How fitting is your abstract domain?

Roberto Giacobazzi[1], Isabella Mastroeni[2], and Elia Perantoni[2]

[1] University of Arizona Tucson - Department of Computer Science
rgiacobazzi@gmail.com
[2] University of Verona - Computer Science Department
isabella.mastroeni@univr.it and elia.perantoni@studenti.univr.it

Abstract. Abstract interpretation offers sound and decidable approximations for undecidable queries related to program behavior. The effectiveness of an abstract domain is entirely reliant on the abstract domain itself, and the worst-case scenario is when the abstract interpreter provides a response of "don't know", indicating that anything could happen during runtime. Conversely, a desirable outcome is when the abstract interpreter provides information that exceeds a specified level of precision, resulting in a more precise answer. The concept of completeness relates to the level of precision that is forfeited when performing computations within the abstract domain. Our focus is on the domain's ability to express program behaviour, which we refer to as adequacy. In this paper, we present a domain refinement strategy towards adequacy and a simple sound proof system for adequacy, designed to determine whether an abstract domain is capable of providing satisfactory responses to specified program queries. Notably, this proof system is both language and domain agnostic, and can be readily incorporated to support static program analysis.

Keywords: Abstract interpretation · Abstract domain precision · Static analysis.

1 Introduction

The accuracy of an abstract interpretation depends upon many factors [13]. (1) The quality of the abstract domain: In this case the abstract domain has to represent in the most precise way all the intermediate invariants that hold at each program point along a computation trace [10, 11, 19, 5, 6]. (2) The precision of the fixpoint strategy: In this case an appropriate fixpoint strategy can improve the precision of the analysis either by delaying widening/narrowing or dynamic trace partition [8, 2, 12, 27]. (3) The way the code is written: In this case, the non-compositional nature of the precision of abstract interpretation can be influenced by the way the code is assembled [20, 3]. All these factors imply that the design of an optimal (aka, sound and complete) abstract interpretation for static program analysis is a very complex task.

In this paper, we address the first of the above mentioned factors: The quality of the abstract domain. The standard approach is based on the notion of abstract domain refinement [14]. The goal of domain refinement is to remove false alarms by enhancing the expressive power of the abstract domain. In particular, in the last two decades, the notion of completeness, with their global and local refinements [17, 5]—the latter called Abstract Interpretation Repair (AIR), perfectly captures the structure of an abstract domain that will produce no false alarm for a given program. This notion has also been weakened towards what is known as *partial completeness* [7] where the precision is evaluated in a metric space built over the abstract domain.

Rather than focusing on eliminating false alarms to achieve completeness, or on weaker forms of completeness that tolerate some level of false alarms, our study is concerned with evaluating the expressiveness of an abstract domain with respect to its ability to represent intermediate invariants in the computation. In particular, when dealing with abstract analysis, it is widely recognized that an abstract domain must allow for the possibility of making no statements regarding the behavior of a computation. This characteristic is critical for enabling the analysis to provide decidable answers to otherwise undecidable questions. The primary objective of adequacy is to prevent excessively imprecise statements with respect to a given concrete assertion. When the level of imprecision exceeds a certain threshold, it is deemed unacceptable, namely not *fitting*, for the purposes of abstract computation. This domain property will be referred to as *adequacy* of the abstract domain.

Like completeness, an abstract domain may be deemed not *adequate* because it is overly abstract. In this scenario, we may question whether it is feasible to *adjust* the abstract domain to ensure adequacy, just as it is possible to eliminate false alarms for achieving completeness by refinement [17, 5]. This proposed approach to achieving adequacy involves a refinement strategy that ensures the refined abstract domain approximates computed invariants more accurately than a specified bound τ. However, precisely as it happens for completeness, while the refinement approach is crucial for gaining a deep understanding of adequacy by identifying the elements necessary for enforcing it, it needs the computation of the function/semantics, which may render it an undecidable characterization.

To address this issue, in the context of program analysis, we also introduce a simple proof system, based on structural induction, to verify whether the approximate invariants generated by the abstract interpreter within an abstract domain A are kept below a the given (abstract) bound τ. This is achieved by proving the validity of triples holding on a program $\mathtt{r}$ w.r.t. an input concrete assertions $\mathtt{c}$ and an output one $\mathtt{d}$. If $[\![\mathtt{r}]\!]^\sharp_\mathsf{A}$ is the abstract semantics of $\mathtt{r}$ given by an abstract interpreter defined on the abstract domain A, namely $[\![\mathtt{r}]\!]^\sharp_\mathsf{A} : \mathsf{A} \to \mathsf{A}$, then if the triple, denoted $\tau \vdash_\mathsf{A} \langle \mathtt{c} \rangle\ \mathtt{r}\ \langle \mathtt{d} \rangle$, is derived in the proof system then we can say that A is adequate for $\mathtt{r}$, namely the computation of $\mathtt{r}$ semantics on A is kept under the bound τ. The proof system we introduce, which we refer to as the *abstract domain adequacy logic*, is very simple and can be efficiently checked online using program analysis tools.

Paper roadmap. Sect. 2 provides an overview of abstract interpretation and programming language semantics. We chose to treat regular commands as the general programming language to establish a language-agnostic framework. In Sect. 3, we recall abstract domain completeness and introduce the novel concept of abstract domain adequacy. Sect. 4 outlines the procedure for *adjusting* an abstract domain towards adequacy by refining it. Finally, in Sect. 5, we present a sound proof system for adequacy, and Sect. 6 concludes the paper with closing remarks and potential future directions.

Related works. Adequacy is a novel approach here proposed for dealing with abstract domain precision. Completeness in abstract interpretation [17] is the closest notion to adequacy, at least in its origin. As we will demonstrate in Sect. 3, the two concepts are incomparable, meaning that neither one is stronger than the other. Nonetheless, the proof system and domain refinement procedures for both concepts employ similar strategies. Specifically, the proof system inherits the locality of the notion of *local completeness* introduced in [4] and [6], which also forms the basis for the refinement strategy in *abstract interpretation repair* (AIR) [5]. As a consequence, the proof systems for global completeness in [19] and local completeness in [6] are logically incomparable with ours. However, the idea of setting a bound on the approximation, which may be weaker than the abstract observation, is a novel aspect of our work.

In the context of local adequacy, as it happens AIR, in general, no optimal (most abstract) refinement exists. Therefore, the process of achieving adequacy can only provide sub-optimal solutions. It should be noted that completeness and adequacy are distinct problems, and as such, the computed refinements produced by the two methods may lead to different domains. An adequate domain may still be incomplete, and thus not a solution in terms of AIR, and vice versa, a solution in terms of AIR may not ensure a bounded abstract computation, and therefore may not be adequate. The technical differences are discussed in Section 3.

In [7], the concept of partial completeness is introduced as a means of evaluating the accuracy of an abstract interpretation. This is accomplished by incorporating the abstract domain into a (quasi) metric space, which relaxes the requirement for local completeness to hold up to a metric neighbor of the exact (complete) solution. Along with completeness, partial completeness is also not comparable to adequacy. The latter is not intended to provide any quantitative estimate of the quality of an abstraction but rather an answer of whether the resulting invariant is below (is approximated by) a given bound τ in the abstract domain. This guarantees that at least the information in τ will be included in the computed approximate invariant.

2 Background

If $\mathtt{S}$, $\wp(\mathtt{S})$ denotes the powerset of $\mathtt{S}$. If $f : \mathtt{S} \to \mathtt{T}$, then we often abuse notation by calling f also its additive lifting $f : \wp(\mathtt{S}) \to \wp(\mathtt{T})$ to sets of values:

$f(X) \stackrel{\text{def}}{=} \{\ f(x) \,|\, x \in X \subseteq \mathtt{S}\ \}$. If $f : \mathtt{S} \longrightarrow \mathtt{T}$ and $g : \mathtt{T} \to \mathtt{U}$, we denote by $g \circ f$ (or simply gf) their composition. If $f : \mathtt{S} \to \mathtt{S}$, and $n \in \mathbb{N}$ we define $f^n : \mathtt{S} \to \mathtt{S}$ inductively as $f^0 \stackrel{\text{def}}{=} id_{\mathtt{S}}$ (the identity on $\mathtt{S}$), $f^{n+1} \stackrel{\text{def}}{=} f \circ f^n$. In a partial ordered structure C, we use $\leq_{\mathsf{C}}$ to denote the partial order relation, $\vee_{\mathsf{C}}$ for lub, $\wedge_{\mathsf{C}}$ for glb, $\top_{\mathsf{C}}$ and $\bot_{\mathsf{C}}$ for respectively the greatest and the least elements (we avoid the pedex C when clear from the context). A function f between ordered structures is monotone if it preserve the order, i.e., $\mathtt{c} \leq_{\mathsf{C}} \mathtt{d} \Rightarrow f(\mathtt{c}) \leq_{\mathsf{C}} f(\mathtt{d})$. It is additive if it preserves arbitrary lubs (co-additivity is dually defined).

2.1 Abstract Interpretation

Abstract interpretation [10, 11], is a formal framework for approximating programs semantics defined on a concrete domain C, by means of some abstraction A of C. Given complete lattices C and A, a pair of functions $\alpha : \mathsf{C} \to \mathsf{A}$ and $\gamma : \mathsf{A} \to \mathsf{C}$ forms a Galois connection (GC for shorts) if for any $\mathtt{c} \in \mathsf{C}$ and $\mathtt{a} \in \mathsf{A}$ we have $\alpha(\mathtt{c}) \leq_{\mathsf{A}} \mathtt{a} \Leftrightarrow \mathtt{c} \leq_{\mathsf{C}} \gamma(\mathtt{a})$. In this case, α (resp. γ) is the abstraction/left adjoint (resp. concretization/right adjoint), and it is additive (resp. co-additive). Co-additive functions $g : \mathsf{A} \to \mathsf{C}$ admits left adjoint $g^- \stackrel{\text{def}}{=} \lambda \mathtt{c}.\ \bigwedge_{\mathsf{A}} \{\ \mathtt{a} \,|\, \mathtt{c} \leq_{\mathsf{C}} f(\mathtt{a})\ \}$. An *upper closure operator* (uco for shorts) $\rho : \mathsf{C} \to \mathsf{C}$ on a poset C is monotone, idempotent, and extensive, i.e., $\forall x \in \mathsf{C}.\ \mathtt{c} \leq_{\mathsf{C}} \rho(\mathtt{c})$. If in a GC $\alpha \circ \gamma = id_{\mathsf{A}}$ then it is a Galois insertion (GI) and $\gamma \circ \alpha$, simply written $\gamma\alpha$, is an uco. Let us denote by $Abs(\mathsf{C})$ the class of abstract domains (GI or uco) of C. It is well known that $Abs(\mathsf{C})$ is isomorphic to the lattice of all ucos on C, therefore when dealing with GI and clear from the context, we abuse notation by denoting as A both the domain of abstract objects ($\mathsf{A} = \gamma\alpha(\mathsf{C})$) and the closure operator ($\mathsf{A} = \gamma\alpha$). Given an abstract domain $\mathsf{A} \in Abs(\mathsf{C})$ ($\mathsf{A} = \gamma\alpha$) and a concrete function $f : \mathsf{C} \to \mathsf{C}$, an abstract function $f^\sharp_{\mathsf{A}} : \mathsf{A} \to \mathsf{A}$ is a *sound* approximation of f when $\alpha \circ f \leq_{\mathsf{A}} f^\sharp_{\mathsf{A}} \circ \alpha$[3]. The best correct approximation (bca) of f in A is the function $f^{\mathsf{A}} \stackrel{\text{def}}{=} \alpha \circ f \circ \gamma$, any other abstraction is less precise. In this case soundness as domain property is defined as $\mathsf{A} \circ f \leq_{\mathsf{C}} \mathsf{A} \circ f \circ \mathsf{A}$.

2.2 Regular Commands

Following [28, 4] (see also [29]) we consider the language $\mathrm{Reg}_{\mathrm{Exp}}$ of regular commands in the top of Fig. 1 (where $\oplus$ denotes non-deterministic choice and $*$ is the Kleene closure), parametric on a grammar of expressions Exp. This language is general enough to represent control-flow graphs of basic expressions and therefore it covers simple deterministic imperative languages.

The Concrete Semantics. Let $\mathrm{Reg}_{\mathrm{Exp}}$ be a regular language. We assume the basic transfer expressions have a semantics $(\!|\cdot|\!) : \mathrm{Exp} \to \mathsf{C} \to \mathsf{C}$ on a complete lattice C such that $(\!|\mathtt{e}|\!)$ is an additive function. The concrete semantics [28]

[3] By $\leq_{\mathsf{A}}$ we denote the partial order relation on A.

$$\mathrm{Reg}_{\mathrm{Exp}} \ni \mathtt{r} ::= \mathtt{e} \mid \mathtt{r};\mathtt{r} \mid \mathtt{r} \oplus \mathtt{r} \mid \mathtt{r}^* \qquad \mathtt{e} \in \mathrm{Exp}$$

$$\begin{aligned} \mathfrak{L} &= \mathrm{Reg}_{\mathfrak{L}\,\mathrm{Exp}} \\ \mathfrak{L}\,\mathrm{Exp} \ni \mathtt{e} &::= \mathbf{skip} \mid x := \mathtt{a} \mid \mathtt{b}? \\ \mathrm{AExp} \ni \mathtt{a} &::= x \mid n \mid \mathtt{a} + \mathtt{a} \mid \mathtt{a} - \mathtt{a} \mid \mathtt{a} * \mathtt{a} \\ \mathrm{BExp} \ni \mathtt{b} &::= \mathbf{tt} \mid \mathbf{ff} \mid \mathtt{a} = \mathtt{a} \mid \mathtt{a} \leq \mathtt{a} \mid \mathtt{a} < \mathtt{a} \mid \mathtt{b} \wedge \mathtt{b} \mid \neg \mathtt{b} \\ \mathit{Var} \ni x &\text{ (variables)}, \qquad n \in \mathbb{Z} \text{ (values)} \end{aligned}$$

Fig. 1. The syntax of $\mathrm{Reg}_{\mathrm{Exp}}$, parametric on Exp, and of $\mathfrak{L}$.

