Theoretical Computer Science 823 (2020) 1–25

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

On collecting semantics for program analysis

Gianluca Amato *, Maria Chiara Meo *, Francesca Scozzari *

Università di Chieti–Pescara, Pescara, Italy

ARTICLE INFO

Article history:
Received 31 January 2019
Received in revised form 6 February 2020
Accepted 17 February 2020
Available online 19 February 2020

Keywords:
Abstract interpretation
Static analysis
Collecting semantics
Galois connection

ABSTRACT

Reasoning on a complex system in the abstract interpretation theory starts with a formal description of the system behavior specified by a collecting semantics. We take the common point of view that a collecting semantics is a very precise semantics from which other abstractions may be derived. We elaborate on both the concepts of precision and derivability, and introduce a notion of adequacy which tell us when a collecting semantics is a good choice for a given family of abstractions. We instantiate this approach to the case of first-order functional programs by considering three common collecting semantics and some abstract properties of functions. We study their relative precision and give a constructive characterization of the classes of abstractions which are adequate for the collecting semantics.

1. Introduction

Abstract interpretation, introduced by Cousot and Cousot [14,15], is a framework for approximating the behavior of systems. It has been used both for developing static analysis, verification methods and for studying relations among semantics at different levels of abstraction. The main point of abstract interpretation is to replace the formal semantics of a system with an abstract semantics computed over a domain of abstract objects, which describe the properties of the system we are interested in.

Given a system e, we assume that its semantics $[\![e]\!]$ is an element of a set C of concrete properties, called the *concrete domain*. In most cases we only need to know some abstract properties of the semantics of a system. Examples of abstract properties are: *the program terminates for all the input values* or *the program output is a value in the interval* $[0, 42]$. Given a set A of abstract properties (*abstract domain*) we are interested in, each concrete property in C may enjoy several abstract properties in A. The relationship between concrete and abstract properties may be formalized in many different ways, but one of the most common approaches consists in fixing an *approximation ordering* $\leq_A$ on the set of abstract properties A, such that $a_1 \leq_A a_2$ when a_1 is a stronger property than a_2, and defining an *abstraction function* $\alpha_A : C \to A$ which maps every concrete property to the strongest (smallest) abstract property it enjoys. This is what is presented in Cousot and Cousot [16] under the "existence of a best abstract approximation assumption".

An abstract interpretation problem consists in answering whether the semantics $[\![e]\!]$ of a system e enjoys a given property a. In theory, this can be verified by computing the strongest abstract property enjoyed by the system e, which is $\alpha_A([\![e]\!])$, and checking that it enjoys the property a we are interested in, that is $\alpha_A([\![e]\!]) \leq_A a$. In practice, since $[\![e]\!]$ is not generally computable, the standard approach to solve this problem is to design an *approximate abstract semantics* $[\![e]\!]^A \in A$ such that

* Corresponding authors.
E-mail addresses: gianluca.amato@unich.it (G. Amato), mariachiara.meo@unich.it (M.C. Meo), francesca.scozzari@unich.it (F. Scozzari).

https://doi.org/10.1016/j.tcs.2020.02.021

$\alpha_A([\![e]\!]) \leq_A [\![e]\!]^A$ and show that $[\![e]\!]^A \leq_A a$. In most cases, $[\![e]\!]^A$ is not designed starting from $[\![e]\!]$, but from a so-called *collecting semantics*.

According to Cousot and Cousot [16], the term collecting semantics may be used to mean either "an instrumented version of the standard semantics in order to gather information about programs executions" or "a version of the standard semantics reduced to essentials in order to ignore irrelevant details about program execution". In this paper, we consider only the second meaning of the term, i.e., we view a collecting semantics as an abstraction of the standard semantics.

One might ask which are the properties that an abstract interpretation enjoys which entitle the abstraction to be called a collecting semantics. It is common to find in the literature the statement that the collecting semantics is a very precise abstraction from which all the other analyses may be derived. Here we want to elaborate and formalize this statement. In particular, we discuss what is the right concept of precision of abstractions to be used in this context, and what is behind the statement that an analysis may be derived from another.

In the context of first-order functional programs, we consider collecting semantics commonly used in static analysis of programs which first appeared in Cousot and Cousot [17]. We introduce three collecting semantics CS_1, CS_2 and CS_3 and discuss the relative precision among them, showing that CS_2 is a suitable collecting semantics for all properties, while CS_1 and CS_3 are only suitable for some properties of functions. We define a category of abstract interpretations and their morphisms based on the notion of Galois connection, discuss the role of initial objects and constructively characterize a class of abstractions for which the collecting semantics are initial, i.e., are well suited as collecting semantics. Moreover, we show how common abstractions of functional properties, such as strictness, totality, convergence, divergence and monotonicity, relate to the collecting semantics.

1.1. Plan of the paper

In Section 2 we explore the concept of precision of abstraction, show some examples of abstraction for properties of functions and discuss how they can be compared starting from the above notion of precision. Section 3 introduces the two collecting semantics CS_1 and CS_2. We compare the precision of the collecting semantics and formally state their relationships. In Section 4 we define the categorical framework of abstract interpretations and discuss the role of initial objects. Section 5 extends this characterization to an additional collecting semantics CS_3 and shows how it relates to the semantics in Section 3. In Section 6 we discuss what happens when using different formalizations of abstract interpretation and suggest some future work. All the proofs are in the appendix.

The article is an extended and revised version of [3]. The most prominent change is a complete overhaul of the motivations of this work and the reasons which led to our choice for the precision ordering. However, there are many other changes: we have included more examples of abstractions (totality, convergence, divergence, involution, idempotence, monotonicity, boundness), studied a new collecting semantics (CS_3) and significantly expanded the intuitive explanations. Moreover, both the appendix with the proofs and the section on related work are new.

2. Precision, derivability and collecting semantics

In the following, we fix a set C of concrete properties. The standard semantics $[\![e]\!]$ of a system e is an element of C. The concrete domain C may be endowed with additional structure which allows to derive the semantics of a system through structural induction on the description of the system and least fixpoint computations. However, the way $[\![e]\!]$ is defined is going to play only a marginal role in our discussion.

Definition 1 *(Abstract Interpretation).* An *abstract interpretation* for a set C is a pair (A, α_A) where the set A is partially ordered by $\leq_A$ and α_A is a map $C \to A$. If $c \in C$, $a \in A$ and $\alpha_A(c) \leq_A a$ we say that a is a *correct approximation* of c. With an abuse of notation, we will sometimes identify an abstract interpretation (A, α_A) with its abstract domain A.

Often C is endowed with a partial ordering (called *computational ordering*) which is used in fixpoint computations. Computational ordering and approximation ordering may be unrelated, hence we do not require the map α_A in an abstract interpretation to be monotone.

2.1. Abstractions and Galois connections

It is worth noting that the abstraction map in our definition of abstraction is not required to be part of a Galois connection. Many abstractions are more naturally formalized in term of abstraction functions only. As a very elementary example, consider the classical "rule of sign" for checking the correctness of the sign of the product xy of two integer numbers, which exploits the fact that the sign of the multiplication xy can be recovered from the sign of x and the sign of y. If we formalize this rule as an abstract interpretation, we may choose $\mathbb{Z}$ as the concrete domain, $\textsc{Sign} = \{pos, neg, zero\}$ as the abstract domain, and $\alpha_S : \mathbb{Z} \to \textsc{Sign}$ as:

$$\alpha_S(x) = \begin{cases} pos & \text{if } x > 0; \\ neg & \text{if } x < 0; \\ zero & \text{if } x = 0. \end{cases}$$

Note that for Sign we use the flat ordering: a is a correct abstraction of x iff a is exactly $\alpha_S(x)$.

In this case α_S is not part of a Galois connection. Actually, α_S is not even monotone if we use the standard ordering for $\mathbb{Z}$. We may turn the rule of sign into a Galois connection if, for example, we take $C = \mathcal{P}(\mathbb{Z})$ as the concrete domain, $A = \mathcal{P}(\text{Sign})$ as the abstract domain and $\alpha : C \to A$ defined as the pointwise extension of α_S, i.e., $\alpha(X) = \{\alpha_S(x) \mid x \in X\}$, where both A and C are ordered by subset inclusion. In this case $\gamma(X) = \{x \in \mathbb{Z} \mid \alpha_S(x) \in X\}$ is the right adjoint of α.

All the abstractions we show in this paper can be turned into Galois connections only if we change the concrete domain. This actually means selecting a collecting semantics among several possible choices, and it is the main topic of the paper. Only rarely the standard semantics of a programming language directly forms a Galois connection with their common abstractions. A notable example is the T_P like semantics for goal-independent semantic of logic programming [11].

2.2. Examples of abstractions

We take as applicative setting the abstract interpretation of functional programs. For easiness of presentation, we consider a simple case where the concrete domain C is the set $\mathcal{D}_\perp \to \mathcal{D}_\perp$ of unary functions from a base domain $\mathcal{D}_\perp$ to itself, and assume that $\mathcal{D}_\perp$ has a distinguished value $\perp$ which represents non-terminating computations. Depending on the concrete domain, $\perp$ may also encode run-time errors. All the results presented are independent from its concrete meaning. Some of the results which follow depend on the fact that $\mathcal{D}_\perp$ has at least an element different from $\perp$. The case when $\mathcal{D}_\perp = \{\perp\}$ is a singleton is trivial and will not be considered.

Example 2 *(A simple program).* Consider a simple setting where the $\mathcal{D} = \mathbb{N}$ is the set of natural numbers. Then, the program

```
let rec prog n = if (n = 2) then prog(n) else n
```

has the following semantics:

$$[\![prog]\!] = \lambda n. \begin{cases} \perp & \text{if } n = 2 \text{ or } n = \perp; \\ n & \text{otherwise.} \end{cases}$$

An example of a simple and useful abstraction is *strictness*. We say that a function $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$ is strict if $f(\perp) = \perp$. Strictness analysis is used to prove that a function f either diverges or needs its arguments. The next definition formalizes strictness analysis as an abstract interpretation.

Definition 3 *(Strictness).* The *strictness* abstract interpretation [24] is $\text{Str} = (\{str \leq \top\}, \alpha_{str})$ where

$$\alpha_{str}(f) = \begin{cases} str & \text{if } f(\perp) = \perp, \\ \top & \text{otherwise.} \end{cases}$$

Another common abstraction for $\mathcal{D}_\perp \to \mathcal{D}_\perp$ is *constancy*. Constancy captures the fact that a function $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$ either diverges or ignores its arguments, i.e., $f(x) = f(\perp)$ for any $x \in \mathcal{D}_\perp$.

Definition 4 *(Constancy).* The *constancy* abstract interpretation [27] is $\text{Const} = (\{const \leq \top\}, \alpha_{const})$ where

$$\alpha_{const}(f) = \begin{cases} const & \text{if } \forall x \in \mathcal{D}_\perp,\ f(x) = f(\perp), \\ \top & \text{otherwise.} \end{cases}$$

Consider the program *prog* defined in Example 2. Since *prog* uses its argument and does not always diverge, we have $\alpha_{str}([\![prog]\!]) = str$ and $\alpha_{const}([\![prog]\!]) = \top$.

Other examples of simple and useful abstractions are *totality*, *convergence*, *divergence* [17], *involution* and *idempotence* [21]. We say that a function $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$ is convergent if, for any $x \in \mathcal{D}_\perp$, $f(x) \neq \perp$, and it is divergent if, for any $x \in \mathcal{D}_\perp$, $f(x) = \perp$. We say that a function f is idempotent if $f \circ f = f$, where $\circ$ is the standard function composition operator. Moreover f is an involution if $f \circ f = id$.

Definition 5 *(Totality).* The *totality* abstract interpretation is $\text{Tot} = (\{tot \leq \top\}, \alpha_{tot})$ where

$$\alpha_{tot}(f) = \begin{cases} tot & \text{if } \forall x \in D,\ f(x) \neq \perp \\ \top & \text{otherwise.} \end{cases}$$

Definition 6 *(Convergence).* The *convergence* abstract interpretation is $\textsc{Conv} = (\{conv \leq \top\}, \alpha_{conv})$ where

$$\alpha_{conv}(f) = \begin{cases} conv & \text{if } \forall x \in \mathcal{D}_\perp,\ f(x) \neq \perp \\ \top & \text{otherwise.} \end{cases}$$

Definition 7 *(Divergence).* The *divergence* abstract interpretation is $\textsc{Div} = (\{div \leq \top\}, \alpha_{div})$ where

$$\alpha_{div}(f) = \begin{cases} div & \text{if } \forall x \in \mathcal{D}_\perp,\ f(x) = \perp \\ \top & \text{otherwise.} \end{cases}$$

Definition 8 *(Involution).* The *involution* abstract interpretation is $\textsc{Inv} = (\{inv \leq \top\}, \alpha_{inv})$ where

$$\alpha_{inv}(f) = \begin{cases} inv & \text{if } \forall x \in \mathcal{D}_\perp,\ f(f(x)) = x \\ \top & \text{otherwise.} \end{cases}$$

Definition 9 *(Idempotence).* The *idempotence* abstract interpretation is $\textsc{Ide} = (\{ide \leq \top\}, \alpha_{ide})$ where

$$\alpha_{ide}(f) = \begin{cases} ide & \text{if } \forall x \in \mathcal{D}_\perp,\ f(f(x)) = f(x) \\ \top & \text{otherwise.} \end{cases}$$

If we consider the program *prog* defined in Example 2, we have that $\alpha_{ide}([\![prog]\!]) = ide$ since *prog* describes an idempotent function. However, $\alpha_{tot}([\![prog]\!]) = \alpha_{conv}([\![prog]\!]) = \alpha_{div}([\![prog]\!]) = \alpha_{inv}([\![prog]\!]) = \top$ since $[\![prog]\!](2) = \perp$, $[\![prog]\!](1) = 1$ and $[\![prog]\!]([\![prog]\!](2)) = \perp \neq 2$.

Finally, consider the monotonicity and boundness properties. Assume $\mathcal{D}_\perp$ is partially ordered by $\sqsubseteq$. We say that a function $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$ is *monotone* if $\forall x, y \in \mathcal{D}_\perp$, $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$. Moreover $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$ is *bounded* if there exists $d \in \mathcal{D}_\perp$ such that $\forall x \in \mathcal{D}_\perp,\ f(x) \sqsubseteq d$.

Definition 10 *(Monotonicity).* The *monotonicity* abstract interpretation is $\textsc{Mon} = (\{mon \leq \top\}, \alpha_{mon})$ where

$$\alpha_{mon}(f) = \begin{cases} mon & \text{if } \forall x, y \in \mathcal{D}_\perp, x \sqsubseteq y \text{ implies } f(x) \sqsubseteq f(y) \\ \top & \text{otherwise.} \end{cases}$$

Definition 11 *(Boundness).* The *boundness* abstract interpretation is $\textsc{Bou} = (\{bou \leq \top\}, \alpha_{bou})$ where

$$\alpha_{bou}(f) = \begin{cases} bou & \text{if } \exists d \in \mathcal{D}_\perp, \forall x \in \mathcal{D}_\perp,\ f(x) \sqsubseteq d \\ \top & \text{otherwise.} \end{cases}$$

Let us consider the program *prog* defined in Example 2, where $\sqsubseteq$ is the standard order relation "less than or equal to" extended in such a way that $\perp \sqsubseteq 0$. We have that $\alpha_{mon}([\![prog]\!]) = \top$ since $[\![prog]\!](1) = 1 \not\sqsubseteq \perp = [\![prog]\!](2)$. Moreover, it is easy to check that $\alpha_{bou}([\![prog]\!]) = \top$.

