On the Properties of Partial Completeness in Abstract Interpretation (Short Paper)*

Marco Campion, Mila Dalla Preda and Roberto Giacobazzi

University of Verona, Italy

Abstract

We present a weakening of the completeness property in abstract interpretation. Completeness of a static analyzer represents the ideal situation where no false alarms are produced when answering queries on program behavior. Completeness, however, is a very rare condition to be satisfied in practice. We introduce the notion of partial completeness as a weakening of precision, namely, the abstract interpreter may produce a bounded number of false alarms, and then we show the key recursive properties of the class of programs for which an abstract interpreter is partially complete with a given bound of imprecision.

Keywords

Abstract Interpretation, Program Analysis, Computability, Partial Completeness

1. Introduction

Abstract interpretation [2, 3, 4] is a general theory for the design of sound-by-construction program analysis tools. The abstract interpretation of a program $P \in \mathsf{Programs}$[1], denoted by $[\![P]\!]^A$, consists of an abstract domain A, often specified by a pair of abstraction $\alpha_A : C \to A$ and concretization $\gamma_A : A \to C$ monotone maps, abstracting some concrete properties of interest C and an interpreter $[\![P]\!]^A : A \to A$, designed for the language used to specify P and on the abstract domain A. The structure of the abstract domain is given by a partial order $\leq_A$ that expresses the relative precision of its objects: If $a, b \in A$ and $a \leq_A b$ then b over approximates a. We are interested in inferring program invariants, hence our concrete semantics $[\![P]\!] : \wp(\mathbb{S}) \to \wp(\mathbb{S})$ will be the standard collecting denotational semantics (e.g., see [5, Chapter 5] or [6, Chapter 3.5]) operating on a domain of sets of memory of states $\wp(\mathbb{S})$. *Soundness* means that if program P satisfies the condition $[\![P]\!]^A \alpha_A(S) \leq_A Q$ for the input $S \in \wp(\mathbb{S})$ and output specification $Q \in A$, then $[\![P]\!]S \subseteq \gamma_A(Q)$. When the converse holds we have precision, or *completeness* of the abstract interpreter, and therefore of the analysis. This

Proceedings of the 23rd Italian Conference on Theoretical Computer Science, Rome, Italy, September 7-9, 2022

* This work is an extended abstract of our recent results published in [1].

marco.campion@univr.it (M. Campion); mila.dallapreda@univr.it (M. Dalla Preda); roberto.giacobazzi@univr.it (R. Giacobazzi)

0000-0002-1099-3494 (M. Campion); 0000-0003-2761-4347 (M. Dalla Preda); 0000-0002-9582-3960 (R. Giacobazzi)

[1] For our purposes, Programs contains programs written in the simple while-language defined in [5, Chapter 2, page 12]

represents the ideal situation where no false alarms are produced. Completeness, however, is a *very rare* condition to be satisfied in practice, therefore, static program analysis needs to, in some way, deal with *incompleteness*. Moreover, the experience tells us that there are results that are "*more incomplete*", i.e., less precise, than others, and this depends upon the way the program is written and the way the abstract interpreter is implemented [7]. As an example, consider the following program

$$\begin{aligned} P \ ::= \ & \textbf{while } x > 1 \textbf{ do } x := x - 2 \\ & \textbf{if } x = 1 \textbf{ then } x := 10 \\ & \textbf{if } x = 2 \textbf{ then } x := 100 \end{aligned}$$