$[\![\cdot]\!] : \mathrm{Reg}_{\mathrm{Exp}} \to \mathsf{C} \to \mathsf{C}$ of regular commands is inductively defined as follows: Let $\mathtt{c} \in \mathsf{C}$

$$\begin{array}{ll} [\![\mathtt{e}]\!]\mathtt{c} \stackrel{\text{def}}{=} (\!|\mathtt{e}|\!)\mathtt{c} & [\![\mathtt{r}_1 \oplus \mathtt{r}_2]\!]\mathtt{c} \stackrel{\text{def}}{=} [\![\mathtt{r}_1]\!]\mathtt{c} \vee_{\mathsf{c}} [\![\mathtt{r}_2]\!]c \\ [\![\mathtt{r}_1;\mathtt{r}_2]\!]\mathtt{c} \stackrel{\text{def}}{=} [\![\mathtt{r}_2]\!]([\![\mathtt{r}_1]\!]\mathtt{c}) & [\![\mathtt{r}^*]\!]\mathtt{c} \stackrel{\text{def}}{=} \bigvee_{\mathsf{c}} \left\{ \; [\![\mathtt{r}]\!]^n\mathtt{c} \;\middle|\; n \in \mathbb{N} \; \right\} \end{array}$$

The Abstract Semantics. Let $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$, the abstract semantics of regular commands $[\![\cdot]\!]^\sharp_{\mathsf{A}} : \mathrm{Reg}_{\mathrm{Exp}} \to \mathsf{A} \to \mathsf{A}$ on the abstract domain A is defined by structural induction as follows:

$$\begin{aligned} [\![\mathtt{e}]\!]^\sharp_{\mathsf{A}}\mathtt{a} &\stackrel{\text{def}}{=} [\![\mathtt{e}]\!]^{\mathsf{A}}\mathtt{a} \\ [\![\mathtt{r}_1;\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\mathtt{a} &\stackrel{\text{def}}{=} [\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}([\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}}\mathtt{a}) \\ [\![\mathtt{r}_1 \oplus \mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\mathtt{a} &\stackrel{\text{def}}{=} [\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}}\mathtt{a} \vee_{\mathsf{A}} [\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\mathtt{a} \\ [\![\mathtt{r}^*]\!]^\sharp_{\mathsf{A}}\mathtt{a} &\stackrel{\text{def}}{=} \bigvee_{\mathsf{A}} \left\{ \; ([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^n\mathtt{a} \;\middle|\; n \in \mathbb{N} \; \right\} \end{aligned}$$

where we recall that $[\![\mathtt{e}]\!]^{\mathsf{A}}$ is the bca in A of $[\![\mathtt{e}]\!]$.
By structural induction we can prove that this abstract semantics is monotonic and correct, i.e., $\alpha \circ [\![\mathtt{r}]\!] \leq_{\mathsf{A}} [\![\mathtt{r}]\!]^\sharp_{\mathsf{A}} \circ \alpha$ (or equivalentely $\alpha \circ [\![\mathtt{r}]\!] \circ \gamma \leq_{\mathsf{A}} [\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}$).

Programs. We consider standard basic transfer functions for expressions used in deterministic while programs: no-op instruction, assignments and Boolean guards, i.e., we consider the regular language $\mathfrak{L}$ defined in Fig. 1. Hence, we have to deal with integer variables and with stores. Let us denote $\mathit{Var}(\mathtt{r})$ the set of all the variables in $\mathtt{r} \in \mathrm{Reg}_{\mathrm{Exp}}$, and let $\mathsf{C} \stackrel{\text{def}}{=} \wp(\mathbb{M})$ the concrete domain of sets of stores, where the store $\mathtt{m} \in \mathbb{M}$ is a function associating values to a set of variables, i.e., $\mathtt{m} : V \to \mathbb{Z}$, $V \subseteq_f \mathit{Var}$ (finite subset).
In particular, the basic transfer function semantics $(\!|\mathtt{e}|\!) : \mathsf{C} \to \mathsf{C}$ for the expressions of $\mathfrak{L}$, is defined as: $\mathtt{M} \in \mathsf{C}$ (i.e., $\mathtt{M} \subseteq \mathbb{M}$)

$$\begin{aligned} (\!|\mathbf{skip}|\!)\mathtt{M} &\stackrel{\text{def}}{=} \mathtt{M} \\ (\!|x := \mathtt{a}|\!)\mathtt{M} &\stackrel{\text{def}}{=} \left\{ \; \mathtt{m}[x \mapsto (\!|\mathtt{a}|\!)\,\mathtt{m} \;\middle|\; \mathtt{m} \in \mathtt{M} \; \right\} \\ (\!|\mathtt{b}?|\!)\mathtt{M} &\stackrel{\text{def}}{=} \left\{ \; \mathtt{m} \in \mathtt{M} \;\middle|\; (\!|\mathtt{b}|\!)\,\mathtt{m} = \mathbf{tt} \; \right\} = \mathtt{M} \cap (\!|\mathtt{b}|\!) \end{aligned}$$

where $(\!|\mathsf{a}|\!) : \mathbb{M} \to \mathbb{Z}$ and $(\!|\mathsf{b}|\!) : \mathbb{M} \to \{\mathbf{tt}, \mathbf{ff}\}$ are the standard evaluation semantics for arithmetic and boolean expressions, respectively, and where we denote $(\!|\mathsf{b}|\!) \stackrel{\text{def}}{=} \{\ \mathbb{m} \in \mathbb{M} \mid (\!|\mathsf{b}|\!)\,\mathbb{m} = \mathbf{tt}\ \}$ the truth semantics of b.
Note that, the concrete semantics of regular language defined above instantiated to $\mathfrak{L}$ corresponds precisely to the denotational semantics defined [9] starting from standard operational semantics of non deterministic choice and iteration [29].

3 From Completeness to Adequacy

Abstract domain completeness is the standard approach used for the characterization of abstract domain precision w.r.t. the computation of the semantics [17]. Let us recall its formal definition.

3.1 Abstract Domain Completeness and its limits

Let $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$ be an abstraction of C and $f : \mathsf{C} \to \mathsf{C}$ a concrete computation on C, e.g., the program semantics, then the abstract function $f^\sharp_\mathsf{A}$ is said to be a *complete/precise* [11, 17] approximation of f on A if $\alpha \circ f = f^\sharp_\mathsf{A} \circ \alpha$. Intuitively, it means that if we abstract $\mathsf{c} \in \mathsf{C}$ (the input of f), we apply the f approximation $f^\sharp_\mathsf{A}$, we obtain the same abstract element that we would have been obtained by abstracting the result of f applied directly on c (without an initial abstraction). Note that, if there exists a complete approximation $f^\sharp_\mathsf{A}$ of f, then f^A is itself complete. Completeness of f^A intuitively means that f^A is the most precise approximation of f and, in therefore, completeness can be characterized as a domain property. $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$ is said to be a complete abstraction for f if $\mathsf{A} \circ f \circ \mathsf{A} = \mathsf{A} \circ f$.

In a more general setting, let $f : \mathsf{C}_1 \to \mathsf{C}_2$ be a function on complete lattices C_i (potentially different), and let $\mathsf{A}_i \in \mathit{Abs}(\mathsf{C}_i)$ (for $i \in \{1, 2\}$) be abstractions, respectively, of input and output domains. In this case, we say that $\langle \mathsf{A}_1, \mathsf{A}_2 \rangle$ is a pair of complete abstract domains for f if $\mathsf{A}_2 \circ f = \mathsf{A}_2 \circ f \circ \mathsf{A}_1$.

Note that, this is a global property, since it requires the equality of two functions on all inputs, i.e., $\forall \mathsf{c} \in \mathsf{C}.\ \mathsf{A} \circ f(\mathsf{c}) = \mathsf{A} \circ f \circ \mathsf{A}(\mathsf{c})$. This makes completeness an extremely strong property and indeed, as proved in [4], it holds for all programs in a Turing complete programming language only for trivial abstract domains. This means that the only abstract domains that are complete for all programs are the straightforward ones: the identical abstraction, making abstract and concrete semantics the same, and the top abstraction, making all programs equivalent by abstract semantics. In particular, the authors show that while global completeness can be hard/impossible to achieve, it could well happen that completeness holds locally, i.e. just for some store properties, proving so far what is called local completeness [4], namely $\mathsf{A} \circ f(\mathsf{c}) = \mathsf{A} \circ f \circ \mathsf{A}(\mathsf{c})$ for a fixed point $\mathsf{c} \in \mathsf{C}$. This weakening makes precision strongly dependent on the point c, implying that among points with the same abstraction we may have

both complete and incomplete points.

However, both completeness characterizations do not really deal with the loss of precision due to the *choice* of the abstract observation, since it characterizes only whether there is an *extra* loss of precision due to the *computation* on observed/abstracted data (compared with the observation of the concretely computed result). For instance, the $\top$ abstraction, which cannot distinguish any information, is trivially complete, even if it represents the total loss of precision in the observation of data. Therefore, completeness is unable to account for this type of information loss caused by abstraction, which is why we strive to introduce a new concept of precision for abstract domains that can designate $\top$ as entirely imprecise. Graphically, the idea is depicted in Fig. 2 for program semantics. Namely, (local) completeness can only avoid the error depicted as blue area, namely the error due to computation on abstract data compared with the abstraction of the concrete computation, while we aim at bounding the total error depicted, namely the error due to the choice of the abstract domain, independently from its potential completeness.

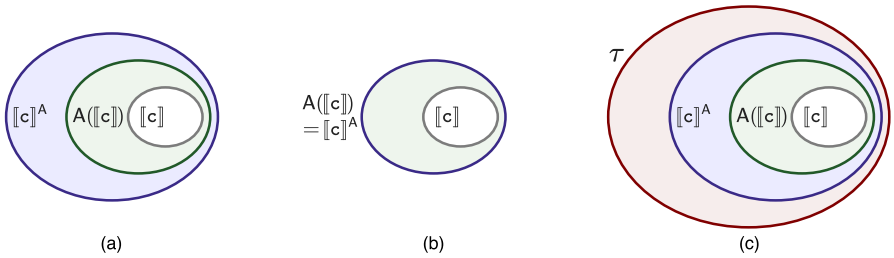

Fig. 2. Program semantics abstraction (a). Complete semantic abstraction (b). Adequacy (c).

3.2 Abstract Domain Adequacy

In order to move the attention to the whole (abstract) image of all the elements with the same abstraction, loosing the dependency on the chosen point imposed by local completeness while keeping a locality of the property, we propose a novel approach referred to as abstract domain adequacy. We can say that abstract domain adequacy allows us to bound (strictly below a fixed threshold) the total amount of approximation due to data abstraction and abstract computation[4]. The idea is to fix a generic bound τ (at the limit $\top$) that we want to confine the loss of precision, namely such that any abstract computation is strictly over-approximated by τ.

[4] Similarly to what happens with completeness, adequacy of any sound abstract operator implies the adequacy of the best correct approximation.

Definition 1 (Abstract domain adequacy w.r.t. τ).
Let C *be a concrete domain,* $f : \mathsf{C} \to \mathsf{C}$, $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$ *one of its abstractions,* $\tau \in \mathsf{A}$ *(i.e.,* $\mathsf{A} = \gamma\alpha$ *and* $\gamma\alpha(\tau) = \tau \in \mathsf{C}$*). Then* A *is adequate w.r.t.* τ *for* f *if*

(Global) $\widetilde{\mathbb{C}}^{\mathsf{A}}(f)_\tau \Leftrightarrow \forall \mathsf{c} \in \mathsf{C}.\, \mathsf{A} \circ f \circ \mathsf{A}(\mathsf{c}) \lesssim_{\mathsf{c}} \tau$ *and*
(Local) $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathsf{c}}(f)_\tau \Leftrightarrow \mathsf{A} \circ f \circ \mathsf{A}(\mathsf{c}) \lesssim_{\mathsf{c}} \tau$

when $\tau = \top$ *we simply call* A *adequate for* f*, denoted* $\widetilde{\mathbb{C}}^{\mathsf{A}}(f)$ *and* $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathsf{c}}(f)$*.*

Example 1. Consider $\mathsf{C} = \wp(\mathbb{Z})$ and $\mathsf{A} = \mathit{Sign}$, whose abstract counterpart is depicted on the left of Fig. 3.

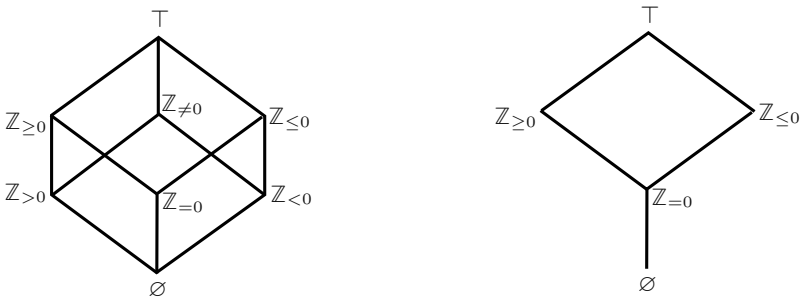

Fig. 3. Abstract domains for signs: *Sign* (left) and Sign_1 (right).