2.3. Comparing abstract interpretations

In order to compare different abstract interpretations, we need to define a notion of relative precision[1] among them. Of course, one might say that there is already a standard notion of relative precision, which is the existence of a Galois connection among the abstract domains. We agree that Galois connections are quite convenient for comparing abstractions, but we do not think that this choice should go unquestioned, and we want to provide more elaborate arguments in favor of this approach. This is because in our setting abstract interpretations are defined through the use of abstraction functions only, and there should be a good reason for introducing Galois connections.

As a first, naïf approach, we may say that, given two abstract interpretations (A, α_A) and (B, α_B) for a concrete domain C, we have that (A, α_A) is more precise than (B, α_B) when all the properties computed by α_B may be recovered from the properties computed by α_A, that is when α_B factors through α_A, i.e., there is a map $\alpha : A \to B$ such that $\alpha_B = \alpha \circ \alpha_A$.

However, this definition of precision is too weak for many purposes. Most of the time, we are not able to compute the abstraction of the concrete semantics $\alpha_A([\![e]\!])$ but only an over-approximation of the real value through an abstract semantics $[\![e]\!]^A \geq_A \alpha_A([\![e]\!])$. In this case, knowing $[\![e]\!]^A$ does not say anything about $\alpha_B([\![e]\!])$, since we have not required α to be monotone.

[1] We use the term *relative precision* to indicate that we deal with precision among abstract domains, since in some work *precision* refers to completeness properties of an abstract domain, see for instance [15,23,6].

Example 12. We say that a function $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$ is *defined constant* if there is $d \in \mathcal{D}$ such that $\forall x \in \mathcal{D}_\perp,\ f(x) = d$. Consider the abstract interpretation (A, α_A) such that $A = \{defcon <_A defconstr <_A \top\}$ and

$$\alpha_A(f) = \begin{cases} defcon & \text{if } f \text{ is defined constant} \\ defconstr & \text{if } f \text{ is strict} \\ \top & \text{otherwise} \end{cases}$$

Note that, although $\alpha_A(f) = defconstr$ exactly when f is strict, the value *defconstr* does not represent the property of being strict but the property of being strict or defined constant.

We may define $\alpha : A \to \textsc{Str}$ such that $\alpha = \{defcon \mapsto \top, defconstr \mapsto str, \top \mapsto \top\}$, and we have $\alpha \circ \alpha_A = \alpha_{str}$. However, α is not monotone: if we have a program e and we know that $[\![e]\!]^A = defconstr \geq \alpha_A([\![e]\!])$, we cannot conclude that $str \geq \alpha_{str}([\![e]\!])$, i.e., that $[\![e]\!]$ is strict.

If we also require monotonicity of α in the definition of precision, things go better. In this case, if we know that $[\![e]\!]^A \geq_A \alpha_A([\![e]\!])$, then we also know $\alpha([\![e]\!]^A) \geq_B \alpha_B([\![e]\!])$, hence an approximate result for the abstract interpretation A gives an approximate result for the abstract interpretation B.

Definition 13 *(Relative α-precision).* Given abstract interpretations (A, α_A) and (B, α_B) over the domain C, we say that A is more α-precise than B when there is a monotone map $\alpha : A \to B$ such that $\alpha \circ \alpha_A = \alpha_B$.

It turns out that the trivial abstraction (C, id) with the flat ordering on C is the most α-precise abstraction. Actually, if (A, α_A) is an abstraction, in order to show that C is more α-precise than A it is enough to choose $\alpha = \alpha_A$, which is monotone since the ordering of C is flat.

2.4. Relative precision and verification

Relative α-precision is still not satisfactory in all the contexts. If abstract interpretation were used for analysis only, it might be a good choice. However, abstract interpretation is also used for verification. In this case, we do not really want to compute $\alpha_A([\![e]\!])$. On the contrary, we fix a property $a \in A$ and want to decide whether $\alpha_A([\![e]\!]) \leq_A a$.

Definition 14 *(Abstract interpretation problem).* Let C be a concrete domain and e be a system whose *concrete semantics* is $[\![e]\!] \in C$. Given an abstract interpretation (A, α) for C and an abstract property $a \in A$, an *abstract interpretation problem* consists in deciding whether

$$\alpha([\![e]\!]) \leq_A a \ .$$

According to this new perspective, if we have abstract interpretations A and B, we might say that A is more precise than B when each abstract interpretation problem over the domain B may be transformed into an equivalent abstract interpretation problem over the domain A. In this way, if we hypothetically have an oracle for verifying properties over A, we can use it to verify properties over B. We may formalize this point of view as follows.

Definition 15 *(Relative γ-precision).* Given two abstract interpretations (A, α_A) and (B, α_B) over the domain C, we say that A is more γ-precise than B when there is $\gamma : B \to A$ such that, for each $c \in C$ and $b \in B$,

$$\alpha_B(c) \leq_B b \iff \alpha_A(c) \leq_A \gamma(b) \ .$$

We show with a couple of examples that the two variants of relative precision are different from one another.

Example 16. Consider the abstract interpretation $A = \mathcal{D}_\perp \cup \{\top\}$ with $d <_A \top$ for each $d \in \mathcal{D}_\perp$. The idea is that $\top$ stands for the trivial property enjoyed by every function, while $d \in \mathcal{D}_\perp$ is the property of being constantly equal to d. In other words:

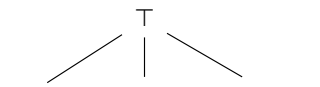
$$\alpha_A(f) = \begin{cases} d & \text{if } \forall x \in \mathcal{D}_\perp,\ f(x) = d, \\ \top & \text{otherwise} \end{cases}$$

There is a monotone map $\alpha : A \to \textsc{Const}$ given by

$$\alpha(a) = \begin{cases} const & \text{if } a \neq \top, \\ \top & \text{if } a = \top \end{cases}$$

and $\alpha_{const} = \alpha \circ \alpha_A$. Therefore, A is more α-precise than *const*. However, there is no direct way to decide whether a map f is constant by having a verifier for the abstract interpretation problems in A. Actually to check whether $\alpha_{const}(f) \leq_A const$, we would need to solve a possibly infinite number of problems $\alpha_A(f) \leq d$ for each $d \in \mathcal{D}_\perp$. Therefore, A is not more γ-precise than Const.

Example 17. Consider the domain (A, α_A) given by



$$\alpha_A(f) = \begin{cases} str & \text{if } f \text{ is strict} \\ defcon & \text{if } f \text{ is defined constant} \\ other & \text{otherwise} \end{cases}$$

and (B, α_B) which is defined in exactly the same way but without the $\top$ element. Note that α_A never returns $\top$. Then, A is more γ-precise than B: we just define $\gamma : B \to A$ as the injection of A in B. However, there is no monotone map $\alpha : A \to B$ such that $\alpha_B = \alpha \circ \alpha_A$ since there is no way to map $\top$ consistently.

Although in the general case the two concepts of precision are different, there are several ways in which they may collapse. This happens, for example, when α_A is surjective.

Proposition 18. *If α_A is surjective, and A is more γ-precise than B, then A is more α-precise than B and there is a Galois connection $\langle \alpha, \gamma \rangle : A \leftrightarrows B$ such that $\alpha \circ \alpha_A = \alpha_B$.*

Note that while the trivial abstraction (C, id) is the most α-precise abstract interpretation, the same is not true when considering γ-precision.

Example 19. The trivial abstraction is not more γ-precise than Str. This would amount to the existence of a function $\gamma : \text{Str} \to C$ such that $\alpha_{str}(f) = str$ iff $f = \gamma(str)$. In other word, this would be possible only if there were a unique strict function, which is obviously not true.

In the general case, the fact that (A, α_A) is more α-precise and γ-precise than (B, α_B) does not imply that $\langle \alpha, \gamma \rangle$ is a Galois connection, as shown by the following example.

Example 20. Let $A = (\{str, other, \top\}, \leq_A)$ with $str <_A other <_A \top$. Given a function $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$, we define:

$$\alpha_A(f) = \begin{cases} str & \text{if } f \text{ is strict,} \\ \top & \text{otherwise.} \end{cases}$$

Let $B = (\{str, \top\}, \leq_B)$ with $str <_B \top$ and $\alpha_B(f) = \alpha_A(f)$ for each $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$. Let $\alpha : A \to B$ be the map $\{str \to str, other \to str, \top \to \top\}$ while $\gamma : B \to A$ is the identity map.

We have that (A, α_A) is more α-precise than (B, α_B), since α is monotone and α is the identity on the image of α_A. Moreover, (A, α_A) is more γ-precise than (B, α_B). Actually, $\leq_B$ and $\leq_A$ do coincide on B, hence, for all $b \in B$ we have that

$$\alpha_B(f) \leq_B b \iff \alpha_B(f) \leq_A b \iff \alpha_A(f) \leq_A \gamma(b) \ .$$

However α and γ do not form a Galois connection, since $\alpha(other) \leq str$ but $other \not\leq \gamma(str)$.

2.5. Derivability among semantics

In the intuitive definition of collecting semantics, one of the requirements is that the other abstractions may be *derived* from it. What does it mean that an abstraction (B, α_B) may be derived from (A, α_A)? It seems unavoidable that each sensible definition of "may be derived from A" means that A is more α-precise than B. In the hypothetical case that we are able to compute everything without losing information, we want that going through the collecting semantics instead of the standard semantics does not lose information.

However, derivable also means that if we have an (approximate) constructive definition of an abstract semantics $[\![e]\!]^A \geq_A \alpha_A([\![e]\!])$, this may be used as a guide for a constructive definition of an abstract semantics $[\![e]\!]^B \geq_B \alpha_B([\![e]\!])$.

A common case is when $[\![e]\!] = F_e^\omega(\perp_C)$ for some map $F_e : C \to C$ and the abstract semantics in A is built by simulating in the abstract domain the progression of the concrete iterates. It means that there is a map $F_{A,e} : A \to A$ such that $F^i_{A,e}(\alpha_A(\perp_C)) \geq_A \alpha_A(F^i_e(\perp_C))$ and there is an abstract join $\amalg_A : \wp(A) \to A$ which is a correct approximation of the least upperbound of C, i.e., if $a_i \geq_A \alpha_A(c_i)$ for each $i < \omega$, then $\amalg_A a_i \geq_A \alpha_A(\bigsqcup_i c_i)$. The abstract semantics on A is defined as $[\![e]\!]^A = \amalg_A\{F^i_{A,e}(\alpha_A(\perp_C)) \mid i < \omega\}$.

In this situation, if A is both more α-precise and γ-precise than B, we may define an abstract semantics $[\![e]\!]^B$ as $\amalg_B\{F^i_{B,e}(\alpha_B(\bot_C)) \mid i < \omega\}$ where $F_{B,e} = \alpha \circ F_{A,e} \circ \gamma$ and $\amalg_B(X) = \alpha(\amalg_A\{\gamma(x) \mid x \in X\})$. It is possible to prove that $[\![e]\!]^B \geq_B \alpha_B([\![e]\!])$.

The cornerstone of the correctness proof is the following proposition. We show the proof here since we think it is important to follow the remaining part of this section.

Proposition 21. *Let (A, α_A) and (B, α_B) be abstractions such that A is both more α-precise and γ-precise than B. If $F_{A,e} : A \to A$ preserves correctness of abstractions, i.e., for any $a \in A, c \in C, a \geq_A \alpha_A(c)$ implies $F_{A,e}(a) \geq_A \alpha_A(F_e(c))$, then $F_{B,e} = \alpha \circ F_{A,e} \circ \gamma$ also preserves correctness of abstractions.*

Proof. Given $b \in B$ and $c \in C$, assume $b \geq_B \alpha_B(c)$. Since A is more γ-precise than B, we have

$$\gamma(b) \geq_A \alpha_A(c) \ .$$

By correctness of $F_{A,e}$, it holds that

$$F_{A,e}(\gamma(b)) \geq_A \alpha_A(F_e(c))$$

and by composing with the monotone map α, we have

$$F_{B,e}(b) \geq_B \alpha_B(F_e(c)) \ . \quad \square$$

Although this result is similar to known results on correctness of abstract operators, it is essential in this proof that we track the behavior of the concrete iterates. Therefore, we have three semantics involved: the concrete one and two abstract ones. This is because the two definitions of relative precision are essentially blind on how A and B are related on elements which are not an abstraction of concrete values. If we want to work more freely without having to continuously refer to the concrete semantics, we may require that α and γ form a Galois connection. This is actually not very far from what we already have, as shown in Proposition 18.

Definition 22 *(Relative precision).* Given (A, α_A) and (B, α_B) two abstract interpretations over the domain C, we say that A is more precise than B when there is a Galois connection $\langle \alpha, \gamma \rangle : A \leftrightarrows B$ such that $\alpha_B = \alpha \circ \alpha_A$.

This is essentially the same definition of precision given in [14], with the additional requirement that α should respect the abstraction maps α_A and α_B which have been given beforehand. Note that we use the term "relative" precision since in some papers the term precision is used to mean α-completeness, which is a different concept [23].

Clearly, relative precision is stronger than both α-precision and γ-precision. When we have a Galois connection, correctness of the abstract semantics A w.r.t. the abstract semantics B may be rephrased without involving concrete semantics, as in the next well known result.

Proposition 23. *Given $F_A : A \to A$ monotone, if A is more precise than B through the Galois connection $\langle \alpha, \gamma \rangle$, then $F_B = \alpha \circ F_A \circ \gamma$ is correct, i.e., if $b \geq_B \alpha(a)$, then $F_B(b) \geq_B \alpha(F_A(a))$.*

In the following, we use the definition of relative precision based on Galois connections. Please note that if we only have an abstraction (A, α_A) and a Galois connection $\langle \alpha, \gamma \rangle : A \leftrightarrows B$, then we may define an abstraction over B as $(B, \alpha \circ \alpha_A)$ such that A is more precise than B. However, we are taking a different point of view where abstractions are defined directly from the concrete semantics, so that condition $\alpha_B = \alpha \circ \alpha_A$ should be included explicitly.

2.6. Collecting semantics

The conclusion from the previous sections is that if we want to compare different abstract interpretations, the notion of relative precision based on Galois connection conjugates in a simple way different aspects relative to precision (α- and γ-precision) with aspects relative to derivability between abstract semantics. The latter is particularly important when one of the two semantics we are considering is a collecting semantics, since its main purpose is guiding the design of an abstract semantics.

When we want to design an abstract semantics, we can either take the concrete semantics as the reference, and use a relatively poor mathematical framework, or use the collecting semantics as the reference, and derive the abstract semantics using the Galois connection framework.

Given a family of abstractions $\mathcal{F}$, a question which arises is which is the "best" collecting semantics for the abstractions in $\mathcal{F}$. Obviously, we require the collecting semantics to be more precise than all the abstractions in $\mathcal{F}$. However, we might require something more. In the general case, if A is more precise than B, there are several Galois connections $\langle \alpha, \gamma \rangle : A \leftrightarrows B$ such that $\alpha \circ \alpha_A = \alpha_B$. If (S, α_S) is a collecting semantics for a family $\mathcal{F}$, it would be better to have a unique Galois connection from S to each abstraction in $\mathcal{F}$.

Definition 24 *(Adequate collecting semantics).* Given a domain C and a family $\mathcal{F}$ of abstractions, an abstraction (S, α_S) is a collecting semantics adequate for $\mathcal{F}$ when, for each abstraction $(A, \alpha_A) \in \mathcal{F}$, there is a unique Galois connection $\langle \alpha, \gamma \rangle : S \leftrightarrows A$ such that $\alpha \circ \alpha_S = \alpha_A$.

