The Best of Abstract Interpretations

ROBERTO GIACOBAZZI, University of Arizona, USA
FRANCESCO RANZATO, University of Padova, Italy

We study *"the best of abstract interpretations"*, that is, the best possible abstract interpretations of programs. Abstract interpretations are inductively defined by composing abstract transfer functions for the basic commands, such as assignments and Boolean guards. However, abstract interpretation is not compositional: even if the abstract transfer functions of the basic commands are the best possible ones on a given abstract domain A this does not imply that the whole inductive abstract interpretation of a program p is still the best in A. When this happens we are in the optimal scenario where the abstract interpretation of p coincides with the abstraction of the concrete interpretation of p. Our main contributions are threefold. Firstly, we investigate the computability properties of the class of programs having the best possible abstract interpretation on a fixed abstract domain A. We show that this class is, in general, not straightforward and not recursive. Secondly, we prove the impossibility of achieving the best possible abstract interpretation of any program p either by an effective compilation of p or by minimally refining or simplifying the abstract domain A. These results show that the program property of having the best possible abstract interpretation is not trivial and, in general, hard to achieve. We then show how to prove that the abstract interpretation of a program is indeed the best possible one. To this aim, we put forward a program logic parameterized on an abstract domain A which infers triples $[pre]_A$ p $[post]_A$. These triples encode that the inductive abstract interpretation of p on A with abstract input $pre \in A$ gives $post \in A$ as abstract output and this is the best possible in A.

CCS Concepts: • **Theory of computation → Logic and verification**; **Semantics and reasoning**; **Pre- and post-conditions**; **Program analysis**; **Abstraction**.

Additional Key Words and Phrases: Abstract interpretation, program analysis, best correct approximation, proof system, logic of programs.

ACM Reference Format:
Roberto Giacobazzi and Francesco Ranzato. 2025. The Best of Abstract Interpretations. *Proc. ACM Program. Lang.* 9, POPL, Article 46 (January 2025), 31 pages. https://doi.org/10.1145/3704882

1 Introduction

Cousot and Cousot [1979] defined already in their seminal POPL1979 paper the notion of *best correct approximation* (bca, originally called "best correct upper approximation" and denoted $\bar{\tau}$ in [Cousot and Cousot 1979, Theorem 7.2.0.3]) of a generic predicate transformer to provide a necessary and sufficient condition for the soundness of an abstract interpretation: any abstract interpretation is sound if and only if it over-approximates the bca [Cousot and Cousot 1979, Theorem 7.2.0.3]. As the terminology hints, the bca of a predicate transformer f represents the best possible function that soundly approximates f in a given abstract domain A. Given a predicate transformer $f : C \to C$ defined in a domain C of concrete program properties (e.g., sets of program stores for imperative programs), and an abstract domain A as specified in the Galois connection-based

Authors' Contact Information: Roberto Giacobazzi, Department of Computer Science, University of Arizona, Tucson, AZ, USA, giacobazzi@arizona.edu; Francesco Ranzato, Dipartimento di Matematica, University of Padova, Padova, Italy, ranzato@math.unipd.it.

ACM 2475-1421/2025/1-ART46
https://doi.org/10.1145/3704882

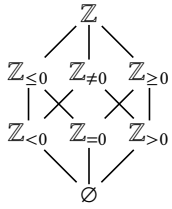

Fig. 1. The Abstract Domain Sign.

abstract interpretation framework, i.e., by abstraction and concretization maps, resp. $\alpha : C \to A$ and $\gamma : A \to C$, the bca of f is defined as $f^A \triangleq \alpha \circ f \circ \gamma : A \to A$. Among all sound approximations of f, f^A is the most precise one—that is, *the best*—w.r.t. the usual pointwise ordering between functions.

There are several advantages of being bca with respect to other, possibly non-bca, sound abstract interpretations: (1) bcas are the *most precise* over-approximations in A, hence, they provide the least number of false alarms and yield an absolute upper bound to the precision that A can achieve; (2) bcas are *systematically induced* by the concrete inductive program semantics $[\![\cdot]\!]$—i.e., the predicate transformer semantics—and by the abstract domain A; (3) bcas induce *extensional* program equivalences $\sim_A$, that is, $\mathsf{p} \sim_A \mathsf{q} \Leftrightarrow \alpha \circ [\![\mathsf{p}]\!] \circ \gamma = \alpha \circ [\![\mathsf{q}]\!] \circ \gamma$ is an extensional program relation *à la* Rice [1953], and $[\![\mathsf{p}]\!] = [\![\mathsf{q}]\!] \Rightarrow \mathsf{p} \sim_A \mathsf{q}$ holds, meaning that the output of best abstract interpretations *is not affected by how the code under analysis is written.*