Let $f \stackrel{\text{def}}{=} \lambda \mathsf{c}.\, \{\, 2 * n \,|\, n \in \mathsf{c} \,\}$ and let $\mathsf{c} \stackrel{\text{def}}{=} \{0, 2, 4, 6, 8\}$, then we have $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathsf{c}}(f)$ since $\mathit{Sign} \circ f \circ \mathit{Sign}(\mathsf{c}) = \mathit{Sign} \circ f(\{\, n \in \mathbb{Z} \,|\, n \geq 0 \,\}) = \mathit{Sign}(\{\, 2n \,|\, n \geq 0, n \in \mathbb{Z} \,\}) = \{\, n \in \mathbb{Z} \,|\, n \geq 0 \,\} \subsetneq \mathbb{Z}$, but $\neg\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathsf{c}'}(f)$ with $\mathsf{c}' \stackrel{\text{def}}{=} \{-1, 0, 1\}$, since $\mathit{Sign} \circ f \circ \mathit{Sign}(\mathsf{c}') = \mathit{Sign} \circ f(\mathbb{Z}) = \mathit{Sign}(2\mathbb{Z}) = \mathbb{Z}$, which is the top, hence it does not hold globally, i.e., $\neg\widetilde{\mathbb{C}}^{\mathsf{A}}(f)$.
If we consider $g = \lambda \mathsf{c}.\, \{\, n^2 + 2 \,|\, n \in \mathsf{c} \,\}$, and we consider $\tau = \mathbb{Z}_{\geq 0}$, then we have that $\forall \mathsf{c}.\, \mathit{Sign} \circ g \circ \mathit{Sign}(\mathsf{c}) \subseteq \{\, n \in \mathbb{Z} \,|\, n > 0 \,\} \subsetneq \tau$. Hence, $\widetilde{\mathbb{C}}^{\mathsf{A}}(g)_\tau$.

In general, completeness and adequacy are not comparable when $\exists \mathsf{c}.\, \mathsf{A}f(\mathsf{c}) = \top$, since in this case we could have completeness, i.e., $\mathsf{A} \circ f(\mathsf{c}) = \mathsf{A} \circ f \circ \mathsf{A}(\mathsf{c})$ but we cannot satisfy adequacy, i.e., $\mathsf{A} \circ f \circ \mathsf{A}(\mathsf{c}) = \mathsf{A} \circ f(\mathsf{c}) = \top$. Following the other direction, it is clear that adequacy cannot imply completeness. A similar reasoning can be done when we consider adequacy w.r.t. $\tau \neq \top$. In general, we can prove the following relation.

Proposition 1. *Let* C *be a concrete domain,* $f : \mathsf{C} \to \mathsf{C}$ *and* $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$*. Then for any* $\tau \in \mathsf{A}$*, if* $\forall \mathsf{c} \in \mathsf{C}.\, \mathsf{A} \circ f(\mathsf{c}) \lesssim_{\mathsf{c}} \tau$ *then* A *complete imply* A *globally adequate w.r.t.* τ *for* f*.*

Example 2. Let us consider C and f in Example 1, we have shown above that $\neg\widetilde{\mathbb{C}}^{\mathsf{A}}(f)$, but it is trivial to show, and well known, that *Sign* is complete for f, since $\forall \mathsf{c} \in \mathsf{C}.\ \mathit{Sign} \circ f \circ \mathit{Sign}(\mathsf{c}) = \mathit{Sign} \circ f(\mathsf{c})$, since f leaves in the resulting set precisely the same signs that are in the input set, but $\top = \mathbb{Z} = \mathit{Sign} \circ f(\{-1, 0, 1\}) = \mathit{Sign}(\{-2, 0, 2\})$. Hence, in this case, completeness does not imply adequacy. On the other side, let $h \stackrel{\text{def}}{=} \lambda \mathsf{c}.\ \{\ (n+2)^2 \,|\, n \in \mathsf{c}\ \}$ of the same example, which satisfies adequacy (w.r.t. $\top = \mathbb{Z}$) since $\forall \mathsf{c} \in \mathsf{C}.\ \mathit{Sign} \circ h \circ \mathit{Sign}(\mathsf{c}) \subseteq \{\ n \in \mathbb{Z} \,|\, n \geq 0\ \}$, but it is not complete, e.g., $\mathit{Sign} \circ h \circ \mathit{Sign}(\{-1, 0, 1\}) = \mathit{Sign} \circ h(\mathbb{Z}) = \mathit{Sign}\{\ (n+2)^2 \,|\, n \in \mathbb{Z}\ \} = \{\ n \in \mathbb{Z} \,|\, n \geq 0\ \}$, while $\mathit{Sign} \circ h(\{-1, 0, 1\}) = \mathit{Sign}(\{1, 2, 3\}) = \{\ n \in \mathbb{Z} \,|\, n > 0\ \}$. Hence, in general, adequacy does not imply completeness.

3.3 Adequacy for $\mathfrak{L}$ programs

Let us consider now abstract domain adequacy for $\mathfrak{L}$ programs, namely where $f = [\![\mathtt{r}]\!]$, for some $\mathtt{r} \in \mathfrak{L}$. In this case, the concrete domain is $\mathsf{C} = \wp(\mathbb{M})$ and $\mathsf{A} \in \mathit{Abs}(\wp(\mathbb{M}))$ (namely $\leq_{\mathsf{C}}$ is $\subseteq$, while $\leq_{\mathsf{A}}$ still depends on the abstract domain, and in particular on the abstraction α) and we say that A is adequate for $\mathtt{r}$. For the sake of readability, we will write $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathsf{c}}(\mathtt{r})$ instead of $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathsf{c}}([\![\mathtt{r}]\!])$ (analogous for global adequacy). In this context, we look as properties of elements in $\mathfrak{L}$ related with adequacy.

In [4] the authors introduce a notion of expressability which has a strong relation with completeness. Formally, b is expressible in A if its truth semantics is an element of A, i.e., $(\!|\mathsf{b}|\!) \in \mathsf{A}$. In particular, the authors show that if the semantics of $\mathsf{b}?$ is local complete then b and $\neg\mathsf{b}$ are expressible, while the other implication does not always holds (Lemma III.2 [4]).
As far as adequacy is concerned, the situation is slightly different: while adequacy seems too weak to imply b to be expressible, it is instead implied by the expressability of b.

Definition 2 (b quasi-expressible w.r.t. τ). *Let $\mathsf{C} = \wp(\mathbb{M})$ be the concrete domain, $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$, $\tau \in \mathsf{A}$. A boolean expression b, such that $(\!|\mathsf{b}|\!) \leq \tau$, is quasi-expressible w.r.t. τ in A if there exists b' expressible in A, i.e, $(\!|\mathsf{b}'|\!) \in \mathsf{A}$, such that $(\!|\mathsf{b}|\!) \subseteq (\!|\mathsf{b}'|\!)$, $\mathsf{A}((\!|\mathsf{b}|\!)) = \mathsf{A}((\!|\mathsf{b}'|\!)) = (\!|\mathsf{b}'|\!) \subsetneq \tau$.*

In other words, a quasi-expressible, w.r.t. τ, boolean expression b has an abstract semantics approximated in A strictly under τ. It is trivial to observe that, if b is not quasi-expressible, w.r.t. τ then $\mathsf{A}((\!|\mathsf{b}|\!)) = \tau$. For instance, consider again the sign domains in Fig. 3. Then the boolean expressions $x \leq 0$ and $0 < x$ are expressible (and therefore quasi-expressible) in *Sign*, since $x \leq 0$ is expressed by $\{\ n \,|\, n \leq 0\ \}$ and $0 < x$ by $\{\ n \,|\, n < 0\ \}$. Note that $0 < x$ is only quasi-expressible, instead, in Sign_1. If we consider the interval domain *Int* and $\tau = [0, +\infty]$, then $10 < x$ is still expressible also w.r.t. τ since its semantics, i.e., $[11, +\infty]$ is strictly contained in τ, $x > 0 \wedge x \ \mathit{mod}\ 2 = 0$, i.e., the set of even numbers greater or equal to 2, is only quasi-expressible w.r.t. τ, since its semantics is contained in the semantics of $x \geq 2$, which is $[2, +\infty] \subsetneq \tau$, while

$x < 100$ is not expressible w.r.t. τ since its semantics is $[-\infty, 99]$, and there is no semantics strictly contained in τ which contains $[-\infty, 99]$.

Theorem 1. *Given* $\mathsf{b} \in \mathrm{BExp}$*, in the hypotheses of Def. 2*

1. *If* b *is quasi-expressible w.r.t* $\tau \in \mathsf{A}$ *then* $\widetilde{\mathbb{C}}^{\mathsf{A}}(\mathsf{b}?)_\tau$*;*
2. *If* b *is not quasi-expressible w.r.t.* τ *then* $\forall \mathsf{c} \in \mathsf{C}.\mathsf{A}(\mathsf{c}) \supseteq \tau$ *we have* $\neg\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathsf{c}}(\mathsf{b}?)_\tau$*.*

Proof.

1. Let b be quasi-expressible w.r.t. τ, hence there exists $\mathsf{b}' \in \mathrm{BExp}$ such that $\mathsf{A}(\langle\!|\mathsf{b}|\!\rangle) = \mathsf{A}(\langle\!|\mathsf{b}'|\!\rangle) = \langle\!|\mathsf{b}'|\!\rangle \subsetneq \tau$ and let $\mathsf{c} \in \mathsf{C}$. Then
$$\begin{aligned}\mathsf{A}[\![\mathsf{b}?]\!]\mathsf{A}(\mathsf{c}) = \mathsf{A}\langle\!|\mathsf{b}?|\!\rangle\mathsf{A}(\mathsf{c}) &= \mathsf{A}(\mathsf{A}(\mathsf{c}) \cap \langle\!|\mathsf{b}|\!\rangle) \subseteq \mathsf{A}(\mathsf{A}(\mathsf{c}) \cap \mathsf{A}(\langle\!|\mathsf{b}|\!\rangle)) \\ &= \mathsf{A}(\mathsf{A}(\mathsf{c}) \cap \mathsf{A}(\langle\!|\mathsf{b}'|\!\rangle)) = \mathsf{A}(\mathsf{c}) \cap \mathsf{A}(\langle\!|\mathsf{b}'|\!\rangle) \\ &\subseteq \mathsf{A}(\langle\!|\mathsf{b}'|\!\rangle) = \langle\!|\mathsf{b}'|\!\rangle \subsetneq \tau\end{aligned}$$
2. Suppose b is not quasi-expressible w.r.t. τ, i.e., $\mathsf{A}(\langle\!|\mathsf{b}|\!\rangle) = \tau$, and $\langle\!|\mathsf{b}|\!\rangle \subseteq \tau$, and suppose $\mathsf{A}(\mathsf{c}) \supseteq \tau$, then $\langle\!|\mathsf{b}|\!\rangle \subseteq \tau \subseteq \mathsf{A}(\mathsf{c})$ and therefore we trivially have $\mathsf{A}[\![\mathsf{b}?]\!]\mathsf{A}(\mathsf{c}) = \mathsf{A}(\mathsf{A}(\mathsf{c}) \cap \langle\!|\mathsf{b}|\!\rangle) = \mathsf{A}(\langle\!|\mathsf{b}|\!\rangle) = \tau$.

In other words, when b is quasi-expressible we have the adequacy of the abstract domain w.r.t. its semantics, when we have adequacy of the abstract domain w.r.t. its semantics, we cannot say anything about the possibility to express b, in particular if $\mathsf{A}(\mathsf{c}) \subsetneq \tau$ we always have adequacy w.r.t. τ.

Corollary 1. *Given* $\mathsf{b} \in \mathrm{BExp}$ *and* $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$

1. *If* b *is quasi-expressible then* $\widetilde{\mathbb{C}}^{\mathsf{A}}(\mathsf{b})$*;*
2. *If* b *is not quasi-expressible (w.r.t.* $\top$*), then* $\forall \mathsf{c} \in \mathsf{C}.\, \mathsf{A}(\mathsf{c}) = \top$ *we have* $\neg\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathsf{c}}(\mathsf{b}?)$*.*

4 Adjusting abstract domains

In this section, we wonder whether it is possible to *adjust* abstract domains in order to make them locally adequate. First, we realize that, differently from completeness, even when we deal with different input and output abstract domains, i.e., $\mathsf{A}_1 \circ f \circ \mathsf{A}_2$, we have only one possible direction for adjusting the domains, we can only refine them.
A second observation is that it is not always possible to adjust an abstract domain towards adequacy.
Finally, even when possible there may not exist a shell [17], namely a *most abstract* refinement guaranteeing adequacy, but anyway we can adjust the domain towards an optimal refinement guaranteeing adequacy.
For the sake of simplicity, in the following we consider the same domain of input and output, but similar results hold in the most general case.

Proposition 2. *Let* C *be concrete the domain,* $f : \mathsf{C} \to \mathsf{C}$*, and* $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$*. Let* $\mathsf{c} \in \mathsf{C}$ *and* $\tau \in \mathsf{A}$*, if* $f(\mathsf{c}) \not\leq_{\mathsf{C}} \tau$ *then the abstract domain* A *cannot be adequate.*

For a generic τ, $f(\mathsf{c}) \not\leq_{\mathsf{C}} \tau$ means that $\tau \leq_{\mathsf{C}} f(\mathsf{c})$ or not comparable, when $\tau = \top$, it means $f(\mathsf{c}) = \top$. The result is trivial, since if A is such that $\mathsf{A} \circ f \circ \mathsf{A}(\mathsf{c}) \leq_{\mathsf{C}} \tau$ then also $f(\mathsf{c}) \leq_{\mathsf{C}} \tau$ by extensivity, contradicting the hypothesis. Hence surely adequacy is violated for any possible abstraction A. For instance, if we consider f of Example 1, $\mathsf{c} = \left\{ \, n \in \mathbb{Z} \,\middle|\, n \geq 0 \, \right\}$ and $\tau = \left\{ \, n \in \mathbb{Z} \,\middle|\, n > 0 \, \right\}$, then $f(\mathsf{c}) = \left\{ \, 2n \in \mathbb{Z} \,\middle|\, n \geq 0 \, \right\} \not\leq_{\mathsf{C}} \tau$ meaning that there not exist abstract domains A adequate for f on c.