and the set of values $\{2, 4\}$. Clearly $[\![P]\!]\{2, 4\} = \{0\}$. However, when P is analyzed by an abstract interpreter $[\![P]\!]^{\mathsf{Int}}$ (e.g., see [6, Chapter 4.5]) which considers abstract arithmetical operators as the best correct approximation (bca for short), defined on the abstract domain of intervals $\mathsf{Int} \triangleq \{[a, b] \mid a, b \in \mathbb{Z} \cup \{-\infty, +\infty\}, a \leq b\} \cup \{\bot_{\mathsf{Int}}\}$ [2] or defined as the bca for Int, namely, $[\![P]\!]^{\mathsf{Int}}_{\text{bca}} \triangleq \alpha_{\mathsf{Int}} \circ [\![P]\!] \circ \gamma_{\mathsf{Int}} \circ \alpha_{\mathsf{Int}}$, then it may exhibit *different levels of imprecision*. Recall that Int abstracts sets of integer values and it contains all intervals $[a, b]$ such that $a, b \in \mathbb{Z}^{\pm\infty}$ where $\mathbb{Z}^{\pm\infty} \triangleq \mathbb{Z} \cup \{-\infty, +\infty\}$, and $a \leq b$. So, for the input of P we get $\alpha_{\mathsf{Int}}(\{2, 4\}) = [2, 4]$, while for the output we obtain the single point $\alpha_{\mathsf{Int}}([\![P]\!]\{2, 4\}) = [0, 0]$. Then, for the two mentioned analysis of P we get $[\![P]\!]^{\mathsf{Int}}_{\text{bca}}\{2, 4\} = [0, 10]$ and $[\![P]\!]^{\mathsf{Int}}\alpha_{\mathsf{Int}}(\{2, 4\}) = [0, 100]$. Note the three different interval properties obtained by the concrete, bca and abstract evaluations of P over Int with input $\{2, 4\}$: $[0, 0] \leq_{\mathsf{Int}} [0, 10] \leq_{\mathsf{Int}} [0, 100]$. These discrepancies are what we want to *measure*.

To this end, we consider a weaker form of metric function that is made specific for the elements of an abstract domain, namely, it is compatible with the underlying ordering relation $\leq_A$, hence taking into account the presence of incomparable elements. This distance function incorporates both the qualitative comparison given by the partial order and a quantitative comparison (Section 2). By exploiting this idea, we can introduce the notion of ε*-partial completeness* of an abstract domain A with respect to a given program and a given (set of) input values (Section 3). A partially complete abstract interpretation allows *some* false-alarms to be reported, but their number is bounded. In this case, the imprecision of the abstract interpreter is bounded by ε, namely, the distance between the property represented by the abstraction of the concrete semantics and the result of the abstract interpretation on the given input, is at most ε. Given a tolerance ε of imprecision and a set of inputs I, we want to answer the following questions (Section 4): *Can we implement a program analyzer such that all programs having I as input are ε-partial complete? Can we prove if our program analysis has or not a bounded level of imprecision?*

2. Quasi-metrics on Abstract Domains

Our goal is to derive the bound of imprecision of an abstract interpreter with respect to a given measure over the abstract domain. For this reason, we need a metric to compare the elements of the abstract domain according to their relative degree of precision. We refer to the

weaker notion of *quasi-metric* introduced in [8] on a non-empty set S. This is a metric function $\delta : S \times S \to \mathbb{Q}_{\geq 0}$ whose symmetry property may not hold. A set endowed with a quasi-metric is called *quasi-metric space*. Let $\mathcal{A}(C)$ be the set of all abstract domains A abstracting some set of concrete properties C and having ordering relation $\leq_A$ [2].

Definition 2.1 (Quasi-metrics A-compatible). *We say that the distance function $\delta_A : A \times A \to \mathbb{N}^{\infty}_{\geq 0} \cup \{\bot\}$ is a quasi-metric A-compatible for $A \in \mathcal{A}(C)$ if for all $a_1, a_2, a_3 \in A$, it satisfies the following axioms:*

(i) $a_1 = a_2 \;\Leftrightarrow\; \delta_A(a_1, a_2) = 0$

(ii) $a_1 \leq_A a_2 \;\Leftrightarrow\; \delta_A(a_1, a_2) \neq \bot$

(iii) $a_1 \leq_A a_2 \leq_A a_3 \;\Rightarrow\; \delta_A(a_1, a_3) \leq \delta_A(a_1, a_2) + \delta_A(a_2, a_3)$