In the rest of the paper, we analyze some common collecting semantics and abstractions for the case of first-order functional languages, we study their relative precision, and we characterize the class of abstractions for which the collecting semantics are adequate.

3. Collecting semantics for functional programs

3.1. Collecting semantics

We define two abstract interpretations CS_1 and CS_2 which are commonly used as collecting semantics for the concrete domain $\mathcal{D}_\perp \to \mathcal{D}_\perp$.

Definition 25 *(Collecting Semantics* CS_1*).* We define the abstract interpretation CS_1 on $\mathcal{D}_\perp \to \mathcal{D}_\perp$ as:

$$\mathsf{CS}_1 = (\mathcal{P}(\mathcal{D}_\perp) \xrightarrow{\cup} \mathcal{P}(\mathcal{D}_\perp), \alpha_{\mathsf{CS}1})$$

where $\mathcal{P}(\mathcal{D}_\perp) \xrightarrow{\cup} \mathcal{P}(\mathcal{D}_\perp)$ is the set of complete join-morphisms[2] ordered pointwise,

$$\alpha_{\mathsf{CS}_1}(f) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp).f(X)$$

and $f(X)$ is the image of f through X.

Note that, differently from the definition in Cousot and Cousot [17], CS_1 is restricted to maps $\mathcal{P}(\mathcal{D}_\perp) \xrightarrow{\cup} \mathcal{P}(\mathcal{D}_\perp)$ which are complete join-morphisms, otherwise there would be multiple abstract objects which approximate exactly the same set of concrete functions. This will be important when proving the adequacy of the CS_1 semantics. A different solution could be to change CS_1 to $\mathcal{D}_\perp \to \mathcal{P}(\mathcal{D}_\perp)$.

Example 26. The restriction to join-morphisms is required since the approximation of any function $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$ in a function $\phi : \mathcal{P}(\mathcal{D}_\perp) \xrightarrow{\cup} \mathcal{P}(\mathcal{D}_\perp)$ is actually completely characterized from the behavior of ϕ on singletons. For instance, given $d \in \mathcal{D}$, consider the following functions ϕ_0 and ϕ_1:

$$\phi_0 = \lambda X. \begin{cases} \emptyset & \text{if } X = \emptyset, \\ \{\perp\} & \text{otherwise,} \end{cases} \qquad \phi_1 = \lambda X. \begin{cases} \emptyset & \text{if } X = \emptyset, \\ \{\perp\} & \text{if } |X| = 1, \\ \mathcal{D}_\perp & \text{otherwise.} \end{cases}$$

Both ϕ_0 and ϕ_1 approximates only a single function, namely the function which diverges for any input, but ϕ_0, which is a complete join-morphisms, seems to be a cleaner choice.

Definition 27 *(Collecting Semantics* CS_2*).* We define the abstract interpretation CS_2 on $\mathcal{D}_\perp \to \mathcal{D}_\perp$ as:

$$\mathsf{CS}_2 = (\mathcal{P}(\mathcal{D}_\perp \to \mathcal{D}_\perp), \alpha_{\mathsf{CS}_2})$$

where $\mathcal{P}(\mathcal{D}_\perp \to \mathcal{D}_\perp)$ is ordered by standard subset inclusion and

$$\alpha_{\mathsf{CS}_2}(f) = \{f\} \ .$$

3.2. Strictness and collecting semantics

Consider the relation between α_{CS_1} and α_{str}. First of all, note that it is possible to recover $\alpha_{str}(\llbracket e \rrbracket)$ from $\alpha_{\mathsf{CS}_1}(\llbracket e \rrbracket)$, since it holds that $\alpha_{str} = \alpha_{1str} \circ \alpha_{\mathsf{CS}_1}$ where $\alpha_{1str} : \mathsf{CS}_1 \to \textsc{Str}$ is defined as:

$$\alpha_{1str}(\phi) = \begin{cases} str & \text{if } \phi(\{\perp\}) \subseteq \{\perp\} \\ \top & \text{otherwise} \end{cases}$$

[2] A function ϕ is a complete join morphism when $\phi(\cup_{i \in I} X_i) = \cup_{i \in I} \phi(X_i)$ for any index set I.

for each $\phi : \mathcal{P}(\mathcal{D}_\perp) \xrightarrow{\cup} \mathcal{P}(\mathcal{D}_\perp)$. Therefore CS_1 is more α-precise than strictness. Moreover, consider the problem $\alpha_{str}([\![e]\!]) \leq str$. It is immediate to show that this is equivalent to $\alpha_{\mathrm{CS}_1}([\![e]\!])(\{\perp\}) \subseteq \{\perp\}$. In turn, this is equivalent to $\alpha_{\mathrm{CS}_1}([\![e]\!]) \leq \phi_{str}$ by defining

$$\phi_{str} = \lambda X. \begin{cases} \emptyset & \text{if } X = \emptyset, \\ \{\perp\} & \text{if } X = \{\perp\}, \\ D_\perp & \text{otherwise.} \end{cases}$$

This happens because the functions correctly approximated by ϕ_{str} are exactly all the strict functions. The problem $\alpha_{str}([\![e]\!]) \leq \top$ is always true, and it is equivalent to $\alpha_{\mathrm{CS}_1}([\![e]\!]) \leq \top_{\mathrm{CS}_1}$ where $\top_{\mathrm{CS}_1}(X) = \mathcal{D}_\perp$ for any non empty X. Therefore, each strictness problem may be reduced to a problem on the collecting semantics CS_1. If we define $\gamma_{1str} : \textsc{Str} \to \mathrm{CS}_1$ as:

$$\gamma_{1str}(str) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp). \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{\perp\} & \text{if } X = \{\perp\} \\ \mathcal{D}_\perp & \text{otherwise} \end{cases}$$

$$\gamma_{1str}(\top) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp). \begin{cases} \emptyset & \text{if } X = \emptyset, \\ \mathcal{D}_\perp & \text{otherwise} \end{cases}$$

we have that CS_1 is more γ-precise than Str and $\langle \alpha_{1str}, \gamma_{1str} \rangle$ is a Galois connection.

Note that the same holds for the collecting semantics CS_2. Actually, CS_2 is more α-precise than Str by taking

$$\alpha(F) = \begin{cases} str & \text{if } \forall f \in F,\ f(\perp) = \perp \\ \top & \text{otherwise.} \end{cases}$$

Moreover $\alpha_{str}([\![e]\!]) \leq str$ is equivalent to $\alpha_{\mathrm{CS}_2}([\![e]\!]) \subseteq F_{str}$ where $F_{str} = \{f \mid \alpha_{str}(f) = str\}$. This make CS_2 more γ-precise than Str by defining $\gamma : \mathrm{CS}_2 \to \textsc{Str}$ such that $\gamma(str) = F_{str}$ and $\gamma(\top) = \mathcal{D}_\perp \to \mathcal{D}_\perp$.

Analogously to the strictness property, it is easy to show that also the convergence, divergence and totality properties may be reduced to a problem on the collecting semantics CS_1 by defining

$$\phi_{conv} = \lambda X. \begin{cases} \emptyset & \text{if } X = \emptyset, \\ \mathcal{D} & \text{otherwise,} \end{cases} \qquad \phi_{div} = \lambda X. \begin{cases} \emptyset & \text{if } X = \emptyset, \\ \{\perp\} & \text{otherwise,} \end{cases}$$

$$\phi_{tot} = \lambda X. \begin{cases} \emptyset & \text{if } X = \emptyset, \\ \mathcal{D}_\perp & \text{if } \perp \in X, \\ \mathcal{D} & \text{otherwise.} \end{cases}$$

The analogous result for CS_2 is immediate.

3.3. Constancy and collecting semantics

Not all the abstract interpretation problems may be reduced to problems in a given collecting semantics. Consider the problem $\alpha_{const}([\![e]\!]) \leq const$. It may be easily reduced to a problem in CS_2, by $\alpha_{\mathrm{CS}_2}([\![e]\!]) \subseteq F_{const}$ where $F_{const} = \{f \mid \alpha_{const}(f) = const\}$, however it cannot be reduced to a problem in CS_1. We must be careful to understand what this means. It is still possible to recover $\alpha_{const}(f)$ from $\alpha_{\mathrm{CS}_1}(f)$: actually, $\alpha_{const} = \alpha_{1const} \circ \alpha_{\mathrm{CS}_1}$ where $\alpha_{1const} : \mathrm{CS}_1 \to \textsc{Const}$ is defined as

$$\alpha_{1const}(\phi) = \begin{cases} const & \text{if } \forall x \in \mathcal{D}_\perp.\ \phi(\{x\}) \cap \phi(\{\perp\}) \neq \emptyset, \\ \top & \text{otherwise.} \end{cases}$$

Therefore, CS_1 is more α-precise than Const. However, there is no element in CS_1 which corresponds to the set of all the constant functions. Actually, if f is an unknown constant function, we have that $f(x)$ might be potentially equal to any value $x \in \mathcal{D}_\perp$. Therefore, the only abstract object $\phi \in \mathcal{P}(\mathcal{D}_\perp) \xrightarrow{\cup} \mathcal{P}(\mathcal{D}_\perp)$ which is a correct approximation for all the constant functions is the greatest element

$$\lambda X. \begin{cases} \emptyset & \text{if } X = \emptyset, \\ \mathcal{D}_\perp & \text{otherwise.} \end{cases}$$

Unfortunately, this is a correct approximation for all the functions, not only the constant ones. More in general, CS_1 cannot express any constraint relating the results of a function for different inputs (it is a so-called *non-relational domain*).

Hence, there is no ϕ_{const} such that $\alpha_{const}(f) \leq const$ is equivalent to $\alpha_{\mathrm{CS}_1}(f) \leq \phi_{const}$, i.e., CS_1 is not more γ-precise than Const. Therefore, the collecting semantics CS_1 is not well suited to analyze the constancy property.

3.4. Involution, idempotence, monotonicity and collecting semantics

Analogously to the constancy property, if we consider the involution property and the problem $\alpha_{inv}(\llbracket e \rrbracket) \leq inv$, we have that it may be reduced to a problem in CS_2, by $\alpha_{\mathrm{CS}_2}(\llbracket e \rrbracket) \subseteq F_{inv}$ where $F_{inv} = \{f \mid \alpha_{inv}(f) = inv\}$. However, as for the previous case, it cannot be reduced to a problem in CS_1, even if $\alpha_{inv} = \alpha_{1inv} \circ \alpha_{\mathrm{CS}_1}$ where $\alpha_{1inv} : \mathrm{CS}_1 \to \textsc{Inv}$ is defined as

$$\alpha_{1inv}(\phi) = \begin{cases} inv & \text{if } \forall x \in \mathcal{D}_\perp.\ \phi(\phi(\{x\})) \supseteq \{x\}, \\ \top & \text{otherwise.} \end{cases}$$

In fact, since there is no element in CS_1 which corresponds to the set of all the involutions, there is no ϕ_{inv} such that $\alpha_{inv}(f) \leq inv$ is equivalent to $\alpha_{\mathrm{CS}_1}(f) \leq \phi_{inv}$. The same holds for Ide, Mon and Bou.

Therefore, the collecting semantics CS_1 is not well suited to analyze the involution, idempotence, monotonicity and boundness properties.

3.5. Relationships among collecting semantics

We now explore how the two collecting semantics relate to each other. We notice that both α_{CS_1} and α_{CS_2} factor through the other. Actually, $\alpha_{\mathrm{CS}_2} = \alpha_{12} \circ \alpha_{\mathrm{CS}_1}$ where $\alpha_{12} : \mathrm{CS}_1 \to \mathrm{CS}_2$ is given by

$$\alpha_{12}(\phi) = \{f : \mathcal{D}_\perp \to \mathcal{D}_\perp \mid \forall x \in \mathcal{D}_\perp, f(x) \in \phi(\{x\})\}\ ,$$

for each $\phi \in \mathcal{P}(\mathcal{D}_\perp) \overset{\cup}{\to} \mathcal{P}(\mathcal{D}_\perp)$, while $\alpha_{\mathrm{CS}_1} = \alpha_{21} \circ \alpha_{\mathrm{CS}_2}$ with $\alpha_{21} : \mathrm{CS}_2 \to \mathrm{CS}_1$ given by

$$\alpha_{21}(F) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp).\ \bigcup \{f(X) \mid f \in F\}$$

for each $F \in \mathcal{P}(\mathcal{D}_\perp \to \mathcal{D}_\perp)$. This contrasts with the standard consideration that CS_2 is more precise than CS_1 and not vice versa. However, the different precision between the two is apparent when we note that, for each $\phi \in \mathrm{CS}_1$, we have

$$\alpha_{\mathrm{CS}_1}(\llbracket e \rrbracket) \leq \phi \iff \alpha_{\mathrm{CS}_2}(\llbracket e \rrbracket) \subseteq \alpha_{12}(\phi)\ .$$

However, the converse is not true: given $F \in \mathrm{CS}_2$, in the general case there is no $\phi_F \in \mathrm{CS}_1$ such that

$$\alpha_{\mathrm{CS}_2}(\llbracket e \rrbracket) \subseteq F \iff \alpha_{\mathrm{CS}_1}(\llbracket e \rrbracket) \leq \phi_F\ . \tag{1}$$

Therefore CS_2 is more γ-precise than CS_1 but not vice versa.

Example 28 *(CS_1 is not more γ-precise than CS_2).* Let $F = \{\lambda x.\perp, \lambda x.a\}$ for a given $a \in \mathcal{D}$. Then $\alpha_{\mathrm{CS}_2}(\llbracket e \rrbracket) \subseteq F$ iff e is a program which always diverges or always terminates with result a. If we choose $\phi_F = \alpha_{21}(F)$, which one might think as a sensible choice in (1), we have $\phi_F(X) = \{\perp, a\}$ for any non empty X. Therefore, if e is a program which always diverges or always terminates with result a, we have $\alpha_{\mathrm{CS}_1}(\llbracket e \rrbracket) \leq \phi_F$. However, the same holds for any program which returns both outputs $\perp$ and a for different inputs. Therefore, the left and right hand-sides in (1) are not equivalent. In general, for any ϕ_F we may choose, the condition $\alpha_{\mathrm{CS}_1}(\llbracket e \rrbracket) \leq \phi_F$ is not able to select functions which always return the same value.

The following proposition summarizes the previous arguments.

Proposition 29. *The following holds:*

- *there is no Galois connection $\langle \alpha_{12}, \gamma_{12} \rangle : \mathrm{CS}_1 \leftrightharpoons \mathrm{CS}_2$ such that $\alpha_{\mathrm{CS}_2} = \alpha_{12} \circ \alpha_{\mathrm{CS}_1}$*
- *α_{21} has a right adjoint, which is α_{12}*
- *α_{1str} has a right adjoint which is $\gamma_{1str} : \textsc{Str} \to \mathrm{CS}_1$ defined as:*

$$\gamma_{1str}(str) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp).\ \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{\perp\} & \text{if } X = \{\perp\} \\ \mathcal{D}_\perp & \text{otherwise} \end{cases}$$

$$\gamma_{1str}(\top) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp).\ \begin{cases} \emptyset & \text{if } X = \emptyset, \\ \mathcal{D}_\perp & \text{otherwise} \end{cases}$$

- *there is no Galois connection $\langle \alpha_{1const}, \gamma_{1const} \rangle : \mathrm{CS}_1 \leftrightharpoons \textsc{Const}$ such that $\alpha_{const} = \alpha_{1const} \circ \alpha_{\mathrm{CS}_1}$.*
- *there is no Galois connection $\langle \alpha_{1inv}, \gamma_{1inv} \rangle : \mathrm{CS}_1 \leftrightharpoons \textsc{Inv}$ such that $\alpha_{inv} = \alpha_{1inv} \circ \alpha_{\mathrm{CS}_1}$. The same holds for Ide, Mon and Bou.*

4. Adequacy of collecting semantics

In this section we elaborate on the concepts of adequacy for a family of abstractions. We define a category whose objects are abstract interpretations and whose morphisms are transformations from more precise to less precise abstractions (see, e.g., Asperti and Longo [8] for standard definitions in category theory).

