Partial (In)Completeness in Abstract Interpretation

Limiting the Imprecision in Program Analysis

MARCO CAMPION, University of Verona, Italy
MILA DALLA PREDA, University of Verona, Italy
ROBERTO GIACOBAZZI, University of Verona, Italy

Imprecision is inherent in any decidable (sound) approximation of undecidable program properties. In abstract interpretation this corresponds to the release of false alarms, e.g., when it is used for program analysis and program verification. As all alarming systems, a program analysis tool is credible when few false alarms are reported. As a consequence, we have to live together with false alarms, but also we need methods to control them. As for all approximation methods, also for abstract interpretation we need to estimate the accumulated imprecision during program analysis. In this paper we introduce a theory for estimating the error propagation in abstract interpretation, and hence in program analysis. We enrich abstract domains with a weakening of a metric distance. This enriched structure keeps coherence between the standard partial order relating approximated objects by their relative precision and the effective error made in this approximation. An abstract interpretation is precise when it is complete. We introduce the notion of partial completeness as a weakening of precision. In partial completeness the abstract interpreter may produce a bounded number of false alarms. We prove the key recursive properties of the class of programs for which an abstract interpreter is partially complete with a given bound of imprecision. Then, we introduce a proof system for estimating an upper bound of the error accumulated by the abstract interpreter during program analysis. Our framework is general enough to be instantiated to most known metrics for abstract domains.

CCS Concepts: • **Theory of computation** → **Program analysis**; **Abstraction**; **Invariants**; **Computability**; **Logic and verification**; *Denotational semantics*; Semantics and reasoning.

Additional Key Words and Phrases: Abstract Interpretation, Abstract Domain, Program Analysis, Partial Completeness

ACM Reference Format:
Marco Campion, Mila Dalla Preda, and Roberto Giacobazzi. 2022. Partial (In)Completeness in Abstract Interpretation: Limiting the Imprecision in Program Analysis. *Proc. ACM Program. Lang.* 6, POPL, Article 59 (January 2022), 31 pages. https://doi.org/10.1145/3498721

1 INTRODUCTION

Program analysis has been studied for over half century and is a major method to aid programmers and software engineers in producing reliable artifacts. Abstract interpretation [Cousot 2021; Cousot and Cousot 1977, 1979] is a general theory for the design of sound-by-construction program analysis tools. The abstract interpreter provides an approximate solution to a system of recursive equations specifying the semantics of our programming language. Soundness here means that all true alarms are captured and reported by the analysis but also false-alarms may be reported. The presence of

Authors' addresses: Marco Campion, Dipartimento di Informatica, University of Verona, Verona, Italy, marco.campion@univr.it; Mila Dalla Preda, Dipartimento di Informatica, University of Verona, Verona, Italy, mila.dallapreda@univr.it; Roberto Giacobazzi, Dipartimento di Informatica, University of Verona, Verona, Italy, roberto.giacobazzi@univr.it.

2475-1421/2022/1-ART59
https://doi.org/10.1145/3498721

false alarms is in this case unavoidable due to the need of program analysis to make decidable an often undecidable property of interest (e.g. absence of runtime errors, variables values that do not overflow, etc.).

The abstract interpretation of a program P, denoted by $[\![P]\!]^A$, consists of an abstract domain A, often specified by a pair of abstraction α_A and concretization γ_A maps, and an interpreter $[\![\cdot]\!]^A$, 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. Soundness means that if program P satisfies the condition $[\![P]\!]^A \alpha_A(S) \leq_A Q$ for the input S and output specification Q, 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 represents the ideal situation where no false alarms are produced. Completeness, however, is a *very rare* condition to be satisfied in practice. Abstract domains can be refined (e.g., see [Giacobazzi et al. 2000]) in order to achieve completeness but, as observed by Giacobazzi et al. [2015], the completeness refinement may result in a way too concrete abstract domain, making the abstract interpreter inefficient if not boiling down to the concrete interpretation.

The problem. In practice we need to deal with incompleteness [Distefano et al. 2019]. 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 [Bruni et al. 2020]. Consider the following programs $P \triangleq$ **while** $x > 0$ **do** $x := x - 1$, $Q \triangleq$ **while** $x > 1$ **do** $x := x - 2$, and $R \triangleq$ **while** $x > 1$ **do** $x := x - 2$ **endw**; **if** $x = 1$ **then** $x := 100$. Consider the input property $\{10\}$ for the variable x. It is clear that $[\![P]\!]\{10\} = [\![Q]\!]\{10\} = [\![R]\!]\{10\} = \{0\}$. However, when P, Q, and R are analyzed by an abstract interpreter defined on the abstract domain of intervals Int [Cousot and Cousot 1977], they exhibit different levels of imprecision. Recall that Int abstracts sets of integer values and it contains all intervals $[a, b]$ such that $a \in \mathbb{Z} \cup \{-\infty\}$, $b \in \mathbb{Z} \cup \{+\infty\}$ and $a \leq b$. If $\alpha_{\mathsf{Int}} : \wp(\mathbb{Z}) \longrightarrow \mathsf{Int}$ is the abstraction function approximating any property of integers into the closest interval containing it, then for instance $\alpha_{\mathsf{Int}}(\{-3, 0, 2\}) = [-3, 2]$ and $\alpha_{\mathsf{Int}}(\{\mathbb{Z}_{\geq 1}\}) = [1, +\infty]$. For the programs above we have: $[\![P]\!]^{\mathsf{Int}} \alpha_{\mathsf{Int}}(\{10\}) = \alpha_{\mathsf{Int}}([\![P]\!]\{10\}) = [0, 0]$, $[\![Q]\!]^{\mathsf{Int}} \alpha_{\mathsf{Int}}(\{10\}) = [0, 1]$, and $[\![R]\!]^{\mathsf{Int}} \alpha_{\mathsf{Int}}(\{10\}) = [0, 100]$. While $[\![P]\!]^{\mathsf{Int}} \alpha_{\mathsf{Int}}(\{10\})$ is complete, i.e., $[\![P]\!]^{\mathsf{Int}} \alpha_{\mathsf{Int}}(\{10\}) = \alpha_{\mathsf{Int}}([\![P]\!]\gamma_{\mathsf{Int}}(\alpha_{\mathsf{Int}}(\{10\}))) = \alpha_{\mathsf{Int}}([\![P]\!][10, 10])$, both Q and R are incomplete, but the first is definitively less precise than the second on the abstract domain Int: $[0, 1] \subset [0, 100]$.

As always in approximate methods, e.g., in numerical analysis, we would like to measure the imprecision accumulated during program analysis. The standard abstract interpretation framework does not allow us to compare the degree of imprecision of the resulting abstract semantics. Preliminary results in this direction introduced domain-specific measures of imprecision for static analysis, see Section 8 for related works. However, the lack of a general method to model the way the error propagates during abstract interpretation makes extremely difficult to establish bounds on the accumulated error, and hence of the possible amount of false alarms, in program analysis.

Main contribution. In this paper, we formalize the notion of limiting imprecision in abstract interpretation. 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. In the previous example, we can consider the distance in Int to be the length of the minimum path between two comparable intervals in the lattice of intervals when finite, otherwise the distance is set to ∞. For the program P above, the distance between $[0, 0]$ and $[0, 1]$ is 1 while the distance between $[0, 1]$ and $[0, 100]$ is 100. Intuitively, this simple metric encodes the number of spurious elements added in the output of the abstract execution w.r.t. the abstraction of the concrete one.

This idea formalizes the intuition that the abstract computation of an abstract interpreter for a program Q with input $\{10\}$ is *more imprecise* than the abstract semantics $\alpha_{\mathsf{Int}}(\llbracket Q \rrbracket \gamma_{\mathsf{Int}}(\alpha_{\mathsf{Int}}(\{10\})))$, which is not by chance said *best correct approximation* (bca for short) of $\llbracket Q \rrbracket \{10\}$. Same happens for R, which is less precise than Q. 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. 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 results of the abstraction of the concrete semantics and the result of the abstract interpretation on the given input, is at most ε.

In Section 3 we introduce the notion of quasi-metric A-compatible with respect to a given abstract domain A. The notion of ε-partial completeness for an abstract interpreter defined on an abstract domain enriched with a A-compatible quasi-metric is introduced in Section 4. This corresponds to a weakening of the notion of local completeness in abstract interpretation as introduced in [Bruni et al. 2021], which is in turn a further weakening of standard (global) completeness [Cousot and Cousot 1979; Giacobazzi et al. 2000]. Indeed, any program complete for A is also ε-partial complete for all $\varepsilon \geq 0$. In Section 6 we study the main recursive properties of the class of all programs for which an abstract interpreter is ε-partial complete. We show that these classes are infinite, in general non-extensional, sets, i.e., they do not form an index-set of partial recursive functions [Rogers 1992]. We also prove that the distance chosen for measuring the relative imprecision between abstract elements, plays an important role in the class of partial completeness. If for every ε and input S we can always find an element whose distance from $\bot_A$ is strictly greater than ε, then we can always constructively build a program that is outside the ε-partial completeness class. An abstract domain satisfying the above condition will be said holding *unlimited imprecision*. We also prove that if we consider non-ε-trivial abstract domains (i.e., abstract domains endowed with a A-compatible distance whose ε-partial completeness class is different from the set of all possible programs) with unlimited imprecision, then for any bound ε, the ε-partial incompleteness class is a non-recursively enumerable set. The same result is obtained for the ε-partial completeness when A has infinite ascending chains and is non-ε-trivial. This means that, in general, we cannot automate the procedure of deciding whether the abstract interpretation of a given program on a given input satisfy a given precision bound. For this reason, in Section 7, we introduce a sound proof system that is able to derive Hoare-like triples $\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ P\ [\,\mathrm{Post}, \varepsilon\,]$ asserting that, the abstraction of the post-condition Post has a distance at most ε from the abstraction of the concrete semantics of P on input Pre. The value ε next to the post-condition can be interpreted as an upper bound for the imprecision accumulated during the computation of the abstract interpreter of the program P with input Pre. By instantiating the post-condition with the output of our abstract interpreter, i.e., proving $\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ P\ [\,\gamma_A(\llbracket P \rrbracket^A \alpha_A(\mathrm{Pre})), \varepsilon\,]$, we can prove that P with input Pre belongs to the ε-partial completeness class. The same applies for proving an upper bound for the imprecision of the bca w.r.t. the concrete semantics, namely, $\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ P\ [\,\alpha_A(\llbracket P \rrbracket \gamma_A(\alpha_A(\mathrm{Pre}))), \beta\,]$. By a slight adjustment of the rules of our proof system for the basic commands (viz., assignments and boolean guards), we obtain a new proof system denoted $\Vdash_{\boldsymbol{A}}$, that can soundly prove the upper bound of imprecision between the bca and the abstract interpreter, i.e., $\Vdash_{\boldsymbol{A}} [\,\gamma_A(\alpha_A(\mathrm{Pre}))\,]\ P\ [\,\gamma_A(\llbracket P \rrbracket^A \alpha_A(\mathrm{Pre})), \eta\,]$. By exploiting both $\vdash_{\boldsymbol{A}}$ and $\Vdash_{\boldsymbol{A}}$, we can infer three different upper bounds of inaccuracy between: The concrete computation, the bca, and the abstract interpreter. Section 8 discusses the most relevant related works and Section 9 concludes and highlights future works.

2 BACKGROUND

2.1 Order and Measure Theory

Sets. Given two sets S and T, $\wp(S)$ denotes the powerset of S, $S \setminus T$ denotes the set-difference between S and T, $\overline{S}$ denotes the complement of S with respect to some universe set determined by the context, $S \subseteq T$ denotes sets inclusion while $S \subset T$ (or $S \subsetneq T$) denotes strict sets inclusion, $|S|$ denotes the cardinality where S is finite if $|S| < \omega$, countably infinite if $|S| = \omega$, countable if $|S| \leq \omega$. A binary relation $\sim$ over a set S is a subset of the Cartesian product $\sim \subseteq S \times S$. We will emphasize the set S on which a binary relation $\sim$ is defined by the subscript $\sim_S$ except for the straightforward equivalence relation $=$ unless it has a different definition. We denote with $\mathbb{N}$, $\mathbb{Q}$ and $\mathbb{R}$ the sets of all, respectively, natural, rational and real numbers. We will use subscripts in order to limit their range, while the superscript symbol ∞ denotes the inclusion of the infinite symbol. For example, $\mathbb{Q}^{\infty}_{\geq 0}$ denotes the set of all non-negative rational numbers with the infinity element. As calculation rules, any sum or difference that involves the symbol ∞, returns ∞.

Functions. We denote with $f : S \to T$ a *totally* defined function and with $f : S \nrightarrow T$ a *partially* defined function. If $f : S \nrightarrow T$ then $f(x)\downarrow$ denotes that $f(x)$ is defined, $dom(f) \triangleq \{x \in S \mid f(x)\downarrow\}$ denotes the domain of f, and given a subset $X \subseteq S$, $f(X) \triangleq \{f(x) \in T \mid x \in X \cap dom(f)\}$ denotes the image of f on X, where f is defined. Two partial functions $f, g : S \nrightarrow T$ are *extensionally equivalent*, denoted by $f \cong g$, if $dom(f) = dom(g)$ and for all $x \in dom(f) = dom(g)$, $f(x) = g(x)$. Sometimes we use a λ-notation $\lambda x.f(x)$ to emphasize the arguments of a function f. Given $f : S \nrightarrow T$ and $g : T \nrightarrow U$, $g \circ f : S \nrightarrow U$ denotes their composition, where $g \circ f(x) = g(f(x))$ when $f(x)\downarrow$ and $g(f(x))\downarrow$, otherwise $g \circ f$ is not defined on x, denoted $g(f(x))\uparrow$.

Measure and Metric. A σ-algebra on a set X is a collection of subsets of X that includes X itself, is closed under complement and is closed under countable unions. The definition implies that it also includes the empty set $\varnothing$ and that it is closed under countable intersections. Consider a σ-algebra A over X. The tuple (X, A) is called a *measurable space*. A *measure* is a non-negative countably additive set function. A function $\mu : A \to \mathbb{R}^{\infty}_{\geq 0}$ is called a measure if it satisfies the following properties: (1) $\forall S \in A.\ \mu(S) \geq 0$ (non-negativity); (2) $\mu(\varnothing) = 0$ (null empty set); (3) if $S_i \in A$ is a countable sequence of disjoint sets, then $\mu(\bigcup_i S_i) = \sum_i \mu(S_i)$ (countable additivity). The triple (X, A, μ) is called a *measure space*. A *metric* is a function that defines a distance between pairs of elements of a set S. A metric on a non-empty set S is a map $d : S \times S \to \mathbb{R}_{\geq 0}$ that $\forall x, y, z \in S$ satisfies (1) the identity of indiscernibles $d(x, y) = 0 \Leftrightarrow x = y$, (2) symmetry $d(x, y) = d(y, x)$, (3) triangle inequality $d(x, y) + d(y, z) \geq d(x, z)$. A set provided with a metric is called *metric space*.

Order theory. A set L endowed with a partial order relation $\leq_L$ is called a partially ordered set, or briefly *poset*, and it is denoted by $\langle L, \leq_L \rangle$. We will use also the *strict* poset relation $<_L$ such that for all $x, y \in L$, $x <_L y$ iff $x \leq_L y$ and $x \neq y$. We say that y *covers* x, written $x \lessdot_L y$, if $x < y$ and there is no element $z \in L$ such that $x < z < y$. A subset $Y \subseteq L$ of a poset $L = \langle L, \leq_L \rangle$ is a chain if for all $y_1, y_2 \in Y$, $y_1 \leq_L y_2$ or $y_2 \leq_L y_1$. A sequence $(l_n)_{n \in \mathbb{N}} = \{l_n \mid n \in \mathbb{N}\}$ of elements in L is an ascending chain if $n \leq m$ implies $l_n \leq_L l_m$. Similarly, a sequence $(l_n)_n$ is a descending chain if $n \leq m$ implies $l_m \leq_L l_n$. A sequence $(l_n)_{n \in \mathbb{N}}$ eventually stabilizes if and only if there exists $n_0 \in \mathbb{N}$ such that for every $n \in \mathbb{N}$: $n \geq n_0$ implies $l_n = l_{n_0}$. A poset L has finite height if and only if all chains are finite. L satisfies the *Ascending Chain Condition* (ACC) if and only if all ascending chains eventually stabilize. Similarly, it satisfies the *Descending Chain Condition* (DCC) if and only if all descending chains eventually stabilize. L satisfies both the ACC and DCC if and only if it has finite height. A poset $\langle L, \leq_L \rangle$ is called a join-semilattice if each two-element subset $\{a, b\} \subseteq L$ has a join (i.e. least upper bound), and is called a meet-semilattice if each two-element subset has a meet (i.e.

greatest lower bound), denoted by $a \vee_L b$ and $a \wedge_L b$ respectively. $\langle L, \leq_L \rangle$ is called a lattice if it is both a join- and a meet-semilattice. This definition makes $\vee_L$ and $\wedge_L$ binary operations. A lattice $\langle L, \leq_L \rangle$ is complete when for all subsets $X \subseteq L$, arbitrary lubs $\bigvee_L X$ and glbs $\bigwedge_L X$ exist in L (empty subset included). A *complete lattice* L with partial order $\leq_L$, lub $\vee_L$, glb $\wedge_L$, greatest element (top) $\top_L$, and least element (bottom) $\bot_L$ is denoted by $\langle L, \leq_L, \vee_L, \wedge_L, \top_L, \bot_L \rangle$.

A function $f : L \to L$ over a poset $\langle L, \leq_L \rangle$ is monotone if for all $x, y \in L$ such that $x \leq_L y$, f preserves the order, i.e., $f(x) \leq_L f(y)$. Moreover, f is idempotent if for every $x \in L$. $f(f(x)) = f(x)$, increasing if $x \leq_L f(x)$. If $f, g : S \to L$ and $\langle L, \leq_L \rangle$ is a poset then the pointwise partial order relation is defined by: $f \sqsubseteq g$ when for all $x \in S$, $f(x) \leq_L g(x)$. If L is a (complete) lattice then $\langle S \to \langle L, \leq_L \rangle \rangle$ is a (complete) lattice. A function $f : L_1 \to L_2$ between complete lattices is additive (co-additive) if for all $Y \subseteq L_1$, $f(\vee_{L_1} Y) = \vee_{L_2} f(Y)$ ($f(\wedge_{L_1} Y) = \wedge_{L_2} f(Y)$). Also, f is continuous (co-continuous) when f preserves lubs (glbs) of chains in L_1. The Knaster–Tarski theorem guarantees that if L is a complete lattice and $f : L \to L$ a monotone function, then the set of fixpoints of f in L is also a complete lattice. As a consequence, since complete lattices cannot be empty (they must contain supremum of empty set), the theorem in particular guarantees the existence of at least one fixpoint of f, and even the existence of a least (or greatest) fixpoint, denoted $\mathsf{lfp}(f)$ (resp. $\mathsf{gfp}(f)$). Moreover, if $f : L \to L$ is continuous then $\mathsf{lfp}(f) = \bigvee_{L\, n \in \mathbb{N}} f^n(\bot_L)$, where, for all $n \in \mathbb{N}$ and $x \in L$, f^n is inductively defined by: $f^0(x) \triangleq x$ and $f^{n+1}(x) \triangleq f(f^n(x))$.

