}, \mathcal{H}_{id} \rangle$ complete for $[\![P]\!]$. It is easy to show that $\mathcal{C}_{[\![P]\!]}^{\mathcal{H}}(\mathcal{H}_{id})$ is the abstract domain $\curlyvee\big(\{n\{2\}^{\mathbb{N}} \mid n \in 2\mathbb{Z}+1\}\big)$, where $\{2\}^{\mathbb{N}} \overset{\text{def}}{=} \{2^k \mid k \in \mathbb{N}\}$. Then $\mathcal{C}_{[\![P]\!]}^{\mathcal{H}}(\mathcal{H}_{id})([\![P]\!](\mathcal{H}(\langle 3, 5 \rangle))) = \mathcal{C}_{[\![P]\!]}^{\mathcal{H}}(\mathcal{H}_{id})([\![P]\!](\mathbb{Z}, 5)) = \mathcal{C}_{[\![P]\!]}^{\mathcal{H}}(\mathcal{H}_{id})(\langle \mathbb{Z}, \{5, 10\} \rangle) = \langle \mathbb{Z}, 5\{2\}^{\mathbb{N}} \rangle$, $\mathcal{C}_{[\![P]\!]}^{\mathcal{H}}(\mathcal{H}_{id})([\![P]\!](\langle 3, 5 \rangle)) = \mathcal{C}_{[\![P]\!]}^{\mathcal{H}}(\mathcal{H}_{id})(\langle \mathbb{Z}, \{10\} \rangle) = \langle \mathbb{Z}, 5\{2\}^{\mathbb{N}} \rangle$.

Expander. Let us consider the left adjoint of the core. In this case we look for the more concrete attacker, which adjoints the same harmless one. This is interpreted by saying that the expander provides the inferior limit to the range of all the attacks making a given policy insecure. In other words it provides the most successful attack for the given policy.

Example 10. Consider the program fragment $P \overset{\text{def}}{=} l := 2h$. We can easily show that the most powerful harmless attacker is the one that cannot distinguish even numbers, i.e., $\curlyvee(\{2\mathbb{Z}, \{1\}, \{3\}, \ldots\})$. Suppose now that the inital observer of the program in output observes $\rho = \{\mathbb{Z}, 2\mathbb{Z} \smallsetminus \{0\}, \{0\}, 2\mathbb{Z}+1, \varnothing\}$. Then we obtain that the most powerful harmless attacker which is more abstract than ρ is Par. At this point the compressor provides the most concrete abstraction, whose core is exactly Par. We can show that this abstraction is $\curlyvee(\{2\mathbb{Z}+1, \{0\}, \{2\}, \{4\}, \ldots, \})$. We can interpret this abstraction as the most powerful malicious attacker, namely the one that is able to exploit as much as possible the failure of the non-interference policy, since it can disclose the exact value of h. Any more abstract domain, has to confuse some even numbers, for instance it can confuse 0 with 2, which means that it cannot distinguish when $h = 0$ and $h = 1$.

Compressor. Finally, let us consider the right adjoint of the completeness shell. In this case we look for the most abstract declassification policy which cannot capture what is indeed released by the attacker observing the program. Also in this case, we interpret this abstraction as a superior limit to the range of all the policies which are inadequate to protect a program from an attacker.

Example 11. Consider the program $P \overset{\text{def}}{=} \textbf{if } h = 0 \textbf{ then } l := 0 \textbf{ else } l := |l|(h/|h|)$, where $|x|$ is the absolute value of x. Let the declassification policy $\phi = \{\mathbb{Z}, \geq 0, < 0, \varnothing\}$. The wlp^P is $\{l = 0 \mapsto h = 0, l > 0 \mapsto h > 0, l < 0 \mapsto h < 0\}$ hence, the information released is $\phi' = \{\mathbb{Z}, \geq 0, \neq 0, \leq 0, > 0, 0, < 0, \varnothing\}$. If we compute the compressor then we obtain $\phi'' = \lambda X.\, \mathbb{Z}$, which means that every policy between ϕ' and ϕ'' is not able to protect the program.