1.1 The Problem

It turns out that the property of being bca *is not compositional*, meaning that even if the abstract interpretation of a program p is defined by leveraging abstract predicate transformers for the basic commands of p, e.g., assignments and Boolean guards, which are bcas, it may well happen that this abstract interpretation of p is not the best. For instance, the functional composition of two bcas, in general, is not the bca of the composition, namely, by composing bcas, we may lose precision. As a simple example, consider the following two programs defined on integer variables

$$\mathsf{p}_1(x) \triangleq \textbf{if } \mathit{even}(x) \textbf{ then } x \textbf{ else } x+1 \qquad \text{and} \qquad \mathsf{p}_2(x) \triangleq \textbf{if } \mathit{even}(x) \textbf{ then } 0 \textbf{ else } 1$$

and their sequential composition $\mathsf{p}_1;\mathsf{p}_2$ such that $[\![\mathsf{p}_1;\mathsf{p}_2]\!] = \lambda X \in \wp(\mathbb{Z}).\{0\}$. In the case of the well-known interval analysis Int [Cousot and Cousot 1977], we have that:

$$\mathsf{p}_1^{\mathsf{Int}}[0,1] = [0,0] \sqcup_{\mathsf{Int}} ([1,1]+1) = [0,0] \sqcup_{\mathsf{Int}} [2,2] = [0,2]\,, \quad \mathsf{p}_2^{\mathsf{Int}}[0,2] = [0,0] \sqcup_{\mathsf{Int}} [1,1] = [0,1]\,,$$
$$(\mathsf{p}_1;\mathsf{p}_2)^{\mathsf{Int}}[0,1] = [0,0] \subsetneq [0,1] = \mathsf{p}_2^{\mathsf{Int}}\mathsf{p}_1^{\mathsf{Int}}[0,1]\,,$$

showing that the bca property is not compositional. Consider $\mathsf{q}_1(x) \triangleq x := x+1;\ x := x-1$ and $\mathsf{q}_2(x) \triangleq x := x+1$, and the sign abstract domain Sign in Figure 1, where the notation of its abstract values provides their concrete meaning. For the sequential composition $\mathsf{q}_1;\mathsf{q}_2$, we have that:

$$[\![\mathsf{q}_1]\!](\mathbb{Z}_{>0}) = \mathbb{Z}_{>0} \qquad [\![\mathsf{q}_2]\!](\mathbb{Z}_{\geq 0}) = \mathbb{Z}_{>0} \qquad [\![\mathsf{q}_1;\mathsf{q}_2]\!](\mathbb{Z}_{>0}) = \mathbb{Z}_{>1}$$
$$\mathsf{q}_1^{\mathsf{Sign}}(\mathbb{Z}_{>0}) = (x := x-1)^{\mathsf{Sign}}(x := x+1)^{\mathsf{Sign}}(\mathbb{Z}_{>0}) = \mathbb{Z}_{\geq 0} \neq \mathbb{Z}_{>0} = \mathsf{Sign}([\![\mathsf{q}_1]\!](\mathbb{Z}_{>0}))$$
$$\mathsf{q}_2^{\mathsf{Sign}}(\mathbb{Z}_{\geq 0}) = (x := x+1)^{\mathsf{Sign}}(\mathbb{Z}_{\geq 0}) = \mathbb{Z}_{>0} = \mathsf{Sign}([\![\mathsf{q}_2]\!](\mathbb{Z}_{\geq 0}))$$
$$(\mathsf{q}_1;\mathsf{q}_2)^{\mathsf{Sign}}(\mathbb{Z}_{>0}) = \mathsf{q}_2^{\mathsf{Sign}}\mathsf{q}_1^{\mathsf{Sign}}(\mathbb{Z}_{>0}) = \mathbb{Z}_{>0} = \mathsf{Sign}([\![\mathsf{q}_1;\mathsf{q}_2]\!](\mathbb{Z}_{>0}))\,.$$

This example shows that $(\mathsf{q}_1;\mathsf{q}_2)^{\mathsf{Sign}}(\mathbb{Z}_{>0})$ provides $\mathbb{Z}_{>0}$ as output in Sign, which is the best abstraction in Sign of the concrete output $\mathbb{Z}_{>1}$. However, this best abstraction for the composite program

$\mathrm{q}_1;\mathrm{q}_2$ does not arise from the composition of the two best abstractions for the subprograms q_1 and q_2, because $\mathrm{q}_1^{\mathsf{Sign}}(\mathbb{Z}_{>0}) = \mathbb{Z}_{\geq 0}$, which is not the best abstraction in Sign of the concrete output $\mathbb{Z}_{>0}$.