Definition 30 *(Category of abstract interpretations).* We call $\mathbb{AI}(C)$ the category whose objects are abstract interpretations for C and whose morphisms are Galois connection $\langle \alpha, \gamma \rangle : A \leftrightharpoons B$ such that $\alpha_B = \alpha \circ \alpha_A$.

Collecting semantics are a starting point when designing new abstract interpretations. If an abstract interpretation (A, α_A) is designed starting from the collecting semantics (S, α_S), it means that (S, α_S) should be more precise than (A, α_A), hence there should be a map from (S, α_S) to (A, α_A) in our category $\mathbb{AI}(C)$. Moreover, it would be preferable to have a unique way of deriving (A, α_A) as a Galois connection from (S, α_S), according to the concept of adequacy introduced in the previous sections. In the categorical settings, this leads to the notion of initiality: if (S, α_S) is initial for a given full subcategory $\mathbb{D}$ of $\mathbb{AI}(C)$, it means that (S, α_S) is adequate for the family of abstractions in $\mathbb{D}$.

Given a collecting semantics (S, α_S), we are interested in characterizing the maximal full subcategory $\mathbb{D}$ of $\mathbb{AI}(C)$ such that (S, α_S) is initial for $\mathbb{D}$. Such subcategories immediately induce a taxonomy on program analysis which precisely characterizes the program properties suitable for a given collecting semantics. In the rest of the section, we show that for the two collecting semantics CS_1 and CS_2, such a full subcategory can be constructively described.

4.1. The collecting semantics CS_2

We first show that CS_2 is initial for all the abstract interpretations which *have enough joins*, and that this is the largest class of abstract interpretations which enjoys this property.

Definition 31 *(Having enough joins).* We say that the abstract interpretation (A, α_A) has enough joins when $\bigvee_A X$ exists for each X which is a subset of the image of α_A.

Obviously, if A is a complete join-semilattice, than (A, α_A) has enough joins.

Example 32. Let $(\mathcal{D}_\perp, \sqsubseteq)$ be a poset such that $\perp \sqsubseteq d$ for each $d \in \mathcal{D}_\perp$. We say that a function $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$ has a maximum value $d \in \mathcal{D}$ if there exists $y \in \mathcal{D}_\perp$ such that for each $x \in \mathcal{D}_\perp$, $f(x) \sqsubseteq f(y) = d$. Consider the abstract interpretation Max= $(\mathcal{D}_\perp, \alpha_{max})$ such that

$$\alpha_{max}(f) = \begin{cases} f(y) & \text{if } f(y) \in \mathcal{D} \text{ and } \forall x \in \mathcal{D}_\perp,\ f(x) \sqsubseteq f(y) \\ \perp & \text{otherwise} \end{cases}$$

Note that $\alpha_{max}(f) = d \in \mathcal{D}$ when d is the maximum value of f. We have that Max has enough joins if and only if $\mathcal{D}$ is a complete join-semilattice. Therefore, if $\mathcal{D}_\perp = \mathbb{N} \cup \{\perp\}$ with the standard ordering on natural numbers, then $\mathcal{D}_\perp$ has not enough joins.

Theorem 33. *The full subcategory of all the abstract interpretations which have enough joins is the largest class of abstractions for which the collecting semantics* CS_2 *is initial.*

4.2. The collecting semantics CS_1

We now show that the collecting semantics CS_1 is initial for a large class of abstract interpretations, which can be constructively characterized.

Definition 34 *(Mix of functions).* We say that a function $g : \mathcal{D}_\perp \to \mathcal{D}_\perp$ is a mix of the set of functions $F \subseteq \mathcal{D}_\perp \to \mathcal{D}_\perp$ iff for each $x \in \mathcal{D}_\perp$ there exists $f \in F$ such that $g(x) = f(x)$.

Definition 35 *(Mixable interpretations).* Let (A, α_A) be an abstract interpretation such that A has enough joins. We call (A, α_A) *mixable* if, for any set of functions $F \subseteq \mathcal{D}_\perp \to \mathcal{D}_\perp$, whenever g is a mix of functions in F, we have $\alpha_A(g) \leq \bigvee_{f \in F} \alpha_A(f)$.

An interpretation is mixable when deciding whether $\alpha(f) \leq a$ may be done by looking at the values of f for a single element of the domain at a time. This observation is formalized by the following lemma.

Lemma 36. *Let* (A, α_A) *be a mixable interpretation. Then* $\alpha_A(f) \leq a \iff \forall x \in \mathcal{D}_\perp\ \exists f' \text{ s.t. } f'(x) = f(x) \wedge \alpha_A(f') \leq a$.

In the previous lemma, the interesting direction is $\Longleftarrow$, since for the other direction it is enough to take $f' = f$.

Example 37. We apply the above intuitive characterization to the previously defined abstractions.

- The strictness abstract interpretation in Definition 3 is mixable, since we only need to check the single value of $f(\bot)$ in order to decide whether a function is strict.
- The constancy, involution, idempotence, monotonicity and boundness abstract interpretations in Definitions 4, 8, 9, 10 and 11 are not mixable, since we need to compare the values computed by f for different arguments.
- The totality, convergence and divergence abstract interpretations in Definitions 5, 6 and 7 are mixable, since we just need to check the value of f for (many) single arguments, without the need of comparing them.

We now show that all the mixable interpretations may be designed starting from the collecting semantics CS_1.

Theorem 38. *The full subcategory of all the mixable abstract interpretations is the largest class of abstractions for which the collecting semantics* CS_1 *is initial.*

5. The downward closed semantics CS_3

Another commonly used collecting semantics for functional programs is the downward closed collecting semantics defined in Cousot and Cousot [17].

Given a poset $(X, \leq_X)$ and $Y \subseteq X$, we denote by $\downarrow Y$ the *downward closure* of Y, i.e. the set

$$\downarrow Y = \{x \in X \mid \exists y \in Y, x \leq_X y\} \ .$$

Y is *downward closed* if $\downarrow Y = Y$ and we denote by $\mathcal{P}^{\downarrow}(X)$ the set of downward closed subsets of X. Note that $(\mathcal{P}^{\downarrow}(X)), \subseteq)$ is a complete lattice with $\emptyset$ as the bottom element and union as the join.

In our setting the poset X is $\mathcal{P}(\mathcal{D}_\bot)$ with subset ordering. Therefore, given $\Theta \subseteq \mathcal{P}(\mathcal{D}_\bot)$, we have

$$\downarrow \Theta = \{Y \mid \exists X \in \Theta, Y \subseteq X\} \ .$$

Definition 39 *(Collecting semantics* $\overline{\mathrm{CS}_3}$*).* We introduce the abstract interpretation

$$\overline{\mathrm{CS}_3} = \left(\mathcal{P}(\mathcal{P}(\mathcal{D}_\bot)) \overset{\cup}{\to} \mathcal{P}^{\downarrow}(\mathcal{P}(\mathcal{D}_\bot)), \alpha_{\overline{\mathrm{CS}_3}}\right)$$

where $\mathcal{P}(\mathcal{P}(\mathcal{D}_\bot)) \overset{\cup}{\to} \mathcal{P}^{\downarrow}(\mathcal{P}(\mathcal{D}_\bot))$ is ordered by $\dot{\subseteq}$, the pointwise extension of $\subseteq$, and

$$\alpha_{\overline{\mathrm{CS}_3}}(f) = \lambda\Theta \in \mathcal{P}(\mathcal{P}(\mathcal{D}_\bot)). \downarrow\{f(X) \mid X \in \Theta\} \ .$$

Example 40. Let $f : \mathcal{D}_\bot \to \mathcal{D}_\bot$ given as

$$f(x) = \begin{cases} \bot & \text{if } x = \bot, \\ a & \text{otherwise} \end{cases}$$

for some $a \in \mathcal{D}$. Then

$$\alpha_{\overline{\mathrm{CS}_3}}(f)(\Theta) = \begin{cases} \emptyset & \text{if } \Theta = \emptyset, \\ \{\emptyset\} & \text{if } \Theta = \{\emptyset\}, \\ \{\emptyset, \{\bot\}\} & \text{if } \Theta = \{\emptyset, \{\bot\}\} \vee \Theta = \{\{\bot\}\}, \\ \{\emptyset, \{a\}\} & \text{if } \exists X \in \Theta, X \cap \mathcal{D} \neq \emptyset \wedge \forall X \in \Theta, \bot \notin X, \\ \{\emptyset, \{\bot\}, \{a\}\} & \text{if } \Theta = \{\{\bot\}\} \cup \Xi \wedge \exists X \in \Xi, \\ & \quad X \cap \mathcal{D} \neq \emptyset \wedge \forall X \in \Xi, \bot \notin X, \\ \{\emptyset, \{\bot\}, \{a\}, \{\bot, a\}\} & \text{if } \exists X \in \Theta, X \supset \{\bot\}. \end{cases}$$

The abstraction $\alpha_{\overline{\mathrm{CS}_3}}$ is not directly suitable as a collecting semantics, since it contains many objects which are abstraction of exactly the same set of functions in $\mathcal{D}_\bot \to \mathcal{D}_\bot$.

Example 41. Let $\Phi = \alpha_{\overline{\mathrm{CS}_3}}(f)$ the element of $\overline{\mathrm{CS}_3}$ from Example 40 and let Φ' be the following element of $\overline{\mathrm{CS}_3}$:

$$\Phi'(\Theta) = \begin{cases} \downarrow\{\{\bot, a, b\}\} & \text{if } \exists X \in \Theta, X \supset \{\bot\} \\ \Phi(\Theta) & \text{otherwise} \end{cases}$$

where a and b are two distinct elements of $\mathcal{D}$. Note that if f is a function in $\mathcal{D}_\perp \to \mathcal{D}_\perp$ and Φ' is a correct abstraction of f, then $f(x) \neq b$ for each $x \in \mathcal{D}_\perp$. This is because, according to Φ', we have $f(\{x\})$ is either $\{\perp\}$ or $\{a\}$. Therefore, the element b in the first clause of Φ' is useless: Φ' is a correct approximation of f iff Φ is a correct approximation of f.

Definition 42 *(Collecting semantics CS_3).* We introduce the abstract interpretation

$$\mathrm{CS}_3 = \left(\left\{ \Phi \in \overline{\mathrm{CS}_3} \mid \Phi = \dot{\bigcup} \{ \alpha_{\mathrm{CS}_3}(f) \mid \alpha_{\mathrm{CS}_3}(f) \dot{\subseteq} \Phi \} \right\}, \alpha_{\mathrm{CS}_3} \right)$$

where $\alpha_{\mathrm{CS}_3} = \alpha_{\overline{\mathrm{CS}_3}}$.

Note that Φ in Example 41 is in CS_3 but Φ' is not.

Example 43. The constancy abstraction reduces to CS_3. In fact, in order to check whether a function f is constant, it is enough to verify for each $X \subseteq \mathcal{D}_\perp$ that $f(X)$ is a singleton, i.e., it is an element of $\downarrow\{\{d\} \mid d \in \mathcal{D}_\perp\}$. More precisely, the abstract interpretation problem $\alpha_{const}(f) \leq const$ may be reduced to $\alpha_{\mathrm{CS}_3}(f) \dot{\subseteq} \Phi_{const}$ where

$$\Phi_{const}(\Theta) = \begin{cases} \emptyset & \text{if } \Theta = \emptyset, \\ \{\emptyset\} & \text{if } \Theta = \{\emptyset\}, \\ \{\emptyset\} \cup \{\{d\} \mid d \in \mathcal{D}_\perp\} & \text{otherwise.} \end{cases}$$

Example 44. The boundness abstraction reduces to CS_3. In fact, in order to check whether a function f is bounded, it is enough to verify for each $X \subseteq \mathcal{D}_\perp$ that $f(X)$ is bounded. More precisely, the abstract interpretation problem $\alpha_{bou}(f) \leq const$ may be reduced to $\alpha_{\mathrm{CS}_3}(f) \dot{\subseteq} \Phi_{bou}$ where

$$\Phi_{bou}(\Theta) = \{X \in \mathcal{P}(\mathcal{D}_\perp) \mid \exists Y \in \Theta, |X| \leq |Y|, \exists d \in \mathcal{D}_\perp, \forall x \in X, x \sqsubseteq d\}$$

The condition on cardinalities is needed to ensure that Φ_{bou} is an element of CS_3. Actually, for any function g, the cardinality of $g(X)$ cannot be greater than the cardinality of X.

Let us compare the semantics CS_3 w.r.t. CS_1 and CS_2. First of all, CS_2 is more α-precise than CS_3. Actually, we may define a monotone map $\alpha_{23} : \mathrm{CS}_2 \to \mathrm{CS}_3$ as

$$\alpha_{23}(F) = \lambda\Theta \in \mathcal{P}(\mathcal{P}(\mathcal{D}_\perp)). \downarrow\{f(X) \mid f \in F \wedge X \in \Theta\} \ .$$

Theorem 45. CS_2 *is more precise than* CS_3.

On the other side, CS_3 is more α-precise than CS_1. Let us define a monotone map $\alpha_{31} : \mathrm{CS}_3 \to \mathrm{CS}_1$ as:

$$\alpha_{31}(\Phi) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp). \bigcup \Phi(\{X\}) \ .$$

Theorem 46. CS_3 *is more precise than* CS_1.

Analogously to the previous abstraction, $\alpha_{\mathrm{CS}_2} = \alpha_{32} \circ \alpha_{\mathrm{CS}_3}$ where $\alpha_{32} : \mathrm{CS}_3 \to \mathrm{CS}_2$ is given by

$$\alpha_{32}(\Phi) = \{f : \mathcal{D}_\perp \to \mathcal{D}_\perp \mid \forall x \in \mathcal{D}_\perp, \{f(x)\} \in \Phi(\{\{x\}\})\} \ ,$$

while $\alpha_{\mathrm{CS}_3} = \alpha_{13} \circ \alpha_{\mathrm{CS}_1}$ where $\alpha_{13} : \mathrm{CS}_1 \to \mathrm{CS}_3$ is given by

$$\alpha_{13}(\phi) = \lambda\Theta \in \mathcal{P}(\mathcal{P}(\mathcal{D}_\perp)).\{X \mid \exists Y \in \Theta, X \subseteq \phi(Y), |X| \leq |Y|\} \ .$$

However, neither of them has a right adjoint. Actually, the next proposition shows that CS_1 is strictly less precise than CS_3, which in turn is strictly less precise than CS_2.

Proposition 47. *The following holds:*

- *there is no Galois connection* $\langle \alpha_{13}, \gamma_{13} \rangle : \mathrm{CS}_1 \leftrightharpoons \mathrm{CS}_3$ *such that* $\alpha_{\mathrm{CS}_3} = \alpha_{13} \circ \alpha_{\mathrm{CS}_1}$;
- *there is no Galois connection* $\langle \alpha_{32}, \gamma_{32} \rangle : \mathrm{CS}_3 \leftrightharpoons \mathrm{CS}_2$ *such that* $\alpha_{\mathrm{CS}_2} = \alpha_{32} \circ \alpha_{\mathrm{CS}_3}$.