4 The geometry of completeness semantic transformers

In this section, we introduce a completely symmetric construction for semantic transformers. The problem is: *Can we (minimally) transform the semantics in order to satisfy completeness?* The transformation is made in two steps: first we induce *completeness*, and then we force *monotonicity* by using standard results on function transformers in [7], since in some cases, the completeness transformation may generate not monotone functions. Recall that any function can be transformed to the closest (from below and from above) monotone function by considering the following basic transformers [7]:

$$\mathbb{M}^{\downarrow} \overset{\text{def}}{=} \lambda f.\, \lambda x.\, \bigwedge \left\{\, f(y) \,\middle|\, y \geq x \,\right\} \qquad \mathbb{M}^{\uparrow} \overset{\text{def}}{=} \lambda f.\, \lambda x.\, \bigvee \left\{\, f(y) \,\middle|\, y \leq x \,\right\}$$

Before introducing these transformers we have to understand what we mean by *minimally* transforming semantics. As usual we consider a lattice of functions where maps are point-wise ordered: $f \sqsubseteq g$ iff $\forall x \in C.\, f(x) \leq g(x)$. Hence, a minimal transformation of f finds the closest function, by reducing or increasing the images of f, wrt. a given property we want to hold for f (in this context, completeness). In abstract interpretation this corresponds to find the closest (viz., least abstraction or concretisation) of the semantics such that completeness holds for a given pair of abstractions. In the following, for simplicity, we consider the case of forward completeness.

4.1 Transforming semantics for inducing forward completeness

Moving upwards. Let us consider first the case of increasing a given function $f : C \xrightarrow{\text{m}} C$ in order to induce completeness with respect to two fixed abstractions $\eta, \rho \in uco(C)$. We first observe that such a minimal transformation exists, namely the set $\{h : C \xrightarrow{\text{m}} C \mid f \sqsubseteq h,\ \rho \circ h \circ \eta = h \circ \eta\}$ has the minimal element. The following result proves that we can always minimally increase a given monotone function f in order to induce completeness.

Theorem 12. *The set* $\left\{\ f : C \xrightarrow{m} C \,\middle|\, \rho \circ f \circ \eta = f \circ \eta\ \right\}$ *is an upper closure operator on* $\langle C \xrightarrow{m} C, \sqsubseteq \rangle$.

For any $f \in C \xrightarrow{\text{m}} C$ and $\eta, \rho \in uco(C)$ define

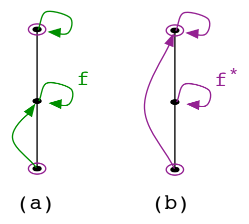
$$\mathbb{F}^{\uparrow}_{\eta,\rho} \overset{\text{def}}{=} \lambda f.\lambda x. \begin{cases} \rho \circ f(x) & \text{if } x \in \eta(C) \\ f(x) & \text{otherwise} \end{cases}$$

Lemma 13. *Let* $f : C \xrightarrow{m} C$. *Then*

$$\mathbb{F}^{\uparrow}_{\eta,\rho}(f) = \bigsqcap \left\{\ h : C \longrightarrow C \,\middle|\, f \sqsubseteq h,\ \rho \circ h \circ \eta = h \circ \eta\ \right\}.$$

The function $\mathbb{F}^{\uparrow}_{\eta,\rho}(f)$ is not the one we look for since it may lack monotonicity. Next example shows that $\mathbb{F}^{\uparrow}_{\eta,\rho}(f)$ may not be monotone for some $f : C \xrightarrow{\text{m}} C$.

Example 14. Consider the pictures on the right. The (purple) circled points are those in ρ and the (green) arrows in the picture (a) represent f. The (purple) arrows in the picture (b) are those of the map obtained from f by applying $\mathbb{F}^{\uparrow}$ which is clearly not monotone.



The lack of monotonicity is due to the fact that, in order to minimally transform f, only the images of the elements in η are modified, leaving unchanged the images of all the other elements. Indeed monotonicity fails when we check it between the new image of one element in η and one outside. We need therefore to apply the transformer $\mathbb{M}^{\uparrow}$ for finding the best *monotone* transformation of f which is complete for the pair of abstractions $\eta, \rho \in uco(C)$. Define $\mathcal{F}^{\uparrow}_{\eta,\rho} \overset{\text{def}}{=} \lambda f.\, \mathbb{M}^{\uparrow} \circ \mathbb{F}^{\uparrow}_{\eta,\rho}(f)$.

Theorem 15. *Let* $f : C \xrightarrow{m} C$.

$$\mathcal{F}^{\uparrow}_{\eta,\rho}(f) = \bigsqcap \left\{\ h : C \xrightarrow{m} C \,\middle|\, f \sqsubseteq h,\ \rho \circ h \circ \eta = h \circ \eta\ \right\}.$$

Moving downwards. We consider the maximal approximation from below of a given $f : C \xrightarrow{\text{m}} C$, making it complete This exists unique under some hypothesis.

