A taxonomy of program analyses

Gianluca Amato, Maria Chiara Meo, and Francesca Scozzari

Dipartimento di Economia, Università di Chieti-Pescara, Italy
{gianluca.amato, mariachiara.meo, francesca.scozzari}@unich.it

Abstract. The design of static analyses of programs in the abstract interpretation theory starts with the choice of a collecting semantics, which is the strongest property we can derive for a program. Starting from well-known collecting semantics for functional programs in the literature, we propose a taxonomy of program properties by considering the sets of abstract interpretations for which the collecting semantics is initial and show that they can be constructively characterized in terms of the abstraction functions.

1 Introduction

Abstract interpretation [7, 8] is a framework for approximating the behavior of discrete systems. The basic idea 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 discrete system e, we assume that its semantics $[\![e]\!]$ is an element of a set C called the *concrete domain*. For instance, consider a simple setting for functional programs[1] where the concrete domain C is the poset of functions $\mathcal{D}_\bot \to \mathcal{D}_\bot$ ordered pointwise and $\mathcal{D}_\bot$ has a distinguished value $\bot$ which represents non-terminating computations. Given the following program over natural numbers:

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

its semantics is

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

where $\mathcal{D}_\bot = \mathbb{N} \cup \{\bot\}$.

In most cases we are only interested in determining some properties of the semantics of a system specified by a set A of abstract objects. Formally, an *abstract interpretation*[2] for a concrete domain C is a pair (A, α) where A is partially ordered by $\leq_A$ (the *approximation ordering*) and $\alpha : C \to A$ maps every semantic property to the strongest (smallest) abstract property it enjoys. An

[1] For easiness of presentation, we do not consider here higher-order functions.

[2] The abstract interpretation framework here used is the one presented in [9] under the "existence of a best abstract approximation assumption". Not all the abstract interpretations may be formalized in this way, such as polyhedral analysis [11, 3, 2, 4, 5].

example of a simple and useful abstraction is *strictness* [13]. We say that a function $f : \mathcal{D}_\perp \to \mathcal{D}_\perp$ is strict if $f(\perp) = \perp$. The *strictness* abstract interpretation is $\textsc{Str} = (\{str, \top\}, \alpha_{str})$ where $str \leq \top$ and

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

Clearly, $\alpha_{str}([\![prog]\!]) = str$ since *prog* describes a strict function.

1.1 Collecting Semantics

In most cases abstract interpretations are defined starting from a so-called *collecting semantics*. According to [9], a collecting semantics is "a version of the standard semantics reduced to essentials in order to ignore irrelevant details about program execution". Thus, a collecting semantics should be an abstraction precise enough that many other abstractions may be derived from it. We say that the abstraction (B, α_B) *may be derived from* (A, α_A), or that (A, α_A) *is more precise than* (B, α_B), when for each $b \in B$ there exists $a \in A$ such that, for any $c \in C$, $\alpha_B(c) \leq b \iff \alpha_A(c) \leq a$.

For example, consider the collecting semantics CS_1 for the concrete domain $C = \mathcal{D}_\perp \to \mathcal{D}_\perp$ defined in [10]: the abstract domain is $\mathcal{P}(\mathcal{D}_\perp) \xrightarrow{\cup} \mathcal{P}(\mathcal{D}_\perp)$, the set of complete join-morphisms from $\mathcal{P}(\mathcal{D}_\perp)$ to itself ordered pointwise, and $\alpha_{\mathrm{CS}_1}(f) = \lambda X \in \mathcal{P}(\mathcal{D}_\perp).f(X)$, where $f(X)$ is the image of f through X. It turns out that strictness may be derived from CS_1. Actually, we have that:

- $\alpha_{str}(f) \leq \top$ is always true, hence it is equivalent to $\alpha_{\mathrm{CS}_1}(f) \leq \lambda X.\mathcal{D}_\perp$;
- $\alpha_{str}(f) \leq str$ means that f is a strict function, which is equivalent to $\alpha_{\mathrm{CS}_1}(\{\perp\}) \subseteq \{\perp\}$, i.e., $\alpha_{\mathrm{CS}_1}(f) \leq \phi_{str}$ by defining

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

However, not all the abstractions may be recovered from CS_1. Consider the property of absence. We say that a function is absent if it ignores its arguments. This gives origin to the *absence* abstract interpretation [16] defined as $\textsc{Abs} = (\{abs, \top\}, \alpha_{abs})$ where $abs \leq \top$ and