(iv) $\forall a_1, a_2 \in A, \varepsilon \in \mathbb{N}_{\geq 0}$ *the predicate* $\delta_A(a_1, a_2) \leq \varepsilon$ *is decidable.*

We allow the quasi-metric between two uncomparable elements to be $\bot$, which represents an undefined distance. An abstract domain $A \in \mathcal{A}(C)$ endowed with a quasi-metric A-compatible δ_A, forms an *abstract quasi-metric space*, denoted by $\boldsymbol{A} \triangleq (A, \delta_A)$. We use $\mathfrak{A}(C)$ to refer to the set of all abstract quasi-metric spaces, and write $\boldsymbol{A} \in \mathfrak{A}(C)$.

As an example of quasi-metric A-compatible, we define the weighted path-length $\delta^{\mathfrak{w}}_A$ as the minimum weighted path, w.r.t. a weight function $\mathfrak{w}$, of intermediate elements between two comparable elements of an abstract domain A. Here $\delta^{\mathfrak{w}}_A$ considers the lattice of A as a weighted directed graph where each edge corresponds to two adjacent abstract elements a, b, that is, $a \leq_A b$ and there are no elements $c \in A$ such that $a \leq_A c \leq_A b$. So for example, $\delta^{\mathfrak{w}}_{\mathsf{Int}}$ over the intervals abstraction having $\mathfrak{w}(\cdot, \cdot) = 1$, i.e., setting to 1 the weight of each edge of Int, returns 9 between $[0, 0]$ and $[0, 10]$ meaning that there are exactly 9 more elements in $[0, 10]$ respect to $[0, 0]$, while $\delta^{\mathfrak{w}}_{\mathsf{Int}}([0, 0], [0, +\infty]) = \infty$ and $\delta^{\mathfrak{w}}_{\mathsf{Int}}([0, 0], [1, 10]) = \bot$ because $[0, 0] \not\leq_{\mathsf{Int}} [1, 10]$.

3. Partial Completeness

Standard completeness in program analysis[3], e.g., see [2, 3, 13], means that no false alarms are returned by analyzing the program with an abstract interpreter on any possible input state. Local completeness, instead, is a recently introduced property [14] that requires completeness only with respect to specific inputs. An ε-partially complete program analyzer *allows* some false alarms to be reported over a considered (set of) input, *but* their amount is bounded by a constant ε which is determined according to a quasi-metric which is compatible with the abstract domain used by the analysis. Formally, given a program $P \in \mathsf{Programs}$, a constant bound $\varepsilon \in \mathbb{N}_{\geq 0}$, a non-empty set of stores $S \in \wp(\mathbb{S})$ and $\boldsymbol{A} \in \mathfrak{A}(\wp(\mathbb{S}))$, we say that $\boldsymbol{A}$ is *ε-partially complete* for P in S if $\alpha_A([\![P]\!]S) \leq^{\varepsilon}_{\delta_A} [\![P]\!]^A \alpha_A(S)$, where $a \leq^{\varepsilon}_{\delta_A} b$ iff $\delta_A(a, b) \leq \varepsilon$. We now have all the ingredients to introduce the notion of ε-partial completeness class of programs.

[2] We consider abstract domains in $\mathcal{A}(C)$ that are either recursive or trivial [9, 1].

[3] In some topics concerning abstract interpretation, completeness is typically recurrent, e.g., in comparative semantics [10, 11] or formal languages [12]. Here we are considering only the field of program analysis.

Definition 3.1 (ε-Partial completeness class). *The partial completeness class of an abstract quasi-metric space $\boldsymbol{A} \in \mathfrak{A}(\wp(\mathbb{S}))$, a constant $\varepsilon \in \mathbb{N}_{\geq 0}$ and a non-empty set of inputs $S \in \wp(\mathbb{S})$, denoted $\mathbb{C}(\boldsymbol{A}, \varepsilon, S) \subseteq \mathsf{Programs}$, is defined as:*

$$\mathbb{C}(\boldsymbol{A}, \varepsilon, S) \triangleq \{P \in \mathsf{Programs} \mid \alpha_A([\![P]\!]S) \leq^{\varepsilon}_{\delta_A} [\![P]\!]^A \alpha_A(S)\}.$$