Theorem 16. *The set* $\{f : C \xrightarrow{m} C \mid \rho \circ f \circ \eta = f \circ \eta\}$ *is a lower closure operator on* $\langle C \xrightarrow{m} C, \sqsubseteq\rangle$ *iff* ρ *is additive.*

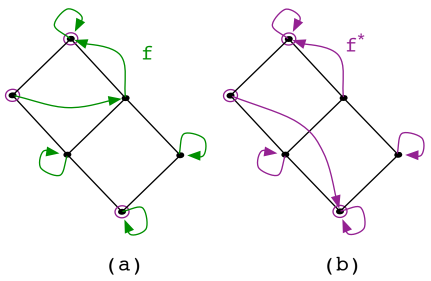
For any $f : C \xrightarrow{\text{m}} C$ and additive closure $\rho \in uco(C)$, define:

$$\mathbb{F}^{\downarrow}_{\eta,\rho} \overset{\text{def}}{=} \lambda f.\lambda x. \begin{cases} \rho^{+} \circ f(x) & \text{if } x \in \eta(C) \\ f(x) & \text{otherwise} \end{cases}$$

Lemma 17. *Let* $f : C \xrightarrow{m} C$, *then*

$$\mathbb{F}^{\downarrow}_{\eta,\rho}(f) = \bigsqcup \left\{ \, h : C \longrightarrow C \,\middle|\, f \sqsupseteq h, \;\; \rho \circ h \circ \eta = h \circ \eta \, \right\}$$

Example 18. Consider the pictures on the right. The (purple) circled points are those in ρ and the (green) arrows on picture (a) are those of f. The (purple) arrows on the picture (b) are those of the map obtained from f by means of $\mathbb{F}^{\downarrow}$, and this map is clearly not monotone.



Again, the lack of monotonicity is due to the fact that, in sake of minimality, the transformers changes the image by f of only some elements, those of η. Again we apply the transformer $\mathbb{M}^{\downarrow}$ for finding the best *monotone* transformation of f. Next theorem shows that it is not necessary a fix-point transformation since the monotone transformer does not change the completeness of functions obtained by $\mathbb{F}^{\downarrow}$. Define $\mathcal{F}^{\downarrow}_{\eta,\rho} \overset{\text{def}}{=} \lambda f.\, \mathbb{M}^{\downarrow} \circ \mathbb{F}^{\downarrow}(f)$.

Theorem 19. *Let* $f : C \xrightarrow{m} C$, *then*

$$\mathcal{F}^{\downarrow}_{\eta,\rho}(f) = \bigsqcup \left\{ \, h : C \xrightarrow{m} C \,\middle|\, f \sqsupseteq h, \;\; \rho \circ h \circ \eta = h \circ \eta \, \right\}.$$

Next result tells us that the completeness transformers, moving in opposite directions in the lattice of functions, are indeed adjoint transformers whenever both exist, namely when the output abstraction ρ is additive, i.e. $\curlyvee(\rho) = \rho$.

Theorem 20. *If* $\rho \in uco(C)$ *is additive then* $(\mathbb{F}^{\uparrow}_{\eta,\rho})^{+} = \mathbb{F}^{\downarrow}_{\eta,\rho}$.

4.2 Transforming semantics for inducing forward incompleteness

Consider the two $\mathscr{F}$-completeness transformers for making functions complete for a given pair of domains, η on the input and ρ on the output. We proved that $\mathcal{F}^{\uparrow}_{\eta,\rho} \in uco(C \xrightarrow{\text{m}} C)$ and $\mathcal{F}^{\downarrow}_{\eta,\rho} \in lco(C \xrightarrow{\text{m}} C)$. We prove that their adjoint functions increase incompleteness. Given a function f, we look for the most distant function with the same complete transformation as f. In particular, when we consider $\mathcal{F}^{\uparrow}_{\eta,\rho}$, we look for the smallest function with the same transformation complete transformation as f which surely includes the maximal amount of incompleteness. A dual reasoning can be done for the other transformation following Janowitz's results.

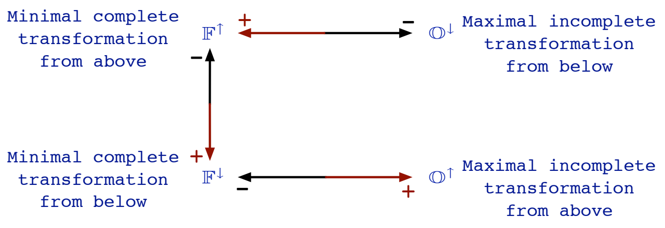

Minimal complete transformation from above
$\mathbb{F}^{\uparrow}$
+
−
$\mathbb{O}^{\downarrow}$
Maximal incomplete transformation from below
−
+
Minimal complete transformation from below
$\mathbb{F}^{\downarrow}$
−
+
$\mathbb{O}^{\uparrow}$
Maximal incomplete transformation from above

Fig. 2. Basic semantic transformers.