2.2 Abstract Interpretation

We consider here the standard abstract interpretation framework as defined by Cousot and Cousot [1977, 1979, 1992a] and based on the correspondence between a domain of concrete or exact properties and a domain of abstract or approximate properties.

Abstract Domains. Abstract domains (also called abstractions) are specified by Galois connection-s/insertions (GCs/GIs for short). In program analysis, concrete and abstract domains are assumed to be complete lattices, resp. $\langle C, \leq_C \rangle$ and $\langle A, \leq_A \rangle$, which are related by abstraction and concretization maps $\alpha_A : C \to A$ and $\gamma_A : A \to C$ that give rise to a GC $(\alpha_A, C, A, \gamma_A)$, that is, for all $a \in A$ and $c \in C$: $\alpha_A(c) \leq_A a \Leftrightarrow c \leq_C \gamma_A(a)$, where we use the subscript to functions α_A and γ_A in order to emphasize the abstract domain A considered. A GC is a GI when $\alpha_A \circ \gamma_A = \lambda x.x$. Let us recall some basic properties of a GC $(\alpha_A, C, A, \gamma_A)$: (1) α_A is additive and γ_A is co-additive; (2) $\gamma_A \circ \alpha_A : C \to C$ is a closure operator, namely, it is a monotone, idempotent and increasing function; (3) if $\rho : C \to C$ is a closure operator then $(\rho, C, \rho(C), \lambda x.x)$ is a GI. In the rest of the paper, we deal with GI which are standard in abstract interpretation (e.g., Sign, Intervals, Octagons, Ellipsoids, etc.) ensuring the existence of abstraction functions[1]. We highlight this hypothesis in the following assumption.

Assumption 1. *We consider only GI-based abstract interpretations.*

If $\alpha_A : C \to A$ is an additive function, then it induces a GC $(\alpha_A, C, A, \alpha_A^+)$ where the concretization $\alpha_A^+ : A \to C$ is defined as right-adjoint of α_A, i.e., $\alpha_A^+(a) \triangleq \vee_C \{c \in C \mid \alpha_A(c) \leq_A a\}$. Dually, if $\gamma_A : C \to A$ is a co-additive function then $(\gamma_A^-, C, A, \gamma_A)$ is a GC where $\gamma_A^- \triangleq \lambda c. \wedge_A \{a \in A \mid c \leq_C \gamma_A(a)\}$ is the left-adjoint of γ_A.

We use $\mathcal{A}(C)$ to denote all the possible abstractions of a concrete domain C satisfying Assumption 1, where $A \in \mathcal{A}(C)$ means that A is an abstract domain of C defined by some GI which is left unspecified. An abstract domain $A \in \mathcal{A}(C)$ is called *strict* if $\gamma_A(\bot_A) = \bot_C$. We say that an element $c \in C$ is *exactly represented* in A if $\gamma_A(\alpha_A(c)) = c$. If $A_1, A_2 \in \mathcal{A}(C)$ then A_1 is equivalent to A_2, denoted by $A_1 \sim_{\mathcal{A}(C)} A_2$, when $\gamma_{A_1}(A_1) = \gamma_{A_2}(A_2)$. The quotient $\mathcal{A}(C)_{/\sim_{\mathcal{A}(C)}}$ is called the lattice

[1]Weaker forms of abstract interpretation are possible, e.g., lacking the abstraction as in polyhedra, but this makes the framework too weak to prove some results presented in Section 6 (more specifically, Theorem 6.11 and Theorem 6.16).

of abstractions because it turns out to be a complete lattice w.r.t. the relative precision ordering: $A_1 \leq_{\mathcal{A}(C)} A_2$ iff for all $c \in C$, $\gamma_{A_1}(\alpha_{A_1}(c)) \leq_C \gamma_{A_2}(\alpha_{A_2}(c))$. Thus, $A_1 \leq_{\mathcal{A}(C)} A_2$ means that A_1 is a more precise abstraction than A_2, or, equivalently, that A_2 abstracts A_1. The following are abstract domain examples of $\mathcal{A}(\wp(\mathbb{Z}))$.

Example 2.1. The Sign $\triangleq \{\mathbb{Z}, -, 0, +, \varnothing\}$ and Parity $\triangleq \{\mathbb{Z}, \mathit{even}, \mathit{odd}, \varnothing\}$ abstract domains for, respectively, sign and parity analysis of integer variables, are a straightforward non-relational abstractions of $\langle \wp(\mathbb{Z}), \subseteq \rangle$ [Cousot and Cousot 1976], i.e., Sign, Parity $\in \mathcal{A}(\wp(\mathbb{Z}))$, where the order relation $\leq_{\mathsf{Sign}}$ is defined as $\varnothing <_{\mathsf{Sign}} 0 <_{\mathsf{Sign}} - <_{\mathsf{Sign}} \mathbb{Z}$ and $\varnothing <_{\mathsf{Sign}} 0 <_{\mathsf{Sign}} + <_{\mathsf{Sign}} \mathbb{Z}$, while $\varnothing <_{\mathsf{Parity}} \mathit{even} <_{\mathsf{Parity}} \mathbb{Z}$ and $\varnothing <_{\mathsf{Parity}} \mathit{odd} <_{\mathsf{Parity}} \mathbb{Z}$. The abstraction maps $\alpha_{\mathsf{Sign}} : \wp(\mathbb{Z}) \to$ Sign and $\alpha_{\mathsf{Parity}} : \wp(\mathbb{Z}) \to$ Parity are defined by:

$$\alpha_{\mathsf{Sign}}(X) \triangleq \begin{cases} \varnothing & \text{if } X = \varnothing, \\ 0 & \text{if } X = \{0\}, \\ + & \text{if } \forall x \in X.\ x \geq 0, \\ - & \text{if } \forall x \in X.\ x \leq 0, \\ \mathbb{Z} & \text{otherwise} \end{cases} \qquad \alpha_{\mathsf{Parity}}(X) \triangleq \begin{cases} \varnothing & \text{if } X = \varnothing, \\ \mathit{even} & \text{if } \forall x \in X.\ x \bmod 2 = 0, \\ \mathit{odd} & \text{if } \forall x \in X.\ x \bmod 2 \neq 0, \\ \mathbb{Z} & \text{otherwise.} \end{cases}$$

where mod is the integer modulo operation. ∎

Example 2.2. The interval abstraction Int [Cousot and Cousot 1976] is a widely used non-relational abstraction since it is efficient and yet able to give useful information to prove, e.g., the absence of arithmetic overflows or out-of-bounds array accesses. Let $\mathbb{Z}^* \triangleq \mathbb{Z} \cup \{-\infty, +\infty\}$ and assume that the standard ordering $\leq$ on $\mathbb{Z}$ is extended to $\mathbb{Z}^*$ in the usual way. Hence:

$$\mathsf{Int} \triangleq \{[a, b] \mid a, b \in \mathbb{Z}^*,\ a \leq b\} \cup \{\bot_{\mathsf{Int}}\}$$

endowed with the standard ordering $\leq_{\mathsf{Int}}$ induced by the interval containment gives rise to a complete lattice, where $\bot_{\mathsf{Int}}$ is the bottom element and $\top_{\mathsf{Int}} \triangleq [-\infty, +\infty]$ is the top element. We have that $\mathsf{Int} \in \mathcal{A}(\wp(\mathbb{Z}))$. Consider the function $\mathit{min} : \wp(\mathbb{Z}) \to \mathbb{Z}^*$ defined as $\mathit{min}(X) \triangleq x$ if $\exists x \in X.\ \forall y \in X.\ x \leq y$, while $\mathit{min}(X) \triangleq -\infty$ otherwise, and the function $\mathit{max} : \wp(\mathbb{Z}) \to \mathbb{Z}^*$ dually defined. The abstraction map $\alpha_{\mathsf{Int}} : \wp(\mathbb{Z}) \to \mathsf{Int}$ is defined by:

$$\alpha_{\mathsf{Int}}(X) \triangleq \begin{cases} \bot_{\mathsf{Int}} & \text{if } X = \varnothing, \\ [\mathit{min}(X), \mathit{max}(X)] & \text{otherwise.} \end{cases}$$

Note that α_{Int} preserves arbitrary unions in $\wp(\mathbb{Z})$ and therefore gives rise to a GI. ∎

Abstract semantics: Correctness and completeness. Let $f : C \to C$ be a concrete monotone (transfer) function—to keep notation simple, we consider unary functions—and let $f^\sharp : A \to A$ be a corresponding monotone abstract (transfer) function defined on some abstraction $A \in \mathcal{A}(C)$. Then, $f^\sharp$ is a *correct* (or *sound*) approximation of f on A when $\alpha_A \circ f \leq_A f^\sharp \circ \alpha_A$ holds. If $f^\sharp$ is correct for f then least fixpoint correctness holds, that is, $\alpha_A(\mathsf{lfp}(f)) \leq_A \mathsf{lfp}(f^\sharp)$ holds. The abstract function $f^\alpha \triangleq \alpha_A \circ f \circ \gamma_A : A \to A$ is called the *best correct approximation* (bca) of f on A, because it turns out that any abstract function $f^\sharp$ is a correct approximation of f iff $f^\alpha \leq_A f^\sharp$. Hence, f^α plays the role of the best possible correct approximation of f on A.

An abstract function $f^\sharp : A \to A$ is a *complete* approximation of f on A when $\alpha_A \circ f = f^\sharp \circ \alpha_A$ holds [Cousot and Cousot 1979]. Our definition of completeness corresponds to the backward completeness defined by Giacobazzi et al. [2000]. When $f^\sharp$ is an abstract transfer function on A used in some static program analysis algorithm, completeness intuitively encodes an optimal precision for $f^\sharp$, meaning that the abstract behavior of $f^\sharp$ on A exactly matches the abstraction in A of the

concrete behaviour of f. If $f^\sharp$ is complete for f then least fixpoint completeness holds (also called fixpoint transfer), i.e., $\alpha_A(\mathsf{lfp}(f)) = \mathsf{lfp}(f^\sharp)$ holds. It turns out that completeness $\alpha_A \circ f = f^\sharp \circ \alpha_A$ holds iff $\alpha_A \circ f = \alpha_A \circ f \circ \gamma_A \circ \alpha_A = f^\alpha \circ \alpha_A$ holds. Thus, the possibility of defining a complete approximation $f^\sharp$ of f on some $A \in \mathcal{A}(C)$ only depends upon the concrete function f and the abstraction A, that is, f^α is the only possible option as complete approximation of f. In the following, we write both "A is complete for f" and "f is complete for A" for $\alpha_A \circ f = f^\alpha \circ \alpha_A$.

3 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. Roughly, a metric allows us to measure the distance between elements in a given set. Intuitively, when applied to an abstract domain $A \in \mathcal{A}(C)$, we would like to have a distance $d : A \times A \longrightarrow \mathbb{R}_{\geq 0}$ which is somehow compatible with the underlying partial order $\leq_A$. For instance, if $a_1 \leq_A a_2 \leq_A a_3$ with $a_1, a_2, a_3 \in A$, then we would expect that $d(a_1, a_2) \leq d(a_1, a_3)$.

The classical definition of distance in measure theory does not work in this situation because in a metric space all pairs of elements can be compared. We relax the classical notion of distance on sets and conjugate it with the underlying order of the elements of an abstract domain.

We refer to the weaker notion of *quasi-metric* introduced by Wilson [1931] on a non-empty set S. This is a metric function $\delta : S \times S \longrightarrow \mathbb{R}_{\geq 0}$ whose symmetry property may not hold. A set endowed with a quasi-metric is called *quasi-metric space*. We adapt this definition of quasi-metric in order to cope with the structure of an abstract domain $A \in \mathcal{A}(C)$ as poset of approximate program properties.

*Definition 3.1 (**Quasi-metrics A-compatible**).* Let $A \in \mathcal{A}(C)$ be an abstract domain with ordering relation $\leq_A$. We say that $\delta_A : A \times A \longrightarrow \mathbb{R}^\infty_{\geq 0} \cup \{\bot\}$ is a *quasi-metric A-compatible* if for all $a_1, a_2, a_3 \in A$, it satisfies the following axioms:

(i) $a_1 = a_2 \iff \delta_A(a_1, a_2) = 0$ (identity of indiscernibles)

(ii) $a_1 \leq_A a_2 \iff \delta_A(a_1, a_2) \neq \bot$ (approximation)

(iii) $a_1 \leq_A a_2 \leq_A a_3 \implies \delta_A(a_1, a_3) \leq \delta_A(a_1, a_2) + \delta_A(a_2, a_3)$ (weak triangle inequality)

□

Here, for all $\varepsilon \in \mathbb{R}_{\geq 0}$: $\varepsilon < \infty$. We allow the quasi-metric between two uncomparable elements to be $\bot$, which represents an undefined distance. Informally, we say that a_1 is approximated by a_2 if and only if the quasi-metric A-compatible between a_1 and a_2 is defined. The value of the distance δ_A between two elements can be interpreted as the *error introduced by the approximation: The lower is the value of the error $\delta_A(a_1, a_2)$, the better is the approximation.* The value $\delta_A(a_1, a_2) = \bot$ expresses naturally the fact that a_2 does not approximate a_1, i.e., $a_1 \not\leq_A a_2$, while equal elements will always have a null quasi-distance. Note that, for every quasi-metric A-compatible, the approximation axiom implies δ_A to embody the underlying poset structure and therefore $\leq_A$ induces the quasi-metric δ_A. We can now adapt the general definition of quasi-metric space to the definition of *abstract quasi-metric space.*

*Definition 3.2 (**Abstract quasi-metric spaces**).* 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 the set of all abstract quasi-metric spaces, and write $\boldsymbol{A} \in \mathfrak{A}(C)$. □

Let us give some examples of quasi-metrics. We formalize the *weighted path-length* distance which considers a discrete lattice A as a weighted directed graph. Let $E_A \subseteq A \times A$ be the set of all pairs

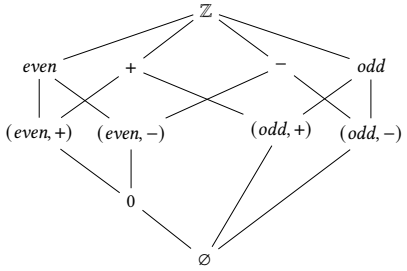

Fig. 1. P ⊓ S abstract domain

(a, b) such that $a \lessdot_A b$. Let $a, b \in A$ with $a \neq b$, we denote with $\mathfrak{C}_a^b$ the set of all possible chains $\mathbf{c} \subseteq E$ such that if $(c, d) \in \mathbf{c}$ then $a \leq_A c \lessdot_A d \leq_A b$. It is clear that if $a \not\leq_A b$ then $\mathfrak{C}_a^b = \varnothing$.

*Definition 3.3 (**Weighted path-length**).* Let $\mathfrak{w} : E_A \to \mathbb{R}_{>0}$ be a weight function. We define $\delta^{\mathfrak{w}} : A \times A \to \mathbb{R}_{\geq 0}^{\infty} \cup \{\bot\}$ with $A \in \mathcal{A}(C)$ such that for every $a, b \in A$:

$$\delta^{\mathfrak{w}}(a, b) \triangleq \begin{cases} 0 & \text{if } a = b, \\ \infty & \text{if } \forall \mathbf{c} \in \mathfrak{C}_a^b.\ |\mathbf{c}| = \omega, \\ \min\left\{ \sum_{e \in \mathbf{c}} \mathfrak{w}(e) \;\middle|\; \begin{array}{l} \mathbf{c} \in \mathfrak{C}_a^b \\ |\mathbf{c}| < \omega \end{array} \right\} & \text{if } \exists \mathbf{c} \in \mathfrak{C}_a^b.\ |\mathbf{c}| < \omega, \\ \bot & \text{if } \mathfrak{C}_a^b = \varnothing. \end{cases}$$

□

It is easy to note that the previous definition fulfills axioms (i)-(iii) of Definition 3.1.

Proposition 3.4. *For all $A \in \mathcal{A}(C)$, $\delta_A^{\mathfrak{w}}$ is a quasi-metric A-compatible.*

Intuitively, $\delta_A^{\mathfrak{w}}$ reflects exactly the underlying abstract domain structure. It counts the minimum weighted path w.r.t. $\mathfrak{w}$ of intermediate elements between two comparable elements of an abstract domain A. Note that $\delta_A^{\mathfrak{w}}$ does not satisfy symmetry and it relates only elements that belong to the same chain. It is clear that $\delta_A^{\mathfrak{w}}$ refines the standard metric associated with the partial order on A, providing a quantitative value to the length of chains separating abstract objects in A. The following are examples of $\delta_A^{\mathfrak{w}}$ instantiated for $A \in \{\mathsf{Sign}, \mathsf{P} \sqcap \mathsf{S}, \mathsf{Int}\}$.

Example 3.5. Consider the $\mathsf{Sign} \triangleq \{\mathbb{Z}, -, 0, +, \varnothing\}$ abstract domain shown in Example 2.1. For all $(a, b) \in E_{\mathsf{Sign}}$, let $\mathfrak{w}(a, b) \triangleq 1$. Then, the weighted path-length $\delta_{\mathsf{Sign}}^{\mathfrak{w}}$ is a quasi-metric Sign-compatible and the pair $(\mathsf{Sign}, \delta_{\mathsf{Sign}}^{\mathfrak{w}}) \in \mathfrak{A}(\wp(\mathbb{Z}))$ forms an abstract quasi-metric space. The following are examples of evaluations of $\delta_{\mathsf{Sign}}^{\mathfrak{w}}$ on some elements of Sign: $\delta_{\mathsf{Sign}}^{\mathfrak{w}}(+, +) = 0$, $\delta_{\mathsf{Sign}}^{\mathfrak{w}}(\varnothing, 0) = 1$, $\delta_{\mathsf{Sign}}^{\mathfrak{w}}(0, \mathbb{Z}) = 2$, $\delta_{\mathsf{Sign}}^{\mathfrak{w}}(\varnothing, \mathbb{Z}) = 3$, $\delta_{\mathsf{Sign}}^{\mathfrak{w}}(\mathbb{Z}, \varnothing) = \bot$, $\delta_{\mathsf{Sign}}^{\mathfrak{w}}(+, -) = \delta_{\mathsf{Sign}}^{\mathfrak{w}}(-, +) = \bot$. ■

Example 3.6. Consider the abstract domain $\mathsf{P} \sqcap \mathsf{S} \in \mathcal{A}(\wp(\mathbb{Z}))$, shown in Fig. 1, which is the reduced product of Sign and Parity abstract domains [Cousot and Cousot 1979]. Let $\mathfrak{w}(\varnothing, (odd, +)) = \mathfrak{w}(\varnothing, (odd, -)) \triangleq 2$ while 1 for the remaining pairs in $E_{\mathsf{P} \sqcap \mathsf{S}}$. Then, the weighted path-length $\delta_{\mathsf{P} \sqcap \mathsf{S}}^{\mathfrak{w}}$ is a quasi-metric $\mathsf{P} \sqcap \mathsf{S}$-compatible. ■

Example 3.7. Consider the abstract domain of intervals $\mathsf{Int} \in \mathcal{A}(\wp(\mathbb{Z}))$, shown in Example 2.2. For every $(a, b) \in E_{\mathsf{Int}}$, let $\mathfrak{w}(a, b) \triangleq 1$. Then, the weighted path-length $\delta_{\mathsf{Int}}^{\mathfrak{w}}$ is a quasi-metric Int-compatible. The intuition of $\delta_{\mathsf{Int}}^{\mathfrak{w}}$ is to count how many more elements one interval has w.r.t. another one. That is, if $i_1, i_2 \in \mathsf{Int}$ and $\delta_{\mathsf{Int}}^{\mathfrak{w}}(i_1, i_2) = k$ for some $k \in \mathbb{N}$, then the interval i_2 contains exactly k more elements than i_1. ■

As another class of distance measure, we consider measurable spaces. Let C be a concrete domain that forms a σ-algebra over a set X, i.e., $C \subseteq \wp(X)$, then the pair (X, C) is a measurable space. We can build over (X, C) a measure function $\mu : C \to \mathbb{R}^{\infty}_{\geq 0}$ that allows us to measure the distance between some property of interest in C, hence defining a quasi-metric A-compatible for $A \in \mathcal{A}(C)$ based on (X, C, μ).