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

It turns out that absence cannot be recovered from CS_1, since there is no element in the abstract domain of CS_1 which exactly corresponds to the set of all the absent functions. However, more precise collecting semantics may be used to derive *abs*, such as CS_2 [10] which has $\mathcal{P}(\mathcal{D}_\perp \to \mathcal{D}_\perp)$ as the abstract domain and $\alpha_{\mathrm{CS}_2}(f) = \{f\}$ as the abstraction function. Then, $\alpha_{abs}(c) \leq abs$ is equivalent to $\alpha_{\mathrm{CS}_2}(f) \subseteq \{f \mid \alpha_{abs}(f) \leq abs\}$. We can also compare the relative precision of two collecting semantics. It is easily shown that CS_1 may be derived from CS_2, since $\alpha_{\mathrm{CS}_1}(f) \leq \phi$ iff $\alpha_{\mathrm{CS}_2}(f) \subseteq \{f' \mid \alpha_{\mathrm{CS}_1}(f') \leq \phi\}$.

2 A category of abstract interpretations

We now formalize the notion of precision which arises from the previous considerations and characterize the collecting semantics using the category theory[3]. We recall that a Galois connection (Gc) is a pair of maps $\langle \alpha, \gamma \rangle : A \leftrightarrows B$ such that $\alpha(a) \leq b \Leftrightarrow a \leq \gamma(b)$. Given a set C, we define the category $\mathbb{AI}(C)$ whose objects are abstract interpretations and whose morphisms from (A, α_A) to (B, α_B) are Galois connections $\langle \alpha, \gamma \rangle : A \leftrightarrows B$ such that $\alpha_B = \alpha \circ \alpha_A$. Identity and composition of arrows work as in the category of Galois connections. This definition of the category of abstract interpretations naturally arises from the practical uses of abstract interpretation and represents a strengthening of the notion of derivability between abstractions, since the existence of a morphism $\langle \alpha, \gamma \rangle : A \leftrightarrows B$ implies that B is derivable from A. The following proposition shows that there is a Gc from CS_2 to CS_1 but not from CS_1 to CS_2, which reflects our intuition that CS_2 is strictly more precise than CS_1, and that CS_1 is a suitable collecting semantics for strictness but not for the absence property.

Proposition 1. *The following properties hold:*

- *there is no Gc* $\langle \alpha_{12}, \gamma_{12} \rangle : \mathrm{CS}_1 \leftrightarrows \mathrm{CS}_2$ *such that* $\alpha_{\mathrm{CS}_2} = \alpha_{12} \circ \alpha_{\mathrm{CS}_1}$*;*
- *there is a Gc* $\langle \alpha_{21}, \gamma_{21} \rangle : \mathrm{CS}_2 \leftrightarrows \mathrm{CS}_1$ *such that* $\alpha_{\mathrm{CS}_1} = \alpha_{21} \circ \alpha_{\mathrm{CS}_2}$*;*
- *there is a Gc* $\langle \alpha_{1str}, \gamma_{1str} \rangle : \mathrm{CS}_1 \leftrightarrows \textsc{Str}$ *such that* $\alpha_{str} = \alpha_{1str} \circ \alpha_{\mathrm{CS}_1}$ *and*

$$\alpha_{1str}(\phi) = \begin{cases} str & \text{if } \phi(\{\perp\}) \subseteq \{\perp\} \\ \top & \text{otherwise.} \end{cases} \qquad \gamma_{1str}(str) = \lambda X. \begin{cases} X & \text{if } X \subseteq \{\perp\} \\ \mathcal{D}_\perp & \text{otherwise} \end{cases}$$
$$\gamma_{1str}(\top) = \lambda X. \mathcal{D}_\perp$$

- *there is no Gc* $\langle \alpha_{1abs}, \gamma_{1abs} \rangle : \mathrm{CS}_1 \leftrightarrows \mathrm{CS}_{abs}$ *such that* $\alpha_{abs} = \alpha_{1abs} \circ \alpha_{\mathrm{CS}_1}$*.*

3 Subcategories of program analyses

When an abstract interpretation (A, α_A) is designed starting from the collecting semantics (S, α_S), it means that (A, α_A) is derivable from (S, α_S), hence there is a map from (S, α_S) to (A, α_A) in our category $\mathbb{AI}(C)$. In addition, it would be preferable to have a canonical way of deriving one from the other. This leads to the notion of initial object in a category: if (S, α_S) is initial for a given full subcategory $\mathbb{D}$ of $\mathbb{AI}(C)$, it means that all the abstract interpretations in $\mathbb{D}$ may be reduced to (S, α_S) in a unique canonical way.