Suppose now $f(\mathsf{c}) \leq_{\mathsf{C}} \tau$, then we may adjust the abstract domain. The following step consists in fixing a first necessary (not sufficient) condition, namely $f \circ \mathsf{A}(\mathsf{c}) \leq_{\mathsf{C}} \tau$. Indeed it is necessary since, if $\mathsf{A} \circ f \circ \mathsf{A}(\mathsf{c}) \leq_{\mathsf{C}} \tau$ then, by extensivity, $f \circ \mathsf{A}(\mathsf{c}) \leq_{\mathsf{C}} \mathsf{A} \circ f \circ \mathsf{A}(\mathsf{c}) \leq_{\mathsf{C}} \tau$. Let us define the following set of elements and domain refinement

$$\mathsf{Ri}^{\tau}_{\mathsf{A}(\mathsf{c})}(\mathsf{A}) \stackrel{\text{def}}{=} \mathsf{A} \boxplus \{\mathsf{d}'\} \text{ with } \mathsf{d}' \in \mathsf{max}\left\{ \, \mathsf{d} \in \mathsf{C} \,\middle|\, \mathsf{d} \geq \mathsf{c},\ f(\mathsf{d}) \leq_{\mathsf{C}} \tau \ \wedge\ \mathsf{A}(\mathsf{c}) = \mathsf{A}(\mathsf{d}) \, \right\}^{5}$$

where the domain operation $\boxplus$ [5] is defined as follows. Let $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$ and $R \subseteq \mathsf{C}$

$$\mathsf{A} \boxplus R \stackrel{\text{def}}{=} \mathcal{M}(\mathsf{A} \cup R)^{6}$$

Proposition 3. *Let* C *be the concrete domain,* $f : \mathsf{C} \to \mathsf{C}$ *monotone, and* $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$*. Suppose* $f(\mathsf{c}) \leq_{\mathsf{C}} \tau$ *and* $f \circ \mathsf{A}(\mathsf{c}) \not\leq_{\mathsf{C}} \tau$ *Then* $\mathsf{A}_i \stackrel{\text{def}}{=} \mathsf{Ri}^{\tau}_{\mathsf{A}(\mathsf{c})}(\mathsf{A})$ *is such that* $f \circ \mathsf{A}_i(\mathsf{c}) \leq_{\mathsf{C}} \tau$*.*

Proof. Let us denote $\mathsf{R} \stackrel{\text{def}}{=} \mathsf{max}\left\{ \, \mathsf{d} \,\middle|\, \mathsf{d} \in \mathsf{C},\ f(\mathsf{d}) \leq_{\mathsf{C}} \tau \ \wedge\ \mathsf{A}(\mathsf{c}) = \mathsf{A}(\mathsf{d}) \, \right\}$ for the sake of readability. Suppose, in particular $\mathsf{A}_i = \mathsf{A} \boxplus \{\mathsf{d}\}$, with $\mathsf{d} \in \mathsf{R}$. Note that $\mathsf{A}_i \in \mathit{Abs}(\mathsf{C})$ by construction, and $\mathsf{A}_i \sqsubseteq \mathsf{A}$. Now, let us observe that $\mathsf{c} \in \mathsf{R}$ since $f(\mathsf{c}) \leq_{\mathsf{C}} \tau$ by hypothesis, hence $\mathsf{c} \leq_{\mathsf{C}} \mathsf{d}$. Then, by monotonicity of A_i, this means that $\mathsf{A}_i(\mathsf{c}) \leq_{\mathsf{C}} \mathsf{A}_i(\mathsf{d}) = \mathsf{d}$, where the last equality holds by construction. But then, by definition of R, we have $f(\mathsf{d}) \leq_{\mathsf{C}} \tau$, and therefore $f \circ \mathsf{A}_i(\mathsf{c}) \leq_{\mathsf{C}} f(\mathsf{d}) \leq_{\mathsf{C}} \tau$.

The following step consists in fixing another necessary (not sufficient) condition, namely $\mathsf{A} \circ f(\mathsf{c}) \leq_{\mathsf{C}} \tau$, which still may not hold. First of all, let us observe that if $f(\mathsf{c}) \leq_{\mathsf{C}} \tau$, being $\tau \in \mathsf{A}$, by monotonicity we have $\mathsf{A} \circ f(\mathsf{c}) \leq_{\mathsf{C}} \mathsf{A}(\tau) = \tau$, hence $\mathsf{A} \circ f(\mathsf{c}) \not\leq_{\mathsf{C}} \tau$ means $\mathsf{A} \circ f(\mathsf{c}) = \tau$. In this case we can refine the output abstract domain to force $\mathsf{A} \circ f(\mathsf{c}) \leq_{\mathsf{C}} \tau$.
Let us define the following refinement

$$\mathsf{Ro}^{\tau}_{\mathsf{A}(\mathsf{c})}(\mathsf{A}) \stackrel{\text{def}}{=} \mathsf{A} \boxplus \{\mathsf{d}'\} \text{ with } \mathsf{d}' \in \mathsf{max}\left\{ \, f(\mathsf{d}) \,\middle|\, \mathsf{c} \leq \mathsf{d} \in \mathsf{C},\ f(\mathsf{d}) \leq_{\mathsf{C}} \tau \ \wedge\ \mathsf{A}(\mathsf{c}) = \mathsf{A}(\mathsf{d}) \, \right\}$$

Proposition 4. *Let* C *be the concrete domain,* $f : \mathsf{C} \to \mathsf{C}$ *monotone, and* $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$*. Suppose* $f(\mathsf{c}) \leq_{\mathsf{C}} \tau$ *and* $\mathsf{A} \circ f(\mathsf{c}) = \tau$*. Then* $\mathsf{A}_o \stackrel{\text{def}}{=} \mathsf{Ro}^{\tau}_{\mathsf{A}(\mathsf{c})}(\mathsf{A})$ *is such that* $\mathsf{A}_o \circ f(\mathsf{c}) \leq_{\mathsf{C}} \tau$*.*

[5] Where max extracts the upper bounds from a set.

[6] Let us recall that $\mathcal{M}$ is the Moore closure, namely the operator closing a set by concrete greatest lower bound, hence making a set a Moore family.

Proof. Let us denote the set $\mathsf{R} \stackrel{\text{def}}{=} \mathsf{max}\left\{\ f(\mathtt{d}) \,\middle|\, \mathtt{d} \in \mathsf{C},\ f(\mathtt{d}) \lesssim_{\mathsf{C}} \tau\ \wedge\ \mathsf{A}(\mathtt{c}) = \mathsf{A}(\mathtt{d})\ \right\}$ for the sake of readability. Suppose, in particular $\mathsf{A}_o = \mathsf{A} \boxplus \{\mathtt{d}'\}$, with $\mathtt{d}' \in \mathsf{R}$. Note that $\mathsf{A}_o \in \mathit{Abs}(\mathsf{C})$ by construction, and $\mathsf{A}_o \sqsubseteq \mathsf{A}$, namely it is more concrete than A, which means that $\mathsf{A}_o \circ f(\mathtt{c}) \leq_{\mathsf{C}} \mathsf{A} \circ f(\mathtt{c}) = \tau$. Now, let us observe that $f(\mathtt{c}) \in \mathsf{R}$ since $f(\mathtt{c}) \lesssim_{\mathsf{C}} \tau$ by hypothesis, hence $f(\mathtt{c}) \leq_{\mathsf{C}} \mathtt{d}'$. Then, by monotonicity of A_o, this means that $\mathsf{A}_o \circ f(\mathtt{c}) \leq_{\mathsf{C}} \mathsf{A}_o(\mathtt{d}') = \mathtt{d}'$, where the last equation holds by construction. But then, by definition of R, we have $\mathtt{d}' \lesssim_{\mathsf{C}} \tau$, and therefore $\mathsf{A}_o \circ f(\mathtt{c}) \leq_{\mathsf{C}} \mathtt{d}' \lesssim_{\mathsf{C}} \tau$.

Now we can make the final step, combining the two transformations.

Theorem 2. *Let* C *be the concrete domain,* $f : \mathsf{C} \to \mathsf{C}$ *monotone, and* $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$. *Suppose* $f(\mathtt{c}) \lesssim_{\mathsf{C}} \tau$, *then let* $\mathsf{A}_i \stackrel{\text{def}}{=} \mathsf{Ri}^{\tau}_{\mathsf{A}(\mathtt{c})}(\mathsf{A})$ *and* $\mathsf{R}^{\tau}_{\mathsf{A}(\mathtt{c})}(\mathsf{A}) \stackrel{\text{def}}{=} \mathsf{Ro}^{\tau}_{\mathsf{A}_i(\mathtt{c})}(\mathsf{A}_i)$, *then* $\mathsf{R}^{\tau}_{\mathsf{A}(\mathtt{c})} \circ f \circ \mathsf{R}^{\tau}_{\mathsf{A}(\mathtt{c})} \lesssim_{\mathsf{C}} \tau$.

Proof. By the hypotheses and by Prop. 3 we have that $f \circ \mathsf{A}_i(\mathtt{c}) \lesssim_{\mathsf{C}} \tau$. Let us denote $\mathtt{c}' \stackrel{\text{def}}{=} \mathsf{A}_i(\mathtt{c})$, then $f(\mathtt{c}') \lesssim \tau$, hence we can build $\mathsf{A}_o \stackrel{\text{def}}{=} \mathsf{Ro}^{\tau}_{\mathsf{A}_i(\mathtt{c}')}(\mathsf{A}_i)$. At this point, by definition and indempotence of A_i we have that

$$\begin{aligned}
\mathsf{Ro}^{\tau}_{\mathsf{A}_i(\mathtt{c}')}(\mathsf{A}_i) &= \mathsf{max}\left\{\ f(\mathtt{d}) \,\middle|\, \mathtt{d} \in \mathsf{C},\ f(\mathtt{d}) \lesssim_{\mathsf{C}} \tau\ \wedge\ \mathsf{A}_i(\mathtt{c}') = \mathsf{A}_i(\mathtt{d})\ \right\} \\
&= \mathsf{max}\left\{\ f(\mathtt{d}) \,\middle|\, \mathtt{d} \in \mathsf{C},\ f(\mathtt{d}) \lesssim_{\mathsf{C}} \tau\ \wedge\ \mathsf{A}_i(\mathsf{A}_i(\mathtt{c})) = \mathsf{A}_i(\mathtt{d})\ \right\} \\
&= \mathsf{max}\left\{\ f(\mathtt{d}) \,\middle|\, \mathtt{d} \in \mathsf{C},\ f(\mathtt{d}) \lesssim_{\mathsf{C}} \tau\ \wedge\ \mathsf{A}_i(\mathtt{c}) = \mathsf{A}_i(\mathtt{d})\ \right\} = \mathsf{Ro}^{\tau}_{\mathsf{A}_i(\mathtt{c})}(\mathsf{A}_i)
\end{aligned}$$

Hence $\mathsf{A}_o = \mathsf{Ro}^{\tau}_{\mathsf{A}_i(\mathtt{c})}(\mathsf{A}_i) = \mathsf{R}^{\tau}_{\mathsf{A}(\mathtt{c})}(\mathsf{A})$ and $\mathsf{A}_o \sqsubseteq \mathsf{A}_i$. Then by Prop. 4 we have that $\mathsf{A}_o \circ f(\mathtt{c}') \lesssim_{\mathsf{C}} \tau$, namely $\mathsf{A}_o \circ f \circ \mathsf{A}_i(\mathtt{c}) \lesssim_{\mathsf{C}} \tau$. But then, by $\mathsf{A}_o \sqsubseteq \mathsf{A}_i$ and by monotonicity of the functions involved, we have $\mathsf{A}_o \circ f \circ \mathsf{A}_o(\mathtt{c}) \leq_{\mathsf{C}} \mathsf{A}_o \circ f \circ \mathsf{A}_i(\mathtt{c}) \lesssim_{\mathsf{C}} \tau$.

Let us consider a very simple example just to show the process.

Example 3. Consider the sign abstraction $\mathsf{A} = \mathit{Sign}$ on the concrete domain $\mathsf{C} = \wp(\mathbb{Z})$ and $g \stackrel{\text{def}}{=} \lambda\mathtt{c}.\left\{\ n+2 \,\middle|\, n \in \mathtt{c}\ \right\}$. Then, if $\mathtt{c} \stackrel{\text{def}}{=} \top = \mathbb{Z}$ we have $g(\mathtt{c}) = \top$, meaning that for the $\top$ element the sign domain cannot be make adequate.
Let us consider now the f function in Example 1, on the same domains and suppose $\tau \stackrel{\text{def}}{=} \mathbb{Z}_{\geq 0}$. Let $\mathtt{c} = \{0, 2, 4\}$, then $f(\mathtt{c}) = \{0, 4, 8\} \subsetneq \tau$, but $f \circ \mathit{Sign}(\mathtt{c}) = f(\mathbb{Z}_{\geq 0}) = \mathbb{Z}_{\geq 0} = \tau$ hence we can try to make the abstract domain adequate for $\mathtt{c}$. Let us observe that for any $n > 0$

$$\mathbb{Z}_{\geq 0} \smallsetminus \{n\} \in \mathsf{max}\left\{\ \mathtt{d} \,\middle|\, f(\mathtt{d}) \subsetneq \mathbb{Z}_{\geq 0}\ \wedge\ \mathit{Sign}(\mathtt{d}) = \mathbb{Z}_{\geq 0}\ \right\}$$

since $f(\mathbb{Z}_{\geq 0} \smallsetminus \{n\}) = \mathbb{Z}_{\geq 0} \smallsetminus \{2n\} \subsetneq \tau$, but $\mathit{Sign}(\mathbb{Z}_{\geq 0} \smallsetminus \{n\}) = \mathbb{Z}_{\geq 0}$ since 0 is still in the result. Let us define $\mathit{Sign}' \stackrel{\text{def}}{=} \mathit{Sign} \boxplus (\mathbb{Z}_{\geq 0} \smallsetminus \{8\})$. Now $f \circ \mathit{Sign}'(\mathtt{c}) = f(\mathbb{Z}_{\geq 0} \smallsetminus \{8\}) = \mathbb{Z}_{\geq 0} \smallsetminus \{16\} \subsetneq \tau$, but unfortunately $\mathit{Sign}' \circ f(\mathtt{c}) = \mathit{Sign}'(\{0, 4, 8\}) = \mathbb{Z}_{\geq 0} = \tau$, hence we have to further refine. Then, we have that

$$\mathbb{Z}_{\geq 0} \smallsetminus \{16\} \in \mathsf{max}\left\{\ f(\mathtt{d}) \,\middle|\, f(\mathtt{d}) \subsetneq \mathbb{Z}_{\geq 0}\ \wedge\ \mathit{Sign}'(\mathtt{d}) = \mathbb{Z}_{\geq 0} \smallsetminus \{8\}\ \right\}$$

Let $\mathit{Sign}'' \stackrel{\text{def}}{=} \mathit{Sign}' \boxplus \mathbb{Z}_{\geq 0} \smallsetminus \{16\}$, and in this case $\mathit{Sign}'' \circ f(\mathtt{c}) = \mathit{Sign}''(\{0, 4, 8\}) = \mathbb{Z}_{\geq 0} \smallsetminus \{16\} \subsetneq \tau$ and $\mathit{Sign}'' \circ f \circ \mathit{Sign}''(\mathtt{c}) = \mathit{Sign}'' \circ f(\mathbb{Z}_{\geq 0} \smallsetminus \{8, 16\})$[7] =

[7] $\mathbb{Z}_{\geq 0} \smallsetminus \{8, 16\}$ is introduced by $\boxplus$ in Sign''.