5.1. Adequacy for CS_3

We now try to characterize the set of abstractions which may be derived from the CS_3 semantics through a Galois connection. This characterization will turn out to be a generalization of the concept of mixable interpretations we have already introduced for CS_1.

Definition 48 *(Set-Mix of functions).* We say that a function $g : \mathcal{D}_\perp \to \mathcal{D}_\perp$ is a set-mix of the set of functions $F \subseteq \mathcal{D}_\perp \to \mathcal{D}_\perp$ iff for each $X \subseteq \mathcal{D}_\perp$ there exists $f \in F$ such that $g(X) \subseteq f(X)$.

Definition 49 *(Set-Mixable interpretations).* Let (A, α_A) be an abstract interpretation such that A has enough joins. We call (A, α_A) *set-mixable* if, for any set of functions $F \subseteq \mathcal{D}_\perp \to \mathcal{D}_\perp$, whenever g is a set-mix of F, then $\alpha_A(g) \leq \bigvee_{f \in F} \alpha_A(f)$.

In other words, an interpretation A is set-mixable when deciding whether $\alpha_A(f) \leq a$ may be done by looking at the values of f for a single subset X of the elements of the domain at a time. The function f enjoys a given property if, for any X, $f(X)$ is an element of a downward closed set of accepted values. This observation is formalized by the following lemma.

Lemma 50. *Let (A, α_A) be a set-mixable interpretation. Then $\alpha_A(f) \leq a \iff \forall X \subseteq \mathcal{D}_\perp$ exists a map f' s.t. $f(X) \subseteq f'(X) \wedge \alpha_A(f') \leq a$.*

Example 51. It is immediate to see that the constancy abstraction is set-mixable, by exploiting the characterization in Example 43. With respect to the previous intuitive characterization, note that, in order to decide whether f is constant, it is enough to check that, for each set X, the image $f(X)$ is a subset of a singleton.

Theorem 52. *The full subcategory of all the set-mixable abstract interpretations is the largest class of abstractions for which the collecting semantics CS_3 is initial.*

Using the characterization of set-mixable abstract interpretations, we may show that CS_3 is adequate for the boundness abstraction.

Proposition 53. *The boundness abstraction is set-mixable.*

Example 54 *(Permutation abstraction).* An example of an abstraction that cannot be derived from CS_3 through Galois connections is the domain of permutations, $\textsc{Perm} = (\{perm \leq \top\}, \alpha_{perm})$ where

$$\alpha_{perm}(f) = \begin{cases} perm & \text{if } f \text{ is bijective,} \\ \top & \text{otherwise.} \end{cases}$$

Actually, in order to check whether f is a permutation, we could check that $f(X)$ has the same cardinality of X for any X and that $f(\mathcal{D}_\perp) = \mathcal{D}_\perp$. However, this is not possible in set-mixable interpretations, since when examining $f(X)$ we may only check whether it is a subset of a given family of good results, not that it is exactly equal to one of these results.

More formally, consider the constant map $f(x) = d_0$ for some $d_0 \in \mathcal{D}_\perp$. Given $X \subseteq \mathcal{D}_\perp$ not empty, we have $f(X) = \{d_0\}$. Consider a permutation f_X such that there exists $x \in X$ with $f_X(x) = d_0$. Then $f(X) = \{d_0\} \subseteq f_X(X)$ with $\alpha_{perm}(f_X) \leq perm$. Hence, if α_{perm} were set-mixable, it would be $\alpha_{perm}(f) = perm$.

Analogously the domains of involution, idempotent and monotone functions cannot be derived from CS_3.

Example 55 *(Involution functions).* Consider the constant map $f(x) = d_0$ for some $d_0 \in \mathcal{D}_\perp$. Given $X \subseteq \mathcal{D}_\perp$ not empty, let $d_1 \in X$ and consider the involution f_X such that

$$f_X = \lambda x \in \mathcal{D}_\perp. \begin{cases} d_0 & \text{if } x = d_1 \\ d_1 & \text{if } x = d_0 \\ x & \text{otherwise.} \end{cases}$$

Note that, when $d_0 = d_1$, f_X is the identity function. We have that $f(X) = \{d_0\} \subseteq f_X(X)$ with $\alpha_{inv}(f_X) \leq inv$. Hence, if α_{inv} were set-mixable, it would be $\alpha_{inv}(f) = inv$.

Example 56 *(Idempotent functions).* Let $d_0, d_1 \in \mathcal{D}_\perp$ with $d_0 \neq d_1$ and consider the non-idempotent function

$$f = \lambda x \in \mathcal{D}_\perp. \begin{cases} d_1 & \text{if } x = d_0 \\ d_0 & \text{otherwise.} \end{cases}$$

Given $X \subseteq \mathcal{D}_\perp$ not empty, we have

$$f(X) = \begin{cases} \{d_1\} & \text{if } X = \{d_0\} \\ \{d_0\} & \text{if } d_0 \notin X \\ \{d_0, d_1\} & \text{otherwise.} \end{cases}$$

Let us consider the idempotent functions

$$f_X = \begin{cases} \lambda y \in \mathcal{D}_\perp . d_1 & \text{if } X = \{d_0\} \\ \lambda y \in \mathcal{D}_\perp . d_0 & \text{if } d_0 \notin X \\ \lambda y \in \mathcal{D}_\perp . \begin{cases} d_0 & \text{if } y = d_0 \\ d_1 & \text{otherwise,} \end{cases} & \text{otherwise.} \end{cases}$$

Therefore for each $X \subseteq \mathcal{D}_\perp$ not empty, $f(X) \subseteq f_X(X)$ with $\alpha_{ide}(f_X) \leq ide$. Hence, if α_{ide} were set-mixable, it would be $\alpha_{ide}(f) = ide$.

Example 57 *(Monotone functions).* Consider the non-monotone function

$$f = \lambda x \in \mathcal{D}_\perp . \begin{cases} d_1 & \text{if } x \sqsubseteq d_0 \\ d_0 & \text{otherwise} \end{cases}$$

where $d_0, d_1 \in \mathcal{D}_\perp$, $d_0 \neq d_1$ and $d_0 \sqsubseteq d_1$. Given $X \subseteq \mathcal{D}_\perp$ not empty, we have

$$f(X) = \begin{cases} \{d_1\} & \text{if } \forall x \in X, x \sqsubseteq d_0 \\ \{d_0\} & \text{if } \forall x \in X, x \not\sqsubseteq d_0 \\ \{d_0, d_1\} & \text{otherwise.} \end{cases}$$

Let us consider the monotone functions

$$f_X = \begin{cases} \lambda y \in \mathcal{D}_\perp . d_1 & \text{if } \forall x \in X, x \sqsubseteq d_0 \\ \lambda y \in \mathcal{D}_\perp . d_0 & \text{if } \forall x \in X, x \not\sqsubseteq d_0 \\ \lambda y \in \mathcal{D}_\perp . \begin{cases} d_0 & \text{if } y \sqsubseteq d_0 \\ d_1 & \text{otherwise,} \end{cases} & \text{otherwise.} \end{cases}$$

Therefore for each $X \subseteq \mathcal{D}_\perp$ not empty, $f(X) \subseteq f_X(X)$ with $\alpha_{mon}(f_X) \leq mon$. Hence, if α_{mon} were set-mixable, it would be $\alpha_{mon}(f) = mon$.

6. Related work

Since the very beginning of the abstract interpretation theory, there have been work on categorical approaches to abstract interpretation, both to show that abstract interpretation can be rephrased in category theory and to exploit category theory results in the abstract interpretation framework (see, for instance, Abramsky [1], Backhouse and Backhouse [9], Panangaden P. [25], Venet [26]). In this paper, we use category theory in a much more limited way.

6.1. Other formalizations of abstract interpretation

The abstract interpretation framework presented in this paper is the same which appears in Cousot and Cousot [16] under the "existence of a best abstract approximation assumption": it is based on an abstraction function and a partial ordering of abstract properties. However, not all the abstract interpretations may be formalized in this way: sometimes it is necessary to resort to a weaker framework where the relation between the concrete domain C and the abstract domain A is given by an approximation relation $\triangleright_A \subseteq A \times C$. When $a \triangleright_A c$ we say that a is a correct abstraction of c. Generally $\triangleright_A$ is upward closed, i.e., if $a \triangleright_A c$ and $a \leq_A a'$, then $a' \triangleright_A c$ (what Cousot and Cousot [16] call "abstract soundness of upper approximations assumption").

A classical example of this situation arises when using numerical abstract domains ([13,4,2,7]) for instance in polyhedral analysis of imperative programs [20]. In this case, we may think that the concrete domain C is the set of execution traces for a program, while the abstract domain P is the set of maps from program points to finite sets of linear inequations on the program variables. We say that an abstract object π is a correct approximation of the trace t (i.e., $\pi \triangleright_P t$) if and only if, for each program point p, every time t passes through p the corresponding assignment of values to variables in t is a point in the polyhedron $\pi(p)$. We cannot formalize this abstract interpretation with an abstraction function, since if t is

infinite, the convex hull of all the assignments for a given program point p might not have a least polyhedral approximation. Another example of this phenomenon is the domain of parallelotopes [5].

The fact that the abstraction $(P, \triangleright_P)$ cannot be formalized with an abstraction function is reflected by the fact that the relation between $(P, \triangleright_P)$ and the standard collecting semantics is not a Galois connection. The collecting semantics in this case is $(S, \triangleright_S)$ where S is the set of maps from program points to the powerset of assignments of values to variables, and $\pi \triangleright_S t$ when for each program point p, every time t passes through p, the corresponding assignment is an element of $\pi(p)$. Although $(S, \triangleright_S)$ is generally considered the reference collecting semantics for $(P, \triangleright_P)$, the relation between the two is given by a monotonic map $\gamma : P \to S$ with the property that if $\pi \triangleright_P t$ then $\gamma(\pi) \triangleright_S t$. However, for the same reason given above, γ has no left adjoint.

We can easily adapt our presentation to this more general definition of abstract interpretation, where the morphism in the category of abstract interpretations now are the concretization functions, that respect the approximation relations.

Note that, in some settings, a standard semantics is already a collecting semantics, in the sense that many common abstraction may be formulated directly as Galois connections of the standard semantics. This is often the case for (constraint) logic programs. The semantics of computed answers [10] is already a collecting semantics: many properties such as sharing, groundness, freeness used in static analysis of logic programs may be formulated as Galois connections from the s-semantics.

6.2. Future work

As for future work, our aim is twofold. From one side, we would like to refine the theoretical framework to reach a clean formal definition of what a collecting semantics is. In this paper we have been focused on the problem of examining when a collecting semantics is adequate for a given abstraction, but we lack a precise a priori definition of what a collecting semantics is. To this question, the paper only gives partial answers: if we have a family of abstractions, we might define a collecting semantics as an initial object in the appropriate subcategory, but if the family is too small, the initial object might be something so abstract that it should not be called a collecting semantics. Actually, it seems reasonable to require that a collecting semantics should be more α-precise than the standard semantics: this means it essentially contains all the information which is present in the standard semantics, although in a form which is easier to deal with in further abstractions.

A different line of research is applying the framework to different settings, for example to the operational semantics of imperative languages. In this case, the standard semantics of a program might be its execution traces (assuming the language is deterministic) while a common collecting semantics is the map associating to each program point the set of all states when the execution reaches that point. Although this is a common collecting semantics, it is not well suited for analyzing the input-output behavior of programs, so different collecting semantics might be proposed for this application. Particularly relevant to this line of research is the work of [18] on temporal abstract interpretation and the work of [12] on abstract interpretation of transition systems.

7. Conclusion

Any static analysis formalized in the abstract interpretation framework is defined starting from a collecting semantics, on which the meaning of the system is defined, and then providing a set of abstract objects which encode the properties we are interested in. The collecting semantics is then a fundamental choice in the design of any abstract interpretation. As pointed out in Cousot and Cousot [19]:

> If the collecting semantics is too abstract, it has a limited expressive power which also limits the scope of the static analysis/verification methods that can be formally derived by abstract interpretation of this collecting semantics. *[omissis]* On the other hand, if the collecting semantics is too precise, then further abstractions are more complex to express.

While we can find in the literature many results on the concrete domain (the semantic framework) and the abstract properties (the abstract domain and its operations), only a few papers have been devoted to systematically study how to choose the collecting semantics. For instance, both Cousot and Cousot [16] and Cousot and Cousot [17] present many collecting semantics, without a general method for choosing the right one.

Mostly authors simply use one of the already defined collecting semantics or, sometime, they invent a new one for a specific abstraction. Some authors have studied how to compute the optimal collecting semantics for a given abstract interpretation (e.g., Giacobazzi [22] limited to the analysis of logic programming). Such an approach forces inventing new collecting semantics (and thus describing new concrete semantics) every time we change the abstract property to be analyzed.

On the contrary, we study three most commonly used collecting semantics for functional programs, we precisely characterize the set of abstract interpretations which can be defined on them and derive a taxonomy on program analysis. More generally, the definition of the sets of mixable and set-mixable abstract interpretations provides a constructive method, applicable to all the abstract interpretations of functional programs, to decide which collecting semantics should be the starting point for the definition of the analysis.

Our formal definition of the category of abstract interpretations also allows us to formally state the relationships between the common collecting semantics and some standard abstract interpretation, and to spread a new light on the significance and choice of the collecting semantics.

As far as we know, this is the first work where the set of abstract interpretations reducible to a given collecting semantics is precisely characterized, leading to the notions of mixable and set-mixable abstract interpretation, which precisely capture these sets of functions.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix. Proofs

Proof (Proposition 18). First of all, we can define a monotone map $\alpha : A \to B$ such that $\alpha_B = \alpha \circ \alpha_A$. Given $a \in A$, consider any $c \in C$ such that $\alpha_A(c) = a$ and take $\alpha(a) = \alpha_B(c)$. In order to prove that α is well-defined, we show that if $\alpha_A(c_1) \leq_A \alpha_A(c_2)$ then $\alpha_B(c_1) \leq_B \alpha_B(c_2)$.

By Definition 15, by $\alpha_B(c_2) \leq_B \alpha_B(c_2)$ we get $\alpha_A(c_2) \leq_A \gamma(\alpha_B(c_2))$. If $\alpha_A(c_1) \leq_A \alpha_A(c_2)$ then we have $\alpha_A(c_1) \leq_A \gamma(\alpha_B(c_2))$ and again by Definition 15 we get $\alpha_B(c_1) \leq_B \alpha_B(c_2)$. This proves that α is well-defined and monotone. Moreover, $\alpha_B = \alpha \circ \alpha_A$ by definition.