Similarly to the completeness case [15], for every ε, the ε-partial completeness class is infinite and non-extensional. It is infinite because for all $\varepsilon \in \mathbb{N}_{\geq 0}$ and $S \in \wp(\mathbb{S})$, $\mathbb{C}(A) \subseteq \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ and $\mathbb{C}(A)$ is infinite, where $\mathbb{C}(A)$ denotes the (global) completeness class of programs. It is also non-extensional because there always exist programs P and Q such that: P is partially complete for A, $[\![P]\!] = [\![Q]\!]$, and Q is not partially complete for A.

4. Recursive Properties of Partial Complete Programs

It is well known that for trivial abstractions (i.e., either the identity or an abstraction having one element only) the corresponding completeness class turns out to be the whole set of programs [13]. Moreover, for all non-trivial abstractions in $\mathcal{A}(\wp(\mathbb{S}))$, its completeness class is strictly contained in $\mathsf{Programs}$ [15]. In this section, we study the counterpart of these results for the case of partial completeness. It turns out that the quasi-metric A-compatible chosen for measuring the imprecision of the analysis, plays an important rule here in order to determine the recursive properties of $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ and, therefore, of the considered program analysis. We first consider the simplest case of abstract quasi-metric spaces of stores $\boldsymbol{A} \in \mathfrak{A}(\wp(\mathbb{S}))$ satisfying the property of having a limited imprecision by $\varepsilon \in \mathbb{N}_{\geq 0}$, i.e., abstractions such that $\forall a \in A : \delta_A(\bot_A, a) \leq \varepsilon$. These include, for instance, the case of finite height lattices with the weighted path-length quasi-metric—complete lattices which are both ACC and DCC.

Theorem 4.1. *If $\boldsymbol{A} \in \mathfrak{A}(\wp(\mathbb{S}))$ has limited imprecision, then we have for all $S \in \wp(\mathbb{S})$: $\exists \varepsilon \in \mathbb{N}_{\geq 0}$. $\mathbb{C}(\boldsymbol{A}, \varepsilon, S) = \mathsf{Programs}$.*

For example, the $\mathsf{Sign} \triangleq \{\mathbb{Z}, -, 0, +, \varnothing\}$ abstraction [16] measured with $\delta^{\mathfrak{w}}_{\mathsf{Sign}}$ having $\mathfrak{w}(\cdot, \cdot) = 1$, has limited imprecision because by for any $\varepsilon \geq 3$, the partial completeness class turns out $\mathbb{C}((\mathsf{Sign}, \delta^{\mathfrak{w}}_{\mathsf{Sign}}), \varepsilon, S) = \mathsf{Programs}$. The difference with respect to the case of standard completeness class $\mathbb{C}(A)$ is that, thanks to the possibility of admitting an upper margin to imprecision (i.e., possible false alarms), there always exists a class of partial completeness with respect to a given bound which includes all programs.

Conversely, an abstract quasi-metric space of stores $\boldsymbol{A} \in \mathfrak{A}(\wp(\mathbb{S}))$ has *unlimited imprecision* when: $\forall \varepsilon \in \mathbb{N}_{\geq 0},\ \exists a \in A.\ \delta_A(\bot_A, a) > \varepsilon$. The abstract quasi-metric space $(\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}})$ is such an example.

Theorem 4.2. *Let $\boldsymbol{A} \in \mathfrak{A}(\wp(\mathbb{S}))$ be any abstract quasi-metric space of stores with unlimited imprecision δ_A and $S \in \wp(\mathbb{S})$. Then, the following equivalence holds: $\exists \varepsilon \in \mathbb{N}_{\geq 0}$. $\mathbb{C}(\boldsymbol{A}, \varepsilon, S) = \mathsf{Programs} \Leftrightarrow A = \wp(\mathbb{S})$.*