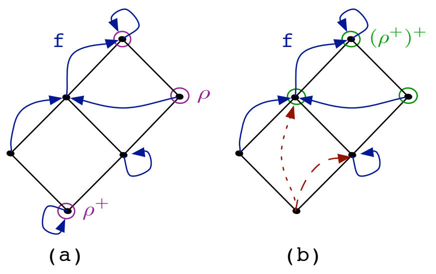
Moving upwards. The corresponding right adjoint, when it exists (see Sec. 3), of $\mathcal{F}^{\downarrow}_{\eta,\rho}$ is $\mathcal{O}^{\uparrow}_{\eta,\rho}(f) \overset{\text{def}}{=} \bigsqcup\{g : C \xrightarrow{\text{m}} C \mid \mathcal{F}^{\downarrow}_{\eta,\rho}(g) = \mathcal{F}^{\downarrow}_{\eta,\rho}(f)\}$. Being $\mathcal{F}^{\downarrow}_{\eta,\rho}$ the composition of two function transformations, we study first the adjoint operation associated with $\mathbb{F}^{\downarrow}$: $\mathbb{O}^{\uparrow}_{\eta,\rho}(f) \overset{\text{def}}{=} \bigsqcup\{g : C \longrightarrow C \mid \mathbb{F}^{\downarrow}_{\eta,\rho}(g) = \mathbb{F}^{\downarrow}_{\eta,\rho}(f)\}$ and then prove that we can always apply the monotonicity transformer afterwards. As observed in Sect. 3, this adjoint operator may not always exist. Moreover, in the previous section we also observed that $\mathbb{F}^{\downarrow}$ may not always exist. Next theorem tells us that the incompleteness transformer exists when ρ^+ is join-uniform, which implicitly says that ρ^+ exists, namely that we also need ρ additive.

Theorem 21. *The transformer $\mathbb{O}^{\uparrow}_{\eta,\rho}(f) \in uco(C \longrightarrow C)$ iff ρ^+ exists and it is join-uniform. In this case*

$$\mathbb{O}^{\uparrow}_{\eta,\rho}(f)(x) = \begin{cases} (\rho^+)^+(f(x)) = \bigvee\left\{\, y \,\middle|\, \rho^+(y) = \rho^+(f(x)) \,\right\} & \text{if } x \in \eta \\ f(x) & \text{otherwise} \end{cases}$$

Note that, also in this case, the transformer does not change all the elements, but only those in η. This implies that, also in this case, the function obtained by applying $\mathbb{O}^{\uparrow}$ is not necessarily monotone, even if applied to a monotone function.

Example 22. Consider $\eta = \rho$. In picture (a) the (purple) circled points are those in ρ (and in ρ^+) and the (blue) arrows corresponds to the function f. In picture (b) we have represented with (green) circled points the closure $(\rho^+)^+$ (which is an uco on the lifted order). We note that the transformation induced by $\mathbb{O}^{\uparrow}$ represented with a dotted line, is non monotone.



By applying $\mathbb{M}^{\downarrow}$ we obtain the monotone one represented with a dashed line.

The example above shows that by applying the transformer $\mathbb{M}^{\downarrow}$ we still obtain a monotone incomplete function. Next result proves that the transformer $\mathbb{M}^{\downarrow}$ does not change the class of complete functions, hence we can always make the monotonicity transformation if necessary, without having to reapply $\mathbb{O}^{\uparrow}$.

Theorem 23.
If $f : C \xrightarrow{m} C$ *then* $\mathbb{F}^{\downarrow}_{\eta,\rho} \circ \mathbb{M}^{\downarrow} \circ \mathbb{O}^{\uparrow}_{\eta,\rho}(f) = \mathbb{F}^{\downarrow}_{\eta,\rho}$ *and* $\mathcal{O}^{\uparrow}_{\eta,\rho}(f) \sqsupseteq \mathbb{M}^{\downarrow} \circ \mathbb{O}^{\uparrow}_{\eta,\rho}(f)$.