Program interpreters, abstract interpreters included, are compositionally designed, namely inductively designed on the program's syntax. Thus, the lack of compositionality of the property of being the best abstract interpreter is the main reason of the dependence between the precision of the abstract interpretation of programs and the way programs are written, namely of the *inherent intensional nature of abstract interpretation* [Bruni et al. 2020]. As a consequence of this fundamental property of abstract interpretation, the following natural questions arise:

- (RQ_1) What is the relationship between the intensionality of abstract interpretation and the lack of compositionality of the best abstract interpreters?
- (RQ_2) Given a program p and an abstract domain A, does the minimal refinement/simplification transform $\mathbb{T}_{\mathrm{p}}(A)$ of A making the abstract interpretation of p on $\mathbb{T}_{\mathrm{p}}(A)$ the best possible one exist?
- (RQ_3) Given a program p and an abstract domain A, can we decide or prove whether the inductive abstract interpretation of p in A is the best possible one?

It is worth remarking that (RQ_3) comes as a generalization of [Cousot and Cousot 1979, Example 7.2.0.5], where Cousot and Cousot presented "a challenge to automatic program synthesizers". In their simple example, the challenge was to synthesize the interval abstract interpretation for

$$\mathrm{p}(x, y) \triangleq \textbf{if } x \leq y \textbf{ then } \langle x, y\rangle$$

in such a way that it is the best possible interpretation on Int. The goal of [Cousot and Cousot 1979, Example 7.2.0.5] was to prove that the following solution

$$\mathrm{p}^{\mathsf{Int}}(\langle x \in [a, b], y \in [c, d]\rangle) = \langle x \in [a, b] \sqcap_{\mathsf{Int}} [-\infty, d], y \in [c, d] \sqcap_{\mathsf{Int}} [a, +\infty]\rangle$$

already given in their first POPL77 paper [Cousot and Cousot 1977, Section 9.2] was correct.

1.2 Contributions

Our first contributions address (RQ_1) and (RQ_3) by exploring the computability aspects of the class of programs that achieve the best inductive abstract interpretation over a fixed abstract domain A, denoted by $\mathrm{BCA}(A)$. The investigation of the recursivity properties of this class of programs is preliminary to any attempt to provide a decision procedure to check whether a given program admits the best inductive abstract interpretation on a given abstract domain. We first show that $\mathrm{BCA}(A)$ is not straightforward, which means that for any non-trivial abstract domain A there always exists a program p_A such that the inductive abstract interpretation of p_A in A is not bca. We also show that for any non-trivial abstract domain A such that, for all programs p, the inductive abstract semantics $[\![\mathrm{p}]\!]_A^\sharp$ is decidable (e.g., in static program analysis), it turns out that fixed an abstract input $a \in A$, there does not exist any effective program compilation τ that transforms a program p into a semantically equivalent program $\tau(\mathrm{p})$ for the concrete input $\gamma(a)$, and at the same time, $\tau(\mathrm{p})$ achieves the bca property on input a. We then prove that the class of programs $\mathrm{BCA}(A)$ is not recursive, meaning that it is undecidable whether the abstract interpretation of a program is the best or not, thus answering the decidability question of (RQ_3). Finally, we show that the bca is the only possible sound abstract semantics which induces an extensional program equivalence. All these contributions have several implications: (1) complete program logics for proving the bca property cannot exist, hence we need to rely upon approximate (sound) logics; (2) Rice's theorem cannot be applied to prove that $\mathrm{BCA}(A)$ is nonrecursive; and (3) programs cannot be compiled into equivalent programs enjoying the bca property; hence, it is not possible to maximize the precision of an abstract interpreter (i.e., make it bca) solely by transforming the code under analysis.

To settle (RQ_2), we show that given an abstract domain A and a program p: (i) the least domain refinement A_r of A such that p satisfies the bca property in A_r, in general, does not exist; (ii) the greatest domain abstraction A_a of A such that p satisfies the bca property in A_a does not exist. These results demonstrate that, unlike the case of completeness [Giacobazzi et al. 2000], the weaker property of being the best possible abstract interpretation in a domain A is inherently harder to achieve, as it cannot be achieved through a minimal transformation of A.