*Definition 3.8 (**μ-distance**).* Let (X, C, μ) be a measure space and $A \in \mathcal{A}(C)$. For all $S^\sharp_1, S^\sharp_2 \in A$, we define $\delta^\mu : A \times A \to \mathbb{R}^{\infty}_{\geq 0} \cup \{\bot\}$ as:

$$\delta^\mu(S^\sharp_1, S^\sharp_2) \triangleq \begin{cases} \mu(\gamma_A(S^\sharp_2)) - \mu(\gamma_A(S^\sharp_1)) & \text{if } S^\sharp_1 \leq_A S^\sharp_2, \\ \bot & \text{otherwise.} \end{cases}$$

□

Example 3.9. Let us consider the measure space $(\mathbb{Z}, \wp(\mathbb{Z}), \mu^c)$, where μ^c is the counting measure such that, for all $S \in \wp(\mathbb{Z})$, $\mu^c(S) \triangleq |S|$ if $|S| < \omega$ and ∞ otherwise. Let us consider also the abstract domain $\mathsf{Int} \in \mathcal{A}(\wp(\mathbb{Z}))$. Then $\delta^{\mu^c}_{\mathsf{Int}}$ is a quasi-metric Int-compatible. Note that $\delta^{\mu^c}_{\mathsf{Int}}$ differs from $\delta^{\mathfrak{w}}_{\mathsf{Int}}$ defined in Example 3.7. For example, $\delta^{\mu^c}_{\mathsf{Int}}([5, +\infty], [0, +\infty]) = \infty \neq 5 = \delta^{\mathfrak{w}}_{\mathsf{Int}}([5, +\infty], [0, +\infty])$. ■

Example 3.10. Let us consider the measure space $(\mathbb{R}^n, \mathfrak{B}(\mathbb{R}^n), \mu^L)$, where $\mathfrak{B}(\mathbb{R}^n)$ is the Borel σ-algebra and μ^L is the Lebesgue measure. Let us consider also an abstract domain $A \in \mathcal{A}(\mathfrak{B}(\mathbb{R}^n))$. Then $\delta^{\mu^L}_A$ is a quasi-metric A-compatible. ■

In the following we use the bold symbol $\boldsymbol{A}$ to indicate the pair (A, δ_A) where $A \in \mathcal{A}(C)$ is an abstract domain and δ_A is a quasi-metric A-compatible. When we want to use a specific quasi-metric A-compatible and / or a specific abstract domain A we explicitly write the pair, e.g., $(\mathsf{Sign}, \delta^{\mathfrak{w}}_{\mathsf{Sign}})$. If δ_A has no superscript, then it is intended as for any quasi-metric A-compatible, otherwise the superscript refers to the specific quasi-metric used. For example, δ_{Sign} refers to any quasi-metric Sign-compatible, whereas $\delta^{\mathfrak{w}}_{\mathsf{Sign}}$ corresponds to the weighted path-length quasi-metric Sign-compatible defined in Example 3.5. In addition, when we say that $\boldsymbol{A}$ satisfies a certain structural property, e.g., ACC, we refer to the abstract domain A in the pair $\boldsymbol{A} = (A, \delta_A)$.

4 PARTIAL COMPLETENESS

Completeness is an ideal (and uncommon) situation that captures the precision of the abstract computation in approximating the dynamic property of interest. In program analysis, completeness is an *highly desirable* property that is extremely hard, if not even impossible, to achieve [Giacobazzi et al. 2000]. For this reason, instead of trying to reach completeness, we need to deal with *incompleteness* and therefore with imprecision. To this end the weaker notion of *local completeness* has been recently introduced by [Bruni et al. 2021] that requires completeness only with respect to specific inputs. An abstract interpretation $f^\sharp$ on the abstract domain $A \in \mathcal{A}(C)$ is locally complete for the function f on input $c \in C$ if $\alpha_A(f(c)) = f^\sharp(\alpha_A(c))$. It is clear that the *global* notion of completeness in [Cousot and Cousot 1979; Giacobazzi et al. 2000] implies local completeness, while the converse does not hold in general. This because local completeness is relative to a specific input $c \in C$, while completeness is relative to the whole concrete domain C.

We propose a further weakening of the property of local completeness that we call *partial (local) completeness* and that introduces a *limit* in the amount of imprecision that is allowed according to a fixed input. In order to simplify the notation, we introduce the notion of ε-*closeness* between abstract elements which specifies the boundaries of allowed imprecision in the abstract domain A w.r.t. δ_A.

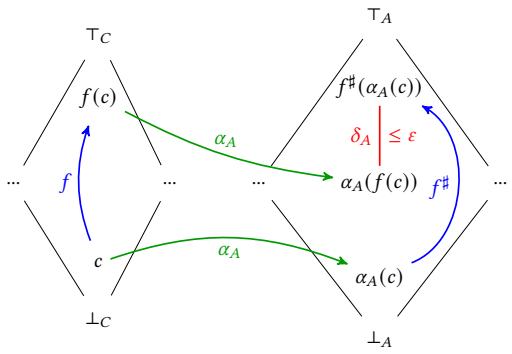

Fig. 2. The general idea of ε-partial completeness

*Definition 4.1 (**ε-Closeness w.r.t. δ_A**).* Let $\boldsymbol{A} \in \mathfrak{A}(C)$ be an abstract quasi-metric space and consider a constant $\varepsilon \in \mathbb{R}_{\geq 0}$. For all $a, b \in A$ such that $a \leq_A b$, we say that b is *ε-close* to a w.r.t. δ_A, denoted $a \approx^{\varepsilon}_{\delta_A} b$, if $\delta_A(a, b) \leq \varepsilon$. □

The concept of closeness encapsulates both the approximation and the relative error, guided by the quasi-metric δ_A. Therefore, two abstract states that are ε-close w.r.t. δ_A differ just for a maximum error quantified by ε.

*Definition 4.2 (**ε-Partial completeness**).* Let $f^\sharp : A \longrightarrow A$ be a correct approximation of $f : C \longrightarrow C$ on $A \in \mathcal{A}(C)$. We say that $f^\sharp$ is an *ε-partial complete* approximation of f on input $c \in C$ if:

$$\alpha_A(f(c)) \approx^{\varepsilon}_{\delta_A} f^\sharp(\alpha_A(c)).$$ □

The general idea of partial completeness is depicted in Fig. 2. Note that by setting $\varepsilon = 0$ we get local completeness, whereas by universal quantifying on $c \in C$ we get the standard (global) completeness. Furthermore, if A is complete for f then the bca f^α is also ε-partial complete for f on $\boldsymbol{A}$ for every $\varepsilon \in \mathbb{R}_{\geq 0}$, δ_A and $c \in C$. This means that, similarly to the completeness case, the possibility of defining an ε-partial complete approximation $f^\sharp$ of f on $\boldsymbol{A} \in \mathfrak{A}(C)$ with input c, only depends upon the concrete function f, the abstraction A, the quasi-metric δ_A, and the input c.

5 PROGRAMS, SEMANTICS, AND ABSTRACT SEMANTICS

5.1 The Syntax

We consider a language of *regular commands* Prog (as defined, e.g., in [Winskel 1993, Chapter 14, Exercise 14.4]) whose syntax is defined as:

$$\mathsf{Prog} \ni \mathsf{r} ::= \mathsf{e} \mid \mathsf{r}; \mathsf{r} \mid \mathsf{r} \oplus \mathsf{r} \mid \mathsf{r}^*$$

This language is general enough to cover deterministic imperative languages as well as other programming paradigms that include, e.g., nondeterministic and probabilistic computations, and equational systems such as Kleene algebras with tests [Kozen 1997]. The expressions $\mathsf{e} \in \mathsf{Exp}$ represent the basic transfer commands and can be instantiated with different kinds of instructions such as (nondeterministic or parallel) assignments, (Boolean) guards or assumptions, error generative primitives, and so on. The term $\mathsf{r}_1; \mathsf{r}_2$ represents sequential composition, the term $\mathsf{r}_1 \oplus \mathsf{r}_2$ represents a choice command that can behave as either r_1 or r_2, while the term r^* is the Kleene iteration of r where r can be executed 0 or any bounded number of times in a sequence. As an abbreviation, we write r^n for the sequence $\mathsf{r}; \ldots; \mathsf{r}$ of n sequential composition of r. For our purposes,

$$\begin{aligned} \mathsf{AExp} \ni \mathrm{a} &::= v \in \mathbb{Z} \mid x_i \in \mathit{Var} \mid \mathrm{a} + \mathrm{a} \mid \mathrm{a} - \mathrm{a} \mid \mathrm{a} * \mathrm{a} \\ \mathsf{BExp} \ni \mathrm{b} &::= \mathbf{t} \mid \mathbf{f} \mid \mathrm{a} = \mathrm{a} \mid \mathrm{a} > \mathrm{a} \mid \mathrm{b} \wedge \mathrm{b} \mid \mathrm{b} \vee \mathrm{b} \mid \neg \mathrm{b} \\ \mathsf{Exp} \ni \mathrm{e} &::= \mathbf{skip} \mid x_i := \mathrm{a} \mid \mathrm{b}? \end{aligned}$$

Fig. 3. Syntax of our deterministic imperative language

$$\begin{aligned} [\![\mathbf{skip}]\!]S &\triangleq S & [\![\mathrm{r}_1;\mathrm{r}_2]\!]S &\triangleq [\![\mathrm{r}_2]\!]([\![\mathrm{r}_1]\!]S) \\ [\![x_i := \mathrm{a}]\!]S &\triangleq \{s[x_i \mapsto (\!|\mathrm{a}|\!)s] \mid s \in S\} & [\![\mathrm{r}_1 \oplus \mathrm{r}_2]\!]S &\triangleq [\![\mathrm{r}_1]\!]S \cup [\![\mathrm{r}_2]\!]S \\ [\![\mathrm{b}?]\!]S &\triangleq \{s \in S \mid (\!|\mathrm{b}|\!)s = \mathbf{t}\} & [\![\mathrm{r}^*]\!]S &\triangleq \cup\{[\![r]\!]^n S \mid n \in \mathbb{N}\} \end{aligned}$$

Fig. 4. Collecting denotational semantics of Prog

we will consider standard basic transfer expressions used in deterministic while-programs with no-op instructions, assignments and Boolean guards, integer expressions and no runtime errors. The syntax of regular expressions Exp to make programs in Prog is shown in Fig. 3. A standard deterministic imperative language (similar to, e.g., the one presented in [Winskel 1993]) can be defined using guarded branching and loop commands as syntactic sugar (used in the examples):

$$\begin{aligned} \textbf{if } \mathrm{b} \textbf{ then } C_1 \textbf{ else } C_2 &\triangleq (\mathrm{b}?; C_1) \oplus (\neg \mathrm{b}?; C_2) \\ \textbf{while } \mathrm{b} \textbf{ do } C &\triangleq (\mathrm{b}?; C)^*; \neg \mathrm{b} \end{aligned}$$

5.2 The Concrete Semantics

We denote the finite set of variables occurring in some syntactic object c by $\mathit{vars}(c) \subseteq \mathit{Var}$. Var is assumed to be a denumerable set of variables indexed by positive integers: $\mathit{Var} \triangleq \{x_i \mid i \in \mathbb{N}_{>0}\}$. The index of variables is omitted in programs that have only one variable x. A program store is a partial function $s \in \mathbb{S} \triangleq \mathit{Var} \nrightarrow \mathbb{Z}$, which can be equivalently specified by a tuple of its defined values $\langle x_1 \mapsto v_1, \ldots, x_n \mapsto v_n \rangle$. The set of all stores is $\mathbb{S} \triangleq \bigcup_{n \in \mathbb{N}} \mathbb{Z}^n$. Single store update is written $s[x_i \mapsto v]$, and a set of stores $S \subseteq \mathbb{S}$ update is defined as: $S[x_i \mapsto v] \triangleq \{s[x_i \mapsto v] \mid s \in S\}$. Given a store function $s \in \mathbb{S}$ let $\mathit{vars}(s) \triangleq \{x_i \in \mathit{Var} \mid s(x_i)\!\downarrow\}$, while for $S \subseteq \mathbb{S}$ let $\mathit{vars}(S) \triangleq \bigcup_{s \in S} \mathit{vars}(s)$.

The semantics of arithmetic and Boolean expressions is respectively defined by the functions $(\!|\cdot|\!) : \mathsf{AExp} \times \mathbb{S} \to \mathbb{Z}$ and $(\!|\cdot|\!) : \mathsf{BExp} \times \mathbb{S} \to \{\mathbf{t}, \mathbf{f}\}$ whose definitions are straightforward and therefore omitted. The collecting (or strongest postcondition) semantics of arithmetic and Boolean expressions is respectively defined by the functions $[\![\mathrm{a}]\!] : \wp(\mathbb{S}) \to \wp(\mathbb{Z})$ and $[\![\mathrm{b}]\!] : \wp(\mathbb{S}) \to \wp(\mathbb{S})$ defined as: $[\![\mathrm{a}]\!]S \triangleq \{(\!|\mathrm{a}|\!)s \in \mathbb{Z} \mid s \in S\}$ and $[\![\mathrm{b}]\!]S \triangleq \{s \in S \mid (\!|\mathrm{b}|\!)s = \mathbf{t}\}$ so that $[\![\mathrm{b}]\!]S \subseteq S$ filters the stores of S making b true.

The collecting denotational program semantics is given by the function $[\![\mathrm{r}]\!] : \wp(\mathbb{S}) \to \wp(\mathbb{S})$ and it is defined in Fig. 4. This is the standard predicate transformer semantics (also called strongest postcondition semantics) since $[\![\mathrm{r}]\!]S \in \wp(\mathbb{S})$ turns out to be the strongest store predicate for the store precondition $S \in \wp(\mathbb{S})$. The terminology "collecting semantics" comes from the fact that for all $\mathrm{r} \in \mathsf{Prog}$, $[\![\mathrm{r}]\!] : \wp(\mathbb{S}) \to \wp(\mathbb{S})$ is an additive function on the complete lattice $\langle \wp(\mathbb{S}), \subseteq \rangle$, so that $[\![\mathrm{r}]\!]S = \cup_{s \in S} [\![\mathrm{r}]\!]\{s\}$ holds. Let $P \in \mathsf{Prog}$ be any program composed by regular expressions defined in Fig. 3. In this case $\lambda s \in \mathbb{S}.[\![P]\!]\{s\}$ represents the partial recursive function computed by P, where $[\![P]\!]\{s\} = \varnothing$ means non-termination of P on input s. Conversely, when P terminates on the input store s we have $[\![P]\!]\{s\} = \{s'\}$ for a suitable store s'. Note that, when P does not contain any assignment to a variable x_i, if $[\![P]\!]\{s\} = \{s'\}$ then $s'(x_i) = s(x_i)$. When $[\![P]\!]$ is applied to a singleton

$$\llbracket e \rrbracket^A S^\sharp \triangleq \alpha_A(\llbracket e \rrbracket \gamma_A(S^\sharp)) \qquad \llbracket \mathsf{r}_1 \oplus \mathsf{r}_2 \rrbracket^A S^\sharp \triangleq \llbracket \mathsf{r}_1 \rrbracket^A S^\sharp \sqcup_A \llbracket \mathsf{r}_2 \rrbracket^A S^\sharp$$
$$\llbracket \mathsf{r}_1; \mathsf{r}_2 \rrbracket^A S^\sharp \triangleq \llbracket \mathsf{r}_2 \rrbracket^A (\llbracket \mathsf{r}_1 \rrbracket^A S^\sharp) \qquad \llbracket \mathsf{r}^* \rrbracket^A S^\sharp \triangleq \sqcup_A \{ (\llbracket r \rrbracket^A)^n S^\sharp \mid n \in \mathbb{N} \}$$

Fig. 5. Abstract denotational program semantics of Prog

$\{s\} \in \wp(\mathbb{S})$, we use the simpler notation $\llbracket P \rrbracket s$ in place of $\llbracket P \rrbracket \{s\}$. We will often abuse notation and represent with $\llbracket P \rrbracket$ both the above mentioned collecting semantics (i.e., a total function from set of stores to set of stores) and the ordinary denotational semantics of P (i.e., a partial function $\lambda s. \llbracket P \rrbracket \{s\}$ from stores to stores where $\varnothing$ corresponds to non-termination).

The collecting semantics is typically used as reference semantics for designing static program analysis [Cousot and Cousot 1977; Miné 2017]. It is well known that not all elements in $\wp(\mathbb{S})$ are recursively enumerable (r.e. for short), hence the semantics of a program. Let $\wp^{\mathrm{re}}(\mathbb{S})$ be the set of all r.e. sets of stores while $\wp^{\mathrm{rec}}(\mathbb{S})$ be the set of all recursive sets of stores. We have that both $\wp^{\mathrm{re}}(\mathbb{S})$ and $\wp^{\mathrm{rec}}(\mathbb{S})$ are denumerable and $\wp^{\mathrm{rec}}(\mathbb{S}) \subsetneq \wp^{\mathrm{re}}(\mathbb{S}) \subsetneq \wp(\mathbb{S})$. Despite the definition of collecting semantics applies to any subset of $\mathbb{S}$, in the following we consider as inputs to semantic functions only sets of stores $S \in \wp(\mathbb{S})$ such that (i) S is r.e. and (ii) S predicates over a finite set of variables, namely $|vars(S)| < \omega$, as is always the case in abstract interpretation. This still allows any variable $x_i \in vars(S)$ to be assigned infinitely many different values by the stores in a set $S \in \wp^{\mathrm{re}}(\mathbb{S})$. Since any input set of stores must have a constructive computable way for building it and programs always manipulate a finite set of variables, we make the following assumption.