At this point, we have that $\mathbb{O}^{\uparrow}$ is the adjoint of a transformer inducing completeness, hence intuitively it induces incompleteness. Clearly, we wonder if there are complete functions which are fix-point of $\mathbb{O}^{\uparrow}$. Next theorem proves that $\mathbb{O}^{\uparrow}$ does not always transform a complete function into an incomplete one. This is the case when no incomplete functions are available.

Theorem 24. *If* $f : C \xrightarrow{m} C$ *then* $\mathbb{O}^{\uparrow}_{\eta,\rho}(f)$ *is complete iff for each* $x \in C$ *we have* $\left\{ y \,\middle|\, \rho^{+}(y) = \rho \circ f \circ \eta(x) \right\} = \{f \circ \eta(x)\}$.

In Fig. 3 (a) we show an example where the condition above holds also for non-trivial functions and closures where $\eta = \rho$. In the picture the closure is represented with (purple) circled points and the map with (green) arrows. This is a non-trivial complete function which is a fix-point of the transformer. In Fig. 3

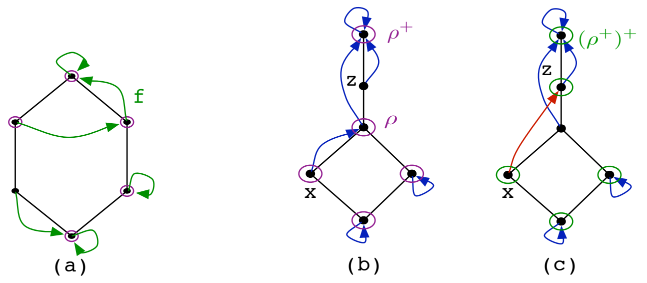


Fig. 3. Incompleteness transformations examples.

(b) and (c) we show a case where a complete function is indeed transformed in an incomplete one by the transformer. Again, in Fig. 3 (b) the closure ρ (and ρ^{+}) is represented with (purple) circled points and the function f, for which the domain is complete, is represented with (blue) arrows. $\mathbb{O}^{\uparrow}_{\eta,\rho}(f)$ is in Fig. 3 (c).

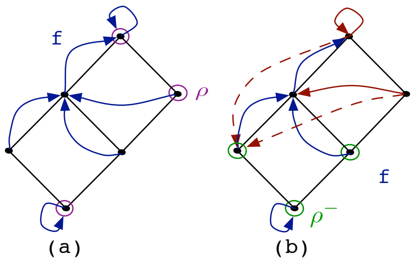
Moving downwards. We close our construction of semantic transformers by characterising an incompleteness transformer associated with the left-adjoint of $\mathcal{F}^{\uparrow}_{\eta,\rho}$, in the sense of expansion: $\mathcal{O}^{\downarrow}_{\eta,\rho}(f) = \bigsqcap\{g : C \xrightarrow{m} C \mid \mathcal{F}^{\uparrow}_{\eta,\rho}(g) = \mathcal{F}^{\uparrow}_{\eta,\rho}(f)\}$ when it exists (see Sect. 3). As in the previous case, we study the following transformation first $\mathbb{O}^{\downarrow}_{\eta,\rho}(f) \overset{\text{def}}{=} \bigsqcap\{g : C \longrightarrow C \mid \mathbb{F}^{\uparrow}_{\eta,\rho}(g) = \mathbb{F}^{\uparrow}_{\eta,\rho}(f)\}$. While $\mathbb{F}^{\uparrow}_{\eta,\rho}$ always exists, $\mathbb{O}^{\downarrow}_{\eta,\rho}$ may not always exist. Next theorem proves that the incompleteness transformer exists when ρ^{-} exists, namely when ρ is meet-uniform.

Theorem 25. *$\mathbb{O}^{\downarrow}_{\eta,\rho}(f) \in lco(C \longrightarrow C)$ iff ρ is meet-uniform. In this case*

$$\mathbb{O}^{\downarrow}_{\eta,\rho}(f)(x) = \begin{cases} \rho^{-}(f(x)) = \bigwedge \{\ y \mid \rho(y) = \rho(f(x))\ \} & \text{if } x \in \eta \\ f(x) & \text{otherwise} \end{cases}$$

Exactly as it happens in all previous cases, the transformer does not change all the elements, but only those in η. This implies that the function obtained by $\mathbb{O}^{\downarrow}$ may not be monotone, even if we start with a monotone function f.

Example 26. Consider the pictures on the right, where $\eta = \rho$. Here we have an example of transformation which returns a non monotone function. We see in picture (b), the closure ρ^{-} denoted with (green) circled points. The transformed map is the one represented in picture (b) with (red) dashed lines. Note that $\mathbb{M}^{\uparrow}(\mathbb{O}^{\downarrow}_{\eta,\rho}(f)) = f$.