As a consequence of the aforementioned impossibility of deciding whether a program admits a best inductive abstract interpretation, as well as the inability to obtain such bca property through code transformation or abstract domain refinement, a natural question arises: *How do we know if the result $b \in A$ of an abstract interpretation of a program* p *with input $a \in A$ is the bca of* p *in A?* A positive answer can tell us whether our analysis of p achieved the best possible result in A. This is a question concerning the quality of the analysis of p, and requires a *meta-analysis*, in the sense of the A^2I framework by Cousot et al. [2019]. To achieve this goal, we design a program logic to prove whether the abstract interpretation of a program is the best possible one in a fixed abstract domain. Of course, due to the impossibility results mentioned above, any such program logic must be approximated, i.e. some proofs can be missed. Our program logic is parameterized on an abstract domain A and infers program triples of the form $[a]_A\ \mathsf{p}\ [b]_A$, where $a, b \in A$. A triple $[a]_A\ \mathsf{p}\ [b]_A$ is valid in our logic if: (i) b is the inductive analysis of p in A on input a; (ii) the inductive analysis of p in A on input a is the bca of p in A with the same input. The most distinctive rules of this bca logic, denoted by $\vdash_{\mathrm{bca}}$, are the following:

$$\frac{\vdash_{\mathrm{bca}} [a]_A\ \mathsf{p}_1\ [c]_A \quad \vdash_{\mathrm{bca}} [c]_A\ \mathsf{p}_2\ [b]_A \quad \gamma(c) = [\![\mathsf{p}_1]\!]\gamma(a)}{\vdash_{\mathrm{bca}} [a]_A\ \mathsf{p}_1; \mathsf{p}_2\ [b]_A}\ (\mathsf{seq}_\gamma)$$

$$\frac{\vdash_{\mathrm{bca}} [a]_A\ \mathsf{p}\ [b]_A \quad b \leq_A a}{\vdash_{\mathrm{bca}} [a]_A\ \mathsf{p}^{\mathrm{fix}}\ [a]_A}\ (\text{abs-inv}) \qquad \frac{\exists n \geq 1.\ \vdash_{\mathrm{bca}} [a]_A\ \overbrace{\mathsf{p}; \cdots ; \mathsf{p}}^{n}\ [b]_A \quad \vdash_{\mathrm{bca}} [a \vee_A b]_A\ \mathsf{p}^{\mathrm{fix}}\ [a \vee_A b]_A}{\vdash_{\mathrm{bca}} [a]_A\ \mathsf{p}^{\mathrm{fix}}\ [a \vee_A b]_A}\ (\text{rec})$$

$$\frac{\vdash_{\mathrm{bca}} [a']_A\ \mathsf{p}\ [b]_A \quad a' \leq_A a \quad [\![\mathsf{p}]\!]^\sharp_A a \leq_A b}{\vdash_{\mathrm{bca}} [a]_A\ \mathsf{p}\ [b]_A}\ (\mathsf{weaken}_{\mathrm{pre}})$$

The rule (seq_γ) provides a sufficient condition for composing bca triples by requiring that the first program p_1 is γ-complete on $\gamma(a)$, i.e, $[\![\mathsf{p}_1]\!]\gamma(a) = \gamma(c)$ holds. In Section 7.1 we argue that this completeness condition *cannot be avoided* in any inductive logic that aims to infer that sequential compositions preserve bcas. In the rules (abs-inv) and (rec), the meaning of the command $\mathsf{p}^{\mathrm{fix}}$ is intended to be the strongest abstract invariant of p for a given abstract precondition $a \in A$, that is, $\mathrm{lfp}(\lambda x \in A.\ a \vee_A [\![\mathsf{p}]\!]^\sharp_A x)$. The rule (abs-inv) is not trivial since it states that an abstract invariant a which is bca, i.e. the triple $[a]_A\ \mathsf{p}\ [b]_A$ is valid for some $b \leq_A a$, is indeed the strongest abstract invariant of p and this is the best possible one in A. On the other hand, while (abs-inv) allows us to derive triples having the shape $[c]_A\ \mathsf{p}^{\mathrm{fix}}\ [c]_A$ only, the recursive rule (rec) can be used to strengthen the abstract precondition $c \in A$ to some $a \leq_A c$ such that the triple $[a]_A\ \mathsf{p}^n\ [c]_A$ is valid for some $n \geq 1$, where p^n denotes the n-th composition $\mathsf{p}; \cdots ; \mathsf{p}$. The intuition is that if the strongest abstract invariant c can be reached from the abstract precondition a after n iterations of p, by preserving the bca property throughout these iterations, then for $\mathsf{p}^{\mathrm{fix}}$ the precondition c can be strengthened to a. Finally, the bca logic includes the rule ($\mathsf{weaken}_{\mathrm{pre}}$) to weaken the abstract precondition a' of a triple $[a']_A\ \mathsf{p}\ [b]_A$ to some $a \geq_A a'$. This rule relies on proving that the A-analysis of p with abstract input a is approximated by b, namely $[\![\mathsf{p}]\!]^\sharp_A a \leq_A b$. On the other hand, as expected, no rule for modifying (weakening or strengthening) the abstract postcondition is available, since the validity of $[a]_A\ \mathsf{p}\ [b]_A$ entails an equality to b, so that the abstract postcondition b cannot be subject to weakening or strengthening.