At this point, Definition 15 may be rewritten as

$$\alpha(\alpha_A(c)) \leq_B b \iff \alpha_A(c) \leq_A \gamma(b) \ .$$

Since α_A is surjective, this means

$$\alpha(a) \leq_B b \iff a \leq_A \gamma(b)$$

for each $a \in A$ and $b \in B$, which is the definition of Galois connection. □

Proof (Proposition 29). • The proof is by contradiction. Assume that there exists $\langle \alpha_{12}, \gamma_{12} \rangle : \mathrm{CS}_1 \leftrightharpoons \mathrm{CS}_2$ such that $\alpha_{\mathrm{CS}_2} = \alpha_{12} \circ \alpha_{\mathrm{CS}_1}$. For $i \in \{0, 1\}$, let $f_i \in \mathcal{D}_\perp \to \mathcal{D}_\perp$ be the function $\lambda x \in \mathcal{D}_\perp . d_i$, where $d_i \in \mathcal{D}_\perp$ and $d_0 \neq d_1$ and let

$$f = \lambda x \in \mathcal{D}_\perp . \begin{cases} d_0 & \text{if } x = d_0 \\ d_1 & \text{otherwise.} \end{cases}$$

Then, by definition of CS_1,

$$\phi_i = \alpha_{\mathrm{CS}_1}(f_i) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp) . \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{d_i\} & \text{otherwise} \end{cases}$$

and

$$\phi = \alpha_{\mathrm{CS}_1}(f) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp) . \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{d_0\} & \text{if } X = \{d_0\} \\ \{d_1\} & \text{if } d_0 \notin X \neq \emptyset \\ \{d_0, d_1\} & \text{otherwise.} \end{cases}$$

By definition of CS_1,

$$\phi_0 \vee \phi_1 = \lambda X \in \mathcal{P}(\mathcal{D}_\perp) . \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{d_0, d_1\} & \text{otherwise} \end{cases}$$

and $\phi \leq \phi_0 \vee \phi_1$. Moreover, since $\langle \alpha_{12}, \gamma_{12} \rangle$ is a Galois connection, it immediately follows that α_{12} is a join-morphism, therefore:

$$\begin{aligned} \alpha_{12}(\phi_0 \vee \phi_1) &= \alpha_{12}(\phi_0) \vee \alpha_{12}(\phi_1) \\ &= \alpha_{12}(\alpha_{\mathrm{CS}_1}(f_0)) \vee \alpha_{12}(\alpha_{\mathrm{CS}_1}(f_1))) \\ &= \alpha_{\mathrm{CS}_2}(f_0) \vee \alpha_{\mathrm{CS}_2}(f_1) \\ &= \{f_0\} \cup \{f_1\} \\ &= \{f_0, f_1\}. \end{aligned}$$

Now, we have a contradiction, since $\phi \leq \phi_0 \vee \phi_1$, while

$$\begin{array}{rl} \alpha_{12}(\phi) = & \alpha_{12}(\alpha_{\mathrm{CS}_1}(f)) = \alpha_{\mathrm{CS}_2}(f) = \\ & \{f\} \not\subseteq \{f_0, f_1\} \qquad\qquad = \alpha_{12}(\phi_0 \vee \phi_1) \end{array}$$

and then the thesis.

- We prove that $\alpha_{21}(F) \leq \phi$ iff $F \subseteq \alpha_{12}(\phi)$. Note that

$$\begin{array}{rl} \alpha_{21}(F) \leq \phi & \iff \forall X \subseteq \mathcal{D}_\perp.\ \bigcup\{f(X) \mid f \in F\} \subseteq \phi(X) \\ & \iff \forall X \subseteq \mathcal{D}_\perp \forall f \in F.\ f(X) \subseteq \phi(X)\ . \end{array}$$

  Since ϕ is a complete join-morphism, the last property holds for all $X \subseteq \mathcal{D}_\perp$, iff it holds for all singletons $\{x\}$ with $x \in \mathcal{D}_\perp$. Then $\alpha_{21}(F) \leq \phi$ is equivalent to

$$\forall x \in \mathcal{D}_\perp \forall f \in F.\ f(x) \in \phi(\{x\})\ ,$$

  namely

$$\forall f \in F\, \forall x \in \mathcal{D}_\perp.\ f(x) \in \phi(\{x\})\ .$$

  Hence we get: $\forall f \in F.\ f \in \alpha_{12}(\phi)$ and therefore $F \subseteq \alpha_{12}(\phi)$.

- Since $\mathcal{P}(\mathcal{D}_\perp) \xrightarrow{\cup} \mathcal{P}(\mathcal{D}_\perp)$ is a complete lattice, it is enough to prove that α_{1str} is a complete join-morphism. Let $S \subseteq \mathcal{P}(\mathcal{D}_\perp) \xrightarrow{\cup} \mathcal{P}(\mathcal{D}_\perp)$. By definition $\bigvee S = \lambda X \in \mathcal{P}(\mathcal{D}_\perp).\bigcup_{\phi \in S} \phi(X)$ and therefore

$$\alpha_{1str}\left(\bigvee S\right) = \begin{cases} str & \text{if } \phi(\{\perp\}) \subseteq \{\perp\} \text{ for each } \phi \in S, \\ \top & \text{otherwise.} \end{cases}$$

  By definition of α_{1str},

$$\alpha_{1str}\left(\bigvee S\right) = \begin{cases} str & \text{if } \alpha_{1str}(\phi) = str \text{ for each } \phi \in S, \\ \top & \text{otherwise} \end{cases}$$

  and therefore $\alpha_{1str}(\bigvee S) = \bigvee_{\phi \in S} \alpha_{1str}(\phi)$. The proof that γ_{1str} is the right adjoint of α_{1str} is straightforward and hence it is omitted.

- The proof is by contradiction. Assume that there exists $\langle \alpha_{1const}, \gamma_{1const} \rangle : \mathrm{CS}_1 \leftrightharpoons \textsc{Const}$ such that $\alpha_{const} = \alpha_{1const} \circ \alpha_{\mathrm{CS}_1}$. For $i \in \{0, 1\}$, let $f_i \in \mathcal{D}_\perp \to \mathcal{D}_\perp$ be the function $\lambda x \in \mathcal{D}_\perp.d_i$, where $d_i \in \mathcal{D}_\perp$ and $d_0 \neq d_1$ and let

$$f = \lambda x \in \mathcal{D}_\perp. \begin{cases} d_0 & \text{if } x = \perp \\ d_1 & \text{otherwise.} \end{cases}$$

  Then, by definition of CS_1,

$$\phi_i = \alpha_{\mathrm{CS}_1}(f_i) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp). \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{d_i\} & \text{otherwise} \end{cases}$$

  and

$$\phi = \alpha_{\mathrm{CS}_1}(f) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp). \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{d_0\} & \text{if } X = \{\perp\} \\ \{d_1\} & \text{if } \perp \notin X \neq \emptyset \\ \{d_0, d_1\} & \text{otherwise.} \end{cases}$$

  By definition of CS_1,

$$\phi_0 \vee \phi_1 = \lambda X \in \mathcal{P}(\mathcal{D}_\perp). \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{d_0, d_1\} & \text{otherwise} \end{cases}$$

  Now, we have a contradiction, since $\phi \leq \phi_0 \vee \phi_1$, while

$$\begin{array}{rl} \alpha_{1const}(\phi) = & \alpha_{1const}(\alpha_{\mathrm{CS}_1}(f)) \\ = & \alpha_{const}(f) = \top \not\leq const \\ = & \alpha_{const}(f_0) \vee \alpha_{const}(f_1) \\ = & \alpha_{1const}(\alpha_{\mathrm{CS}_1}(f_0)) \vee \alpha_{1const}(\alpha_{\mathrm{CS}_1}(f_1)) \\ = & \alpha_{1const}(\phi_0) \vee \alpha_{1const}(\phi_1) \\ = & \alpha_{1const}(\phi_0 \vee \phi_1) \end{array}$$

from which the thesis.

- First, we consider the involution property. The proof is by contradiction. Assume that there exists $\langle \alpha_{1inv}, \gamma_{1inv} \rangle : \mathrm{CS}_1 \leftrightarrows \textsc{Inv}$ such that $\alpha_{inv} = \alpha_{1inv} \circ \alpha_{\mathrm{CS}_1}$. Let $f_1 \in \mathcal{D}_\perp \to \mathcal{D}_\perp$ be the function $\lambda x \in \mathcal{D}_\perp . x$ and

$$f_2 = \lambda x \in \mathcal{D}_\perp . \begin{cases} d_0 & \text{if } x = d_1 \\ d_1 & \text{if } x = d_0 \\ x & \text{otherwise} \end{cases}$$

where for $i = 0, 1$, $d_i \in \mathcal{D}_\perp$ and $d_0 \neq d_1$ and let

$$f = \lambda x \in \mathcal{D}_\perp . \begin{cases} d_0 & \text{if } x = d_1 \\ x & \text{otherwise.} \end{cases}$$

Then, by definition of CS_1,

$$\phi_1 = \alpha_{\mathrm{CS}_1}(f_1) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp) . X$$
$$\phi_2 = \alpha_{\mathrm{CS}_1}(f_2) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp) . \begin{cases} (X \setminus \{d_0\}) \cup \{d_1\} & \text{if } d_0 \in X \text{ and } d_1 \notin X \\ (X \setminus \{d_1\}) \cup \{d_0\} & \text{if } d_1 \in X \text{ and } d_0 \notin X \\ X & \text{otherwise} \end{cases}$$

and

$$\phi = \alpha_{\mathrm{CS}_1}(f) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp) . \begin{cases} (X \setminus \{d_1\}) \cup \{d_0\} & \text{if } d_1 \in X \\ X & \text{otherwise} \end{cases}$$

By definition of CS_1,

$$\phi_1 \vee \phi_2 = \lambda X \in \mathcal{P}(\mathcal{D}_\perp) . \begin{cases} X \cup \{d_1\} & \text{if } d_0 \in X \\ X \cup \{d_0\} & \text{if } d_1 \in X \\ X & \text{otherwise} \end{cases}$$

Now, we have a contradiction, since $\phi \leq \phi_1 \vee \phi_2$, while

$$\begin{aligned} \alpha_{1inv}(\phi) &= \alpha_{1inv}(\alpha_{\mathrm{CS}_1}(f)) \\ &= \alpha_{inv}(f) = \top \not\leq inv \\ &= \alpha_{inv}(f_1) \vee \alpha_{inv}(f_1) \\ &= \alpha_{1inv}(\alpha_{\mathrm{CS}_1}(f_1)) \vee \alpha_{1inv}(\alpha_{\mathrm{CS}_1}(f_2)) \\ &= \alpha_{1inv}(\phi_1) \vee \alpha_{1inv}(\phi_2) \\ &= \alpha_{1inv}(\phi_1 \vee \phi_2) \end{aligned}$$

from which the thesis.

Now, we consider the idempotence property. The proof is by contradiction. Assume that there exists $\langle \alpha_{1ide}, \gamma_{1ide} \rangle : \mathrm{CS}_1 \leftrightarrows \textsc{Ide}$ such that $\alpha_{ide} = \alpha_{1ide} \circ \alpha_{\mathrm{CS}_1}$. For $i \in \{0, 1\}$, let $f_i \in \mathcal{D}_\perp \to \mathcal{D}_\perp$ be the idempotent function $\lambda x \in \mathcal{D}_\perp . d_i$, where $d_i \in \mathcal{D}_\perp$ and $d_0 \neq d_1$ and let

$$f = \lambda x \in \mathcal{D}_\perp . \begin{cases} d_0 & \text{if } x = d_1 \\ d_1 & \text{otherwise.} \end{cases}$$

Then, by definition of CS_1, for $i \in \{0, 1\}$,

$$\phi_i = \alpha_{\mathrm{CS}_1}(f_i) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp) . \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{d_i\} & \text{otherwise} \end{cases}$$

and

$$\phi = \alpha_{\mathrm{CS}_1}(f) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp) . \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{d_0\} & \text{if } X = \{d_1\} \\ \{d_1\} & \text{if } d_1 \notin X \neq \emptyset \\ \{d_0, d_1\} & \text{otherwise.} \end{cases}$$

By definition of CS_1,

$$\phi_0 \vee \phi_1 = \lambda X \in \mathcal{P}(\mathcal{D}_\perp). \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{d_0, d_1\} & \text{otherwise} \end{cases}$$

Now, we have a contradiction, since $\phi \leq \phi_0 \vee \phi_1$, while

$$\begin{aligned} \alpha_{1ide}(\phi) =& \alpha_{1ide}(\alpha_{\mathrm{CS}_1}(f)) \\ =& \alpha_{ide}(f) = \top \not\leq ide \\ =& \alpha_{ide}(f_0) \vee \alpha_{ide}(f_1) \\ =& \alpha_{1ide}(\alpha_{\mathrm{CS}_1}(f_0)) \vee \alpha_{1ide}(\alpha_{\mathrm{CS}_1}(f_1)) \\ =& \alpha_{1ide}(\phi_0) \vee \alpha_{1ide}(\phi_1) \\ =& \alpha_{1ide}(\phi_0 \vee \phi_1) \end{aligned}$$

from which the thesis.

Now, let us consider the monotonicity property. The proof is by contradiction. Assume that there exists $\langle \alpha_{1mon}, \gamma_{1mon} \rangle : \mathrm{CS}_1 \leftrightharpoons \textsc{Mon}$ such that $\alpha_{mon} = \alpha_{1mon} \circ \alpha_{\mathrm{CS}_1}$. For $i \in \{0, 1\}$, let $f_i \in \mathcal{D}_\perp \to \mathcal{D}_\perp$ be the function $\lambda x \in \mathcal{D}_\perp . d_i$, where $d_i \in \mathcal{D}_\perp$, $d_0 \sqsubseteq d_1$, $d_1 \not\sqsubseteq d_0$ and let f be the non-monotone function

$$f = \lambda x \in \mathcal{D}_\perp . \begin{cases} d_0 & \text{if } x = d_1 \\ d_1 & \text{otherwise.} \end{cases}$$

Now, the proof is analogous to the previous case and hence it is omitted.