Informally, if we consider a non-trivial abstract quasi-metric space that has unlimited imprecision and an input S, then independently of how we set a threshold ε of false alarms acceptance, there always exists a program P for which the abstract analysis over A with input S, is not

ε-partially complete, namely $P \notin \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$. By a straightforward padding argument, any of these programs can be extended to an infinite set of programs for which the abstraction is *ε-partially incomplete*. The class of ε-partial *incomplete* programs for $\boldsymbol{A} \in \mathfrak{A}(\wp(\mathbb{S}))$ with input S, is the complement set of $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$, formally: $\overline{\mathbb{C}(\boldsymbol{A}, \varepsilon, S)} = \{P \in \mathsf{Programs} \mid \alpha_A(\llbracket P \rrbracket S) \not\preceq^{\varepsilon}_{\delta_A} \llbracket P \rrbracket^A \alpha_A(S)\}$. Theorem 4.2 implies that any non-trivial abstract domain of stores endowed with an unlimited imprecision δ_A, has an infinite set of programs for which the abstract interpreter is ε-partially incomplete. Conversely, if δ_A has limited imprecision, then, trivially, we can always find a certain level of tolerance that makes the analysis ε-partially complete for all programs.

We now study the computational limits of the class of partially complete and incomplete programs. In this case, the topological structure of A in terms of ascending chains, is fundamental. We say that an abstract quasi-metric space of stores $\boldsymbol{A} \in \mathfrak{A}(\wp(\mathbb{S}))$ is *ε-trivial* for some $\varepsilon \in \mathbb{N}_{\geq 0}$ if $\mathbb{C}(\boldsymbol{A}, \varepsilon, S) = \mathsf{Programs}$. The following theorem shows that the class of programs $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ turns out to be recursively enumerable (r.e.) whenever the abstract domain of stores A satisfies the *Ascending Chain Condition* (ACC).

Theorem 4.3. *If $\boldsymbol{A} \in \mathfrak{A}(\wp(\mathbb{S}))$ is ACC, then for every $S \in \wp(\mathbb{S})$ and $\varepsilon \in \mathbb{N}_{\geq 0}$, $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ is r.e..*

Sketch. Run a dovetail algorithm constructing the set $b \triangleq \bigsqcup_{i \in \mathbb{N}} \{\alpha_A(\llbracket P \rrbracket s_i)\}$ starting from $b = \bot$, where $s_i \in S$ is an enumeration of S. If, at some point of the iterates, there exists $s_i \in S$ such that $\llbracket P \rrbracket s_i \neq \varnothing$, $b = b \sqcup_A \alpha_A(\llbracket P \rrbracket s_i)$ and $\delta_A(b, \llbracket P \rrbracket^A \alpha_A(S)) \leq \varepsilon$, then the algorithm terminates. If $P \in \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ then the convergence of the above algorithms is guaranteed by the ACC of $\boldsymbol{A}$. □

The following theorem states that both $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ and $\overline{\mathbb{C}(\boldsymbol{A}, \varepsilon, S)}$ are non-r.e. sets when $\boldsymbol{A}$ is not ε-trivial and not ACC.

Theorem 4.4. *If $\boldsymbol{A} \in \mathfrak{A}(\wp(\mathbb{S}))$ is not ε-trivial, then $\overline{\mathbb{C}(\boldsymbol{A}, \varepsilon, S)}$ is non-r.e.. Moreover, if $\boldsymbol{A}$ is also not ACC, then $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ is non-r.e..*

As a straightforward corollary, the local completeness class $\mathbb{C}(A, S)$ for A satisfying the ACC property is r.e., while non-r.e. for non-ACC abstractions. Furthermore, if A is not trivial, then $\overline{\mathbb{C}(A, S)}$ is non-r.e. even if A is ACC. Let us notice that Theorems 4.3 and 4.4 provide a further insight into the structure of $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ and its complement class $\overline{\mathbb{C}(\boldsymbol{A}, \varepsilon, S)}$. These theorems prove that, given any non-ACC abstract quasi-metric space of stores $\boldsymbol{A}$, whenever we limit the expected imprecision of our analysis to a bound ε of possible false alarms w.r.t. an input S, we cannot build a procedure that enumerates all programs satisfying that bound or that do not respect that bound, unless the abstract domain is ε-trivial. Therefore, deciding whether a static program analysis can produce or cannot produce some bounded set of false alarms is in general impossible, unless A satisfies the ACC. In this last case, we can at least answer "yes" if the static analysis has a bounded imprecision over a program with a specific input. The ε-partial completeness and incompleteness class of an abstraction are therefore a non-trivial property of programs for which no recursively enumerable procedure may exist which is able to enumerate all of their elements.