1.3 Illustrative Example of the Program Logic

As an easy example that illustrates the above proof principles for reasoning on abstract postconditions which are bcas, we prove that the triple $[\mathbb{Z}_{>0}]_{\text{Sign}}\ (x > 0?\,;\ x := x - 1)^{\text{fix}}\ [\mathbb{Z}_{\geq 0}]_{\text{Sign}}$ is valid, namely $\mathbb{Z}_{\geq 0}$ is the strongest abstract invariant in Sign of the while program

$$\mathrm{p} \triangleq \textbf{while}\ x > 0\ \textbf{do}\ x := x - 1$$

for the abstract precondition $\mathbb{Z}_{>0}$, and $\mathbb{Z}_{\geq 0}$ is indeed the Sign abstraction of the strongest concrete invariant of p with precondition $\gamma_{\text{Sign}}(\mathbb{Z}_{>0})$ (which is $\mathbb{Z}_{\geq 0}$ itself). This can be proved in the logic $\vdash_{\text{bca}}$ as follows. Since for basic commands, we always consider their best correct approximations, we infer the triples $\vdash_{\text{bca}}\ [\mathbb{Z}_{>0}]_{\text{Sign}}\ x > 0?\ [\mathbb{Z}_{>0}]_{\text{Sign}}$ and $\vdash_{\text{bca}}\ [\mathbb{Z}_{>0}]_{\text{Sign}}\ x := x - 1\ [\mathbb{Z}_{\geq 0}]_{\text{Sign}}$. Then, we have that $[\![x > 0?]\!]\gamma_{\text{Sign}}(\mathbb{Z}_{>0}) = \gamma_{\text{Sign}}(\mathbb{Z}_{>0})$ holds, i.e. the test $x > 0?$ is locally γ-complete on $\mathbb{Z}_{>0}$, so that the composition rule (seq_{γ}) can be applied to derive $\vdash_{\text{bca}}\ [\mathbb{Z}_{>0}]_{\text{Sign}}\ x > 0?\,;\ x := x - 1\ [\mathbb{Z}_{\geq 0}]_{\text{Sign}}$. However, since $\mathbb{Z}_{\geq 0} \not\leq_{\text{Sign}} \mathbb{Z}_{>0}$, we cannot apply the (abs-inv) rule: indeed, it is worth observing that $[\mathbb{Z}_{>0}]_{\text{Sign}}\ (x > 0?\,;\ x := x - 1)^{\text{fix}}\ [\mathbb{Z}_{>0}]_{\text{Sign}}$ would not be a valid bca triple. We therefore apply the weakening rule $(\text{weaken}_{\text{pre}})$ because its premise $[\![x > 0?\,;\ x := x - 1]\!]^{\sharp}_{\text{Sign}}\mathbb{Z}_{\geq 0} = \mathbb{Z}_{\geq 0} \leq_{\text{Sign}} \mathbb{Z}_{\geq 0}$ holds, to infer $\vdash_{\text{bca}}\ [\mathbb{Z}_{\geq 0}]_{\text{Sign}}\ x > 0?\,;\ x := x - 1\ [\mathbb{Z}_{\geq 0}]_{\text{Sign}}$. The rule (abs-inv) can now be applied to derive $\vdash_{\text{bca}}\ [\mathbb{Z}_{\geq 0}]_{\text{Sign}}\ (x > 0?\,;\ x := x-1)^{\text{fix}}\ [\mathbb{Z}_{\geq 0}]_{\text{Sign}}$. As the final proof step, we apply (rec) with $n = 1$, because we have already proved $\vdash_{\text{bca}}\ [\mathbb{Z}_{>0}]_{\text{Sign}}\ x > 0?\,;\ x := x - 1\ [\mathbb{Z}_{\geq 0}]_{\text{Sign}}$ and $\mathbb{Z}_{\geq 0} = \mathbb{Z}_{>0} \vee_{\text{Sign}} \mathbb{Z}_{\geq 0}$, thus the premises of (rec) hold and $\vdash_{\text{bca}}\ [\mathbb{Z}_{>0}]_{\text{Sign}}\ (x > 0?\,;\ x := x - 1)^{\text{fix}}\ [\mathbb{Z}_{\geq 0}]_{\text{Sign}}$ can be inferred.