$Sign''(\mathbb{Z}_{\geq 0} \setminus \{16, 32\}) = \mathbb{Z}_{\geq 0} \setminus \{16\} \subsetneq \tau$. In this example, we can also observe that, in general, the refinements do not induce local completeness [5] but only adequacy.

5 Abstract domain adequacy logic

In this section we define a proof system for program analysis of regular commands, parameterized by an abstraction $\mathsf{A} = \gamma\alpha(\mathsf{C})$ on the concrete domain of sets of stores $\mathsf{C} = \wp(\mathbb{M})$. The provable triples of our logic are judgements of the form $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\, \mathtt{r}\, \langle \mathtt{d} \rangle$, with $\mathtt{c}, \mathtt{d} \in \mathsf{C}$, $\tau \in \mathsf{A}$ and $\mathtt{r} \in \mathfrak{L}$. $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\, \mathtt{r}\, \langle \mathtt{d} \rangle$ guarantees that:

1. $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})_{\tau}$;
2. $[\![\mathtt{r}]\!]^{\sharp}_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lesssim_{\mathsf{A}} \tau_{\alpha} \stackrel{\text{def}}{=} \alpha(\tau)$.

The rules are provided in Fig. 4. It is worth noting that, the rule `relax` is added, even if less common, since the rules work with the concrete elements $\mathtt{c}$, and the rule `seq` could be not directly applicable also in cases where the first statement has an output assertion greater than the input assertion of the following statement. In this case the rule $\mathtt{relax}_2$ may not be used or may increase imprecision getting closer to τ.
For instance, consider $\mathtt{r} = \mathtt{r}_1; \mathtt{r}_2$, and consider as output property of $\mathtt{r}_1$ the property $\mathtt{c} = \{0, 10, 20, 30\}$, while the input property for $\mathtt{r}_2$ is $\mathtt{c}' = \{0, 10, 30\}$, then we would have to reduce $\mathtt{c}$ or enlarge $\mathtt{c}'$ in order to apply rule `seq`. It should be clear that the first direction is surely preferable for not losing precision and only the rule `relax` allows to follow such direction.

Lemma 1. *Let* $\mathsf{C} = \wp(\mathbb{M})$, $\mathsf{A} \in Abs(\mathsf{C})$ *(*$\mathsf{A} = \gamma\alpha$*),* $\tau \in \mathsf{A}$ *and* $\tau_{\alpha} \stackrel{\text{def}}{=} \alpha(\tau)$*. Then* $\forall \mathtt{c} \in \mathsf{C}$ *we have that* $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})_{\tau}$ *iff* $\alpha([\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c})) \lesssim_{\mathsf{A}} \tau_{\alpha}$.

Proof. Suppose $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})_{\tau}$, namely such that $\mathsf{A}[\![\mathtt{r}]\!]\mathsf{A}(\mathtt{c}) \subsetneq \tau$, written in terms of the GI it is $\gamma\alpha[\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c}) \subsetneq \tau$, then by monotonicity of α and by properties of GI, we have that

$$\alpha[\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c}) = \alpha\gamma\alpha[\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c}) \leq_{\mathsf{A}} \tau_{\alpha}$$

Suppose, towards contradiction, that $\alpha[\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c}) = \tau_{\alpha}$, then since $\gamma(\tau_{\alpha}) = \gamma\alpha(\tau) = \tau$ being $\tau \in \mathsf{A}$, we would have also

$$\mathsf{A}[\![\mathtt{r}]\!]\mathsf{A}(\mathtt{c}) = \gamma\alpha[\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c}) = \gamma(\tau_{\alpha}) = \tau$$

contradicting the hypothesis, hence $\alpha[\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c}) \lesssim_{\mathsf{A}} \tau_{\alpha}$.
Suppose now $\alpha[\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c}) \lesssim_{\mathsf{A}} \tau_{\alpha}$, then by monotonicity of γ, we have

$$\mathsf{A}[\![\mathtt{r}]\!]\mathsf{A}(\mathtt{c}) = \gamma\alpha[\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c}) \subseteq \gamma(\tau_{\alpha}) = \tau$$

Suppose, towards contradiction, that $\gamma\alpha[\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c}) = \gamma(\tau_{\alpha})$ then being γ one-to-one, this would imply $\alpha[\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c}) = \tau_{\alpha}$ contradicting the hypothesis. Hence $\mathsf{A}[\![\mathtt{r}]\!]\mathsf{A}(\mathtt{c}) \subsetneq \tau$.

$$\texttt{transfer:}\ \frac{\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{e})_{\tau}}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{e}\ \langle [\![\mathtt{e}]\!]\gamma\alpha(\mathtt{c}))\rangle}$$

$$\texttt{relax:}\ \frac{\mathtt{c}' \subseteq \mathtt{c} \subseteq \gamma\alpha(\mathtt{c}')\quad \tau \vdash_{\mathsf{A}} \langle \mathtt{c}'\rangle\ \mathtt{r}\ \langle \mathtt{d}'\rangle\quad \mathtt{d} \subseteq \mathtt{d}' \subseteq \gamma\alpha(\mathtt{d})}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}\ \langle \mathtt{d}\rangle}$$

$$\texttt{relax}_2\texttt{:}\ \frac{\mathtt{c} \subseteq \mathtt{c}'\quad \tau \vdash_{\mathsf{A}} \langle \mathtt{c}'\rangle\ \mathtt{r}\ \langle \mathtt{d}'\rangle\quad \alpha(\mathtt{d}') \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lesssim_{\mathsf{A}} \tau_{\alpha}}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}\ \langle \mathtt{d}\rangle}$$

$$\texttt{seq:}\ \frac{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}_1\ \langle \mathtt{d}'\rangle\quad \tau \vdash_{\mathsf{A}} \langle \mathtt{d}'\rangle\ \mathtt{r}_2\ \langle \mathtt{d}\rangle}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}_1; \mathtt{r}_2\ \langle \mathtt{d}\rangle}\qquad \texttt{iterate:}\ \frac{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}\ \langle \mathtt{d}\rangle\quad \alpha(\mathtt{d}) \leq_{\mathsf{A}} \alpha(\mathtt{c}) \lesssim_{\mathsf{A}} \tau_{\alpha}}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}^*\ \langle \mathtt{c}\rangle}$$

$$\texttt{join:}\ \frac{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}_1\ \langle \mathtt{d}_1\rangle\quad \tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}_2\ \langle \mathtt{d}_2\rangle\quad \alpha(\mathtt{d}_1 \cup \mathtt{d}_2) \lesssim_{\mathsf{A}} \tau_{\alpha}}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}_1 \oplus \mathtt{r}_2\ \langle \mathtt{d}_1 \cup \mathtt{d}_2\rangle}$$

Fig. 4. A logic for adequacy w.r.t. τ ($\tau_{\alpha} \stackrel{\text{def}}{=} \alpha(\tau)$).

Theorem 3. *Let* $\mathtt{c}, \mathtt{d} \in \mathsf{C} = \wp(\mathbb{M})$, $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$ *(*$\mathsf{A} = \gamma\alpha$*),* $\tau \in \mathsf{A}$ *and* $\tau_{\alpha} \stackrel{\text{def}}{=} \alpha(\tau)$. *If* $\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}\ \langle \mathtt{d}\rangle$ *then* (1) $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})_{\tau}$ *and* (2) $[\![\mathtt{r}]\!]^{\sharp}_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lesssim_{\mathsf{A}} \tau_{\alpha}$.

Proof. We prove the soundness of the rule system in Fig. 4 by structural induction on the derivation tree of $\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}\ \langle \mathtt{d}\rangle$, by distinguishing the cases depending on the last rule applied.

(`transfer`): In this case (1) holds by hypothesis, being the premix of the rule. Then, by Lemma 1 we have $\alpha([\![\mathtt{e}]\!]\gamma\alpha(\mathtt{c})) \lesssim_{\mathsf{A}} \tau_{\alpha}$, and by definition of $[\![\mathtt{e}]\!]^{\sharp}_{\mathsf{A}}$ we have

$$[\![\mathtt{e}]\!]^{\sharp}_{\mathsf{A}}\alpha(\mathtt{c}) = (\alpha[\![\mathtt{e}]\!]\gamma)\alpha(\mathtt{c}) = \alpha([\![\mathtt{e}]\!]\gamma\alpha(\mathtt{c})) \lesssim_{\mathsf{A}} \tau_{\alpha}$$

(`relax`): First of all, let us observe that the hypotheses, by monotonicity and idempotence of $\gamma\alpha$, imply $\gamma\alpha(\mathtt{c}) = \gamma\alpha(\mathtt{c}')$, and therefore, being γ one-to-one, this means that $\alpha(\mathtt{c}) = \alpha(\mathtt{c}')$. Analogously, $\alpha(\mathtt{d}) = \alpha(\mathtt{d}')$. This means that $\alpha([\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c})) = (\alpha[\![\mathtt{r}]\!]\gamma)\alpha(\mathtt{c}) \leq_{\mathsf{A}} [\![\mathtt{r}]\!]^{\sharp}_{\mathsf{A}}\alpha(\mathtt{c})$ by definition of bca, but then $[\![\mathtt{r}]\!]^{\sharp}_{\mathsf{A}}\alpha(\mathtt{c}) = [\![\mathtt{r}]\!]^{\sharp}_{\mathsf{A}}\alpha(\mathtt{c}') \lesssim_{\mathsf{A}} \tau_{\alpha}$ since by hypothesis $\tau \vdash_{\mathsf{A}} \langle \mathtt{c}'\rangle\ \mathtt{r}\ \langle \mathtt{d}'\rangle$, and therefore by inductive hypothesis $[\![\mathtt{r}]\!]^{\sharp}_{\mathsf{A}}\alpha(\mathtt{c}') \leq_{\mathsf{A}} \alpha(\mathtt{d}') \lesssim_{\mathsf{A}} \tau_{\alpha}$. But then, by definition of bca and by transitivity of $\leq_{\mathsf{A}}$ we have $\alpha([\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c})) \lesssim_{\mathsf{A}} \tau_{\alpha}$. Hence, by Lemma 1 we have (1). As far as (2) is concerned, by we have proved above we have

$$[\![\mathtt{r}]\!]^{\sharp}_{\mathsf{A}}\alpha(\mathtt{c}) = [\![\mathtt{r}]\!]^{\sharp}_{\mathsf{A}}\alpha(\mathtt{c}') \leq_{\mathsf{A}} \alpha(\mathtt{d}') = \alpha(\mathtt{d}) \qquad \text{and} \qquad \alpha(\mathtt{d}) = \alpha(\mathtt{d}') \lesssim_{\mathsf{A}} \tau_{\alpha}$$

($\texttt{relax}_2$): Note that, by hypothesis $\tau \vdash_{\mathsf{A}} \langle \mathtt{c}'\rangle\mathtt{r}\langle \mathtt{d}'\rangle$, hence by inductive hypothesis,

as before, we have $[\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}') \leq_{\mathsf{A}} \alpha(\mathtt{d}') \lesssim_{\mathsf{A}} \tau_\alpha$, and therefore

$$\alpha[\![\mathtt{r}]\!]\gamma\alpha(\mathtt{c}) \leq_{\mathsf{A}} [\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} [\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}') \lesssim_{\mathsf{A}} \tau_\alpha$$

Then by Lemma 1 this implies $\mathsf{A}[\![\mathtt{r}]\!]\mathsf{A}(\mathtt{c}) \subsetneq \tau$ (condition (1)), and moreover, again by the rule hypotheses and by inductive hypothesis, we have

$$[\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} [\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}') \leq_{\mathsf{A}} \alpha(\mathtt{d}') \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lesssim_{\mathsf{A}} \tau_\alpha$$

proving condition (2).

(seq): Let us prove (1). By inductive hypotheses, we have that $[\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}') \lesssim_{\mathsf{A}} \tau_\alpha$ and $[\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{d}') \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lesssim_{\mathsf{A}} \tau_\alpha$, then by definition of bca and of abstract semantics, and by the monotonicity of the abstract semantics we have that

$$\alpha[\![\mathtt{r}_1;\mathtt{r}_2]\!]\gamma\alpha(\mathtt{c}) \leq_{\mathsf{A}} [\![\mathtt{r}_1;\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) = [\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}([\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c})) \leq_{\mathsf{A}} [\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{d}') \lesssim_{\mathsf{A}} \tau_\alpha$$

Hence, by Lemma 1, we have $\mathsf{A}[\![\mathtt{r}]\!]\mathsf{A}(\mathtt{c}) \subsetneq \tau$. As far as (2) is concerned, we have already proved that $[\![\mathtt{r}_1;\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} [\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{d}')$, and $[\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{d}') \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lesssim_{\mathsf{A}} \tau_\alpha$ by hypothesis (2) on $\mathtt{r}_2$, and therefore by transitivity we have the thesis.