Assumption 2. *In the following, unless specified otherwise, we consider as inputs to semantic functions only sets of stores that are r.e. and that predicate over a finite set of variables.* [2]

5.3 The Abstract Semantics

An abstraction of stores is specified as an abstract domain $A \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$. The main source of imprecision in program analysis is due to the fact that the composition of two bca may not be the bca of the composition of the corresponding concrete functions. We define the abstract semantics for an abstract domain $A \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ by structural induction on the syntax of Prog starting from the bca of the basic commands $\mathsf{e} \in \mathsf{Exp}$ and then composing them.

This is formalized by the abstract semantics $\llbracket \mathsf{r} \rrbracket^A : A \to A$ shown in Fig. 5, which provides a correct approximation of the concrete collecting semantics. Recall that in a GI, the abstract join $\sqcup_A$ is always the bca of the concrete join $\cup$. Let us comment further the abstract semantics in Fig. 5:

(i) The abstract semantics of basic transfer expressions $\mathsf{e} \in \mathsf{Exp}$ is defined as the bca on A. We assume that the bca of basic transfer commands is computable in Prog. Although in program analysis this is a recurrent assumption, this choice can be easily weakened by using any approximation of basic transfer expressions in the abstract semantics, at the price of further loss of precision. This will result in larger estimated error, without affecting the results later presented in Section 6.

(ii) $\llbracket \mathsf{r}_1; \mathsf{r}_2 \rrbracket^A$ is compositionally defined and, in general, does not coincide with the bca of $\llbracket \mathsf{r}_1; \mathsf{r}_2 \rrbracket$, while the composition of complete functions is always complete as recalled in Section 2.2.

[2] Richer languages manipulating an unbound number of variables, e.g., by recursion, can be considered at the price of complicating the model and replacing variable finiteness with abstract domains defined as functions from natural numbers $n \in \mathbb{N}$ to Galois connections on a concrete domain with n variables (e.g., see [Bruni et al. 2020; Venet 1996]).

(iii) $[\![\mathsf{r}_1 \oplus \mathsf{r}_2]\!]^A$ relies on the abstract lub $\sqcup_A$ which is the complete (therefore best correct) approximation of the concrete lub on $\wp^{\mathrm{re}}(\mathbb{S})$. Thus, completeness is preserved: If $[\![\mathsf{r}_1]\!]^A$ and $[\![\mathsf{r}_2]\!]^A$ are complete abstractions of $[\![\mathsf{r}_1]\!]$ and $[\![\mathsf{r}_2]\!]$, then $[\![\mathsf{r}_1 \oplus \mathsf{r}_2]\!]^A$ is a complete abstraction of $[\![\mathsf{r}_1 \oplus \mathsf{r}_2]\!]$. A similar reasoning can be applied to $[\![\mathsf{r}^*]\!]^A$.

(iv) It is easy to check (by structural induction on r) that the abstract semantics in Fig. 5 is monotonic: $\forall \mathsf{r} \in \mathsf{Prog}$, if $S_1^\sharp \leq_A S_2^\sharp$ then $[\![\mathsf{r}]\!]^A S_1^\sharp \leq_A [\![\mathsf{r}]\!]^A S_2^\sharp$, and correct: $\forall \mathsf{r} \in \mathsf{Prog}$ and $S^\sharp \in A$: $[\![\mathsf{r}]\!]\gamma_A(S^\sharp) \subseteq \gamma_A([\![\mathsf{r}]\!]^A S^\sharp)$.

In general, the abstract semantics in Fig. 5 does not define a static program analysis since it may not terminate. Termination can be enforced by using a widening operator $\nabla_A : A \times A \longrightarrow A$ [Cousot and Cousot 1977]. From now on, we will assume the following.

Assumption 3. *Given an abstract domain $A \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$, we only consider static program analysis $[\![\mathsf{r}]\!]^A : A \longrightarrow A$ that terminates in a finite number of steps on all inputs $S^\sharp \in A$.*

It is worth noting that the soundness property guaranteed by the abstract interpretation framework and Assumption 3 imply that for all programs $P \in \mathsf{Prog}$ and set of stores $S \in \wp^{\mathrm{re}}(\mathbb{S})$: $[\![P]\!]S \neq \varnothing \implies [\![P]\!]^A \alpha_A(S) \neq \bot_A$. This is essential in order for the abstract interpreter to soundly approximate all possible concrete executions. This condition is trivially satisfied by abstract domains with a finite number of elements or that are ACC. Otherwise a widening operation extrapolating unstable bounds is necessary to force termination [Cousot and Cousot 1977].

It is worth remarking that the abstract semantics defined in Fig. 5 is a correct approximation of the concrete collecting semantics defined in Fig. 4, *but*, in general, it is not its bca. Moreover, for all $A \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$, $P \in \mathsf{Prog}$ and $S \in \wp^{\mathrm{re}}(\mathbb{S})$, the following inequalities hold:

$$\alpha_A([\![P]\!]S) \;\leq_A\; \alpha_A([\![P]\!]\gamma_A(\alpha_A(S))) \;\leq_A\; [\![P]\!]^A \alpha_A(S).$$

5.4 Recursive Abstract Domains

Static program analysis always relies upon recursive, namely decidable, abstractions. This is because the analysis of programs requires decidable answers to undecidable questions regarding the dynamic behavior of programs. The notion of recursive abstract interpretation has been studied in [Cousot et al. 2018]. In the following, we formalize this notion for a generic GC based abstract interpretation.

*Definition 5.1 (**Recursive abstract domains of stores**).* An abstract domain $A \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ is *recursive* if:

(i) For every store $s \in \mathbb{S}$, $\alpha_A(\{s\})$ is computable;

(ii) For every $S^\sharp \in A$, the set of stores $\gamma_A(S^\sharp)$ is recursive, namely, $\gamma_A(S^\sharp) \in \wp^{\mathrm{rec}}(\mathbb{S})$;

(iii) The partial order relation $\leq_A$ is decidable. □

Note that (iii) implies that the equivalence relation between abstract elements is decidable. Indeed, by the antisymmetry property satisfied by all partial order relations, if $a_1 \leq_A a_2$ and $a_2 \leq_A a_1$ then $a_1 = a_2$. The intuition is that the elements of any recursive abstract domain of stores represent recursive sets of stores, therefore, they are isomorphic to a suitable subset of $\wp^{\mathrm{rec}}(\mathbb{S})$. Besides recursive abstract domains, we also consider trivial abstractions of stores.

*Definition 5.2 (**Trivial abstractions of stores**).* The *trivial abstractions* in $\mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ are:

- $id \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ denoting the (least) identical abstraction: $\forall S \in \wp^{\mathrm{re}}(\mathbb{S})$, $\alpha_{id}(S) = S = \gamma_{id}(S)$
- $\top^{\mathbb{S}} \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ denoting the greatest abstraction: $\forall S \in \wp^{\mathrm{re}}(\mathbb{S})$, $\alpha_{\top^{\mathbb{S}}}(S) = \mathbb{S}$. □

In the following, we only consider program analyzers as specified by GI-based abstract interpreters (Assumption 1) over strict ($\gamma_A(\bot_A) = \varnothing$) and recursive or trivial abstract domains. This assumption is often left implicit in program analysis, we emphasize it through the following

Assumption 4. *We consider abstract domains in $\mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ that are either strict recursive or trivial.*

As a consequence of Assumption 4, each non-trivial abstract domain is assumed to be recursive and strict. Note that $\top^{\mathbb{S}}$ is a recursive abstract domain but it is not strict, while *id* is strict but it is not recursive because $\gamma_{id}(S)$ might not be recursive. It is important to note that the GI-based abstract interpretations and the strictness conditions allow us to consider non-trivial abstract domains of stores that exactly represent the empty set $\varnothing$, i.e., non-termination.

Proposition 5.3. *For every non-trivial $A \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ we have $\gamma_A(\alpha_A(\varnothing)) = \varnothing$.*

The following definition recalls the notion of *completeness class of programs*, firstly introduced in [Giacobazzi et al. 2015], as the class of all programs for which a given abstract domain is complete, and extends this notion to the case of locally complete abstract interpretations.

*Definition 5.4 (**Completeness classes**).* Given $A \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$, its *completeness class* and *local completeness class* $\mathbb{C}(A) \subseteq \mathsf{Prog}$ and $\mathbb{C}(A, S) \subseteq \mathsf{Prog}$ are defined for $S \in \wp^{\mathrm{re}}(\mathbb{S})$ as follows:

$$\begin{array}{rcl} \mathbb{C}(A) & \triangleq & \{P \in \mathsf{Prog} \mid \forall S \in \wp^{\mathrm{re}}(\mathbb{S}).\ \alpha_A(\llbracket P \rrbracket S) = \llbracket P \rrbracket^A \alpha_A(S)\} \\ \mathbb{C}(A, S) & \triangleq & \{P \in \mathsf{Prog} \mid \alpha_A(\llbracket P \rrbracket S) = \llbracket P \rrbracket^A \alpha_A(S)\}. \end{array}$$

□

Giacobazzi et al. [2015] proved that $\mathbb{C}(A)$ is infinite by a straightforward padding argument, since **skip** $\in \mathbb{C}(A)$ for any A and sequential composition of complete commands is still complete. Moreover $\mathbb{C}(A)$ is non-extensional because there always exist programs P and Q such that: P is complete for A, $\llbracket P \rrbracket = \llbracket Q \rrbracket$, and Q is not complete for A (e.g., see [Bruni et al. 2020; Giacobazzi 2008; Laviron and Logozzo 2009] for examples of program transformations designed to alter the abstract semantics, either by improving precision as in program analysis by hints, or by worsening precision as in code obfuscation).

6 CLASSES OF PARTIAL COMPLETE PROGRAMS

Standard completeness in program analysis, e.g., see [Cousot and Cousot 1977, 1979; Giacobazzi et al. 2000], means that no false alarms are returned by analyzing the program with an abstract interpreter on any possible input state. Local completeness, instead, corresponds to have no false alarms for some specific input states. 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.

6.1 Partial Complete Programs

In this section we introduce the class of programs for which the abstract interpreter returns an ε-partially complete analysis, i.e., the abstract interpreter under the given input, never outputs an abstract value whose quasi-metric distance from the abstraction of the concrete one exceeds ε.

*Definition 6.1 (**Decidable quasi-metrics A-compatible**).* Let $A \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$, we say that $\delta_A : A \times A \to \mathbb{Q}^{\infty}_{\geq 0} \cup \{\bot\}$ is a *decidable* quasi-metric A-compatible if it satisfies the three axioms of Definition 3.1 and the following axiom:

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

Decidability of the compatible quasi-metric is important in order to be able to decide the closeness of the abstract semantics and the abstraction of the concrete one. Note that, by considering only

recursive or trivial abstract domains in $\mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ (Assumption 4), any $A \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ contains a finite or countably infinite number of r.e. sets. This means that $|A| \leq \omega = |\mathbb{N}| = |\mathbb{Q}_{\geq 0}|$. Therefore, it is reasonable to consider the range of a decidable quasi-metric A-compatible function over $\mathbb{Q}_{\geq 0}$ instead of the non-negative real numbers as in the original definition. It is easy to note that the weighted path-length quasi-metrics defined in Examples 3.5-3.7 for, respectively, Sign, $\mathsf{P} \sqcap \mathsf{S}$ and Int, are all decidable. An abstract domain $A \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ endowed with a decidable quasi-metric A-compatible δ_A, forms an abstract quasi-metric space of stores $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$. δ_A filters the imprecision injected by the abstract analysis over A, and it measures only the inaccuracy that we want to track and hence limit. In the following, otherwise stated, we consider only decidable quasi-metrics. We can now define when a program is ε-partial complete for an abstract quasi-metric space of stores and a set of inputs.

*Definition 6.2 (**ε-Partially complete programs**).* Consider a program $P \in \mathsf{Prog}$, a constant bound $\varepsilon \in \mathbb{Q}_{\geq 0}$, a non-empty set of input stores $S \in \wp^{\mathrm{re}}(\mathbb{S})$ and an abstract quasi-metric space $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$. $\boldsymbol{A}$ is *ε-partially complete* for P in S if the following holds:

$$\alpha_A(\llbracket P \rrbracket S) \approx^{\varepsilon}_{\delta_A} \llbracket P \rrbracket^A \alpha_A(S). \qquad \square$$

We have all the ingredients to introduce the notion of ε-partial completeness class as a mapping from abstract quasi-metric spaces of stores $\mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$, a constant $\varepsilon \in \mathbb{Q}_{\geq 0}$ and a set of inputs, to the set of all programs for which the abstraction is ε-partially complete. Formally:

$$\mathbb{C} : \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S})) \times \mathbb{Q}_{\geq 0} \times \wp(\wp^{\mathrm{re}}(\mathbb{S})) \longrightarrow \wp(\mathsf{Prog}).$$

*Definition 6.3 (**ε-Partial completeness class**).* The *partial completeness class* of an abstract quasi-metric space $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$, a constant $\varepsilon \in \mathbb{Q}_{\geq 0}$ and a non-empty family of inputs $\mathfrak{F} \subseteq \wp^{\mathrm{re}}(\mathbb{S})$, denoted $\mathbb{C}(\boldsymbol{A}, \varepsilon, \mathfrak{F}) \subseteq \mathsf{Prog}$, is defined as:

$$\mathbb{C}(\boldsymbol{A}, \varepsilon, \mathfrak{F}) \triangleq \{P \in \mathsf{Prog} \mid \forall S \in \mathfrak{F}.\ \alpha_A(\llbracket P \rrbracket S) \approx^{\varepsilon}_{\delta_A} \llbracket P \rrbracket^A \alpha_A(S)\}. \qquad \square$$

From now on, we will focus on classes of partial completeness with a single input that will be denoted with S instead of $\{S\}$. Because recursive and trivial abstract domains of stores are either finite or countably infinite, we can define $\mathbb{C}(\boldsymbol{A}, \infty, S) \triangleq \cup_{\varepsilon \in \mathbb{Q}_{\geq 0}} \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$. If $P \in \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ then the abstraction of the collecting semantics and the abstract semantics, respectively on S and $\alpha_A(S)$, are ε-close w.r.t. δ_A. Roughly, a partial completeness class $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ is the set of all programs whose abstract interpretation on abstraction A and input S *can* produce false alarms, *but* the error introduced is not greater than ε according to δ_A. The following proposition is a straightforward result from Definition 5.4 of completeness and Definition 6.3 of partial completeness classes.

Proposition 6.4. *For every $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$, the following hold:*

(i) $\mathbb{C}(A) \subseteq \mathbb{C}(A, S) = \mathbb{C}(\boldsymbol{A}, 0, S) \subseteq \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$

(ii) $\forall \varepsilon, \xi \in \mathbb{Q}_{\geq 0}.\ \varepsilon \leq \xi \implies \mathbb{C}(\boldsymbol{A}, \varepsilon, S) \subseteq \mathbb{C}(\boldsymbol{A}, \xi, S)$

(iii) $\forall \mathfrak{F}, \mathfrak{F}' \subseteq \wp^{\mathrm{re}}(\mathbb{S}).\ \mathfrak{F} \subseteq \mathfrak{F}' \implies \mathbb{C}(\boldsymbol{A}, \varepsilon, \mathfrak{F}') \subseteq \mathbb{C}(\boldsymbol{A}, \varepsilon, \mathfrak{F})$

(iv) $\forall \mathfrak{F}, \mathfrak{F}' \subseteq \wp^{\mathrm{re}}(\mathbb{S}).\ \mathbb{C}(\boldsymbol{A}, \varepsilon, \mathfrak{F} \cup \mathfrak{F}') = \mathbb{C}(\boldsymbol{A}, \varepsilon, \mathfrak{F}) \cap \mathbb{C}(\boldsymbol{A}, \varepsilon, \mathfrak{F}')$

(v) $\mathbb{C}(\boldsymbol{A}, \infty, S) = \mathsf{Prog}$

Forcing a zero closeness means requiring no false alarms on input S, i.e., local completeness, while each complete abstraction is also partially complete for every S. Moreover, weakening closeness increases monotonically the set of partially complete programs, while by increasing the considered set of inputs we might restrict the partial completeness class. In the following, when

we omit the set of inputs from the partial completeness class, we intend as for every S, that is $\mathbb{C}(\boldsymbol{A}, \varepsilon) \triangleq \bigcap_{S \in \wp^{\mathrm{re}}(\mathbb{S})} \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$.

Example 6.5. Consider the abstract quasi-metric space $(\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}) \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{Z}))$. Let us consider the class of 0-partial complete programs w.r.t. $(\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}})$, i.e., $\mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 0, S)$ which does not admit any imprecision on $S \in \wp^{\mathrm{re}}(\mathbb{S})$, and the class $\mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 1, S)$, which admits only one spurious element on the interval output. We define $P_{\mathrm{abs}} \in \mathsf{Prog}$ as:

$$\begin{aligned} \mathrm{P}_{\mathrm{abs}} &\triangleq \textbf{if } x \geq 0 \textbf{ then skip else } x := x * (-1) \\ &= (x \geq 0;\ \textbf{skip}) \oplus (x < 0?;\ x := x * (-1)) \end{aligned}$$

which, computes the absolute value of integer variables. Consider the input sets $S_1 = \{0, 2, 5\}$ and $S_2 = \{-1, 3, 7\}$. We have the following concrete and abstract evaluations:

$$\alpha_{\mathsf{Int}}(\llbracket P_{\mathrm{abs}} \rrbracket S_1) = \alpha_{\mathsf{Int}}(\llbracket \textbf{skip} \rrbracket S_1) = \alpha_{\mathsf{Int}}(S_1) = [0, 5]$$
$$\llbracket P_{\mathrm{abs}} \rrbracket^{\mathsf{Int}} \alpha_{\mathsf{Int}}(S_1) = \llbracket P_{\mathrm{abs}} \rrbracket^{\mathsf{Int}} [0, 5] = \llbracket \textbf{skip} \rrbracket^{\mathsf{Int}} [0, 5] = [0, 5]$$

$$\alpha_{\mathsf{Int}}(\llbracket P_{\mathrm{abs}} \rrbracket S_2) = \alpha_{\mathsf{Int}}(\llbracket \textbf{skip} \rrbracket \{3, 7\} \cup \llbracket x := x * (-1) \rrbracket \{-1\}) = \alpha_{\mathsf{Int}}(\{1, 3, 7\}) = [1, 7]$$
$$\llbracket P_{\mathrm{abs}} \rrbracket^{\mathsf{Int}} \alpha_{\mathsf{Int}}(S_2) = \llbracket \textbf{skip} \rrbracket^{\mathsf{Int}} [0, 7] \sqcup_{\mathsf{Int}} \llbracket x := x * (-1) \rrbracket^{\mathsf{Int}} [-1, -1] = [0, 7] \sqcup_{\mathsf{Int}} [1, 1] = [0, 7]$$

Then, clearly $P_{\mathrm{abs}} \in \mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 0, S_1)$, $P_{\mathrm{abs}} \notin \mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 0, S_2)$, while $P_{\mathrm{abs}} \in \mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 1, S_2)$. ■

Similarly to the completeness case [Giacobazzi et al. 2015], for every ε, the ε-partial completeness class is infinite and non-extensional. It is infinite because for all $\varepsilon \in \mathbb{Q}_{\geq 0}$ and $S \in \wp^{\mathrm{re}}(\mathbb{S})$, by Proposition 6.4 $\mathbb{C}(A) \subseteq \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ and $\mathbb{C}(A)$ is infinite. Therefore $|\mathbb{C}(\boldsymbol{A}, \varepsilon, S)| = \omega$. It is also non-extensional because there always exist programs P and Q such that: P is partially complete for A, $\llbracket P \rrbracket = \llbracket Q \rrbracket$, and Q is not partially complete for A, as shown by the following example.