2 Background

2.1 Functions, Orders, and Computability

Given two sets S and T, $\wp(S)$ denotes the powerset of S, $S \smallsetminus T$ denotes the set-difference between S and T, $\overline{S}$ denotes the complement of S with respect to some universe set to be determined by the context, $S \subsetneq T$ denotes strict inclusion, $|S|$ denotes the cardinality of S, ω denotes the first limit ordinal (thus $|\mathbb{N}| = |\omega|$), $f : S \longrightarrow T$ denotes a totally defined function, $f : S \pfun T$ a partially defined function. If $f : S \pfun T$ then $f(x)\!\downarrow$ denotes that f is defined on x, while $f(x)\!\uparrow$ denotes that f is not defined on x; the domain and range of f are, resp., $\text{dom}(f) \triangleq \{x \in S \mid f(x)\!\downarrow\}$ and $\text{rng}(f) \triangleq \{f(x) \in T \mid x \in S \cap \text{dom}(f)\}$. Given a subset $X \subseteq S$, $f(X) \triangleq \{f(x) \in T \mid x \in X \cap \text{dom}(f)\}$ denotes the image of f on X, where f is defined.

A poset P w.r.t. a partial order relation $\leq$ is denoted by $\langle P, \leq\rangle$. If a poset $\langle L, \leq\rangle$ is complete lattice then $\vee_L$ denotes its lub, $\wedge_L$ its glb, $\top_L$ its top, $\bot_L$ its bottom. If $f, g : S \longrightarrow L$ and $\langle L, \leq\rangle$ is a poset then the pointwise partial order relation $f \dot{\leq} g$ is defined as usual: for all $x \in S$, $f(x) \leq g(x)$. A function $f : L_1 \longrightarrow L_2$ between complete lattices is additive (co-additive) if for all $Y \subseteq L$, $f(\vee_{L_1} Y) = \vee_{L_2} f(Y)$ ($f(\wedge_{L_1} Y) = \wedge_{L_2} f(Y)$). Recall that any monotone function $f : L \longrightarrow L$ on a complete lattice L always has least and greatest fixpoints, denoted, resp., by $\text{lfp}(f)$ and $\text{gfp}(f)$.

Let us recall some basic notions in computability theory (reference textbooks include [Rogers 1987; Soare 1980]). In the following, the terms *computable* (in some given model of computation, e.g., Turing Machines), *effective* and *recursive* are synonym. Recursive functions are total computable functions while partial recursive functions are partially defined computable functions. We assume a surjective acceptable enumeration, also called coding, of partial recursive functions: For any $e \in \mathbb{N}$, φ_e denotes the partial recursive function of index (code) e. As usual, we assume that partial recursive functions are defined over some infinite denumerable domain $\mathbb{D}$ which includes the natural numbers, i.e. $\mathbb{N} \subseteq \mathbb{D}$, so that, for any $e \in \mathbb{N}$, $\varphi_e : \mathbb{D} \pfun \mathbb{D}$. Partial recursive functions φ_e are associated with Turing Machines, or, equivalently, programs in some Turing complete programming language $\mathscr{L}$. The semantics of $\mathscr{L}$ is therefore a recursive function $[\![\cdot]\!] : \mathscr{L} \longrightarrow (\mathbb{D} \pfun \mathbb{D})$ such that for any $\mathrm{p} \in \mathscr{L}$, $[\![\mathrm{p}]\!] : \mathbb{D} \pfun \mathbb{D}$ is a partial recursive function, whose domain is often denoted by