Finally, let us consider the boundness property. The proof is by contradiction. Assume that there exists $\langle \alpha_{1bou}, \gamma_{1bou} \rangle : \mathrm{CS}_1 \leftrightharpoons \textsc{Bou}$ such that $\alpha_{bou} = \alpha_{1bou} \circ \alpha_{\mathrm{CS}_1}$. Let g be a non-bounded function and for each $d \in \mathcal{D}_\perp$ let us consider the constant function $f_d = \lambda x \in \mathcal{D}_\perp . g(d)$. Moreover let $F = \{f_d \mid d \in \mathcal{D}_\perp\}$. By construction for each $d \in \mathcal{D}_\perp$ there exists $f_d \in F$ such that $g(d) = f_d(d)$ and $\alpha_{bou}(f_d) = bou$. Then, by definition of CS_1,

$$\phi = \alpha_{\mathrm{CS}_1}(g) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp). \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{g(d) \mid d \in X\} & \text{otherwise} \end{cases}$$

and

$$\phi_d = \alpha_{\mathrm{CS}_1}(f_d) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp). \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{g(d)\} & \text{otherwise} \end{cases}$$

By definition of CS_1,

$$\bigvee_{f_d \in F} \phi_d = \bigvee_{d \in \mathcal{D}_\perp} \phi_d = \lambda X \in \mathcal{P}(\mathcal{D}_\perp). \begin{cases} \emptyset & \text{if } X = \emptyset \\ \{g(d) \mid d \in \mathcal{D}_\perp\} & \text{otherwise} \end{cases}$$

Now, we have a contradiction, since $\phi \leq \bigvee_{f_d \in F} \phi_d$, while

$$\begin{aligned} \alpha_{1bou}(\phi) =& \alpha_{1bou}(\alpha_{\mathrm{CS}_1}(g)) \\ =& \alpha_{bou}(f) = \top \not\leq bou \\ =& \bigvee_{f_d \in F} \alpha_{bou}(f_d) \\ =& \bigvee_{f_d \in F} \alpha_{1bou}(\alpha_{\mathrm{CS}_1}(f_d)) \\ =& \bigvee_{f_d \in F} \alpha_{1bou}(\phi_d) \\ =& \alpha_{1bou}(\bigvee_{f_d \in F} \phi_d) \end{aligned}$$

from which the thesis. □

Proof (Theorem 33). Assume (A, α_A) is an abstract interpretation which has enough joins. We first show that there exists a Galois connection $\langle \alpha, \gamma \rangle$ from CS_2 to (A, α_A) and prove that $\alpha \circ \alpha_{\mathsf{CS}_2} = \alpha_A$. We define

$$\alpha(F) = \bigvee\nolimits_A \left\{ \alpha_A(f) \mid \alpha_{\mathsf{CS}_2}(f) \subseteq F \right\} \ .$$

First of all, note that $\alpha(F) = \bigvee_A \{\alpha_A(f) \mid f \in F\}$. We begin by showing that $\alpha \circ \alpha_{\mathsf{CS}_2} = \alpha_A$. We have that:

$$\alpha(\alpha_{\mathsf{CS}_2}(f)) \ = \ \bigvee\nolimits_A \left\{ \alpha_A(f') \mid f' \in \alpha_{\mathsf{CS}_2}(f) \right\} \ = \ \bigvee\nolimits_A \left\{ \alpha_A(f') \mid f' \in \{f\} \right\} \ = \ \bigvee\nolimits_A \{\alpha_A(f)\} \ = \ \alpha_A(f) \ .$$

We now prove that α has a right adjoint. It is enough to show that α is a complete join-morphism.

$$\alpha\left(\bigcup_i F_i\right) = \bigvee\nolimits_A \left\{ \alpha_A(f) \mid f \in \bigcup_i F_i \right\} = \bigvee\nolimits_A \bigcup_i \{\alpha_A(f) \mid f \in F_i\}$$
$$= \bigvee_i \bigvee\nolimits_A \{\alpha_A(f) \mid f \in F_i\} = \bigvee_i \alpha(F_i) \ .$$

We need to prove that given any $\langle \alpha', \gamma' \rangle : \mathsf{CS}_2 \leftrightarrows (A, \alpha_A)$ such that $\alpha' \circ \alpha_{\mathsf{CS}_2} = \alpha_A$, then we have that $\alpha = \alpha'$ and $\gamma = \gamma'$. Since γ' and γ are right adjoints to α and α' respectively, it is enough to prove that $\alpha = \alpha'$. Given $F \in \mathcal{P}(\mathcal{D}_\perp \to \mathcal{D}_\perp)$, since α' is a complete join-morphism, we have that

$$\alpha'(F) = \alpha'\left(\bigcup \{\{f\} \mid f \in F\}\right) = \bigvee\nolimits_A \{\alpha'(\{f\}) \mid f \in F\} =$$
$$\bigvee\nolimits_A \{\alpha'(\alpha_{\mathsf{CS}_2}(f)) \mid f \in F\} = \bigvee\nolimits_A \{\alpha_A(f) \mid f \in F\} = \alpha(F).$$

Finally, assume that there exists a Galois connection $\langle \alpha, \gamma \rangle$ from CS_2 to (A, α_A) with $\alpha \circ \alpha_{\mathsf{CS}_2} = \alpha_A$. We want to prove that (A, α_A) has enough joins. Let $F \subseteq \mathcal{D}_\perp \to \mathcal{D}_\perp$, since a left adjoint preserves all joins in its domain, we have:

$$\alpha(F) = \bigvee \{\alpha(\{f\}) \mid f \in F\} = \bigvee \{\alpha(\alpha_{\mathsf{CS}_2}(f)) \mid f \in F\} = \bigvee \{\alpha_A(f) \mid f \in F\} \ .$$

Then $\bigvee \{\alpha_A(f) \mid f \in F\}$ exists, i.e., (A, α_A) has enough joins, which concludes the proof. □

Proof (Lemma 36). The direction $\Longrightarrow$ is obvious, it is enough to take $f' = f$. For the other direction, for each $x \in \mathcal{D}_\perp$ let us call f_x a function such that $f_x(x) = f(x)$ and $\alpha_A(f_x) \leq a$. Then, $f(x) = f_x(x)$ is obtained by mixing the maps f_x's, and since (A, α_A) is mixable, then $\alpha_A(f) \leq a$. □

Proof (Theorem 38). Assume (A, α_A) is a mixable interpretation. We define a Galois connection $\langle \alpha, \gamma \rangle$ from CS_1 to (A, α_A) and prove that $\alpha \circ \alpha_{\mathsf{CS}_1} = \alpha_A$:

$$\alpha(\phi) = \bigvee \left\{ \alpha_A(f) \mid \alpha_{\mathsf{CS}_1}(f) \leq \phi \right\}$$
$$\gamma(a) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp). \bigcup \{f(X) \mid \alpha_A(f) \leq a\} \ .$$

First of all, we show that $\alpha \circ \alpha_{\mathsf{CS}_1} = \alpha_A$. We have

$$\alpha(\alpha_{\mathsf{CS}_1}(f)) = \bigvee \left\{ \alpha_A(f') \mid \alpha_{\mathsf{CS}_1}(f') \leq \alpha_{\mathsf{CS}_1}(f) \right\} \ .$$

Note that $\alpha_{\mathsf{CS}_1}(f') \leq \alpha_{\mathsf{CS}_1}(f)$ if and only if $f = f'$. Therefore $\alpha(\alpha_{\mathsf{CS}_1}(f)) = \alpha_A(f)$.

We now prove that $\langle \alpha, \gamma \rangle$ is a Galois connection. It is obvious that α and γ are monotone. Note that

$$\phi \leq \gamma(a) \iff \forall X \subseteq \mathcal{D}_\perp. \phi(X) \subseteq \bigcup \{f(X) \mid \alpha_A(f) \leq a\}$$
$$\iff \forall X \subseteq \mathcal{D}_\perp \forall y \in \phi(X) \exists f (\alpha_A(f) \leq a \wedge y \in f(X)) \ .$$

Since ϕ is a complete join-morphism, the last property holds for all $X \subseteq \mathcal{D}_\perp$ iff it holds for all singletons $\{x\}$ with $x \in \mathcal{D}_\perp$. Then $\phi \leq \gamma(a)$ is equivalent to

$$\forall x \in \mathcal{D}_\perp \forall y \in \phi(\{x\}) \exists f \ (\alpha_A(f) \leq a \wedge y = f(x)) \ .$$

Note that quantifying over all $x \in \mathcal{D}_\perp$ and $y \in \phi(\{x\})$ is the same as quantifying over all $x \in \mathcal{D}_\perp$ and $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$ correctly abstracted by ϕ. Hence we get:

$$
\begin{aligned}
&\forall x \in \mathcal{D}_\perp \forall y \in \phi(\{x\})\, \exists f (\alpha_A(f) \leq a \wedge y = f(x)) \\
\iff &\forall f' (\alpha_{\mathrm{CS}_1}(f') \leq \phi \to \forall x \in \mathcal{D}_\perp \exists f (\alpha_A(f) \leq a \wedge f'(x) = f(x)) \\
&\text{[since } (A, \alpha_A) \text{ is mixable]} \\
\iff &\forall f' (\alpha_{\mathrm{CS}_1}(f') \leq \phi \to \alpha_A(f') \leq a) \\
\iff &\bigvee \{\alpha_A(f) \mid \alpha_{\mathrm{CS}_1}(f) \leq \phi\} \leq a \\
\iff &\alpha(\phi) \leq a \ .
\end{aligned}
$$

We need to prove that if $\langle \alpha', \gamma' \rangle : \mathrm{CS}_1 \leftrightharpoons (A, \alpha_A)$ is such that $\alpha' \circ \alpha_{\mathrm{CS}_1} = \alpha_A$, then $\alpha = \alpha'$ and $\gamma = \gamma'$. Since γ' and γ are right adjoints to α and α' respectively, it is enough to prove $\alpha = \alpha'$. Given $\phi \in \mathcal{P}(\mathcal{D}_\perp) \xrightarrow{\cup} \mathcal{P}(\mathcal{D}_\perp)$, since ϕ is a complete join-morphisms, we have:

$$\phi = \bigvee \{\alpha_{\mathrm{CS}_1}(f) \mid \alpha_{\mathrm{CS}_1}(f) \leq \phi\} \ .$$

Since α' has a right adjoint, it is a complete join-morphism, hence

$$\alpha'(\phi) = \bigvee \{\alpha'(\alpha_{\mathrm{CS}_1}(f)) \mid \alpha_{\mathrm{CS}_1}(f) \leq \phi\} = \bigvee \{\alpha_A(f) \mid \alpha_{\mathrm{CS}_1}(f) \leq \phi\} = \alpha(\phi).$$

Finally, we show that, given any abstract interpretation (A, α_A), if there is a Galois connection $\langle \alpha, \gamma \rangle : \mathrm{CS}_1 \leftrightharpoons A$ such that $\alpha \circ \alpha_{\mathrm{CS}_1} = \alpha_A$, then (A, α_A) is mixable.

First of all, we prove that A has enough joins. Let $F \subseteq \mathcal{D}_\perp \to \mathcal{D}_\perp$, since a left adjoint preserves all joins in its domain, we have:

$$\alpha(\bigvee \{\alpha_{\mathrm{CS}_1}(f) \mid f \in F\}) = \bigvee \{\alpha(\alpha_{\mathrm{CS}_1}(f)) \mid f \in F\} = \bigvee \{\alpha_A(f) \mid f \in F\} \ .$$

Then $\bigvee \{\alpha_A(f) \mid f \in F\}$ exists, i.e., (A, α_A) has enough joins.

Now, given a function g, assume g is a mix of the functions in F. We need to prove that $\alpha_A(g) \leq \bigvee_{f \in F} \alpha_A(f)$. Since $\forall x \in \mathcal{D}_\perp \exists f \in F$ s.t. $g(x) = f(x)$, it follows that for any $x \in \mathcal{D}_\perp$ there exists $f \in F$ such that $\alpha_{\mathrm{CS}_1}(g)(\{x\}) = \alpha_{\mathrm{CS}_1}(f)(\{x\})$, and thus $\alpha_{\mathrm{CS}_1}(g)(\{x\}) \subseteq \bigcup_{f \in F} \alpha_{\mathrm{CS}_1}(f)(\{x\})$, from which $\alpha_{\mathrm{CS}_1}(g) \leq \bigvee_{f \in F} \alpha_{\mathrm{CS}_1}(f)$. Since α is a join-morphism, we have that $\alpha(\alpha_{\mathrm{CS}_1}(g)) \leq \alpha(\bigvee_{f \in F} \alpha_{\mathrm{CS}_1}(f)) = \bigvee_{f \in F} \alpha(\alpha_{\mathrm{CS}_1}(f))$, i.e. $\alpha_A(g) \leq \bigvee_{f \in F} \alpha_A(f)$. □

Proof (Theorem 45). First of all, it is immediate to see that $\alpha_{23}(\alpha_{\mathrm{CS}_2}(f)) = \alpha_{\mathrm{CS}_3}(f)$. We now prove that α_{23} is a complete join morphism. Given a family $\{F_i\} \subseteq \mathcal{P}(\mathcal{D}_\perp \to \mathcal{D}_\perp)$, we have

$$
\begin{aligned}
\alpha_{23}(\cup_i F_i)(\Theta) &= \downarrow \{f(X) \mid f \in \cup_i F_i \wedge X \in \Theta\} \\
&= \downarrow \cup_i \{f(X) \mid f \in F_i \wedge X \in \Theta\} \\
&= \cup_i \downarrow \{f(X) \mid f \in F_i \wedge X \in \Theta\} \\
&= \cup_i (\alpha_{23}(F_i)(\Theta)) \\
&= (\dot{\cup}_i \alpha_{23}(F_i))(\Theta) \ . \quad \square
\end{aligned}
$$

Proof (Theorem 46). First of all, it is immediate to see that $\alpha_{31}(\alpha_{\mathrm{CS}_3}(f)) = \alpha_{\mathrm{CS}_1}(f)$. Actually

$$
\begin{aligned}
\alpha_{31}(\alpha_{\mathrm{CS}_3}(f))(X) &= \cup \alpha_{\mathrm{CS}_3}(f)(\{X\}) = \cup \downarrow \{f(Y) \mid Y \in \{X\}\} \\
&= \cup \downarrow \{f(X)\} = f(X) = \alpha_{\mathrm{CS}_1}(f)(X) \ .
\end{aligned}
$$

We now prove that α_{31} is a complete join-morphism. Given a family $\{\Phi_i\} \subseteq \mathcal{P}(\mathcal{P}(\mathcal{D}_\perp)) \xrightarrow{\cup} \mathcal{P}^{\downarrow}(\mathcal{P}(\mathcal{D}_\perp))$, we have

$$
\begin{aligned}
\alpha_{31}(\dot{\cup}_i \Phi_i)(X) &= \cup \left((\dot{\cup}_i \Phi_i)(\{X\})\right) = \cup (\cup_i (\Phi_i(\{X\}))) = \\
&\cup_i (\cup (\Phi_i(\{X\}))) = \cup_i (\alpha_{31}(\Phi_i)(X)) = (\bigvee_i \alpha_{31}(\Phi_i))(X) \ . \quad \square
\end{aligned}
$$

Proof (Proposition 47). • The proof of the first point is by contradiction. In the following, for $i \in \{0, 1\}$, let $f_i, f \in \mathcal{D}_\perp \to \mathcal{D}_\perp$ and let $\phi_i = \alpha_{\mathrm{CS}_1}(f_i), \phi = \alpha_{\mathrm{CS}_1}(f) \in \mathcal{P}(\mathcal{D}_\perp) \xrightarrow{\cup} \mathcal{P}(\mathcal{D}_\perp)$ as defined in the first point of the proof of Proposition 29. We have that $\phi \leq \phi_0 \vee \phi_1$. Assume that there exists $\langle \alpha_{13}, \gamma_{13} \rangle : \mathrm{CS}_1 \leftrightharpoons \mathrm{CS}_3$ such that $\alpha_{\mathrm{CS}_3} = \alpha_{13} \circ \alpha_{\mathrm{CS}_1}$.

Since α_{13} is a join-morphism,

$$\begin{aligned}\alpha_{13}(\phi_0 \vee \phi_1)(\{\{d_0, d_1\}\}) &= (\alpha_{13}(\phi_0) \vee \alpha_{13}(\phi_1))(\{\{d_0, d_1\}\}) \\ &= (\alpha_{13}(\alpha_{\mathrm{CS}_1}(f_0)) \dot{\cup} \alpha_{13}(\alpha_{\mathrm{CS}_1}(f_1)))(\{\{d_0, d_1\}\}) \\ &= \alpha_{\mathrm{CS}_3}(f_0)(\{\{d_0, d_1\}\}) \cup \alpha_{\mathrm{CS}_3}(f_1)(\{\{d_0, d_1\}\}) \\ &= \downarrow\{\{d_0\}\} \cup \downarrow\{\{d_1\}\} \\ &= \{\emptyset, \{d_0\}, \{d_1\}\} \ .\end{aligned}$$

On the other hand,

$$\begin{aligned}\alpha_{13}(\phi)(\{\{d_0, d_1\}\}) =& \alpha_{13}(\alpha_{\mathrm{CS}_1}(f))(\{\{d_0, d_1\}\}) \\ =& \alpha_{\mathrm{CS}_3}(f)(\{\{d_0, d_1\}\}) \\ =& \downarrow\{\{d_0, d_1\}\} = \{\emptyset, \{d_0\}, \{d_1\}, \{d_0, d_1\}\} \\ \not\subseteq& \{\emptyset, \{d_0\}, \{d_1\}\} \\ =& \alpha_{13}(\phi_0 \vee \phi_1)(\{\{d_0, d_1\}\}) \ .\end{aligned}$$

Therefore α_{13} is not monotone and then the thesis.