Example 6.6. Consider the abstract domain $\mathsf{P} \sqcap \mathsf{S}$, shown in Fig. 1. Let us consider the weighted path-length quasi-metric $\mathsf{P}\sqcap\mathsf{S}$-compatible $\delta^{\mathfrak{w}}_{\mathsf{P}\sqcap\mathsf{S}}$, and the partial completeness class $\mathbb{C}((\mathsf{P}\sqcap\mathsf{S}, \delta^{\mathfrak{w}}_{\mathsf{P}\sqcap\mathsf{S}}), 2)$. Consider the following two programs:

$$P \triangleq x := 0; x := x + 1; x := x - 1$$
$$Q \triangleq x := 2; x := x - 1; \textbf{if } x \geq 1 \textbf{ then } x := x - 1 \textbf{ else skip}.$$

It is easy to note that $\llbracket P \rrbracket = \llbracket Q \rrbracket = \lambda S \in \wp^{\mathrm{re}}(\mathbb{Z}).\{\langle x \mapsto 0 \rangle\}$ and $\alpha_{\mathsf{P}\sqcap\mathsf{S}}(\llbracket P \rrbracket S) = \alpha_{\mathsf{P}\sqcap\mathsf{S}}(\llbracket Q \rrbracket S) = \langle x \mapsto 0 \rangle$. It is easy to observe that

$$\llbracket P \rrbracket^{\mathsf{P}\sqcap\mathsf{S}} \alpha_{\mathsf{P}\sqcap\mathsf{S}}(\{\langle x \mapsto v \rangle \mid v \in \mathbb{Z}\}) = \langle x \mapsto \mathit{even} \rangle$$
$$\llbracket Q \rrbracket^{\mathsf{P}\sqcap\mathsf{S}} \alpha_{\mathsf{P}\sqcap\mathsf{S}}(\{\langle x \mapsto v \rangle \mid v \in \mathbb{Z}\}) = \langle x \mapsto \mathbb{Z} \rangle.$$

Consequently, $P \in \mathbb{C}((\mathsf{P} \sqcap \mathsf{S}, \delta^{\mathfrak{w}}_{\mathsf{P}\sqcap\mathsf{S}}), 2)$ while $Q \notin \mathbb{C}((\mathsf{P} \sqcap \mathsf{S}, \delta^{\mathfrak{w}}_{\mathsf{P}\sqcap\mathsf{S}}), 2)$. Note that, even though P is partially complete w.r.t. $\mathbb{C}((\mathsf{P} \sqcap \mathsf{S}, \delta^{\mathfrak{w}}_{\mathsf{P}\sqcap\mathsf{S}}), 2)$, both P and Q are not complete for $\mathsf{P} \sqcap \mathsf{S}$, i.e., $P, Q \notin \mathbb{C}(\mathsf{P} \sqcap \mathsf{S})$. ■

6.2 Recursive Properties of Partial Complete Programs

It is well known that for trivial abstractions the corresponding completeness class turns out to be the whole set of programs [Giacobazzi et al. 2000]. Moreover, for all non-trivial abstractions in $\mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$, its completeness class is strictly contained in Prog [Giacobazzi et al. 2015]. More has been recently proved along this direction: a completeness class is an index set of partial recursive function if and only if the abstraction is trivial [Bruni et al. 2020].

In this section, we study the counterpart of these results for the case of partial completeness. We first consider the simplest case of abstract quasi-metric spaces of stores $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ satisfying the property of having a limited imprecision, i.e., complete lattices A such that the quasi-metric δ_A is bounded. 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. We then consider the more general setting of abstract interpretations over abstract domains where an unlimited imprecision can always be produced by any terminating program analysis by abstract interpretation, e.g., employing widening operations to enforce termination [Cousot and Cousot 1977, 1992b].

*Definition 6.7 (**Abstract quasi-metric spaces with limited imprecision**).* An abstract quasi-metric space of stores $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ has *imprecision limited* by $\varepsilon \in \mathbb{Q}_{\geq 0}$ if for each $a \in A$ we have $\delta_A(\bot_A, a) \leq \varepsilon$. □

The following result is immediate and helps us to understand the relation between abstract domains with limited imprecision and the class of partial completeness properties of programs.

Proposition 6.8. *If $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ has limited imprecision, then we have*

$$\forall S \in \wp^{\mathrm{re}}(\mathbb{S}).\ \exists \varepsilon \in \mathbb{Q}_{\geq 0}.\ \mathbb{C}(\boldsymbol{A}, \varepsilon, S) = \mathsf{Prog}\,.$$

Proof. By definition of abstract domain with limited imprecision, there exists $\varepsilon \in \mathbb{Q}_{\geq 0}$ such that if $a \in A$ then $\delta_A(\bot_A, a) \leq \varepsilon$. Consider $n \in \mathbb{Q}_{\geq 0}$ such that $n \geq \varepsilon$. Then for any S we have $\mathsf{Prog} = \mathbb{C}(\boldsymbol{A}, \infty) = \cup_{m \leq n} \mathbb{C}(\boldsymbol{A}, m) \subseteq \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$. □

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. This corresponds to allow the largest possible imprecision, viz., amount of incompleteness, making the condition of being complete (viz., precise) vacuous.

Example 6.9. Consider the abstract domain Sign with $\delta^{\mathfrak{w}}_{\mathsf{Sign}}$ as Sign-compatible quasi-metric. $(\mathsf{Sign}, \delta^{\mathfrak{w}}_{\mathsf{Sign}})$ has clearly limited imprecision. Indeed for all $n \geq 3$, $\mathbb{C}((\mathsf{Sign}, \delta^{\mathfrak{w}}_{\mathsf{Sign}}), n) = \mathsf{Prog}$. ■

The case of abstract domains with unlimited imprecision is less straightforward and reflects precisely into the theory of partial completeness. An abstract quasi-metric space of stores $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ has *unlimited imprecision* when: $\forall \varepsilon \in \mathbb{Q}_{\geq 0},\ \exists a \in A.\ \delta_A(\bot_A, a) > \varepsilon$. Clearly, the unlimited imprecision property can only be satisfied by abstract quasi-metric space of stores with a (countably) infinite number of elements.

Proposition 6.10. *Let $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ be an abstract quasi-metric space of stores with unlimited imprecision. Then $|A| = \omega$.*

Next theorem generalizes a result in [Giacobazzi et al. 2015] to the case of classes of partial complete programs.

Theorem 6.11. *Let $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ be any abstract quasi-metric space of stores with unlimited imprecision δ_A and $S \in \wp^{\mathrm{re}}(\mathbb{S})$. Then:*

$$\exists \varepsilon \in \mathbb{Q}_{\geq 0}.\ \mathbb{C}(\boldsymbol{A}, \varepsilon, S) = \mathsf{Prog} \iff A = \mathit{id}.$$

Proof. ($\Leftarrow$) is straightforward because when $A = \mathit{id}$ then $\mathbb{C}(\boldsymbol{A}, 0) = \mathbb{C}(A) = \mathsf{Prog}$. Therefore, by Proposition 6.4, we can conclude $\mathbb{C}(\boldsymbol{A}, \varepsilon, S) = \mathsf{Prog}$.
($\Rightarrow$) By contradiction, we assume that:

(1) $A \neq \mathit{id}$ and, by Assumption 4, A is a strict abstract domain, i.e., with abstraction and concretization functions, respectively α_A and γ_A, such that $\gamma_A(\alpha_A(\varnothing)) = \varnothing$;

(2) $\boldsymbol{A}$ has unlimited imprecision, i.e., $\forall \varepsilon \in \mathbb{Q}_{\geq 0}.\ \exists a \in A.\ \delta_A(\bot_A, a) > \varepsilon$;

(3) there exists $\xi \geq 0$ such that $\mathbb{C}(\boldsymbol{A}, \xi, S) = \mathsf{Prog}$.

By (2) and (3) we have that there exists $a \in A$ such that $\bot_A \neq a$, otherwise if $\bot_A = a$ then $\delta_A(\bot_A, a) = 0$, while we have $\delta_A(\bot_A, a) > \xi$ with $\xi \in \mathbb{Q}_{\geq 0}$. Hence by (1), $\varnothing \subset \gamma_A(a)$ (recall that in any GI γ_A is injective).

We are now in the position of building a program that does not belong to $\mathbb{C}(\boldsymbol{A}, \xi, S)$, therefore leading to an absurd. Consider a non-terminating program P_w such that for any store $s \in \mathbb{S}$, $[\![P_w]\!]s = \varnothing$, while the abstract interpreter is incomplete on it, i.e., for any non-empty $X \subseteq \mathbb{S}$: $[\![P_w]\!]^A \alpha_A(X) \neq \bot_A$. Define $P_a \in \mathsf{Prog}$ as a program such that for any store $s \in \mathbb{S}$ we have that $[\![P_a]\!]s \in \gamma_A(a)$ and in particular $[\![P_a]\!]^A([\![P_w]\!]^A \alpha_A(S)) = a$. Turing completeness of Prog ensures that we can compute any finite store in $\gamma_A(a)$. Moreover, being $\gamma_A(a)$ and $\gamma_A(\alpha_A(S))$ both recursive sets, and $[\![\cdot]\!]^A$ a program, then we can build P_a in Prog as above.

We prove that if

$$P \triangleq P_w; P_a$$

then $\delta_A(\alpha_A([\![P]\!]S), [\![P]\!]^A \alpha_A(S)) > \xi$ hence $P \notin \mathbb{C}(\boldsymbol{A}, \xi, S)$. It is clear that

$$[\![P]\!]S = [\![P_a]\!]([\![P_w]\!]S) = [\![P_a]\!]\varnothing = \varnothing$$

and therefore $\alpha_A([\![P]\!]S) = \bot_A$. Analogously, with the abstract semantics we have:

$$[\![P]\!]^A \alpha_A(S) = [\![P_a]\!]^A([\![P_w]\!]^A \alpha_A(S)) = a$$

Therefore: $\delta_A(\alpha_A([\![P]\!]S), [\![P]\!]^A \alpha_A(S)) = \delta_A(\bot_A, a) > \xi$, hence, $P \notin \mathbb{C}(\boldsymbol{A}, \xi, S)$ which contradicts assumption (3). We can conclude that A must be the identical abstraction of stores. □

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 $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ with input S, is the complement set of $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$, formally: $\overline{\mathbb{C}(\boldsymbol{A}, \varepsilon, S)} = \{P \in \mathsf{Prog} \mid \alpha_A([\![P]\!]S) \not\approx^{\varepsilon}_{\delta_A} [\![P]\!]^A \alpha_A(S)\}$.

Corollary 6.12. *If $\boldsymbol{A}$ is a non-trivial abstract quasi-metric space with unlimited imprecision, then for all $\varepsilon \in \mathbb{Q}_{\geq 0}$, $|\overline{\mathbb{C}(\boldsymbol{A}, \varepsilon, S)}| = \omega$.*

This means 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.

Example 6.13. Consider the abstract domain of intervals $\mathsf{Int} \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{Z}))$, shown in Example 2.2, endowed with a quasi-metric Int-compatible δ_{Int}, such that δ_{Int} has unlimited imprecision. Recall that Int is not ACC. This means that there are infinite strictly ascending chains, such as for instance $[0,1] \leq_{\mathsf{Int}} [0,2] \leq_{\mathsf{Int}} \cdots \leq_{\mathsf{Int}} [0,n], \ldots$. Hence, a proper widening operator is required in order to enforce convergence of the abstract Kleene iterates of the abstract interpreter. The standard interval widening consists in replacing any unstable upper bound with $+\infty$ and any unstable lower bound with $-\infty$. Let us define in Fig. 6 the "delayed" widening $\nabla^i_{\mathsf{Int}} : \mathsf{Int} \times \mathsf{Int} \to \mathsf{Int}$, where $\#_{iter}$ indicates the current number of iteration in the loop. That is, ∇^i_{Int} does not immediately abort unstable computations by pushing to infinity, but it delays its application after i iterations. This is particularly useful when the first few iterates of the loop differ from the following ones, and

$$[a,b]\ \nabla^{i}_{\mathsf{Int}}\ [c,d] \triangleq \left[\begin{cases} a & \text{if } a \le c \\ c & \text{if } c < a \text{ and } \#_{iter} \le i \\ -\infty & \text{otherwise} \end{cases}, \begin{cases} b & \text{if } b \ge d \\ d & \text{if } d > b \text{ and } \#_{iter} \le i \\ +\infty & \text{otherwise} \end{cases}\right]$$

Fig. 6. The widening operator in Example 6.13

it is always a good idea to start extrapolating only after having accumulated a few iterations. In this way, the widening can make a more educated guess about the loop behavior. After a finite, fixed number of i iterations, we revert to widening so that termination of the abstract interpreter is preserved. Consider the following program:

$$P_n \triangleq x := n; \textbf{while } x > 1 \textbf{ do } x := x - (n-1)$$

where $n \in \mathbb{N}_{\ge 1}$ is a constant and the expressions n and $(n-1)$ are assumed to be resolved before giving the program as input to the concrete and abstract interpreter (e.g., by running a specializer on the code like the pre-processor for the C language which replaces all the occurrences of symbols n with the corresponding constant number). Clearly, for all input stores the while-loop of P_n terminates after one iteration if the condition $x > 1$ is satisfied. Indeed, for all $n \ge 1$ and $S \in \wp^{\mathrm{re}}(\mathbb{Z})$, we have $[\![P_n]\!]S = \{\langle x \mapsto 1\rangle\}$, which corresponds to $\alpha_{\mathsf{Int}}([\![P_n]\!]S) = \alpha_{\mathsf{Int}}(\{\langle x \mapsto 1\rangle\}) = [1,1]$. This means that all P_n are extensionally equivalent. Consider now the abstract denotational semantics for Int abstract domain of stores, where we replace the abstract lub $\sqcup_{\mathsf{Int}}$ in the loop head with the widening $\nabla^{2}_{\mathsf{Int}}$ which forces the convergence after two iterations. If $n = 1$ we get trivially $\langle x \mapsto [1,1]\rangle$. For $n \ge 2$, at the first iteration of the while-loop we have:

$$[\![x > 1]\!]^{\mathsf{Int}}\langle x \mapsto [n,n]\rangle = \langle x \mapsto [n,n]\rangle$$
$$[\![x := x - (n-1)]\!]^{\mathsf{Int}}\langle x \mapsto [n,n]\rangle = \langle x \mapsto [1,1]\rangle$$
$$\langle x \mapsto [n,n]\rangle\ \nabla^{2}_{\mathsf{Int}}\ \langle x \mapsto [1,1]\rangle = \langle x \mapsto [1,n]\rangle$$

while for the second iteration:

$$[\![x > 1]\!]^{\mathsf{Int}}\langle x \mapsto [1,n]\rangle = \langle x \mapsto [2,n]\rangle$$
$$[\![x := x - (n-1)]\!]^{\mathsf{Int}}\langle x \mapsto [2,n]\rangle = \langle x \mapsto [(3-n),1]\rangle$$
$$\langle x \mapsto [1,n]\rangle\ \nabla^{2}_{\mathsf{Int}}\ \langle x \mapsto [(3-n),1]\rangle = \langle x \mapsto [(3-n),n]\rangle.$$

Finally, at the third iteration we get a fixpoint. The final abstract output of the abstract semantics is given by the lub between

$$[\![x \le 1]\!]^{\mathsf{Int}}\langle x \mapsto [1,n]\rangle \sqcup_{\mathsf{Int}} [\![x \le 1]\!]^{\mathsf{Int}}\langle x \mapsto [(3-n),n]\rangle$$

which is $\langle x \mapsto [(3-n),1]\rangle$. Therefore, the abstract semantics of P_n for all inputs $S \in \wp^{\mathrm{re}}(\mathbb{Z})$ is

$$[\![P_1]\!]^{\mathsf{Int}}\alpha_{\mathsf{Int}}(S) = \langle x \mapsto [1,1]\rangle \qquad [\![P_{n\ge 2}]\!]^{\mathsf{Int}}\alpha_{\mathsf{Int}}(S) = \langle x \mapsto [(3-n),1]\rangle.$$

Note that $\{P_n\}_{n\in\{1,2\}} \subset \mathbb{C}(\mathsf{Int})$ while $\{P_n\}_{n>2} \cap \mathbb{C}(\mathsf{Int}) = \varnothing$. That is, all programs in $\{P_n\}_{n>2}$ are incomplete for Int. Moreover, we can arbitrary worsen the result of our static analysis on P_n by selecting a larger constant n. This implies that, since δ_{Int} has unlimited imprecision by assumption, if $P_n \in \mathbb{C}((\mathsf{Int},\delta_{\mathsf{Int}}),\varepsilon)$ for some $n > 2$ and $\varepsilon \in \mathbb{Q}_{\ge 0}$, then there exists a constant $m \in \mathbb{N}_{>0}$ such that, for every $S \in \wp^{\mathrm{re}}(\mathbb{Z})$, the output of the abstract interpreter is

$$[\![P_{n+m}]\!]^{\mathsf{Int}}\alpha_{\mathsf{Int}}(S) = \langle x \mapsto [(3-(n+m)),1]\rangle$$

and $\langle x \mapsto [(3-n),1]\rangle$ is not ε-close to $\langle x \mapsto [(3-(n+m)),1]\rangle$, namely

$$\langle x \mapsto [(3-n),1]\rangle \not\approx^{\varepsilon}_{\delta_{\mathsf{Int}}} \langle x \mapsto [(3-(n+m)),1]\rangle.$$

This implies that $P_{n+m} \notin \mathbb{C}((\mathsf{Int}, \delta_{\mathsf{Int}}), \varepsilon)$. Observe that, even though both $x > 1$ and $x \leq 1$ are exactly representable in Int with the intervals, respectively, $[2, +\infty]$ and $[-\infty, 1]$, the transfer function for $[\![x \leq 1]\!] : \wp^{\mathrm{re}}(\mathbb{Z}) \to \wp^{\mathrm{re}}(\mathbb{Z})$ is incomplete with respect to Int. This imprecision can be arbitrarily widened, without modifying the extensional behavior of the program. This makes possible to foil any partial complete abstraction with respect to any constant bound $\varepsilon \in \mathbb{Q}_{\geq 0}$. ■

We now study the computational limits of the class of partially complete and incomplete programs with respect to a given abstract quasi-metric space, ε bound, and set of input stores S. We begin with a definition of ε-triviality for an abstract quasi-metric space of stores.