$W_{\mathsf{p}} \triangleq \mathrm{dom}(\varphi_{\mathsf{p}})$. W.l.o.g., we identify $\mathscr{L}$ with its enumeration, so that we assume that $[\![\mathsf{p}]\!] = \varphi_{\mathsf{p}}$. A set $S \subseteq \mathbb{D}$ is recursively enumerable (r.e.) if there exists $\mathsf{p} \in \mathscr{L}$ such that $S = \mathrm{dom}(\varphi_{\mathsf{p}})$, or, equivalently, $S = \mathrm{rng}(\varphi_{\mathsf{p}}) = [\![\mathsf{p}]\!](\mathbb{D})$. It is known that S is r.e. iff S is the range of a total computable function or empty, namely, iff S is empty or $S = [\![\mathsf{p}]\!](\mathbb{D}) = \mathrm{rng}(\varphi_{\mathsf{p}})$ for some program $\mathsf{p} \in \mathscr{L}$ such that $[\![\mathsf{p}]\!]$ is a total recursive function. A set $S \subseteq \mathbb{D}$ is recursive iff S and $\overline{S}$ are r.e. The set of all r.e. subsets of $\mathbb{D}$ is denoted by $\wp^{\mathrm{re}}(\mathbb{D})$. It is known [Soare 1980, Union Theorem 1.9, Chapter II] that $\langle \wp^{\mathrm{re}}(\mathbb{D}), \subseteq \rangle$ is a distributive lattice (i.e., union and intersection distribute over each other) where $\varnothing$ and $\mathbb{D}$ are, resp., the bottom and top elements, and that for the set of recursive subsets $\wp^{\mathrm{rec}}(\mathbb{D}) \triangleq \{S \in \wp(\mathbb{D}) \mid S, \overline{S} \in \wp^{\mathrm{re}}(\mathbb{D})\}$, it turns out that $\langle \wp^{\mathrm{rec}}(\mathbb{D}), \subseteq \rangle$ is a Boolean algebra, namely, a distributive lattice with complementation [Soare 1980, 1.15 (ii), Chapter II].

2.2 Abstract Interpretation

When the semantics of a programming language is specified on a given concrete domain of properties C, abstract interpretation [Cousot 2021; Cousot and Cousot 1977] is the *de facto* standard method to specify abstract semantics, that is, semantics defined on an abstract domain A of approximate program properties. Concrete and abstract domains are typically complete lattices to guarantee the existence of join and meet operations used in the definition of concrete and abstract semantics. Since several abstractions are possible, we use subscripts such as $\leq_A$ and $\vee_A$ to disambiguate the underlying carrier set A and omit the subscripts in the case of C. Given complete lattices C and A, a pair of functions $\alpha : C \longrightarrow A$ and $\gamma : A \longrightarrow C$ forms a Galois connection (GC) when for all $c \in C$, $a \in A$, $\alpha(c) \leq_A a \Leftrightarrow c \leq \gamma(a)$ holds. The lattices C and A are called, resp., concrete and abstract domain, and α and γ are called, resp., abstraction and concretization maps. W.l.o.g. [Cousot 2021, Section 11.6], we only consider GCs such that $\alpha\gamma = id_A$, called Galois insertions (GIs), where α is surjective, or, equivalently, γ is injective. Let us recall that α is additive, γ is co-additive, $\gamma\alpha$ is an (upper) closure operator, that is, $\gamma\alpha : C \longrightarrow C$ is a monotone, idempotent and extensive (i.e., $id_C \mathrel{\dot{\leq}_C} \gamma\alpha$ holds) function, and $\gamma(A) \subseteq C$ is closed under arbitrary glbs. The class of abstract domains of C is $\mathrm{Abs}(C) \triangleq \{\langle A, \leq_A, \alpha, \gamma\rangle \mid \alpha : C \longrightarrow A,\ \gamma : A \longrightarrow C \text{ is a GI}\}$, and we write $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$, or simply $A \in \mathrm{Abs}(C)$, to mean that A is an abstract domain related to C by the abstraction and concretization maps α and γ. The domain $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$ is disjunctive when γ is additive (e.g., Sign in Figure 1 is disjunctive). When notationally convenient, we simply use A in place of the closure operator $\gamma\alpha : C \longrightarrow C$, e.g., $\mathsf{Int}(\{-3, 0, 2\}) = [-3, 2]$ for the interval domain Int. Given two abstract domains $A_{\alpha_A,\gamma_A}, B_{\alpha_B,\gamma_B} \in \mathrm{Abs}(C)$, B is a *refinement* of A and A a *simplification* (or *abstraction*) of B, denoted by $B \sqsubseteq A$, when $\gamma_A(A) \subseteq \gamma_B(B)$ holds, i.e., when B is at least as expressive as A. An abstract domain $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$ is *trivial* if: either (a) $\gamma\alpha = \lambda x.x$ holds, i.e. A boils down to an isomorphic representation of the concrete domain C, and in this case we use the notation id_{Abs}; or (b) $\gamma\alpha = \lambda x.\top_C$ holds, i.e., A is a singleton domain which can express precisely the greatest element $\top_C$ only, and here we denote it by $\top_{\mathrm{Abs}}$. When C is a complete lattice, $\langle \mathrm{Abs}(C), \sqsubseteq, \sqcup, \sqcap, \top_{\mathrm{Abs}}, id_{\mathrm{Abs}}\rangle$ denotes the so-called *lattice of abstract interpretations* [Cousot and Cousot 1979, Section 8], meaning that there exist the most concrete simplification (i.e., lub $\sqcup$) and the most abstract refinement (i.e., glb $\sqcap$) of any family of abstract domains.