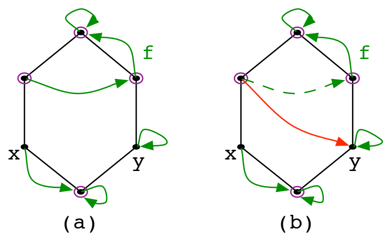
Theorem 27.
If $f : C \xrightarrow{m} C$ then $\mathbb{F}^{\uparrow}_{\eta,\rho} \circ \mathbb{M}^{\uparrow} \circ \mathbb{O}^{\downarrow}_{\eta,\rho}(f) = \mathbb{F}^{\uparrow}_{\eta,\rho}$ and $\mathcal{O}^{\downarrow}_{\eta,\rho}(f) \sqsubseteq \mathbb{M}^{\uparrow} \circ \mathbb{O}^{\downarrow}_{\eta,\rho}(f)$.

Also in this case, we consider the case when this trasformation induces incompleteness. Namely, we wonder when $\rho \circ \mathbb{O}^{\downarrow}_{\eta,\rho}(f) \circ \eta \neq \mathbb{O}^{\downarrow}_{\eta,\rho}(f) \circ \eta$.

Theorem 28. *If $f : C \xrightarrow{m} C$ then $\mathbb{O}^{\downarrow}_{\eta,\rho}(f)$ is complete iff for each $x \in C$ we have $\{\ y \mid \rho(y) = \rho \circ f \circ \eta(x)\ \} = \{f \circ \eta(x)\}$.*

An example of complete function which is left unchanged by $\mathbb{O}^{\downarrow}$ is the same shown before for $\mathbb{O}^{\uparrow}$ (Fig. 3 (a)). Next example shows, instead, a case where a complete function is indeed transformed into an incomplete one by $\mathbb{O}^{\downarrow}$.

Example 29. Consider the pictures on the right where $\eta = \rho$. Again the closure is represented with (purple) circled points. In the picture (a) we have the original function f, for which the domain is complete, as we can simply verify. In (b) we have the function that we obtain by applying the transformer $\mathbb{O}^{\downarrow}$ to f.



4.3 Transforming semantics for transforming program security

Transforming functions corresponds to transform semantics [10]. In the following we consider the meaning of transformed semantics in language-based security.

Inducing completeness. Assume η be disjunctive, i.e. $\curlyvee(\eta) = \eta$. We consider $\mathbb{F}^{\uparrow}_{\mathcal{H}^{\phi}_{\eta},\mathcal{H}_{\rho}}(wlp^{P}_{\eta})$ and $\mathbb{F}^{\downarrow}_{\mathcal{H}^{\phi}_{\eta},\mathcal{H}_{\rho}}(wlp^{P}_{\eta})$, where $\mathcal{H}^{\phi^{+}}_{\eta} = \lambda\langle X^{\mathtt{H}}, X^{\mathtt{L}}\rangle.\ \langle \phi^{+}(X^{\mathtt{H}}), \eta^{+}(X^{\mathtt{L}})\rangle$. The transformations above, show that whenever we have $wlp^{P}_{\eta}(X) = \langle H, \eta(L)\rangle$, and $X \in \mathcal{H}_{\rho}$ then $\mathbb{F}^{\uparrow}_{\mathcal{H}^{\phi}_{\eta},\mathcal{H}_{\rho}}(wlp^{P}_{\eta})(X) = \langle \phi(H), \eta(L)\rangle$ by idempotence of η, and $\mathbb{F}^{\downarrow}_{\mathcal{H}^{\phi}_{\eta},\mathcal{H}_{\rho}}(wlp^{P}_{\eta})(X) = \langle \phi^{+}(H), \eta(L)\rangle$ by Th. 1(1). This result tells us that, while by using the core we can transform the output observation ρ for characterizing attackers, we cannot transform the input observation η.

Example 30. Let $P \overset{\text{def}}{=}$ **while** $(h > 0)$ **do** $(h := h - 1; l := h)$ **endw**. Suppose the attacker can observe the identity. The *wlp* is $\{l = 0 \mapsto\ h \geq 0, l \neq 0 \mapsto h = 0\}$. This program is insecure for the policy $\phi = \lambda x.\,\mathbb{Z}$, which says that nothing has to flow. The secure semantic transformation is $\mathbb{F}^{\uparrow}_{\mathcal{H}^{\phi}_{id},\mathcal{H}_{id}}(wlp^{P}_{id})$: $\{l = 0 \mapsto\ h \in \mathbb{Z}, l \neq 0 \mapsto h \in \mathbb{Z}\}$ which, for example, is the semantics of the transformed program $P' \overset{\text{def}}{=} l_1 := l; P; l := l_1$.