(iterate): Let us prove that if we have $\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}\ \langle \mathtt{d}\rangle$ with $\alpha(\mathtt{d}) \leq_{\mathsf{A}} \alpha(\mathtt{c})$, then we have $\tau \vdash_{\mathsf{A}} \langle \mathtt{c}\rangle\ \mathtt{r}^*\ \langle \mathtt{c}\rangle$. We first prove by induction on $n \geq 1$ that $([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^n\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d})$. The base ($n = 1$) is the hypothesis of the rule, suppose it holds for $([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^n$, let us prove for $n+1$, recalling that by structural inductive hypothesis we have that $[\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \subsetneq \tau_\alpha$, and by the rule hypothesis we have $\alpha(\mathtt{d}) \leq_{\mathsf{A}} \alpha(\mathtt{c})$

$$\begin{aligned}([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^{n+1}\alpha(\mathtt{c}) &= (([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^n \circ [\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})\alpha(\mathtt{c}) \\ &= ([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^n([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c})) \leq_{\mathsf{A}} ([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^n\alpha(\mathtt{d}) \\ &\leq_{\mathsf{A}} ([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^n\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lesssim_{\mathsf{A}} \tau_\alpha\end{aligned}$$

where the last relations hold by inductive hypothesis on n. Hence, for all $n \geq 1$ we have $([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^n\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lesssim_{\mathsf{A}} \tau_\alpha$. At this point we have condition (2)

$$\begin{aligned}\alpha([\![\mathtt{r}^*]\!]\gamma\alpha(\mathtt{c})) &\leq_{\mathsf{A}} [\![\mathtt{r}^*]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) = \bigvee\nolimits_{\mathsf{A}\{n\geq 0\}}([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^n\alpha(\mathtt{c}) \\ &= ([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^0\alpha(\mathtt{c}) \vee_{\mathsf{A}} \bigvee\nolimits_{\mathsf{A}\{n\geq 1\}}([\![\mathtt{r}]\!]^\sharp_{\mathsf{A}})^n\alpha(\mathtt{c}) \\ &\leq_{\mathsf{A}} \alpha(\mathtt{c}) \vee_{\mathsf{A}} \alpha(\mathtt{d}) \leq_{\mathsf{A}} \alpha(\mathtt{c}) \lesssim_{\mathsf{A}} \tau_\alpha\end{aligned}$$

by additivity of α and by the rule hypothesis $\alpha(\mathtt{c}) \lesssim_{\mathsf{A}} \tau_\alpha$. Finally, condition (1) comes by Lemma 1.

(join): Let us prove (2). By inductive hypotheses we have that $[\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}_1) \lesssim_{\mathsf{A}} \tau_\alpha$ and that $[\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}_2) \lesssim_{\mathsf{A}} \tau_\alpha$. Hence, by additivity of α we

have

$$\begin{aligned}\alpha([\![\mathtt{r}_1 \oplus \mathtt{r}_2]\!]\gamma\alpha(\mathtt{c})) &= (\alpha[\![\mathtt{r}_1 \oplus \mathtt{r}_2]\!]\gamma)\alpha(\mathtt{c}) \leq_{\mathsf{A}} [\![\mathtt{r}_1 \oplus \mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \\ &= [\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \curlyvee_{\mathsf{A}} [\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}_1) \curlyvee_{\mathsf{A}} \alpha(\mathtt{d}_2) \\ &= \alpha(\mathtt{d}_1 \cup \mathtt{d}_2) \lesssim_{\mathsf{A}} \tau_\alpha\end{aligned}$$

By Lemma 1 what we have proved implies also (1), proving the thesis.

Corollary 2. *Let* $\mathsf{C} = \wp(\mathbb{M})$, $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$ *(*$\mathsf{A} = \gamma\alpha$*). If* $\vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ \mathtt{r}\ \langle \mathtt{d} \rangle$ *then* (1) $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})$ *and* (2) $[\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lesssim_{\mathsf{A}} \top_{\mathsf{A}}$.

Let us investigate about completeness of this rule system.

Theorem 4. *Let* $\mathtt{c}, \mathtt{d} \in \mathsf{C} = \wp(\mathbb{M})$, $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$ *(*$\mathsf{A} = \gamma\alpha$*),* $\mathtt{r} \in \mathfrak{L}$, $\tau \in \mathsf{A}$ *and* $\tau_\alpha \stackrel{\text{def}}{=} \alpha(\tau)$. (1) $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})_\tau$ *and* (2) $[\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lesssim_{\mathsf{A}} \tau_\alpha$ *do not imply* $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\, \mathtt{r}\, \langle \mathtt{d} \rangle$.

Proof (Sketch). We provide an example that cannot be deduced by using the rule system. Consider $\mathtt{r} \stackrel{\text{def}}{=} (x := x * 2; x < 0?)$, let us denote by $\mathtt{r}_1 \stackrel{\text{def}}{=} x := x * 2$ and by $\mathtt{r}_2 \stackrel{\text{def}}{=} x \leq 0?$. Consider the sign domain $\mathsf{A} \stackrel{\text{def}}{=} \mathit{Sign} = \gamma_{\mathtt{S}}\alpha_{\mathtt{S}}$ in Fig. 3 and let us denote by $\alpha_{\mathtt{S}} : \wp(\mathbb{Z}) \to \mathit{Sign}$, and $\gamma_{\mathtt{S}}$ the function associating with each abstract element the represented set, e.g., $\gamma_{\mathtt{S}}(\mathbb{Z}_{\leq 0}) = \{\ n \in \mathbb{Z} \,|\, n \leq 0\ \}$, and let $\tau \stackrel{\text{def}}{=} \gamma(\mathbb{Z}_{\leq 0})$. Let $\mathtt{c} \stackrel{\text{def}}{=} \{-10, 0, 10\}$ and $\mathtt{d} \stackrel{\text{def}}{=} \{-10\}$ Then we have that

(1) $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})_\tau$, since $\mathit{Sign}[\![\mathtt{r}]\!]\mathit{Sign}(\{-10, 0, 10\}) = \mathit{Sign}[\![\mathtt{r}]\!]\mathbb{Z} = \mathit{Sign}(\{\ n \,|\, n < 0\ \}) = \{\ n \,|\, n < 0\ \} \subsetneq \tau$;

(2) $[\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\mathit{Sign}(\mathtt{c}) = [\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\mathit{Sign}(\{-10, 0, 10\}) = [\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\mathbb{Z} = \mathbb{Z}_{<0} = \alpha_{\mathtt{S}}(\mathtt{d}) \lesssim_{\mathit{Sign}} \mathbb{Z}_{\leq 0}$, where trivially $\mathbb{Z}_{\leq 0} = \alpha_{\mathtt{S}}(\tau)$

But, $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r}_1)_\tau$ does not hold, since $\mathit{Sign}[\![\mathtt{r}_1]\!]\mathit{Sign}(\{-10, 0, 10\}) = \mathit{Sign}[\![\mathtt{r}_1]\!]\mathbb{Z} = \mathit{Sign}(\mathbb{Z}) = \mathbb{Z} \not\subseteq \tau$, and therefore we cannot find $\mathtt{d}' \in \wp(\mathbb{Z})$ such that $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle \mathtt{r}_1 \langle \mathtt{d}' \rangle$, making not possible to apply rule `seq`.

The logic above becomes complete if we add/substitute rule `(seq)` with the following rule computing code semantics

$$\mathtt{seq}_1\colon \frac{\tau \vdash_{\mathsf{A}} \langle [\![\mathtt{r}_1]\!]\gamma\alpha(\mathtt{c}) \rangle\ \mathtt{r}_2\ \langle \mathtt{d} \rangle}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ \mathtt{r}_1; \mathtt{r}_2\ \langle \mathtt{d} \rangle}$$

and the rule, with infinite premixes

$$\mathtt{iterate}_1\colon \frac{\forall n \in \mathbb{N}.\, \tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ \mathtt{r}^n\ \langle \mathtt{d} \rangle}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ \mathtt{r}^*\ \langle \mathtt{d} \rangle}$$

where $\mathtt{r}^n$ is a syntactic sugar defined as : $\mathtt{r}^0 \stackrel{\text{def}}{=} \mathbf{skip}$ and $\mathtt{r}^{n+1} \stackrel{\text{def}}{=} \mathtt{r}^n; \mathtt{r}$. But it is worth noting that in this way the logic itself becomes undecidable requiring to compute a code semantics and to prove infinite conditions for being applied. Let us denote as $\tau \models_{\mathsf{A}} \langle \mathtt{c} \rangle\ \mathtt{r}\ \langle \mathtt{d} \rangle$ the derivation in the rule system in Fig. 4 extended with rules $\mathtt{seq}_1$ and $\mathtt{iterate}_1$.

Theorem 5. *Let* $\mathtt{c},\mathtt{d} \in \mathsf{C} = \wp(\mathbb{M})$, $\mathsf{A} \in \mathit{Abs}(\mathsf{C})$ *(*$\mathsf{A} = \gamma\alpha$*),* $\mathtt{r} \in \mathfrak{L}$, $\tau \in \mathsf{A}$ *and* $\tau_\alpha \stackrel{\text{def}}{=} \alpha(\tau)$*. If* (1) $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})_\tau$ *and* (2) $[\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq \alpha(\mathtt{d}) < \tau$ *then we have* $\tau \vDash_{\mathsf{A}} \langle \mathtt{c}\rangle\, \mathtt{r}\, \langle \mathtt{d}\rangle$.

Proof. Let us prove by structural induction on the language $\mathfrak{L}$.

$\mathtt{r} = \mathtt{e}$: If (1) and (2) hold, then trivially by rule `(transfer)` we have $\tau \vDash_{\mathsf{A}} \langle \mathtt{c}\rangle\, \mathtt{e}\, \langle \mathtt{d}\rangle$.

$\mathtt{r} = \mathtt{r}_1 \oplus \mathtt{r}_2$: Let us consider condition (2), if $[\![\mathtt{r}_1 \oplus \mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lneq_{\mathsf{A}} \tau_\alpha$ then we have that $[\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \curlyvee_{\mathsf{A}} [\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lneq_{\mathsf{A}} \tau_\alpha$, implying that we have both the relations $[\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{Q}) \lneq_{\mathsf{A}} \tau_\alpha$ and $[\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{Q}) \lneq_{\mathsf{A}} \tau_\alpha$. By properties of bca, this means that $(\alpha[\![\mathtt{r}_1]\!]\gamma)\alpha(\mathtt{c}) \leq_{\mathsf{A}} [\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \lneq_{\mathsf{A}} \tau_\alpha$, but then by Lemma 1 we have $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r}_1)_\tau$. Analogously we also have $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r}_2)_\tau$. Hence, by inductive hypothesis implies that $\tau \vDash_{\mathsf{A}} \langle \mathtt{c}\rangle\, \mathtt{r}_1\, \langle \mathtt{d}\rangle$ and $\tau \vDash_{\mathsf{A}} \langle \mathtt{c}\rangle\, \mathtt{r}_2\, \langle \mathtt{d}\rangle$, and therefore, by rule `(join)` we have that $\tau \vDash_{\mathsf{A}} \langle \mathtt{c}\rangle\, \mathtt{r}\, \langle \mathtt{d}\rangle$, being $\alpha(\mathtt{d} \vee \mathtt{d}) = \alpha(\mathtt{d}) \lneq_{\mathsf{A}} \tau_\alpha$.

$\mathtt{r} = \mathtt{r}_1;\mathtt{r}_2$: Suppose $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})_\tau$ together with $[\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}(\alpha(\mathtt{c})) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lneq_{\mathsf{A}} \tau_\alpha$. First of all let us observe that, by definition $[\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}(\alpha(\mathtt{c})) = [\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}([\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}}(\alpha(\mathtt{c})))$. Then, by monotonicity and by construction of abstract semantics we have

$$[\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}(\alpha[\![\mathtt{r}_1]\!]\gamma\alpha(\mathtt{c})) \leq_{\mathsf{A}} [\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}([\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}}(\alpha(\mathtt{c}))) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lneq_{\mathsf{A}} \tau_\alpha$$

which is condition (2) for the hypothesis of rule $(\mathtt{seq}_1)$. As far as (1) is concerned let us observe that

$$\mathsf{A}[\![\mathtt{r}_2]\!]\mathsf{A}([\![\mathtt{r}_1]\!]\gamma\alpha(\mathtt{c})) = \gamma\alpha[\![\mathtt{r}_2]\!]\gamma\alpha[\![\mathtt{r}_1]\!]\gamma\alpha(\mathtt{c}) \subseteq \gamma([\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}(\alpha[\![\mathtt{r}_1]\!]\gamma\alpha(\mathtt{c})))$$

where the last holds by definition of $[\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}$, and for what we have proved above we have

$$\gamma([\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}(\alpha[\![\mathtt{r}_1]\!]\gamma\alpha(\mathtt{c}))) \subseteq \gamma(\tau_\alpha) = \tau$$

Finally, by Lemma 1 it must be $\gamma([\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}(\alpha[\![\mathtt{r}_1]\!]\gamma\alpha(\mathtt{c}))) \subsetneq \tau$, since we proved above that $[\![\mathtt{r}_2]\!]^\sharp_{\mathsf{A}}(\alpha[\![\mathtt{r}_1]\!]\gamma\alpha(\mathtt{c})) \lneq_{\mathsf{A}} \tau_\alpha$. Therefore we have (1), i.e., $\mathsf{A}[\![\mathtt{r}_2]\!]\mathsf{A}([\![\mathtt{r}_1]\!]\gamma\alpha(\mathtt{c})) \subsetneq \tau$. At this point, by inductive hypothesis we conclude that $\tau \vDash_{\mathsf{A}} \langle [\![\mathtt{r}_1]\!]\gamma\alpha(\mathtt{c})\rangle\, \mathtt{r}_2\, \langle \mathtt{d}\rangle$, and by rule $(\mathtt{seq}_1)$ we derive $\tau \vDash_{\mathsf{A}} \langle \mathtt{c}\rangle\, \mathtt{r}_1;\mathtt{r}_2\, \langle \mathtt{d}\rangle$.