*Definition 6.14 (**ε-trivial Abstract quasi-metric spaces**).* An abstract quasi-metric space of stores $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ is *ε-trivial* for some $\varepsilon \in \mathbb{Q}_{\geq 0}$ if $\mathbb{C}(\boldsymbol{A}, \varepsilon, S) = \mathsf{Prog}$. □

Of course ε-trivial abstract domains induce recursive classes of partial complete programs: If $\boldsymbol{A}$ is ε-trivial, then both $\mathbb{C}(\boldsymbol{A}, \varepsilon, S) = \mathsf{Prog}$ and $\overline{\mathbb{C}(\boldsymbol{A}, \varepsilon, S)} = \varnothing$ are recursive sets.
In the following, we show some simple examples of ε-trivial abstract domains.

Example 6.15. $(id, \delta^{\mathfrak{w}}_{id})$ and $(\top^{\mathbb{S}}, \delta^{\mathfrak{w}}_{\top^{\mathbb{S}}})$ are both n-trivial for all $n \in \mathbb{Q}_{\geq 0}$. The $(\mathsf{Sign}, \delta^{\mathfrak{w}}_{\mathsf{Sign}})$ abstract quasi-metric space is n-trivial for $n \geq 3$, while $(\mathsf{P} \sqcap \mathsf{S}, \delta^{\mathfrak{w}}_{\mathsf{P}\sqcap\mathsf{S}})$ is n-trivial for $n \geq 4$. ■

The following theorem shows that the class of programs $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ turns out to be r.e. whenever the abstract domain of stores A satisfies the ACC property.

Theorem 6.16. *If $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ is ACC, then for every $S \in \wp^{\mathrm{re}}(\mathbb{S})$ and $\varepsilon \in \mathbb{Q}_{\geq 0}$, $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ is r.e..*

Proof. Clearly, if $A = \top^{\mathbb{S}}$ then $\mathbb{C}(\boldsymbol{A}, \varepsilon, S) = \mathsf{Prog}$ and therefore r.e. (recursive in this case). Suppose $A \neq \top^{\mathbb{S}}$ and ACC. We show the sketch of a possible algorithm that terminates if and only if $P \in \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$:

(1) run the abstract interpreter $[\![P]\!]^A$ with input the recursive set of stores $\alpha_A(S)$. Suppose that $[\![P]\!]^A \alpha_A(S) = a$;

(2) if $a = \bot_A$, then the algorithm terminates (i.e., P is locally complete by the soundness property of abstract interpretation);

(3) otherwise ($a \neq \bot_A$), let us consider an enumeration $\{s_i\}_{i\in\mathbb{N}}$ of all $s \in S$ (recall that S is r.e. by Assumption 2);

(4) set $b \triangleq \bot_A$. Run a dovetail algorithm constructing $b \triangleq \bigsqcup_{i\in\mathbb{N}} \{\alpha_A([\![P]\!]s_i)\}$. The dovetail algorithm performs the first step of $[\![P]\!]s_0$ on the first store $s_0 \in S$; next, it performs the first step on the second store s_1 and the second step on the first store s_0; next, it performs the first step of the third store s_2, the second step of the second store s_1, and the third step on the first store s_0; and so on. When the program $[\![P]\!]s_i$ terminates (i.e., $[\![P]\!]s_i \neq \varnothing$) on some $s_i \in S$, then (since α_A is total recursive) the element $\alpha_A([\![P]\!]s_i)$ is added to b, namely: $b = b \sqcup_A \alpha_A([\![P]\!]s_i)$;

(5) if, at some point of the iterates, there exists $s_i \in S$ such that $[\![P]\!]s_i \neq \varnothing$, $b = b \sqcup_A \alpha_A([\![P]\!]s_i)$ and $\delta_A(b, a) \leq \varepsilon$, because $\delta_A(b, a) \leq \varepsilon$ is a decidable predicate by Definition 3.1 of quasi-metric A-compatible, then the algorithm terminates.

If $P \in \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ then the convergence of the above algorithms solving the recursive equation $b = b \sqcup_A \alpha_A([\![P]\!]s_i)$ is guaranteed by the ACC property assumption of $\boldsymbol{A}$. □

The general algorithm proposed in the proof of Theorem 6.16, allows us to systematically prove the ε-partial completeness of any program $P \in \mathsf{Prog}$ on any input $S \in \wp^{\mathrm{re}}(\mathbb{S})$ w.r.t. any ACC abstract quasi-metric space $\boldsymbol{A}$, while it does not give an answer for programs $P \notin \mathbb{C}(\boldsymbol{A}, \varepsilon, S)$.

Example 6.17. Consider the Sign abstract domain endowed with $\delta^{\mathfrak{w}}_{\mathsf{Sign}}$ and the following program:

$$P \triangleq \textbf{if } x \geq 1 \textbf{ then } x := x - 1 \textbf{ else skip}$$

We want to check whether P is in $\mathbb{C}((\mathsf{Sign}, \delta^{\mathfrak{w}}_{\mathsf{Sign}}), 1, \mathbb{Z}_{\geq 0})$, i.e., the class of programs whose concrete and abstract evaluations on the positive integers are 1-close w.r.t. $\delta^{\mathfrak{w}}_{\mathsf{Sign}}$, that is, the abstract interpreter must respect the condition: $\alpha_{\mathsf{Sign}}(\llbracket P \rrbracket \mathbb{Z}_{\geq 0}) \preccurlyeq_{\mathsf{Sign}} \llbracket P \rrbracket^{\mathsf{Sign}} \alpha_{\mathsf{Sign}}(\mathbb{Z}_{\geq 0})$. Following the algorithm sketched above we get: $\llbracket P \rrbracket^{\mathsf{Sign}} \alpha_{\mathsf{Sign}}(\mathbb{Z}_{\geq 0}) = \mathbb{Z}$. By constructing the set $b = b \sqcup_{\mathsf{Sign}} \alpha_{\mathsf{Sign}}(\llbracket P \rrbracket s_i)$ where $s_i \in \mathbb{Z}_{\geq 0}$ is an enumeration of $\mathbb{Z}_{\geq 0}$, we get:

$$\begin{aligned}
b &= \varnothing \\
b &= \varnothing \sqcup_{\mathsf{Sign}} \alpha_{\mathsf{Sign}}(\llbracket P \rrbracket \{0\}) = 0 \not\approx^{1}_{\delta^{\mathfrak{w}}_{\mathsf{Sign}}} \mathbb{Z} \\
b &= 0 \sqcup_{\mathsf{Sign}} \alpha_{\mathsf{Sign}}(\llbracket P \rrbracket \{1\}) = 0 \sqcup_{\mathsf{Sign}} 0 = 0 \not\approx^{1}_{\delta^{\mathfrak{w}}_{\mathsf{Sign}}} \mathbb{Z} \\
b &= 0 \sqcup_{\mathsf{Sign}} \alpha_{\mathsf{Sign}}(\llbracket P \rrbracket \{2\}) = 0 \sqcup_{\mathsf{Sign}} + = + \approx^{1}_{\delta^{\mathfrak{w}}_{\mathsf{Sign}}} \mathbb{Z}
\end{aligned}$$

at this point the algorithm terminates, hence we can conclude that $P \in \mathbb{C}((\mathsf{Sign}, \delta^{\mathfrak{w}}_{\mathsf{Sign}}), \mathbb{Z}_{\geq 0}, 1)$. Note that, for checking whether $P \in \mathbb{C}((\mathsf{Sign}, \delta^{\mathfrak{w}}_{\mathsf{Sign}}), \mathbb{Z}_{\geq 0}, 0)$ the algorithm keeps running the check $b = + \sqcup_{\mathsf{Sign}} \alpha_{\mathsf{Sign}}(\llbracket P \rrbracket \{n\}) = + \sqcup_{\mathsf{Sign}} + = +$ and $+ \not\approx^{0}_{\delta^{\mathfrak{w}}_{\mathsf{Sign}}} \mathbb{Z}$ for all $n \in \mathbb{Z}_{\geq 0}$. This because we have $P \notin \mathbb{C}(\mathsf{Sign})$ and hence $P \notin \mathbb{C}((\mathsf{Sign}, \delta^{\mathfrak{w}}_{\mathsf{Sign}}), \mathbb{Z}_{\geq 0}, 0)$. ∎

We can trivially extend the use of the algorithm proposed in Theorem 6.16 to the local completeness class of programs which are complete with respect to ACC abstract domains of stores and a set of input stores.

Corollary 6.18. *For every $A \in \mathcal{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ and $S \in \wp^{\mathrm{re}}(\mathbb{S})$, if A is ACC then $\mathbb{C}(A, S)$ is r.e..*

The following theorem proves 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 6.19. *If $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ 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..*

Proof. The proof is made by showing that the first order predicate defining the class of ε-partially complete (resp. incomplete) programs (which can be seen as a subset of natural numbers since each program in Prog can be mapped through the Gödel numbering $\mathfrak{g} : \mathsf{Prog} \to \mathbb{N}$ to a natural number) is classified in the arithmetical hierarchy as Π_2 (resp. Σ_2). Let $\varphi_P : \mathbb{S} \nrightarrow \mathbb{S}$ be the partial recursive function associated to program $P \in \mathsf{Prog}$ (the concrete interpreter) and, without loss of generality, $\varphi_A : \mathsf{Prog} \times A \to A$ be the total recursive function representing the abstract interpreter. We define the predicate $R(\boldsymbol{A}, \varepsilon, P, s, n)$, with $s \in \mathbb{S}$ and $n \in \mathbb{N}$, as follows: $R(\boldsymbol{A}, \varepsilon, P, s, n) \Leftrightarrow \varphi_P(s) \downarrow \text{ in } n \text{ steps} \wedge \delta_A(\alpha_A(\varphi_P(s)), \varphi_A(P, \alpha_A(s))) \leq \varepsilon$. Because $\delta_A(a, b) \leq \varepsilon$ is decidable by definition of δ_A, the predicate defining R is recursive, i.e., $R(\boldsymbol{A}, \varepsilon, P, s, n) \in \Sigma_0 = \Pi_0$. We can rewrite the decidability of the ε-partial completeness and incompleteness classes as follows:

$$\begin{aligned}
P \in \mathbb{C}(\boldsymbol{A}, \varepsilon, S) &\Leftrightarrow \forall s \in S.\ \exists n \in \mathbb{N}.\ R(\boldsymbol{A}, \varepsilon, P, s, n) \\
P \in \overline{\mathbb{C}(\boldsymbol{A}, \varepsilon, S)} &\Leftrightarrow \exists s \in S.\ \forall n \in \mathbb{N}.\ \neg R(\boldsymbol{A}, \varepsilon, P, s, n).
\end{aligned}$$

Table 1. Recursive properties of completeness, local completeness and partial completeness classes of programs and their respective complement classes

	Class	Recursive Property	Proof	Conditions
Completeness	$\mathbb{C}(A)$	non-r.e.	[Giacobazzi et al. 2015]	A non-trivial
	$\overline{\mathbb{C}(A)}$	non-r.e.	[Giacobazzi et al. 2015]	A non-trivial
Local Completeness	$\mathbb{C}(A, S)$	r.e.	Corollary 6.18	A ACC
	$\overline{\mathbb{C}(A, S)}$	non-r.e.	Corollary 6.20	A non-trivial
Partial Completeness	$\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$	r.e.	Theorem 6.16	$\boldsymbol{A}$ ACC
	$\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$	non-r.e.	Theorem 6.19	$\boldsymbol{A}$ non-ε-trivial, non-ACC
	$\overline{\mathbb{C}(\boldsymbol{A}, \varepsilon, S)}$	non-r.e.	Theorem 6.19	$\boldsymbol{A}$ non-ε-trivial

By observing the bounded quantifiers before $R(\boldsymbol{A}, \varepsilon, P, s, n)$ and $\neg R(\boldsymbol{A}, \varepsilon, P, s, n)$, we can conclude that $R(\boldsymbol{A}, \varepsilon, P, s, n) \in \Pi_2$ and $R(\boldsymbol{A}, \varepsilon, P, s, n) \in \Sigma_2$. This proves that $\mathbb{C}(\boldsymbol{A}, \varepsilon, S)$ and $\overline{\mathbb{C}(\boldsymbol{A}, \varepsilon, S)}$ are both non-r.e. sets. □

Corollary 6.20. *If $A \notin \{id, \top^{\mathbb{S}}\}$ then the local incompleteness class $\overline{\mathbb{C}(A, S)}$ is non-r.e., moreover if A is also not ACC then the local completeness class $\mathbb{C}(A, S)$ is non-r.e..*

Let us notice that the proofs of Theorems 6.16 and 6.19 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, automating the proof that an abstract domain is ε-partially complete or ε-partially incomplete for a given program and a given set of input stores— i.e., deciding whether a static program analysis can produce or cannot produce some bounded set of false alarms—is in general impossible. 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. Table 1 summarizes the state of the art concerning the recursive properties of these classes.

7 ESTIMATING THE IMPRECISION OF AN ABSTRACT INTERPRETER

The above results of non-recursive enumerability for non-ACC abstract domains of the class of partial complete programs prevent us from developing any complete procedure to establish whether an abstraction is ε-partial complete or incomplete for a given program and input stores. In this section we introduce a sound proof system that is able to overstimate a bound of imprecision generated by the abstract interpreter when analyzing P with input S over $\boldsymbol{A}$.

Given an abstract quasi-metric space of stores $\boldsymbol{A} \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$, a program $P \in \mathrm{Prog}$, a pre-condition and a post condition predicates over $\mathbb{S}$, our goal is to derive a triple $[\,\mathrm{Pre}\,]\ P\ [\,\mathrm{Post}, \varepsilon\,]$ such that the abstraction of the post-condition Post is ε-close to the abstraction of the concrete

$$\frac{}{\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ \mathbf{skip}\ [\,\mathrm{Pre}, 0\,]}\ [\mathbf{skip}] \qquad \frac{b? \in \mathbb{C}(\boldsymbol{A}, \varepsilon, \mathrm{Pre})}{\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ b?\ [\,[\![b?]\!]\gamma_A(\alpha_A(\mathrm{Pre})), \varepsilon\,]}\ [\mathbf{bool}]$$

$$\frac{x := a \in \mathbb{C}(\boldsymbol{A}, \varepsilon, \mathrm{Pre})}{\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ x := a\ [\,[\![x := a]\!]\gamma_A(\alpha_A(\mathrm{Pre})), \varepsilon\,]}\ [\mathbf{assign}]$$

$$\frac{\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ P_1\ [\,\mathrm{Mid}, \beta\,] \quad \vdash_{\boldsymbol{A}} [\,\mathrm{Mid}\,]\ P_2\ [\,\mathrm{Post}, \varepsilon\,] \quad \vdash_{\boldsymbol{A}} [\,[\![P_1]\!]\mathrm{Pre}\,]\ P_2\ [\,[\![P_2]\!]\mathrm{Mid}, \eta\,]}{\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ P_1; P_2\ [\,\mathrm{Post}, \varepsilon + \eta\,]}\ [\mathbf{seq}]$$

$$\frac{\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ P\ [\,\mathrm{Post}, \varepsilon\,] \quad \alpha_A(\mathrm{Post}) \approx^{\beta}_{\delta_A} \alpha_A(\mathrm{Post}')}{\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ P\ [\,\mathrm{Post}', \varepsilon + \beta\,]}\ [\mathbf{weaken}] \quad \frac{\vdash_{\boldsymbol{A}} [\,\gamma_A(\alpha_A(\mathrm{Pre}))\,]\ P\ [\,\mathrm{Pre}, \varepsilon\,]}{\vdash_{\boldsymbol{A}} [\,\gamma_A(\alpha_A(\mathrm{Pre}))\,]\ P^*\ [\,\mathrm{Pre}, \varepsilon\,]}\ [\mathbf{star}]$$

$$\frac{\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ P_1\ [\,\mathrm{Post}_1, \beta\,] \quad \vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ P_2\ [\,\mathrm{Post}_2, \varepsilon\,] \quad \oplus\text{-Bound}(\boldsymbol{A}, \mathfrak{b})}{\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ P_1 \oplus P_2\ [\,\mathrm{Post}_1 \vee \mathrm{Post}_2, \mathfrak{b}(\beta, \varepsilon)\,]}\ [\mathbf{join}]$$

Fig. 7. The proof system $\vdash_{\boldsymbol{A}}$

evaluation of P with input the pre-condition Pre, namely:

$$[\,\mathrm{Pre}\,]\ P\ [\,\mathrm{Post}, \varepsilon\,] \Leftrightarrow \alpha_A([\![P]\!]\mathrm{Pre}) \approx^{\varepsilon}_{\delta_A} \alpha_A(\mathrm{Post}).$$

This generalizes the well-known Hoare triples for partial correctness, by keeping track in ε of the imprecision measured by δ_A and accumulated by the abstract interpreter defined on the abstract domain A. Of course we allow an undefined amount of imprecision, here denoted by the symbol ∞, that is, $\varepsilon \in \mathbb{Q}^{\infty}_{\geq 0}$. To simplify notation, we represent the predicates Pre, Post as sets of stores instead of first order predicate. For example, the set $\{-2, 5\} \in \wp^{\mathrm{re}}(\mathbb{Z})$ is used to represent a more verbose expression such as $x = -2 \vee x = 5$. The proof system is shown in Fig. 7.