Inducing incompleteness. Let us consider the incompleteness transformers $\mathbb{O}^{\uparrow}_{\mathcal{H}^{\phi}_{\eta},\mathcal{H}_{\rho}}(wlp^{P}_{\eta})$ and $\mathbb{O}^{\downarrow}_{\mathcal{H}^{\phi}_{\eta},\mathcal{H}_{\rho}}(wlp^{P}_{\eta})$ where, when the necessary conditions on η and ϕ hold, $\mathcal{H}^{\phi^{++}}_{\eta}(X) = \langle \phi^{++}(X^{\mathtt{H}}), \eta^{++}(X^{\mathtt{L}})\rangle$ and $\mathcal{H}^{\phi^{-}}_{\eta}(X) = \langle \phi^{-}(X^{\mathtt{H}}), \eta^{-}(X^{\mathtt{L}})\rangle$. The problem is that the completeness equation with ϕ holds if ϕ is partitioning [1]. This is a problem since it implies that ϕ is not meet-uniform and ϕ^{+} cannot be join-uniform. Hence we don't have an optimal transformation of wlp^{P}_{η} but rather only maximal incomplete transformations. This is interpreted by noting that a semantic transformer corresponds to an active attacker that wants to exploit its *activity* for disclosing more private information. A program that does not reveal to an active attacker more than what is revealed to a passive one is called *robust* [23]. Therefore there is no best active attacker that can extract all about private data. An attacker can only decide what it wants to disclose and consequently actively transform the code.

Example 31. Let $P \overset{\text{def}}{=} h := h\ mod\ 2$; **if** $h = 0$ **then** $l := 0$ **else** $l := 1$ and $\rho = \eta = id$, $\phi = Par$. wlp^{P}_{id} is $\{l = 0 \mapsto h \in 2\mathbb{Z}, l = 1 \mapsto h \in 2\mathbb{Z}{+}1\}$ namely we disclose parity of h. Suppose the attacker wants to distinguish in addition if h is 0 or not. Then we consider the partitioning closure $\sigma \overset{\text{def}}{=} \curlyvee(\{\{0\}, 2\mathbb{Z} \smallsetminus \{0\}, 2\mathbb{Z} + 1\})$. For what we said above, we don't have a best transformation allowing to observe 0, hence the attacker has to choose to disclose weaker information about h, for example $\{0, 2\}$ instead of 0. We can consider the semantics: $\{l = 0 \mapsto h \in \{0, 2\}, l = 1 \mapsto h \in 2\mathbb{Z}{+}1\}$ approximating $\mathbb{O}^{\downarrow}_{\mathcal{H}^{\sigma}_{id},\mathcal{H}_{id}}(wlp^{P}_{id})$ which, for example, is the semantics of **if** $h \leq 2 \vee h\ mod\ 2 = 1$ **then** P **else** $l := 2$

5 Discussion

We proved that standard abstract interpretation methods, based on Galois connections, can provide an adequate model for reasoning about transformations of both abstract domains and semantics. While the abstract domain side is more traditional in static program analysis, in particular in the field of abstraction design, the semantic side is completely new. In this paper we proved that a completely symmetric construction holds for both semantic and domain transformers, sharing the same geometric structure which is based on the lifting of Galois connections higher order, from the objects of computation to the space of abstract domains and predicate transformers. This shows that abstract interpretation, as originally developed in [6], may have a universal validity not only for approximating semantics but also on reasoning about its own methods. The key aspect in this construction is completeness, which is the driving force for transforming domains and semantics to achieve a given precision degree. We used language-based security as an application ground for interpreting our transformations, but the validity of these results are general. For instance possible applications of the basic semantic transformers for achieving maximal incompleteness are in code obfuscation. In this case an obfuscated program fighting against an attacker which performs static analysis driven reverse engineering can be viewed as the maximal incomplete transformation of the program with respect to the abstractions used by the analyser. Similarly minimal complete transformations can be used in abstract model checking for isolating temporal sub-logics which are complete for a given abstract system to analyse.

Acknowledgements. Part of the material developed in this paper was conceived with Francesco Ranzato and Elisa Quintarelli. We are grateful to the endless discussions we had together during the last decade, discussions that helped us in better understanding the beauty of abstract interpretation theory.