$\mathtt{r} = \mathtt{r}_1^*$: Suppose $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})_\tau$ together with $[\![\mathtt{r}]\!]^\sharp_{\mathsf{A}}(\alpha(\mathtt{c})) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lneq_{\mathsf{A}} \tau_\alpha$. By definition of abstract semantics, we have $\bigvee_{\mathsf{A}}([\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}})^n\alpha(\mathtt{c}) = [\![\mathtt{r}_1^*]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lneq_{\mathsf{A}} \tau_\alpha$, which implies, by definition of least upper bound, that for all n, we have $([\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}})^n\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lneq_{\mathsf{A}} \tau_\alpha$. As before, we can also show that

$$\alpha[\![\mathtt{r}_1^n]\!]\gamma\alpha(\mathtt{c}) \leq_{\mathsf{A}} [\![\mathtt{r}_1^n]\!]^\sharp_{\mathsf{A}}\alpha(\mathtt{c}) = ([\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}})^n\alpha(\mathtt{c}) \leq_{\mathsf{A}} \alpha(\mathtt{d}) \lneq_{\mathsf{A}} \tau_\alpha$$

since, by induction and by definition, we can observe that $[\![\mathtt{r}_1^n]\!]^\sharp_{\mathsf{A}} = ([\![\mathtt{r}_1]\!]^\sharp_{\mathsf{A}})^n$. Now, by Lemma 1 we also have $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r}_1^n)_\tau$ for each $n \in \mathbb{N}$, hence by inductive hypothesis we have that $\forall n \in \mathbb{N}.\, \tau \vDash_{\mathsf{A}} \langle \mathtt{c}\rangle\, \mathtt{r}_1^n\, \langle \mathtt{d}\rangle$. Finally, by rule $(\mathtt{iterate}_1)$ we have that $\tau \vDash_{\mathsf{A}} \langle \mathtt{c}\rangle\, \mathtt{r}_1^*\, \langle \mathtt{d}\rangle$.

Let us consider some simple examples of derivation. For the sake of simplicity, since the following programs have only one variable x, we abuse notation identifying the domain of stores $\mathbb{M}$ with the domain of values $\mathbb{Z}$. Hence, in the following examples, we will define A directly on $\wp(\mathbb{Z})$.

Example 4. Let us consider the regular command in Reg

$$\mathtt{r} \stackrel{\text{def}}{=} (x \leq 0?; x := x * 2)^*; 0 \leq x?$$

and $\tau = \top$. Let us consider the abstract domain $\mathsf{A} = \mathit{Sign}$ (let $\mathsf{A} = \gamma_{\mathsf{S}}\alpha_{\mathsf{S}}$) on the left of Fig. 3. For the sake of readability, in the following we identify each abstract element with its meaning, e.g., $\mathbb{Z}_{\leq 0}$ with $\{n \mid n \leq 0\}$. As observe previously, both the boolean expressions in $\mathtt{r}$ are expressible (and therefore quasi-expressible) in *Sign*, since $x \leq 0$ is expressed by $\mathbb{Z}_{\leq 0}$ and $0 \leq x$ by $\mathbb{Z}_{\geq 0}$. By Th. 1 we have that both $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{d}}(x \leq 0?)$ and $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{d}}(0 < x?)$ hold for any $\mathtt{d}$. Let us consider an input property $\mathtt{c} \stackrel{\text{def}}{=} \{-100, -10, 0\}$ ($\mathit{Sign}(\mathtt{c}) = \mathbb{Z}_{\leq 0}$), then $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(x \leq 0?)$ and therefore by rule `(transfer)`:

$$\frac{\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(x \leq 0?)}{\vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ x \leq 0?\ \langle \mathbb{Z}_{\leq 0} \rangle}$$

being $[\![x \leq 0?]\!]\mathit{Sign}(\mathtt{c}) = [\![x \leq 0?]\!]\mathbb{Z}_{\leq 0} = \mathbb{Z}_{\leq 0}$.
Now, note that $\mathit{Sign}\,[\![x := x * 2]\!]\,\mathit{Sign}(\mathbb{Z}_{\leq 0}) = \mathit{Sign}\,[\![x := x * 2]\!]\,\mathbb{Z}_{\leq 0} = \mathbb{Z}_{\leq 0} \subsetneq \top$, hence $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathbb{Z}_{\leq 0}}(x := x * 2)$, and therefore we can apply again rule `(transfer)`

$$\frac{\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathbb{Z}_{\leq 0}}(x := x * 2)}{\vdash_{\mathsf{A}} \langle \mathbb{Z}_{\leq 0} \rangle\ x := x * 2\ \langle \mathbb{Z}_{\leq 0} \rangle}$$

since $[\![x := x * 2]\!]\mathit{Sign}(\mathbb{Z}_{\leq 0}) = \mathbb{Z}_{\leq 0}$. Finally by rule `(seq)`

$$\frac{\vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ x \leq 0?\ \langle \mathbb{Z}_{\leq 0} \rangle \quad \vdash_{\mathsf{A}} \langle \mathbb{Z}_{\leq 0} \rangle\ x := x * 2\ \langle \mathbb{Z}_{\leq 0} \rangle}{\vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ x \leq 0?; x := x * 2\ \langle \mathbb{Z}_{\leq 0} \rangle}$$

Now, we can apply rule `(iterate)` obtaining

$$\frac{\vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ x \leq 0?; x := x * 2\ \langle \mathbb{Z}_{\leq 0} \rangle \quad \mathit{Sign}(\mathbb{Z}_{\leq 0}) = \mathbb{Z}_{\leq 0} = \mathit{Sign}(\mathtt{c}) \subsetneq \top}{\vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ (x \leq 0?; x := x * 2)^*\ \langle \mathtt{c} \rangle}$$

Finally, by Th. 1 we have $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(0 \leq x?)$, implying again by rule `(transfer)`

$$\frac{\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(0 \leq x?)}{\vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ 0 \leq x?\ \langle \mathbb{Z}_{=0} \rangle}$$

since $[\![0 \leq x]\!]\mathit{Sign}(\mathtt{c}) = [\![0 \leq x]\!]\mathbb{Z}_{\leq 0} = \mathbb{Z}_{=0}$ In this way we can conclude, by rule `(seq)`, that $\vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ (x \leq 0?; x := x * 2)^*; 0 \leq x?\ \langle \mathbb{Z}_{=0} \rangle$, meaning that $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})$.

Example 5. Let us consider the regular program $\mathtt{r} \stackrel{\text{def}}{=} \mathtt{r}_1 \oplus \mathtt{r}_2$, where

$$\begin{array}{l} \mathtt{r}_1 \stackrel{\text{def}}{=} (0 < x?; x := x - 1)^* \\ \mathtt{r}_2 \stackrel{\text{def}}{=} (x < 100?; x := x + 1)^* \end{array}$$

Let us consider as abstract domain $\mathsf{A} = \mathit{Int}$ (let $\mathsf{A} = \gamma_{\mathrm{I}}\alpha_{\mathrm{I}}$), $\tau = \gamma_{\mathrm{I}}([0, +\infty])$ and $\mathtt{c} \stackrel{\text{def}}{=} \{0, 10, 20, 30\}$ ($\alpha_{\mathrm{I}}(\mathtt{c}) = [0, 30]$). In the following, for the sake of readability, we identify $[n, m]$ with its meaning $\{i \mid n \leq i \leq m\}$. As already observed, in the considered abstract domain $0 < x$ is expressible w.r.t. τ, while $x < 100$ is not. By Th. 1, we have $\forall \mathtt{d} \in \mathsf{C}$ that $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{d}}(0 < x?)_\tau$ while we have that $\forall \mathtt{d} \subsetneq \tau$ $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{d}}(x < 100?)_\tau$, since $\mathit{Int}([\![x < 100?]\!]\mathtt{d}) \subseteq [0, 100] \subsetneq \tau$.
Let us consider $\mathtt{r}_1$ first. By rule `(transfer)`

$$\frac{\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(0 < x?)_\tau}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ 0 < x?\ \langle [1, 30] \rangle}$$

Now we have that $[\![x := x - 1]\!]\mathit{Int}([1, 30]) = [0, 29]$, hence again by rule `(transfer)` we derive

$$\frac{\widetilde{\mathbb{C}}^{\mathsf{A}}_{[1,30]}(x := x - 1)_\tau}{\tau \vdash_{\mathsf{A}} \langle [1, 30] \rangle\ x := x - 1\ \langle [0, 29] \rangle}$$

being $\mathit{Int}([\![x := x - 1]\!]\mathit{Int}([1, 30]))[0, 29] \subsetneq \tau$, and therefore by rule `(seq)` we have $\vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ 0 < x?; x := x - 1\ \langle [0, 29] \rangle$. Now, by rule `(iterate)`, we prove

$$\frac{\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ 0 < x?; x := x - 1\ \langle [0, 29] \rangle \quad \mathit{Int}([0, 29]) = [0, 29] \subseteq \mathit{Int}(\mathtt{c}) = [0, 30] \subsetneq \tau}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ (0 < x?; x := x - 1)^*\ \langle \mathtt{c} \rangle}$$

Let us consider $\mathtt{r}_2$. By rule `(transfer)`

$$\frac{\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(x < 100?)_\tau}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ x < 100?\ \langle [0, 30] \rangle}$$

Now we have that $[\![x := x + 1]\!][0, 30]) = [1, 31]$, hence again by rule `(transfer)` we derive $\vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ x := x + 1\ \langle [1, 31] \rangle$, and therefore $\vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ x < 100?; x := x + 1\ \langle [1, 31] \rangle$ by rule `(seq)`. In this case, unfortunately $\mathit{Int}([1, 31]) = [1, 31] \not\subseteq \mathit{Int}(\mathtt{c}) = [1, 30]$, hence we cannot apply rule `(iterate)`. Let us consider, instead $\mathtt{c}' \stackrel{\text{def}}{=} \{0, 10, 20, 30, 100\}$ ($\mathit{Int}(\mathtt{c}') = [0, 100]$). Then by rule `(transfer)`

$$\frac{\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}'}(x < 100?)_\tau}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}' \rangle\ x < 100?\ \langle [0, 99] \rangle}$$

then being $[\![x := x + 1]\!][0, 99]) = [1, 100]$, we have by rule `(transfer)`

$$\frac{\widetilde{\mathbb{C}}^{\mathsf{A}}_{[0,99]}(x := x + 1)_\tau}{\tau \vdash_{\mathsf{A}} \langle [0, 99] \rangle\ x := x + 1\ \langle [1, 100] \rangle}$$

and therefore $\vdash_{\mathsf{A}} \langle \mathtt{c}' \rangle\ x < 100?; x := x + 1\ \langle [1, 100] \rangle$ by rule `(seq)`. Now by rule `(iterate)`

$$\frac{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}' \rangle\ x < 100?; x := x + 1\ \langle [1, 100] \rangle \quad \mathit{Int}([1, 100]) = [1, 100] \subseteq \mathit{Int}(\mathtt{c}') = [0, 100] \subsetneq \tau}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c}' \rangle\ (x < 100?; x := x + 1)^*\ \langle \mathtt{c}' \rangle}$$

At this point, by rule $(\mathtt{relax}_2)$

$$\frac{\mathtt{c} \subseteq \mathtt{c}' \quad \tau \vdash_{\mathsf{A}} \langle \mathtt{c}' \rangle\ (x < 100?; x := x + 1)^*\ \langle \mathtt{c}' \rangle}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ (x < 100?; x := x + 1)^*\ \langle \mathtt{c}' \rangle}$$

and therefore we can apply rule `(join)`

$$\frac{\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ (0 < x?; x := x - 1)^*\ \langle \mathtt{c} \rangle \quad \tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ (x < 100?; x := x + 1)^*\ \langle \mathtt{c}' \rangle \quad \alpha(\mathtt{c}') \lesssim_{\mathsf{A}} \tau_\alpha}{\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ \mathtt{r}\ \langle \mathtt{c}' \rangle}$$

since $\mathtt{c} \cup \mathtt{c}' = \mathtt{c}'$, meaning that $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})_\tau$.

Example 6. In this example, we show that we can derive limits also for diverging loops. Consider $\mathtt{r} = (x \leq 0?; x := 2{*}x)^*$, $\mathsf{A} = \mathit{Int}$ (let $\mathsf{A} = \gamma_{\mathrm{I}}\alpha_{\mathrm{I}}$) and $\tau \stackrel{\text{def}}{=} [-\infty, 0]$. In this case we can observe that if we start from a finite $\mathtt{c}$ then it is not possible to apply rule `iterate` since we enlarge at least one bound of the interval, while if we compute adequacy on a point including the limit we can prove adequacy.
Let $\mathtt{c}$ be such that $\alpha_{\mathrm{I}}(\mathtt{c}) = [n, 0] \subsetneq [-\infty, 0]$, then $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ x \leq 0?\ \langle \mathtt{c} \rangle$, while $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ x := 2 * x\ \langle \mathtt{d} \rangle$ with $-2n \in \mathtt{d}$, hence $\alpha_{\mathrm{I}}(\mathtt{d}) \not\subseteq \alpha_{\mathrm{I}}(\mathtt{c})$. It is trivial to observe that the only case in which we obtain a result contained in the starting point $\mathtt{c}$ is when $\mathtt{c} = [-\infty, n]$ $(n \leq 0)$, since in this case we can derive, in the proof system, that $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ x := 2 * x\ \langle \mathtt{d} \rangle$ with $\mathtt{d} \subseteq [-\infty, 2n] \subseteq [-\infty, n]$.
Let, for instance $n = -10$, then trivially $\tau \vdash_{\mathsf{A}} \langle [-\infty, -10] \rangle\ x \leq 0?\ \langle [-\infty, -10] \rangle$, $\tau \vdash_{\mathsf{A}} \langle [-\infty, -10] \rangle\ x := 2 * x\ \langle [-\infty, -20] \rangle$, hence, by using the composition rule, we prove $\tau \vdash_{\mathsf{A}} \langle [-\infty, -10] \rangle\, x \leq 0?; x := 2 * x\, \langle [-\infty, -20] \rangle$ and being $[-\infty, -20] \subseteq [-\infty, -10]$, we can derive $\tau \vdash_{\mathsf{A}} \langle [-\infty, -10] \rangle\, x \leq 0?; x := 2 * x\, \langle [-\infty, -10] \rangle$. Hence, $\forall \mathtt{c} \in \left\{\ [-\infty, n] \,\middle|\, n < 0\ \right\}$ we have that $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathtt{r})_{[-\infty, 0]}$.