Let us give an intuition for each rule. The triples derived by [**bool**] and [**assign**] rules represent the basic inductive assumptions on the bound of the imprecision. Note that the post-condition of both rules corresponds to the bca of the concrete semantics. Rule [**weaken**] allows us to derive a weaker post-condition by choosing another output Post′ that over approximates Post and is more inaccurate of β. In this case, by the triangle inequality axiom of δ_A, we get as new bound $\varepsilon + \beta$. The rule [**seq**] for sequential composition requires the additional hypothesis that $\vdash_{\boldsymbol{A}} [\,[\![P_1]\!]\mathrm{Pre}\,]\ P_2\ [\,[\![P_2]\!]\mathrm{Mid}, \eta\,]$ which states that the abstract output of the concrete semantics of P_2 with input Mid is η-close respect to having as input $[\![P_1]\!]\mathrm{Pre}$. This fixes the maximal output distance of P_2 with input Mid. Here η could be obtained as a function depending on the β-distance of the inputs starting from $[\![P_1]\!]\mathrm{Pre}$. Note that, in this case, the worst case imprecision bound consists in the sum of the bound η generated by the computation $\alpha_A([\![P_2]\!]\mathrm{Mid})$ and the distance with the abstraction of the post-condition chosen. Rule [**star**] captures an abstract invariant with an amount of maximal error quantified in ε. Because the join operator is always complete and the elements building P^* are all along a chain, the $*$ operator does not lose further precision w.r.t. what P already does. This rule does not consider the use of a widening operator and therefore it assumes that the abstract interpreter terminates (e.g., when $\boldsymbol{A}$ is ACC). The case of widening-based abstract interpreters always ensuring termination is later discussed in Section 9. The last rule [**join**] involves the join operator. Even though the abstract join operator preserves completeness, the case where one of the two (or both) operands are incomplete, is not straightforward. The problem here stems in the fact that the resulting imprecision bound could not be determined by knowing only the imprecision on P_1 and P_2. This is because the quasi-metric distance between the concrete execution of $P_1 \oplus P_2$ and the join of the two post-conditions, relies on the underlying structure of the abstract quasi-metric space considered. For this reason, the premise of rule [**join**] asks for the validity of

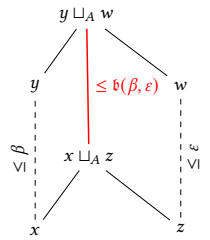

Fig. 8. The $\oplus$-Bound condition

the predicate $\oplus$-Bound which induces a topological property on the abstract domain as depicted in Fig. 8, and defined as follows:

*Definition 7.1 (**$\oplus$-Bound**).* Let $\boldsymbol{A} \in \mathfrak{A}(\wp^{\text{re}}(\mathbb{S}))$ and $\mathfrak{b} : \mathbb{Q}^{\infty}_{\geq 0} \times \mathbb{Q}^{\infty}_{\geq 0} \to \mathbb{Q}^{\infty}_{\geq 0}$. The predicate $\oplus$-Bound$(\boldsymbol{A}, \mathfrak{b})$ is true if the function $\mathfrak{b}$ satisfies the following condition for all $x, y, z, w \in A$ and $\beta, \varepsilon \in \mathbb{Q}^{\infty}_{\geq 0}$: $\delta_A(x, y) \leq \beta \ \wedge \ \delta_A(z, w) \leq \varepsilon \implies \delta_A(x \sqcup_A z, y \sqcup_A w) \leq \mathfrak{b}(\beta, \varepsilon)$. □

Note that Definition 7.1 ranges over any $x, z \in A$, this means that it also holds when $x \leq_A z$ and viceversa (the same applies for y and w). Fig. 8 is an example where, respectively, x, z and y, w are incomparable. Recall that $\delta_A(x, y) \leq \beta$ implies $x \leq_A y$. The predicate $\oplus$-Bound captures a specific structural property of the quasi-metric space $\boldsymbol{A}$: It says that for every four abstract elements that are related as in Fig. 8, the corresponding join has a limited error bounded by the function $\mathfrak{b}$ which depends on the two bound β and ε. This topological property depends on both the complete lattice forming A and on the quasi-metric A-compatible considered. Clearly, the constant function $\mathfrak{b}(\beta, \varepsilon) \triangleq \infty$ satisfies the predicate $\oplus$-Bound for every $\boldsymbol{A} \in \mathfrak{A}(\wp^{\text{re}}(\mathbb{S}))$, though giving no information on a possible bound. As an example of application, in the following proposition we prove that the predicate $\oplus$-Bound is valid for the interval abstract domain $(\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}})$ defined in Example 3.7, where $\mathfrak{b}$ corresponds to the sum function.

Proposition 7.2. *Let $\mathfrak{b}(\beta, \varepsilon) \triangleq \beta + \varepsilon$ for all $\beta, \varepsilon \in \mathbb{Q}_{\geq 0}$. Then $\oplus$-Bound$((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), \mathfrak{b})$ holds.*

Proof. Clearly, if either β or ε (or both) are ∞, then $\mathfrak{b}(\beta, \varepsilon) = \infty$ and the predicate is trivially true. Let us consider four finite intervals $[a, b], [c, d], [x, y], [z, w] \in \mathsf{Int}$ such that $\delta^{\mathfrak{w}}_{\mathsf{Int}}([a, b], [c, d]) = \beta \neq \infty$ and $\delta^{\mathfrak{w}}_{\mathsf{Int}}([x, y], [z, w]) = \varepsilon \neq \infty$. Let us consider the case where $b < x$ and $d < z$, that is, the intervals $[a, b]$ and $[x, y]$ do not share elements (same for $[c, d]$ and $[z, w]$). Let $|[a, b]|$ be the standard cardinality function. By the definition of $\delta^{\mathfrak{w}}_{\mathsf{Int}}$ and the stated assumptions, we know that:

$$|[c, d]| = |[a, b]| + \beta$$
$$|[z, w]| = |[x, y]| + \varepsilon.$$

That is, the intuition of $\delta^{\mathfrak{w}}_{\mathsf{Int}}$ is that if $\delta^{\mathfrak{w}}_{\mathsf{Int}}([a, b], [c, d]) = \beta$ then the interval $[c, d]$ has β elements more than $[a, b]$. This implies that we can count the elements of the union between two intervals as follows:

$$|[a, b] \sqcup_{\mathsf{Int}} [x, y]| = |[a, b]| + |[x, y]| + |[b + 1, x - 1]|$$
$$|[c, d] \sqcup_{\mathsf{Int}} [z, w]| = |[c, d]| + |[z, w]| + |[d + 1, z - 1]|.$$

Note that $|[d+1, z-1]| \leq |[b+1, x-1]|$ because $[d+1, z-1]$ is defined by the intervals $[c, d]$ and $[z, w]$ which are strictly larger than $[a, b]$ and $[x, y]$, respectively. Therefore, we get:

$$\begin{aligned}\delta^{\mathfrak{w}}_{\mathsf{Int}}([a,b] \sqcup_{\mathsf{Int}} [x,y], [c,d] \sqcup_{\mathsf{Int}} [z,w]) &= |[c,d] \sqcup_{\mathsf{Int}} [z,w]| - |[a,b] \sqcup_{\mathsf{Int}} [x,y]| \\ &= \beta + \varepsilon + |[d+1, z-1]| - |[b+1, x-1]| \\ &\leq \beta + \varepsilon.\end{aligned}$$

The other cases follows immediately. □

The following proposition gives a straightforward bound function for the abstract join over abstract domains with finite height where the weighted path-length quasi-metric is used.

Proposition 7.3. *Let $(A, \delta^{\mathfrak{w}}_A) \in \mathfrak{A}(\wp^{\mathrm{re}}(\mathbb{S}))$ be any ACC and DCC abstract quasi-metric domain endowed with the weighted path-length quasi-metric. If we define*

$$\mathfrak{b}(\beta, \varepsilon) \triangleq \begin{cases} 0 & \text{if } \beta = \varepsilon = 0, \\ max\{\sum_{e \in \mathbf{c}} \mathfrak{w}(e) \mid \mathbf{c} \in \mathfrak{C}^{\top_A}_{\bot_A}\} & \text{otherwise.} \end{cases}$$

then $\oplus$-Bound$((A, \delta^{\mathfrak{w}}_A), \mathfrak{b})$ holds.

The following theorem proves the soundness of our proof system.

Theorem 7.4. $\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\ P\ [\,\mathrm{Post}, \varepsilon\,] \implies \alpha_A([\![P]\!]\mathrm{Pre}) \approx^{\varepsilon}_{\delta_A} \alpha_A(\mathrm{Post})$.

Proof. Rule [**exp**] follows immediately since 0-closeness implies equality.
[**weaken**]: $\alpha_A([\![P]\!]\mathrm{Pre}) \approx^{\varepsilon}_{\delta_A} \alpha_A(\mathrm{Post}) \approx^{\beta}_{\delta_A} \alpha_A(\mathrm{Post}')$ and, by the weak triangle inequality of δ_A, we conclude that $\alpha_A([\![P]\!]\mathrm{Pre}) \approx^{\varepsilon+\beta}_{\delta_A} \alpha_A(\mathrm{Post}')$.
[**seq**]: By assuming $\alpha_A([\![P_2]\!][\![P_1]\!]\mathrm{Pre}) \approx^{\eta}_{\delta_A} \alpha_A([\![P_2]\!]\mathrm{Mid}) \approx^{\varepsilon}_{\delta_A} \alpha_A(\mathrm{Post})$ and by the weak triangle inequality we get $\alpha_A([\![P_2]\!][\![P_1]\!]\mathrm{Pre}) \approx^{\varepsilon+\eta}_{\delta_A} \alpha_A(\mathrm{Post})$.
[**star**]: We assume $\alpha_A([\![P]\!]\gamma_A(\alpha_A(\mathrm{Pre}))) \approx^{\varepsilon}_{\delta_A} \alpha_A(\mathrm{Pre})$. We need to prove that the following distance holds: $\alpha_A([\![P^*]\!]\gamma_A(\alpha_A(\mathrm{Pre}))) \approx^{\varepsilon}_{\delta_A} \alpha_A(\mathrm{Pre})$. By definition

$$\begin{aligned}\alpha_A([\![P^*]\!]\gamma_A(\alpha_A(\mathrm{Pre}))) &= \alpha_A(\bigcup_n \{[\![P]\!]^n \gamma_A(\alpha_A(\mathrm{Pre}))\}) \\ &= \bigsqcup_n \{\alpha_A([\![P]\!]^n \gamma_A(\alpha_A(\mathrm{Pre})))\}.\end{aligned}$$

By induction, if $n = 0$ we get $\alpha_A(\mathrm{Pre}) \approx^{\varepsilon}_{\delta_A} \alpha_A(\mathrm{Pre})$ and if $n = 1$ $\alpha_A([\![P]\!]\gamma_A(\alpha_A(\mathrm{Pre}))) \approx^{\varepsilon}_{\delta_A} \alpha_A(\mathrm{Pre})$ which holds by the premise of this rule. Assume the inductive hypothesis for n. Then for the $n+1$ step we get:

$$\begin{aligned}\alpha_A([\![P]\!]^{n+1}\gamma_A(\alpha_A(\mathrm{Pre}))) &= \alpha_A([\![P]\!][\![P]\!]^n \gamma_A(\alpha_A(\mathrm{Pre}))) \\ &\leq_A \alpha_A([\![P]\!]\gamma_A(\alpha_A([\![P]\!]^n \gamma_A(\alpha_A(\mathrm{Pre}))))) \approx^{\varepsilon}_{\delta_A} \alpha_A(\mathrm{Pre}).\end{aligned}$$

Therefore, we can conclude $\alpha_A([\![P^*]\!]\gamma_A(\alpha_A(\mathrm{Pre}))) \approx^{\varepsilon}_{\delta_A} \alpha_A(\mathrm{Pre})$.
[**join**]: We know that $\alpha_A([\![P_1]\!]\mathrm{Pre}) \approx^{\beta}_{\delta_A} \alpha_A(\mathrm{Post}_1)$ and $\alpha_A([\![P_2]\!]\mathrm{Pre}) \approx^{\varepsilon}_{\delta_A} \alpha_A(\mathrm{Post}_2)$. Then, by assuming the validity of $\oplus$-Bound, we get:

$$\begin{aligned}\alpha_A([\![P_1 \oplus P_2]\!]\mathrm{Pre}) &= \alpha_A([\![P_1]\!]\mathrm{Pre} \vee [\![P_2]\!]\mathrm{Pre}) \\ &= \alpha_A([\![P_1]\!]\mathrm{Pre}) \vee_A \alpha_A([\![P_2]\!]\mathrm{Pre}) \\ &\approx^{\mathfrak{b}(\beta,\varepsilon)}_{\delta_A} \alpha_A(\mathrm{Post}_1) \vee_A \alpha_A(\mathrm{Post}_2) = \alpha_A(\mathrm{Post}_1 \vee \mathrm{Post}_2).\end{aligned}$$

□

$$\dfrac{\dfrac{x \geq 0? \in \mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 3, \{-1,3,7\}))}{\vdash_{(\mathsf{Int},\delta^{\mathfrak{w}}_{\mathsf{Int}})} [\,\{-1,3,7\}\,]\; x \geq 0?\; [\,\{0,..,7\},3\,]}\;[\mathbf{bool}] \quad \dfrac{}{\vdash_{(\mathsf{Int},\delta^{\mathfrak{w}}_{\mathsf{Int}})} [\,\{0,..,7\}\,]\; \mathbf{skip}\; [\,\{0,..7\},0\,]}\;[\mathbf{skip}]}{(*)} \quad \dfrac{\dfrac{}{\vdash_{(\mathsf{Int},\delta^{\mathfrak{w}}_{\mathsf{Int}})} [\,\{3,7\}\,]\; \mathbf{skip}\; [\,\{3,7\},0\,]}\;[\mathbf{skip}] \quad \alpha_{\mathsf{Int}}(\{3,7\}) \approx^{3}_{\delta^{\mathfrak{w}}_{\mathsf{Int}}} \alpha_{\mathsf{Int}}(\{0,..7\})}{\vdash_{(\mathsf{Int},\delta^{\mathfrak{w}}_{\mathsf{Int}})} [\,\{3,7\}\,]\; \mathbf{skip}\; [\,\{0,..,7\},3\,]}\;[\mathbf{weaken}]}{}\;[\mathbf{seq}]$$

$$\dfrac{\dfrac{x < 0? \in \mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 0, \{-1,3,7\})}{\vdash_{(\mathsf{Int},\delta^{\mathfrak{w}}_{\mathsf{Int}})} [\,\{-1,3,7\}\,]\; x < 0?\; [\,\{-1\},0\,]}\;[\mathbf{bool}] \quad \dfrac{x := x * -1 \in \mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 0, \{-1\})}{\vdash_{(\mathsf{Int},\delta^{\mathfrak{w}}_{\mathsf{Int}})} [\,\{-1\}\,]\; x := x * -1\; [\,\{1\},0\,]}\;[\mathbf{assign}]}{(\#)}\;[\mathbf{seq}]$$

$$\dfrac{\dfrac{(*)}{\vdash_{(\mathsf{Int},\delta^{\mathfrak{w}}_{\mathsf{Int}})} [\,\{-1,3,7\}\,]\; x \geq 0; \mathbf{skip}\; [\,\{0,..,7\},3\,]}\;[\mathbf{seq}] \quad \dfrac{(\#)}{\vdash_{(\mathsf{Int},\delta^{\mathfrak{w}}_{\mathsf{Int}})} [\,\{-1,3,7\}\,]\; x < 0?; x := x * -1\; [\,\{1\},0\,]}\;[\mathbf{seq}]}{\vdash_{(\mathsf{Int},\delta^{\mathfrak{w}}_{\mathsf{Int}})} [\,\{-1,3,7\}\,]\; P\; [\,\{0,..,7\},3\,]}\;[\mathbf{join}]$$

Fig. 9. Derivation of Example 7.6

The fundamental consequence of Theorem 7.4 is reported in the following corollary. Here the class of programs $\mathbb{C}^{\mathrm{bca}}(\boldsymbol{A}, \beta, S)$ is defined as follows:

$$\mathbb{C}^{\mathrm{bca}}(\boldsymbol{A}, \beta, S) \triangleq \{P \in \mathsf{Prog} \mid \alpha_A([\![P]\!]S) \approx^{\beta}_{\delta_A} \alpha_A([\![P]\!]\gamma_A(\alpha_A(S)))\}.$$

Corollary 7.5. *The following implications hold:*

(i) $\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\; P\; [\,[\![P]\!]\gamma_A(\alpha_A(\mathrm{Pre})), \beta\,] \implies P \in \mathbb{C}^{\mathrm{bca}}(\boldsymbol{A}, \beta, \mathrm{Pre})$

(ii) $\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\; P\; [\,\gamma_A([\![P]\!]^A\alpha_A(\mathrm{Pre})), \varepsilon\,] \implies P \in \mathbb{C}(\boldsymbol{A}, \varepsilon, \mathrm{Pre})$

(iii) $\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\; P\; [\,[\![P]\!]\gamma_A(\alpha_A(\mathrm{Pre})), \beta\,] \wedge \vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\; P\; [\,\gamma_A([\![P]\!]^A\alpha_A(\mathrm{Pre})), \varepsilon\,] \Rightarrow \beta \leq \varepsilon$

Intuitively, the class $\mathbb{C}^{\mathrm{bca}}(\boldsymbol{A}, \beta, S)$ collects all the programs whose bca on A with input S, is β-close to the abstraction of their corresponding concrete executions.
The triple $\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\; P\; [\,\gamma_A([\![P]\!]^A\alpha_A(\mathrm{Pre})), \varepsilon\,]$ infers the upper bound of imprecision between the abstraction of the concrete execution of P and its abstract semantics with input S, that is, we prove that $\boldsymbol{A}$ (i.e., the abstract semantics $[\![\cdot]\!]^A$ defined in Fig. 5) is ε-partial complete for P. Conversely, the triple $\vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\; P\; [\,[\![P]\!]\gamma_A(\alpha_A(\mathrm{Pre})), \beta\,]$ allows us to prove the limit of imprecision of the bca of A respect to the abstraction of the concrete semantics on P with input S. β can be considered as the *intrinsic imprecision* of the chosen abstract quasi-metric space $\boldsymbol{A}$ when analyzing P with input S, from which we cannot define a more precise analysis.