References

1. A. Banerjee, R. Giacobazzi, and I. Mastroeni. What you lose is what you leak: Information leakage in declassifivation policies. In *Proc. of the 23th Internat. Symp. on Mathematical Foundations of Programming Semantics* (*MFPS '07*), volume 1514 of *ENTCS*, Amsterdam, 2007. Elsevier.
2. T.S. Blyth and M.F. Janowitz. *Residuation theory.* Pergamon Press, 1972.
3. A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. *ACM Trans. Program. Lang. Syst.*, 19(1):7–47, 1997.
4. P. Cousot. The calculational design of a generic abstract interpreter. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design.* NATO ASI Series F. IOS Press, Amsterdam, 1999.
5. P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theor. Comput. Sci.*, 277(1-2):47–103, 2002.
6. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of Conf. Record of the 4th ACM Symp. on Principles of Programming Languages* (*POPL '77*), pages 238–252, New York, 1977. ACM Press.
7. P. Cousot and R. Cousot. A constructive characterization of the lattices of all retractions, preclosure, quasi-closure and closure operators on a complete lattice. *Portug. Math.*, 38(2):185–198, 1979.
8. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. of Conf. Record of the 6th ACM Symp. on Principles of Programming Languages* (*POPL '79*), pages 269–282, New York, 1979. ACM Press.

9. P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation (Invited Paper). In M. Bruynooghe and M. Wirsing, editors, *Proc. of the 4th Internat. Symp. on Programming Language Implementation and Logic Programming* (*PLILP '92*), volume 631 of *LNCS*, pages 269–295, Berlin, 1992. Springer-Verlag.
10. P. Cousot and R. Cousot. Systematic design of program transformation frameworks by abstract interpretation. In *Proc. of Conf. Record of the Twentyninth Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 178–190, New York, 2002. ACM Press.
11. G. Filé, R. Giacobazzi, and F. Ranzato. A unifying view of abstract domain design. *ACM Comput. Surv.*, 28(2):333–336, 1996.
12. R. Giacobazzi and I. Mastroeni. Domain compression for complete abstractions. In L. D. Zuck et al., editor, *In proc. of the 4th Internat. Conf. on Verification, Model Checking and Abstract Interpretation (VMCAI'03)*, volume 2575 of *LNCS*, pages 146–160, Berlin, 2003. Springer-Verlag.
13. R. Giacobazzi and I. Mastroeni. Abstract non-interference: Parameterizing non-interference by abstract interpretation. In *Proc. of the 31st Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL '04)*, pages 186–197, New York, 2004. ACM-Press.
14. R. Giacobazzi and I. Mastroeni. Adjoining declassification and attack models by abstract interpretation. In S. Sagiv, editor, *Proc. of the European Symp. on Programming (ESOP '05)*, volume 3444 of *LNCS*, pages 295–310, Berlin, 2005. Springer-Verlag.
15. R. Giacobazzi and E. Quintarelli. Incompleteness, counterexamples and refinements in abstract model-checking. In P. Cousot, editor, *Proc. of The 8th Internat. Static Analysis Symp. (SAS'01)*, volume 2126 of *LNCS*, pages 356–373, Berlin, 2001. Springer-Verlag.
16. R. Giacobazzi and F. Ranzato. Refining and compressing abstract domains. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. of the 24th Internat. Colloq. on Automata, Languages and Programming* (*ICALP '97*), volume 1256 of *LNCS*, pages 771–781, Berlin, 1997. Springer-Verlag.
17. R. Giacobazzi and F. Ranzato. Optimal domains for disjunctive abstract interpretation. *Sci. Comput. Program*, 32(1-3):177–210, 1998.
18. 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.
19. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract interpretations complete. *J. of the ACM.*, 47(2):361–416, 2000.
20. R. Giacobazzi, F. Ranzato, and F. Scozzari. Making abstract domains condensing. *ACM Transactions on Computational Logic (ACM-TOCL)*, 6(1):33–60, 2005.
21. R. Giacobazzi and F. Scozzari. A logical model for relational abstract domains. *ACM Trans. Program. Lang. Syst.*, 20(5):1067–1109, 1998.
22. M. F. Janowitz. Residuated closure operators. *Portug. Math.*, 26(2):221–252, 1967.
23. A. Sabelfeld and A.C. Myers. Language-based information-flow security. *IEEE J. on selected ares in communications*, 21(1):5–19, 2003.
24. M. Ward. The closure operators of a lattice. *Ann. Math.*, 43(2):191–196, 1942.