References

[1] M. Campion, M. Dalla Preda, R. Giacobazzi, Partial (in)completeness in abstract interpretation: Limiting the imprecision in program analysis, Proc. ACM Program. Lang. 6 (2022). doi:10.1145/3498721.

[2] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, ACM Press, 1977, pp. 238–252. doi:10.1145/512950.512973.

[3] P. Cousot, R. Cousot, Systematic design of program analysis frameworks, in: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, ACM Press, 1979, pp. 269–282. doi:10.1145/567752.567778.

[4] P. Cousot, Principles of Abstract Interpretation, The MIT Press, Cambridge, Mass., 2021.

[5] G. Winskel, The formal semantics of programming languages: an introduction, MIT press, 1993.

[6] A. Miné, Tutorial on static inference of numeric invariants by abstract interpretation, Foundations and Trends in Programming Languages 4 (2017) 120–372. URL: https://doi.org/10.1561/2500000034. doi:10.1561/2500000034.

[7] R. Bruni, R. Giacobazzi, R. Gori, I. Garcia-Contreras, D. Pavlovic, Abstract extensionality: on the properties of incomplete abstract interpretations, PACMPL 4 (2020) 28:1–28:28. doi:10.1145/3371096.

[8] W. A. Wilson, On quasi-metric spaces, American Journal of Mathematics 53 (1931) 675–684. doi:10.2307/2371174.

[9] P. Cousot, R. Giacobazzi, F. Ranzato, Program analysis is harder than verification: A computability perspective, in: International Conference on Computer Aided Verification, Springer, 2018, pp. 75–95. doi:10.1007/978-3-319-96142-2_8.

[10] P. Cousot, R. Cousot, Inductive definitions, semantics and abstract interpretation, in: Conference Record of the 19th ACM Symp. on Principles of Programming Languages (*POPL '92*), ACM Press, 1992, pp. 83–94.

[11] P. Cousot, R. Cousot, Compositional and inductive semantic definitions in fixpoint, equational, constraint, closure-condition, rule-based and game-theoretic form (Invited Paper), in: P. Wolper (Ed.), Proc. of the 7th Internat. Conf. on Computer Aided Verification (*CAV '95*), volume 939 of *Lecture Notes in Computer Science*, Springer-Verlag, 1995, pp. 293–308.

[12] M. Campion, M. Dalla Preda, R. Giacobazzi, Abstract interpretation of indexed grammars, in: International Static Analysis Symposium, Springer, 2019, pp. 121–139. doi:10.1007/978-3-030-32304-2_7.

[13] R. Giacobazzi, F. Ranzato, F. Scozzari., Making abstract interpretation complete, Journal of the ACM 47 (2000) 361–416. doi:10.1145/333979.333989.

[14] R. Bruni, R. Giacobazzi, R. Gori, F. Ranzato, A logic for locally complete abstract interpretations, in: Proc. 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021), IEEE Computer Society, 2021, pp. 1–13. doi:10.1109/LICS52264.2021.9470608, distinguished paper.

[15] R. Giacobazzi, F. Logozzo, F. Ranzato, Analyzing program analyses, in: Proceedings

of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015, 2015, pp. 261–273. doi:10.1145/2676726.2676987.

[16] P. Cousot, R. Cousot, Static determination of dynamic properties of programs, in: Proceedings of the 2nd International Symposium on Programming, Dunod, Paris, 1976, pp. 106–130. doi:10.1145/390019.808314.