- The proof of the second point is again by contradiction. Let d_0 and d_1 be two elements in $\mathcal{D}_\perp$ and assume that there exists $\langle \alpha_{32}, \gamma_{32} \rangle : \mathrm{CS}_3 \leftrightharpoons \mathrm{CS}_2$ such that $\alpha_{\mathrm{CS}_2} = \alpha_{32} \circ \alpha_{\mathrm{CS}_3}$. Let f be the identity on $\mathcal{D}_\perp$, $f' = f[d_0 \to d_1, d_1 \to d_0]$ and $g = f[d_0 \mapsto d_1, d_1 \mapsto d_1]$. Let $X \subseteq \mathcal{D}_\perp$ and we have several cases:
  - if $d_0 \notin X$ and $d_1 \notin X$, then $g(X) = f(X) = f'(X)$;
  - if $d_0 \in X$ but $d_1 \notin X$, then $g(X) = f'(X)$;
  - if $d_1 \in X$ but $d_0 \notin X$, then $g(X) = f(X)$;
  - if $d_0, d_1 \in X$, then $g(X) \subset f(X) = f'(X)$.

  Therefore, for each $X \subseteq \mathcal{D}_\perp$ we have either $g(X) \subseteq f(X)$ or $g(X) \subseteq f'(X)$, hence $\{g(X)\} \subseteq \{f(X), f'(X)\}$, i.e. $\downarrow\{g(X)\} \subseteq \downarrow\{f(X), f'(X)\} = \downarrow\{f(X)\} \cup \downarrow\{f'(X)\}$. This means $\alpha_{\mathrm{CS}_3}(g) \dot{\subseteq} \alpha_{\mathrm{CS}_3}(f) \dot{\cup} \alpha_{\mathrm{CS}_3}(f')$.
  Since α_{32} is additive, we get

  $$\alpha_{32}(\alpha_{\mathrm{CS}_3}(g)) = \alpha_{\mathrm{CS}_2}(g) = \{g\} \text{ and } \alpha_{32}(\alpha_{\mathrm{CS}_3}(f) \dot{\cup} \alpha_{\mathrm{CS}_3}(f')) = \{f, f'\}.$$

  By monotonicity it should be $\{g\} \subseteq \{f, f'\}$, which is a contradiction. □

Proof (Lemma 50). The direction $\Longrightarrow$ is obvious, it is enough to take $F' = \{f\}$. For the other direction, for each $X \subseteq \mathcal{D}_\perp$ let us call f_X a function such that $f(X) \subseteq f_X(X)$ and $\alpha_A(f_X) \leq a$. Then, $f(X)$ is obtained by mixing the maps f_X's, and since (A, α_A) is set-mixable, then $\alpha_A(f) \leq a$. □

Proof (Theorem 52). Assume (A, α_A) is a set-mixable interpretation. We define a Galois connection $\langle \alpha, \gamma \rangle$ from CS_3 to (A, α_A) and prove that $\alpha \circ \alpha_{\mathrm{CS}_3} = \alpha_A$:

$$\alpha(\Phi) = \bigvee_A \left\{ \alpha_A(f) \mid \alpha_{\mathrm{CS}_3}(f) \dot{\subseteq} \Phi \right\}$$

$$\gamma(a) = \lambda\Theta \in \mathcal{P}(\mathcal{P}(\mathcal{D}_\perp)). \bigcup_{\alpha_A(f) \leq_A a} \downarrow\{f(X) \mid X \in \Theta\} \ .$$

First of all, we show that $\alpha \circ \alpha_{\mathrm{CS}_3} = \alpha_A$. We have

$$\alpha(\alpha_{\mathrm{CS}_3}(f)) = \bigvee_A \left\{ \alpha_A(f') \mid \alpha_{\mathrm{CS}_3}(f') \dot{\subseteq} \alpha_{\mathrm{CS}_3}(f) \right\} \ .$$

Note that $\alpha_{\mathrm{CS}_3}(f') \dot{\subseteq} \alpha_{\mathrm{CS}_3}(f)$ if and only if $f = f'$. Therefore $\alpha(\alpha_{\mathrm{CS}_3}(f)) = \alpha_A(f)$.

We now prove that $\langle \alpha, \gamma \rangle$ is a Galois connection. It is obvious that α and γ are monotone. Note that

$$\begin{aligned}\Phi \dot{\subseteq} \gamma(a) &\iff \forall\Theta \subseteq \mathcal{P}(\mathcal{D}_\perp). \Phi(\Theta) \subseteq \bigcup_{\alpha_A(f) \leq_A a} \downarrow\{f(X) \mid X \in \Theta\} \\ &\iff \forall\Theta \subseteq \mathcal{P}(\mathcal{D}_\perp). \forall Y \in \Phi(\Theta) \\ &\qquad \exists f \ (\alpha_A(f) \leq_A a \wedge \exists Z \in \{f(X) \mid X \in \Theta\} \wedge Y \subseteq Z) \ .\end{aligned}$$

Since Φ is a complete join-morphism, the last property holds for all $\Theta \subseteq \mathcal{P}(\mathcal{D}_\perp)$ iff it holds for all the singletons $\{X\}$ with $X \in \mathcal{P}(\mathcal{D}_\perp)$. Then $\Phi \dot{\subseteq} \gamma(a)$ is equivalent to

$$\forall X \in \mathcal{P}(\mathcal{D}_\perp)\, \forall Y \in \Phi(\{X\})\, \exists f\, (\alpha_A(f) \preceq_A a \wedge Y \subseteq f(X)) \ .$$

Note that quantifying over all $X \in \mathcal{P}(\mathcal{D}_\perp)$ and $Y \in \Phi(\{X\})$ is the same as quantifying over all $X \in \mathcal{P}(\mathcal{D}_\perp)$ and $f' : \mathcal{D}_\perp \to \mathcal{D}_\perp$ correctly abstracted by Φ. Hence we get:

$$\begin{aligned}
&\forall X \in \mathcal{P}(\mathcal{D}_\perp) \forall Y \in \Phi(\{X\})\, \exists f (\alpha_A(f) \preceq a \wedge Y \subseteq f(X)) \\
\iff &\forall f' (\alpha_{\mathrm{CS}_3}(f') \mathrel{\dot\subseteq} \Phi \to \forall X \in \mathcal{P}(\mathcal{D}_\perp) \exists f (\alpha_A(f) \preceq_A a \wedge f'(X) \subseteq f(X)) \\
&\text{[since } (A, \alpha_A) \text{ is set-mixable]} \\
\iff &\forall f' (\alpha_{\mathrm{CS}_3}(f') \preceq \Phi \to \alpha_A(f') \preceq a) \\
\iff &\bigvee_A \{\alpha_A(f) \mid \alpha_{\mathrm{CS}_3}(f) \mathrel{\dot\subseteq} \Phi\} \preceq a \\
\iff &\alpha(\Phi) \preceq_A a \ .
\end{aligned}$$

We need to prove that if $\langle \alpha', \gamma' \rangle : \mathrm{CS}_3 \leftrightharpoons (A, \alpha_A)$ is such that $\alpha' \circ \alpha_{\mathrm{CS}_1} = \alpha_A$, then $\alpha = \alpha'$ and $\gamma = \gamma'$. Since γ' and γ are right adjoints to α and α' respectively, it is enough to prove $\alpha = \alpha'$. Given $\Phi \in \mathcal{P}(\mathcal{P}(\mathcal{D}_\perp)) \xrightarrow{\cup} \mathcal{P}^{\downarrow}(\mathcal{P}(\mathcal{D}_\perp))$, by definition of CS_3 we have that

$$\Phi = \dot\cup \{\alpha_{\mathrm{CS}_3}(f) \mid \alpha_{\mathrm{CS}_3}(f) \mathrel{\dot\subseteq} \Phi\} \ .$$

Since α' has a right adjoint, it is a complete join-morphism, hence

$$\alpha'(\Phi) = \bigvee_A \{\alpha'(\alpha_{\mathrm{CS}_3}(f)) \mid \alpha_{\mathrm{CS}_3}(f) \mathrel{\dot\subseteq} \Phi\} = \bigvee_A \{\alpha_A(f) \mid \alpha_{\mathrm{CS}_3}(f) \mathrel{\dot\subseteq} \Phi\} = \alpha(\Phi).$$

The proof of maximality is analogous to that one of Theorem 38 and hence it is omitted. □

Proof (Proposition 53). The proof is by contradiction. Assume that the boundness abstraction is not set-mixable. Then, by definition of Bou, there exist $g : \mathcal{D}_\perp \to \mathcal{D}_\perp$ and a set of functions $F \subseteq \mathcal{D}_\perp \to \mathcal{D}_\perp$, such that for each $X \subseteq \mathcal{D}_\perp$ there exists $f \in F$ such that $g(X) \subseteq f(X)$ and $\top = \alpha_{bou}(g) \not\preceq \bigvee_{f \in F} \alpha_{bou}(f) = bou$. Since $\alpha_{bou}(g) = \top$, we have that g is not bounded and therefore for each $d \in \mathcal{D}_\perp$ there exists $v_d \in \mathcal{D}_\perp$, $g(v_d) \not\preceq d$. Let $S = \{v_d \mid d \in \mathcal{D}_\perp\}$. By hypothesis there exists $f \in F$ such that $g(S) \subseteq f(S)$. Therefore for each $v_d \in S$ there exists $w_d \in S$ such that $g(v_d) = f(w_d)$. By construction for each $d \in \mathcal{D}_\perp$ we have that $f(w_d) \not\preceq d$. Therefore f is not bounded, $\alpha_{bou}(f) = \top$ and this contradicts the assumption $\bigvee_{f \in F} \alpha_{bou}(f) = bou$. □

References

[1] S. Abramsky, Abstract interpretation, logical relations, and Kan extensions, J. Log. Comput. 1 (1) (1990) 5–40.
[2] G. Amato, S. Di Nardo Di Maio, F. Scozzari, Numerical static analysis with Soot, in: Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program Analysis, SOAP '13, ACM, New York, NY, USA, 2013.
[3] G. Amato, M.C. Meo, F. Scozzari, A taxonomy of program analyses, in: A. Aldini, M. Bernardo (Eds.), Proceedings of the 19th Italian Conference on Theoretical Computer Science, Urbino, Italy, September 18–20, 2018, in: CEUR Workshop Proceeding, vol. 2243, 2018, pp. 213–217, http://ceur-ws.org/Vol-2243/paper21.pdf.
[4] G. Amato, M. Parton, F. Scozzari, A tool which mines partial execution traces to improve static analysis, in: H. Barringer, et al. (Eds.), First International Conference, RV 2010, St. Julians, Malta, November 1–4, 2010, Proceedings, in: Lecture Notes in Computer Science, vol. 6418, Springer, Berlin, Heidelberg, 2010, pp. 475–479.
[5] G. Amato, M. Rubino, F. Scozzari, Inferring linear invariants with parallelotopes, Sci. Comput. Program. 148 (2017) 161–188.
[6] G. Amato, F. Scozzari, Observational completeness on abstract interpretation, Fundam. Inform. 106 (2–4) (2011) 149–173.
[7] G. Amato, F. Scozzari, Random: R-based analyzer for numerical domains, in: N. Bjørner, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning, 18th International Conference, LPAR-18, Mérida, Venezuela, March 11–15, 2012, in: Lecture Notes in Computer Science, vol. 7180, Springer, Berlin, Heidelberg, 2012, pp. 375–382.
[8] A. Asperti, G. Longo, Categories, Types and Structures: An Introduction to Category Theory for the Working Computer Scientist, Foundations of Computing Series, The MIT Press, 1991.
[9] K. Backhouse, R. Backhouse, Safety of abstract interpretations for free, via logical relations and Galois connections, Sci. Comput. Program. 51 (1) (2004) 153–196.
[10] A. Bossi, M. Gabbrielli, G. Levi, M. Martelli, The s-semantics approach: theory and applications, J. Log. Program. 19-20 (1994) 149–197.
[11] M. Comini, G. Levi, M.C. Meo, A theory of observables for logic programs, Inf. Comput. 169 (1) (2001) 23–80.
[12] P. Cousot, Constructive design of a hierarchy of semantics of a transition system by abstract interpretation (extended abstract), Electron. Notes Theor. Comput. Sci. 6 (1997) 77–102.
[13] P. Cousot, R. Cousot, Static determination of dynamic properties of programs, in: Proceedings of the Second International Symposium on Programming, Dunod, Paris, France, 1976, pp. 106–130.
[14] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, ACM Press, New York, NY, USA, Jan. 1977, pp. 238–252.
[15] P. Cousot, R. Cousot, Systematic design of program analysis frameworks, in: POPL '79: Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, ACM Press, New York, NY, USA, Jan. 1979, pp. 269–282.
[16] P. Cousot, R. Cousot, Abstract interpretation frameworks, J. Log. Comput. 2 (4) (Aug. 1992) 511–549.

[17] P. Cousot, R. Cousot, Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages), in: Proceedings of the 1994 International Conference on Computer Languages, IEEE Computer Society Press, Los Alamitos, CA, USA, May 1994, pp. 95–112, invited paper.
[18] P. Cousot, R. Cousot, Temporal abstract interpretation, in: POPL '00: Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM Press, New York, NY, USA, Jan. 2000, pp. 12–25.
[19] P. Cousot, R. Cousot, Abstract interpretation: past, present and future, in: Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, ACM, New York, NY, USA, 2014, pp. 2:1–2:10, http://doi.acm.org/10.1145/2603088.2603165.
[20] P. Cousot, N. Halbwachs, Automatic discovery of linear restraints among variables of a program, in: POPL '78: Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, ACM Press, New York, NY, USA, Jan. 1978, pp. 84–97.
[21] M. de Kruijf, K. Sankaralingam, Idempotent code generation: implementation, analysis, and evaluation, in: Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), CGO '13, IEEE Computer Society, Washington, DC, USA, 2013, pp. 1–12.
[22] R. Giacobazzi, "Optimal" collecting semantics for analysis in a hierarchy of logic program semantics, in: C. Puech, R. Reischuk (Eds.), Proc. 13th International Symposium on Theoretical Aspects of Computer Science, STACS'96, in: Lecture Notes in Computer Science, vol. 1046, Springer, 1996, pp. 503–514.
[23] R. Giacobazzi, F. Ranzato, F. Scozzari, Making abstract interpretations complete, J. ACM 47 (2) (2000) 361–416.
[24] A. Mycroft, The theory and practice of transforming call-by-need into call-by-value, in: B. Robinet (Ed.), International Symposium on Programming: Proceedings of the Fourth 'Colloque International sur la Programmation', Paris, 22–24 April 1980, Springer-Verlag, Berlin, Heidelberg, 1980, pp. 269–281.
[25] P.M.P. Panangaden, A category theoretic formalism for abstract interpretation, Tech. Rep. UUCS-84-005, University of Utah, May 1984.
[26] A. Venet, Abstract cofibered domains: application to the alias analysis of untyped programs, in: Proceedings of the Third International Symposium on Static Analysis, SAS '96, Springer-Verlag, London, UK, 1996, pp. 366–382.
[27] P. Wadler, R.J.M. Hughes, Projections for strictness analysis, in: G. Kahn (Ed.), Functional Programming Languages and Computer Architecture: Portland, Oregon, USA, September 14–16, 1987, Proceedings, Springer-Verlag, Berlin, Heidelberg, 1987, pp. 385–407.