Example 7. Let us consider the regular command in Reg

$$\mathtt{r} \stackrel{\text{def}}{=} (\mathit{even}(x)?; x := x + 1)^*; \oplus(\neg \mathit{even}(x)?; x := x * 2)$$

supposing to add to the language the operator $\mathit{even}(x)$ returning true if x is even, false otherwise, with $\mathsf{A} = \mathit{Int}$ (let $\mathsf{A} = \gamma_{\mathrm{I}}\alpha_{\mathrm{I}}$) and $\tau = \top$. It should be clear that $\mathit{even}(x)$ is not quasi-expressible in A, however we can prove adequacy on some points $\mathtt{c}$. Let $\mathtt{c} = [n, +\infty]$, with $n > 0$, then we have that $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle \mathit{even}(x)? \langle \mathtt{c} \cap 2\mathbb{Z} \rangle$, since we can trivially prove that $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{c}}(\mathit{even}(x))_\tau$. At this point, it holds that $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \cap 2\mathbb{Z} \rangle x := x + 1 \langle [n+1, +\infty] \cap 2\mathbb{Z} + 1 \rangle$ by rule `(transfer)`. Hence, by rule `(seq)` we have $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\, \mathit{even}(x); x := x + 1 \langle [n+1, +\infty] \cap 2\mathbb{Z} + 1 \rangle$, and being $n > 0$ we have $\mathtt{d} = [n + 1, +\infty] \subseteq [n, +\infty]$ meaning that $\alpha_{\mathrm{I}}(\mathtt{d} \cap 2\mathbb{Z} + 1) = \mathtt{d} \subseteq \alpha_{\mathrm{I}}(\mathtt{c}) = \mathtt{c}$, and therefore $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\ (\mathit{even}(x)?; x := x + 1)^*\ \langle \mathtt{c} \rangle$, rule `(seq)` and by rule `(iterate)`.

On the other hand, $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle \neg even(x)?\, \langle \mathtt{c} \cap 2\mathbb{Z}+1 \rangle$, while $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \cap 2\mathbb{Z}+1 \rangle\, x := 2x\, \langle [2n,+\infty] \cap 2\mathbb{Z} \rangle$, both by rule `(transfer)`, with $[n,+\infty] \supseteq [2n,+\infty]$.
Finally, we have that $\mathtt{c} \cup ([2n,+\infty] \cap 2\mathbb{Z}) = [n,+\infty] \cup ([2n,+\infty] \cap 2\mathbb{Z}) = \mathtt{c}$, which means that $\alpha_{\mathrm{I}}(\mathtt{c}) = [n,+\infty] \subsetneq \tau$. Then we conclude, by rule `(join)`, that $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\, \mathtt{r}\, \langle \mathtt{c} \rangle$, i.e., adequacy w.r.t. τ is satisfied.

Example 8. Let us consider a final example with a relational domain such that Octagons [25, 26] for the regular command in Reg

$$\mathtt{r} \stackrel{\text{def}}{=} (y - x \leq 0?; y := y - 1)^*; x - y \leq 0;$$

with $\mathsf{A} = Oct$ (let $\mathsf{A} = \gamma_0 \alpha_0$) and $\tau = \{x \leq 5, y \leq 7\}$. In this case, the guard is quasi-expressible. Let us consider $\mathtt{c} \stackrel{\text{def}}{=} \{y \leq 2, x \leq 1, y - x \leq 2\}$. By Th. 1, we have $\forall \mathtt{d} \in \mathsf{C}$ that $\widetilde{\mathbb{C}}^{\mathsf{A}}_{\mathtt{d}}(y - x \leq 0?)_\tau$, hence, by rule `(transfer)`, $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\, y - x \leq 0?\, \langle \{y \leq 2, x \leq 1, y - x \leq 0\} \rangle$. Then, by rule `(transfer)` we have also that $\tau \vdash_{\mathsf{A}} \langle \{y \leq 2, x \leq 1, y - x \leq 0\} \rangle\, y := y - 1\, \langle \{y \leq 1, x \leq 1, y - x \leq -1\} \rangle$ and by rule `(seq)` we have $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\, y - x \leq 0?; y := y - 1\, \langle \{y \leq 1, x \leq 1, y - x \leq -1\} \rangle$. Then, since $\{y \leq 2, x \leq 1, y - x \leq -1\} \leq_{\mathsf{A}} \mathtt{c}$, we can apply rule `(iterate)` obtaining $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\, (y - x \leq 0?; y := y - 1)^*\, \langle \mathtt{c} \rangle$.
Now, also $x - y \leq 0$ is expressible, hence we have that, by rule `(transfer)`, $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\, x - y \leq 0?\, \langle \{y \leq 2, x \leq 1, x - y \leq 0, y - x \leq 2\} \rangle$, Hence, we can prove that $\tau \vdash_{\mathsf{A}} \langle \mathtt{c} \rangle\, (y - x \leq 0?; y := y - 1)^*; x - y \leq 0?\, \langle \{y \leq 2, x \leq 1, x - y \leq 0, y - x \leq 2\} \rangle$, again by rule `(seq)`, and therefore we prove adequacy on `c`.

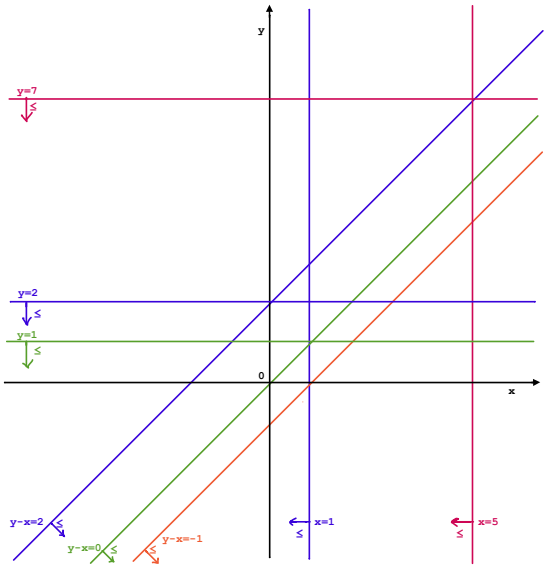

Fig. 5. Graphical representation of octagons in Example 8

6 Conclusions

We introduced the novel notion of adequate abstract domain together with a refinement strategy to adjust abstract domains to make them adequate and a program logic to check whether the abstract interpretation designed on a given abstract domain is adequate. Our program logic is simple and can be checked online during program analysis.
Adequacy is particularly interesting when we are interested to prove whether our abstract interpreter incorporates enough information (here encoded by the bound τ) in the computed invariant. This can have applications beyond program analysis, for instance in language-based security and code protection. In language-based security, as specified by abstract non-interference [21, 23, 15], adequacy could guarantee that certain amount of information is kept secret or dually it is released in the case of declassification, and this could be analyzed also for dynamic code [1, 24]. In code protection the notion of adequacy become stronger than completeness. The standard approach to protect code against program analysis is to make the abstract interpreter maximally imprecise with respect to the given program, which means incomplete [18, 16, 22]. In this case we may replace completeness with adequacy, and imagine code protecting transformations making an abstract interpreter inadequate for the transformed code. When the chosen bound is $\top$ this corresponds to have a code transformation for which the abstract interpreter is totally blind and cannot extract any meaningful information. Finally, for program verification, it could be interesting to exploit the idea introduced for partial completeness [7], i.e., the use of a metric for weakening completeness, for strengthening adequacy by fixing a maximal distance that we want to guarantee from the fixed bound τ.

References

1. Arceri, V., Mastroeni, I.: Analyzing dynamic code: A sound abstract interpreter for evil eval. ACM Trans. Priv. Secur. **24**(2), 10:1–10:38 (2020)
2. Bourdoncle, F.: Abstract interpretation by dynamic partitioning. Journal of Functional Programming **2**(4), 407–435 (1992)
3. Bruni, R., Giacobazzi, R., Gori, R., Garcia-Contreras, I., Pavlovic, D.: Abstract extensionality: on the properties of incomplete abstract interpretations. Proc. ACM Program. Lang. **4**(POPL), 28:1–28:28 (2020). https://doi.org/10.1145/3371096, https://doi.org/10.1145/3371096
4. Bruni, R., Giacobazzi, R., Gori, R., Ranzato, F.: A logic for locally complete abstract interpretations. In: Symposium on Logic in Computer Science, LICS. pp. 1–13. IEEE (2021)
5. Bruni, R., Giacobazzi, R., Gori, R., Ranzato, F.: Abstract interpretation repair. In: Jhala, R., Dillig, I. (eds.) PLDI '22: 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, San Diego, CA, USA, June 13 - 17, 2022. pp. 426–441. ACM (2022)
6. Bruni, R., Giacobazzi, R., Gori, R., Ranzato, F.: A correctness and incorrectness program logic. J. ACM **70**(2) (2023)

7. Campion, M., Preda, M.D., Giacobazzi, R.: Partial (in)completeness in abstract interpretation: limiting the imprecision in program analysis. Proc. ACM Program. Lang. **6**(POPL), 1–31 (2022). https://doi.org/10.1145/3498721, https://doi.org/10.1145/3498721
8. Cousot, P.: Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice. Res. rep. R.R. 88, Laboratoire IMAG, Université scientifique et médicale de Grenoble, Grenoble, France (Sep 1977), 15 p.
9. Cousot, P.: Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. Theor. Comput. Sci. **277**(1-2), 47–103 (2002)
10. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Conference Record of the 4th ACM Symposium on Principles of Programming Languages (*POPL '77*). pp. 238–252. ACM Press (1977)
11. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Conference Record of the 6th ACM Symposium on Principles of Programming Languages (*POPL '79*). pp. 269–282. ACM Press (1979)
12. Cousot, P., Cousot, R.: Comparing the Galois connection and widening/narrowing approaches to abstract interpretation (Invited Paper). In: Bruynooghe, M., Wirsing, M. (eds.) Proc. of the 4th Internat. Symp. on Programming Language Implementation and Logic Programming (*PLILP '92*). Lecture Notes in Computer Science, vol. 631, pp. 269–295. Springer-Verlag (1992)
13. Cousot, P.: Principles of Abstract Interpretation. MIT Press (2021)
14. Filé, G., Giacobazzi, R., Ranzato, F.: A unifying view of abstract domain design. ACM Comput. Surv. **28**(2), 333–336 (1996)
15. Giacobazzi, R., Mastroeni, I.: Adjoining classified and unclassified information by abstract interpretation. Journal of Computer Security **18**(5), 751 – 797 (2010)
16. Giacobazzi, R., Mastroeni, I.: Making abstract interpretation incomplete - modeling the potency of obfuscation. In: Miné, A., Schmidt, D. (eds.) 19th International Static Analysis Symp. (SAS '12). Lecture Notes in Computer Science, vol. 7460, pp. 129 – 145 (2012)
17. Giacobazzi, R., Ranzato, F., Scozzari., F.: Making abstract interpretation complete. Journal of the ACM **47**(2), 361–416 (March 2000)
18. Giacobazzi, R., Jones, N.D., Mastroeni, I.: Obfuscation by partial evaluation of distorted interpreters. In: Kiselyov, O., Thompson, S.J. (eds.) Proceedings of the ACM SIGPLAN 2012 Workshop on Partial Evaluation and Program Manipulation, PEPM 2012, Philadelphia, Pennsylvania, USA, January 23-24, 2012. pp. 63–72. ACM (2012)
19. Giacobazzi, R., Logozzo, F., Ranzato, F.: Analyzing program analyses. In: Rajamani, S.K., Walker, D. (eds.) Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015. pp. 261–273. ACM (2015)
20. Giacobazzi, R., Mastroeni, I.: Making abstract models complete. Mathematical Structures in Computer Science **26**(4), 658–701 (2016)
21. Giacobazzi, R., Mastroeni, I.: Abstract non-interference: A unifying framework for weakening information-flow. ACM Trans. Priv. Secur. **21**(2), 1–31 (2018)
22. Giacobazzi, R., Mastroeni, I., Preda, M.D.: Maximal incompleteness as obfuscation potency. Formal Aspects Comput. **29**(1), 3–31 (2017)
23. Mastroeni, I.: Abstract interpretation-based approaches to security - A survey on abstract non-interference and its challenging applications. In: Banerjee, A., Danvy,

O., Doh, K., Hatcliff, J. (eds.) Semantics, Abstract Interpretation, and Reasoning about Programs: Essays Dedicated to David A. Schmidt on the Occasion of his Sixtieth Birthday, Manhattan, Kansas, USA, 19-20th September 2013. EPTCS, vol. 129, pp. 41–65 (2013)
24. Mastroeni, I., Arceri, V.: Improving dynamic code analysis by code abstraction. In: Lisitsa, A., Nemytykh, A.P. (eds.) Proceedings of the 9th International Workshop on Verification and Program Transformation, VPT@ETAPS 2021, Luxembourg, Luxembourg, 27th and 28th of March 2021. EPTCS, vol. 341, pp. 17–32 (2021)
25. Minè, A.: The octagon abstract domain. In: AST 2001 in WCRE 2001. pp. 310–319. IEEE, IEEE CS Press (October 2001)
26. Miné, A.: The octagon abstract domain. Higher Order Symbol. Comput. **19**(1), 31–100 (2006). https://doi.org/http://dx.doi.org/10.1007/s10990-006-8609-1
27. Müller, M.N., Fischer, M., Staab, R., Vechev, M.: Abstract interpretation of fixpoint iterators with applications to neural networks. Proc. ACM Program. Lang. **7**(PLDI) (2023)
28. O'Hearn, P.W.: Incorrectness logic. Proceedings of the ACM on Programming Languages (POPL) **4**(10) (2020)
29. Winskel, G.: The formal semantics of programming languages: an introduction. MIT press (1993)