Example 7.6. Let us consider the program $P_{\mathrm{abs}} \triangleq (x \geq 0;\ \mathbf{skip}) \oplus (x < 0?;\ x := x * (-1))$ of Example 6.5 and the input $S_2 = \{-1, 3, 7\}$. We know that $[\![P]\!]^{\mathsf{Int}}\alpha_{\mathsf{Int}}(S_2) = [0, 7]$ and by deriving in Fig. 9 the triple $\vdash_{(\mathsf{Int},\delta^{\mathfrak{w}}_{\mathsf{Int}})} [\,\{-1,3,7\}\,]\; P\; [\,\{0,1,..,7\},3\,]$ we can conclude that $P \in \mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 3, S_2)$. From Example 6.5 we know that $P \in \mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 1, S_2)$, and, although the result is sound since $\mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 1, S_2) \subseteq \mathbb{C}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 3, S_2)$, this discrepancy is due to the partial completeness hypothesis of the base rule [**bool**] on $x \geq 0?$ with input S_2 which will be propagated through the other rules. It is easy to note that the abstract semantics corresponds to the bca over Int, i.e., $[\![P]\!]^{\mathsf{Int}}\alpha_{\mathsf{Int}}(S_2) = \alpha_{\mathsf{Int}}([\![P]\!]\gamma_{\mathsf{Int}}(\alpha_{\mathsf{Int}}(S_2)))$, therefore, the derivation in Fig. 9 proves also that $P \in \mathbb{C}^{\mathrm{bca}}((\mathsf{Int}, \delta^{\mathfrak{w}}_{\mathsf{Int}}), 3, S_2)$. ∎

The distance between $\alpha_A([\![P]\!]\gamma_A(\alpha_A(\mathrm{Pre}))) \leq_A [\![P]\!]^A\alpha_A(\mathrm{Pre})$ can be obtained by modifying the rules [**bool**] and [**assign**] as follows:

$$\dfrac{}{\Vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\; x := a\; [\,[\![x := a]\!]\gamma_A(\alpha_A(\mathrm{Pre})), 0\,]}\;[\mathbf{assign}] \quad \dfrac{}{\Vdash_{\boldsymbol{A}} [\,\mathrm{Pre}\,]\; b?\; [\,[\![b?]\!]\gamma_A(\alpha_A(\mathrm{Pre})), 0\,]}\;[\mathbf{bool}]$$

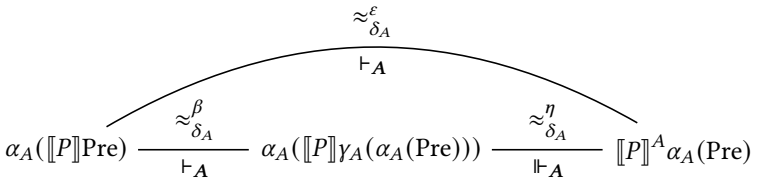

Fig. 10. The distance relation achieved by the two proof systems $\vdash_{\boldsymbol{A}}$ and $\Vdash_{\boldsymbol{A}}$

The other rules remain untouched. We denote this new proof system with $\Vdash_{\boldsymbol{A}}$. Note that, as we are interested in measuring the imprecision added by the abstract interpreter w.r.t. the bca, rules [**bool**] and [**assign**] assert that the evaluation of the base rules, i.e., the regular expressions do not generate imprecision because the abstract semantics considered evaluates $x := a$ and $b?$ as their respective bcas. The soundness proof of $\Vdash_{\boldsymbol{A}}$ follows immediately by $\vdash_{\boldsymbol{A}}$.

Corollary 7.7. *The following implication holds:*

$$\Vdash_{\boldsymbol{A}} [\,\gamma_A(\alpha_A(\mathrm{Pre}))\,]\ P\ [\,\gamma_A([\![P]\!]^A\alpha_A(\mathrm{Pre})), \eta\,] \ \Rightarrow\ \delta_A(\alpha_A([\![P]\!]\gamma_A(\alpha_A(\mathrm{Pre}))), [\![P]\!]^A\alpha_A(\mathrm{Pre})) \leq \eta.$$

Fig. 10 summarizes the results obtained from Corollaries 7.5 and 7.7.

8 RELATED WORK

Completeness is a well known notion in static program analysis by abstract interpretation and the classes of complete programs for a given abstraction have been recently studied [Bruni et al. 2020, 2021; Giacobazzi et al. 2015]. In [Giacobazzi et al. 2015] the authors introduce the notion of completeness class as the set of all programs that are complete with regard to a given abstract domain, together with a sound stratified deductive system for proving the completeness of program analysis over an abstract domain. Giacobazzi et al. [2015] provide a sound proof system for proving completeness of any program respect to an abstract domain of stores. We follow an orthogonal approach and, rather than proving whether a program P is ε-partial complete respect to an abstract quasi-metric space of stores $\boldsymbol{A}$ and a set of stores S, we formalize a sound proof system that is able to overstimate a bound of imprecision generated by the abstract interpreter when analyzing P with input S over $\boldsymbol{A}$. Bruni et al. [Bruni et al. 2020] introduced the concept of completeness cliques as the set of equivalent programs that are complete with regard to an abstract domain. In particular, they prove that there exists a total recursive function that transforms any complete program into a semantically equivalent but incomplete one for a given abstraction. This work does not distinguish among incompleteness results: An analysis is either complete or incomplete but no further formalization is available for reasoning about the level of imprecision associated to an incomplete analysis. The first attempt to weaken the notion of completeness in abstract interpretation has been recently introduced by Bruni et al. [2021]. Here the authors introduced the notion of local completeness, that is, completeness among certain program traces. They provided a logical proof system that combines over and under-approximations of programs behaviors. Our approach can be considered as a further weakening of local completeness, as our aim is to be able to measure and reason about the imprecision induced by program analysis.

A wide literature has addressed the problem of measuring the imprecision of abstract interpretation and static analysis. Among the earliest, Crazzolara Crazzolara [1997] was among the very first proposing the use of quasi-metric spaces, instead of partial orders, as a definition of distance in abstract interpretation. In particular [Crazzolara 1997] applies Banach's contraction principle as

an alternative for Knaster-Tarsky's fixpoint theorem. Conversely, our approach considers quasi-metrics as *external* measures and we follow the classical framework of abstract interpretation in [Cousot and Cousot 1977] and [Cousot and Cousot 1979] for the fixpoint approximation in concrete collecting and abstract semantics. In [Casso et al. 2019] the authors propose distances in logic programming domains for measuring the precision of analysis, while in [Logozzo 2009] the authors introduce the notion of pseudo-distance, a weaker form of metric similar to quasi-metric definition, as an external measure function to quantifying the relative loss of precision induced by numerical abstract domains. Also Sotin [2010] defines measures in $\mathbb{R}^n$ that allow us to quantify the difference in precision between two abstract values of a numeric domain, by comparing the size of their concretizations. This is applied to guess the most appropriate domain to analyze a program, by under-approximating the potentially visited states via random testing and comparing the precision with which different domains would approximate those states. Di Pierro and Wiklicky [2000] also propose a notion of probabilistic abstract interpretation, which allows us to measure the precision of an abstract domain and its operators by using vector spaces instead of partially order sets. However, all the above mentioned papers do not study the properties of families of programs for which the precision of the analysis can be bounded and ways to inductively estimate, in a proof system like approach, an upper bound to the error injected by the abstract interpreter during program analysis. It is worth remarking that, instead of defining domain-specific measures of the imprecision injected by program analysis, we define a general framework based on generic quasi-metric spaces and a weaker notion of completeness, here called partial completeness, that allow us to control the amount of imprecision that we tolerate in the analysis. All the above mentioned notions of distance (e.g., those in [Casso et al. 2019; Logozzo 2009; Sotin 2010]) can be plugged in our theory, resulting in specific classes of partial complete programs and a proof system for estimating an upper bound to the error accumulated by the abstract interpreter.

9 CONCLUSION

Partial completeness opens a new perspective in the field of static program analysis by abstract interpretation. Incompleteness is a common situation in program analysis and partial completeness allows us to refine the notion of incomplete analysis by tuning the amount of *noise* that we allow in the analysis. Partial completeness cannot be easily reduced to standard completeness on a coarser abstract domain. For example, if we consider an abstract domain where the ε-close abstract states are gathered together (e.g., a set of intervals), then this newly obtained abstract domain is formed by elements which represent properties of properties of states. This provides a way to express properties of the analyzer applied to a program P on a given input S and not properties of what P with input S computes. On this new domain, checking ε-partial completeness of P with input S can be transferred to checking standard completeness of the analyzer applied to P on S. Nevertheless, being an abstract interpretation, this new (meta-)analysis may not be itself complete and, therefore, may produce false alarms.

For practical usage, our ambition is to encourage future users/designers of abstract interpretation to use/design abstract domains always equipped with a specific metric of interest. This can be the result of the same design process used for abstract domains: (1) check the hypothetical invariant after a few concrete iterations and (2) design the appropriate shapes to include in the abstract domain. Any deviation from the expected shape (e.g., a larger interval, octagon, ellipsoid) is an error with a corresponding value. The corresponding abstract quasi metric space will be the collection of these objects enriched with a quasi metric expressing the relative precision among them. Our framework provides a way to estimate the imprecision of program analysis and hence it helps the user in compressing the resulting error below a given bound. We believe that *viewing program analysis as an approximate computation*, as we are used to in numerical analysis, can provide a

paradigm shift in the way we use/design abstract domains. This makes the error component, which is unavoidable in program analysis, clear in particular to non-expert users, hence providing the appropriate tools to fully control its propagation.

As future work, we plan to modify the proof system in Fig. 7 in order to deal with widening operators. By definition, widening may easily inject unbound errors in the iterates of the abstract interpreter. To this end, it is interesting to observe that the result of a widening operator over a non-ACC abstract domain A and for a program P with abstract input $S^\sharp$, can be considered as analyzing P with input $S^\sharp$ over a new ACC abstract domain which is built over A by the widening operator chosen and specialized with the program P and input $S^\sharp$.

Our proof system types any program/input pair (P, S) with a bound of the error induced by the abstract interpreter applied to (P, S). In theory we can see the proof system as an abstract interpretation itself, and use our same framework to estimate its power/quality at a higher type. All these ideas relate to a paper by Cousot et al. [2019] and can be considered as a future work to evaluate the precision of our proposed proof system.

Furthermore, we aim to define the notion of *partial completeness cliques*, following the first definition in complexity theory by Asperti [2008], hence extending the results in [Bruni et al. 2020] to the case of partial completeness. As in the case of complete abstractions for a given program, also in the case of partial complete abstract domains the underlying lattice structure plays a central role. For this reason, we plan to study the existence of minimal domain transformers that can ensure the ε-partial completeness of the analysis w.r.t. a given program and constant ε, as done in [Giacobazzi et al. 2000] for the case of standard completeness.

Our work has a strong connection to code obfuscation [Collberg and Nagra 2009]. Code obfuscation are program transformations explicitly designed to degrade the results of program analysis, namely to induce imprecision, and therefore incompleteness [Giacobazzi 2008; Giacobazzi and Mastroeni 2012; Giacobazzi et al. 2017]. Being able to control and quantify the amount of imprecision induced in the abstract interpretation by a code obfuscating program transformation, could allow us to measure the potency of these transformations. This is still one of the main open challenges in software protection [Ceccato et al. 2019; Collberg et al. 2011; Sutter et al. 2019]. In this perspective, our method would provide a very first method to semantically quantify the potency of any code transformation techniques that aim to reduce the precision of program analysis.

Although the decidability requirement of the predicate $\delta_A(a, b) \leq \varepsilon$ resembles the decidability of a Blum's complexity measure [Blum 1967], our definition of quasi-metrics A-compatible is not a Blum complexity measure. We plan to deepen our study in order to formalize a measure of imprecision for abstract interpretation that satisfies the Blum's axioms in order to define partial completeness classes as complexity classes.

We also plan to extend the concept of partial complete abstract interpretation to non-GCs/GIs. For example, to the case of convex polyhedra [Cousot and Halbwachs 1978] and symbolic abstract domains, such as the domain of regular or indexed grammars [Campion et al. 2019].

ACKNOWLEDGMENTS

We wish to thank the anonymous reviewers of POPL2022 for their detailed comments. This work has been partially supported by the grant PRIN2017 (code: 201784YSZ5) "AnalysiS of PRogram Analyses (ASPRA)" and the project "Dipartimenti di Eccellenza 2018-2022" funded by the Italian Ministry of Education, Universities and Research (MIUR).

REFERENCES

Andrea Asperti. 2008. The intensional content of Rice's theorem. *ACM SIGPLAN Notices* 43, 1 (2008), 113–119. https://doi.org/10.1145/1328438.1328455

Manuel Blum. 1967. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM (JACM)* 14, 2 (1967), 322–336. https://doi.org/10.1145/321386.321395

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, Isabel Garcia-Contreras, and Dusko Pavlovic. 2020. Abstract extensionality: on the properties of incomplete abstract interpretations. *PACMPL* 4, POPL (2020), 28:1–28:28. https://doi.org/10.1145/3371096

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. 2021. A Logic for Locally Complete Abstract Interpretations. In *Proc. 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2021)*. IEEE Computer Society, 1–13. https://doi.org/10.1109/LICS52264.2021.9470608 Distinguished paper.

Marco Campion, Mila Dalla Preda, and Roberto Giacobazzi. 2019. Abstract Interpretation of Indexed Grammars. In *International Static Analysis Symposium*. Springer, 121–139. https://doi.org/10.1007/978-3-030-32304-2_7

Ignacio Casso, José F Morales, Pedro López-García, Roberto Giacobazzi, and Manuel V. Hermenegildo. 2019. Computing abstract distances in logic programs. In *International Symposium on Logic-Based Program Synthesis and Transformation*. Springer, 57–72. https://doi.org/10.1007/978-3-030-45260-5_4

Mariano Ceccato, Paolo Tonella, Cataldo Basile, Paolo Falcarin, Marco Torchiano, Bart Coppens, and Bjorn De Sutter. 2019. Understanding the behaviour of hackers while performing attack tasks in a professional setting and in a public challenge. *Empir. Softw. Eng.* 24, 1 (2019), 240–286. https://doi.org/10.1007/s10664-018-9625-6

Christian Collberg and Jasvir Nagra. 2009. *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection*. Addison-Wesley Professional.

Christian S. Collberg, Jack W. Davidson, Roberto Giacobazzi, Yuan Xiang Gu, Amir Herzberg, and Fei-Yue Wang. 2011. Toward Digital Asset Protection. *IEEE Intelligent Systems* 26, 6 (2011), 8–13. https://doi.org/10.1109/MIS.2011.106

Patrick Cousot. 2021. *Principles of Abstract Interpretation*. The MIT Press, Cambridge, Mass.

Patrick Cousot and Radhia Cousot. 1976. Static determination of dynamic properties of programs. In *Proceedings of the 2nd International Symposium on Programming*. Dunod, Paris, 106–130. https://doi.org/10.1145/390019.808314

Patrick Cousot and Radhia Cousot. 1977. 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, 238–252. https://doi.org/10.1145/512950.512973

Patrick Cousot and Radhia Cousot. 1979. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM Press, 269–282. https://doi.org/10.1145/567752.567778

Patrick Cousot and Radhia Cousot. 1992a. Abstract interpretation frameworks. *J. Logic and Comput.* 2, 4 (1992), 511–547. https://doi.org/10.1093/logcom/2.4.511

Patrick Cousot and Radhia Cousot. 1992b. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation (Invited Paper). In *Proc. of the 4th Internat. Symp. on Programming Language Implementation and Logic Programming (PLILP '92) (Lecture Notes in Computer Science, Vol. 631)*, M. Bruynooghe and M. Wirsing (Eds.). Springer-Verlag, 269–295. https://doi.org/10.1007/3-540-55844-6_142

Patrick Cousot, Roberto Giacobazzi, and Francesco Ranzato. 2018. Program analysis is harder than verification: A computability perspective. In *International Conference on Computer Aided Verification*. Springer, 75–95. https://doi.org/10.1007/978-3-319-96142-2_8

Patrick Cousot, Roberto Giacobazzi, and Francesco Ranzato. 2019. A^{2}I: Abstract^2 Interpretation. *Proc. ACM Program. Lang.* 3, POPL, Article 42 (Jan. 2019), 31 pages. https://doi.org/10.1145/3290355

Patrick Cousot and Nicolas Halbwachs. 1978. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM Press, New York, NY, Tucson, Arizona, 84–97. https://doi.org/10.1145/512760.512770

Federico Crazzolara. 1997. Quasi-metric Spaces as Domains for Abstract Interpretation. In *1997 Joint Conf. on Declarative Programming, APPIA-GULP-PRODE'97, Grado, Italy, June 16-19, 1997*, Moreno Falaschi, Marisa Navarro, and Alberto Policriti (Eds.). 45–56.

Alessandra Di Pierro and Herbert Wiklicky. 2000. Measuring the precision of abstract interpretations. In *International Workshop on Logic-Based Program Synthesis and Transformation*. Springer, 147–164.

Dino Distefano, Manuel Fähndrich, Francesco Logozzo, and Peter W. O'Hearn. 2019. Scaling static analyses at Facebook. *Commun. ACM* 62, 8 (2019), 62–70. https://doi.org/10.1145/3338112

Roberto Giacobazzi. 2008. Hiding Information in Completeness Holes - New perspectives in code obfuscation and watermarking. In *Proc. of The 6th IEEE International Conferences on Software Engineering and Formal Methods (SEFM'08)*. IEEE Press., 7–20. https://doi.org/10.1109/SEFM.2008.41

Roberto Giacobazzi, Francesco Logozzo, and Francesco Ranzato. 2015. 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*, Sriram K. Rajamani and David Walker (Eds.). ACM, 261–273. https://doi.org/10.1145/2676726.2676987

Roberto Giacobazzi and Isabella Mastroeni. 2012. Making abstract interpretation incomplete: Modeling the potency of obfuscation. In *International Static Analysis Symposium*. Springer, 129–145. https://doi.org/10.1007/978-3-642-33125-1_11

Roberto Giacobazzi, Isabella Mastroeni, and Mila Dalla Preda. 2017. Maximal incompleteness as obfuscation potency. *Formal Aspects of Computing* 29, 1 (2017), 3–31. https://doi.org/10.1007/s00165-016-0374-2

Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. 2000. Making Abstract Interpretation Complete. *Journal of the ACM* 47, 2 (March 2000), 361–416. https://doi.org/10.1145/333979.333989

Dexter Kozen. 1997. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 19, 3 (1997), 427–443. https://doi.org/10.1145/256167.256195

Vincent Laviron and Francesco Logozzo. 2009. Refining Abstract Interpretation-Based Static Analyses with Hints. In *Proc. of APLAS'09 (Lecture Notes in Computer Science, Vol. 5904)*. Springer-Verlag, 343–358. https://doi.org/10.1007/978-3-642-10672-9_24

Francesco Logozzo. 2009. Towards a Quantitative Estimation of Abstract Interpretations. In *Workshop on Quantitative Analysis of Software* (workshop on quantitative analysis of software ed.). Microsoft. https://www.microsoft.com/en-us/research/publication/towards-a-quantitative-estimation-of-abstract-interpretations/

Antoine Miné. 2017. Tutorial on Static Inference of Numeric Invariants by Abstract Interpretation. *Foundations and Trends in Programming Languages* 4, 3-4 (2017), 120–372. https://doi.org/10.1561/2500000034

Hartley Rogers. 1992. *Theory of recursive functions and effective computability*. The MIT press.

Pascal Sotin. 2010. *Quantifying the precision of numerical abstract domains*. Technical Report HAL Id: inria-00457324. INRIA. https://hal.inria.fr/inria-00457324

Bjorn De Sutter, Christian S. Collberg, Mila Dalla Preda, and Brecht Wyseur. 2019. Software Protection Decision Support and Evaluation Methodologies (Dagstuhl Seminar 19331). *Dagstuhl Reports* 9, 8 (2019), 1–25. https://doi.org/10.4230/DagRep.9.8.1

Arnaud Venet. 1996. Abstract cofibered domains: Application to the alias analysis of untyped programs. In *International Static Analysis Symposium*. Springer, 366–382. https://doi.org/10.1007/3-540-61739-6_53

Wallace Alvin Wilson. 1931. On quasi-metric spaces. *American Journal of Mathematics* 53, 3 (1931), 675–684. https://doi.org/10.2307/2371174

Glynn Winskel. 1993. *The formal semantics of programming languages: an introduction*. MIT press.