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.

[3] Since the very beginning of the abstract interpretation theory, there have been work on categorical approaches to abstract interpretation (see, for instance, [1, 6, 14, 15]).

In the rest of the section, we show that for the two collecting semantics CS_1 e CS_2, such a full subcategory can be constructively described.

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 2 (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.

Theorem 3. *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.*

The maximality of the class of abstract interpretations which have enough joins allows us to completely characterize the analyses which reduce to the collecting semantics CS_2. We now show that the collecting semantics CS_1 is initial for a large class of abstract interpretations, which can be constructively characterized.

Definition 4 (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 5 (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 6. *Let (A, α_A) be a mixable interpretation. Then $\alpha_A(f) \leq a \iff \forall x \in \mathcal{D}_\perp\ \exists f'$ s.t. $f'(x) = f(x) \land \alpha_A(f') \leq a$.*

This lemma can be exploited to characterize mixable abstract interpretations:

- the strictness abstract interpretation is mixable, since we only need to check the single value of $f(\perp)$ in order to decide whether a function is strict;
- the absence abstract interpretation is not mixable, since we need to compare the values computed by f for different arguments;
- the totality abstract interpretation, which decides whether a function is total, i.e., for all $x \in \mathcal{D}_\perp \setminus \{\perp\}$ we have that $f(x) \neq \perp$ is mixable, since we just need to see 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 7. *The full subcategory of all the mixable abstract interpretations is the largest class of abstractions for which the collecting semantics CS_1 is initial.*

4 Conclusion

Static analyses formalized in the abstract interpretation framework are defined starting from a collecting semantics, which is a fundamental choice in the design of any abstract interpretation, but few works in the literature systematically study collecting semantics. [9, 10] present many collecting semantics, without giving a general method for choosing the right one. Mostly authors simply use one of the already defined collecting semantics or invent a new one for a specific abstraction (e.g., [12] for logic programs). On the contrary, we start from the definition of two most commonly used collecting semantics and derive a taxonomy on program analysis. Most importantly we show that the sets of abstract interpretations that can be derived from them can be constructively characterized.

References

1. S. Abramsky. Abstract interpretation, logical relations, and Kan extensions. *Journal of Logic and Computation*, 1(1):5–40, 1990.
2. G. Amato, S. Di Nardo Di Maio, M. C. Meo, F. Scozzari. Descending chains and narrowing on template abstract domains. *Acta Informatica*, 55(6):521–545, 2018.
3. G. Amato, S. Di Nardo Di Maio and F. Scozzari Numerical static analysis with Soot. In *Proc. ACM SIGPLAN SOAP*. ACM Press, 2013.
4. Gianluca Amato and Francesca Scozzari. Observational completeness on abstract interpretation. *Fundamenta Informaticae*, 106(2–4):149–173, 2011.
5. Gianluca Amato and Francesca Scozzari. Random: R-based Analyzer for Numerical Domains. In *Proc. LPAR-18, LNCS*, vol. 7180:375–382. Springer, 2012.
6. K. Backhouse and R. Backhouse. Safety of abstract interpretations for free, via logical relations and galois connections. *Science of Computer Programming*, 51(1):153–196, 2004.
7. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. POPL '77*, pages 238–252. ACM Press, 1977.
8. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Proc. POPL '79*, pages 269–282. ACM Press, 1979.
9. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–549, 1992.
10. P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages). In *Proc. ICCL*, pages 95–112. IEEE Press, 1994.
11. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proc. POPL '78*, pages 84–97, 1978. ACM Press.
12. R. Giacobazzi. "Optimal" collecting semantics for analysis in a hierarchy of logic program semantics. In *Proc. STACS, LNCS*, vol. 1046:503–514. Springer, 1996.
13. A. Mycroft. *The theory and practice of transforming call-by-need into call-by-value.* In *Proc. Int. Symposium on Programming, LNCS*, vol. 83:269–281. Springer, 1980.
14. P. Panangaden and P. Mishra. A category theoretic formalism for abstract interpretation. Technical Report UUCS-84-005, University of Utah, 1984.
15. A. Venet. Abstract cofibered domains: Application to the alias analysis of untyped programs. In *Proc. SAS '96, LNCS*, vol. 1145:366–382. Springer, 1996.
16. P. Wadler and R. J. M. Hughes. *Projections for strictness analysis.* In *Proc. FPCA, LNCS*, vol. 274:385–407. Springer, 1987.