Abstract interpretation is intended to approximate properties of program semantics. Hence, we will often assume that the concrete domain can express the properties of any program with finitely many variables assuming values on a semantic domain $\mathbb{D}$. Moreover, we assume that programs range in a given *Turing complete* programming language $\mathscr{L}$. Both these assumptions provide the computability requirements for the concrete and abstract domains. For the concrete domain, this corresponds to fix $C = \wp^{\mathrm{re}}(\mathbb{D})$, i.e., for each $S \in \wp^{\mathrm{re}}(\mathbb{D})$ there exists a program $\mathsf{p} \in \mathscr{L}$ such that $[\![\mathsf{p}]\!]\mathbb{D} = S$. For the abstract domain, let us observe that static program analysis by abstract

interpretation is intended to associate with the program under inspection a recursive, namely decidable, approximation of its generally undecidable semantics. This is because static program analysis requires decidable answers to undecidable questions such as those expressed as extensional properties of programs, e.g., whether $[\![\mathrm{p}]\!]pre \subseteq post$ holds, for some $pre, post \subseteq \mathbb{D}$. The notion of decidable or recursive abstract interpretation has been studied by Cousot et al. [2018] for the sake of comparing the hardness of analyzing and verifying programs, where an abstract domain is defined as a collection of recursive sets such that the relation of subset inclusion between them is decidable. An abstract domain $A_{\alpha,\gamma}$ is here called *recursive* when the abstraction of any r.e. set is recursive, namely, for any $S \in \wp^{\mathrm{re}}(\mathbb{D})$, $\gamma\alpha(S) \in \wp^{\mathrm{rec}}(\mathbb{D})$ holds. Clearly, the trivial top abstraction closure $\lambda x.\mathbb{D}$ corresponds to a recursive abstract domain $\{\top_A\}$, while the trivial identity abstraction $\lambda x.x$ corresponds to a nonrecursive abstract domain, which is the concrete domain $\wp^{\mathrm{re}}(\mathbb{D})$ itself.

Given an abstract domain $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$ and a monotone concrete operation $f : C \to C$ (a generalization to n-ary functions of type $C^n \to C$ can be easily done componentwise), a monotone abstract function $f^\sharp : A \to A$ is a correct (or sound) approximation of f when $\alpha f \mathrel{\dot\leq} f^\sharp\alpha$ holds. It is known that if $f^\sharp$ is a correct approximation of f then we also have fixpoint correctness, i.e., $\alpha(\mathrm{lfp}(f)) \leq_A \mathrm{lfp}(f^\sharp)$ holds. The *best correct approximation* (bca) of f in A is defined as the abstract function $f^A \triangleq \alpha f\gamma : A \to A$. The term "best" is justified by the well-known fact [Cousot and Cousot 1979, Section 7.2] that an abstract function $f^\sharp : A \to A$ is a correct approximation of f iff $f^A \mathrel{\dot\leq} f^\sharp$ holds, so that the bca f^A turns out to be the most precise, w.r.t. the pointwise ordering $\mathrel{\dot\leq}$, among the correct approximations of f on A.

The abstract function $f^\sharp$ is an α-complete approximation of f (or just α-complete; Giacobazzi et al. [2000] use the term backward complete) if $\alpha f = f^\sharp\alpha$ holds. The abstract domain A is called an α-complete abstraction for f when there exists an α-complete approximation $f^\sharp : A \to A$ of f on the abstract domain A, and if this is the case then necessarily $f^\sharp = f^A$ holds, i.e., $f^\sharp$ is the bca. This form of completeness intuitively means that the abstract behaviour of $f^\sharp$ on A, i.e. the function $f^\sharp\alpha : C \to A$, exactly matches the abstraction in A of the behaviour of f, i.e. the function αf. In an α-complete approximation $f^\sharp$, the only loss of precision is due to the abstract domain and not to the definition of the abstract function $f^\sharp$ itself. Let us also recall that *local* α-completeness means that the α-completeness equation holds on some subset S of the concrete domain, that is, if $S \subseteq C$ then $f^\sharp$ is a locally α-complete on S when for all $c \in S$, $\alpha f(c) = f^\sharp\alpha(c)$ holds [Bruni et al. 2021, 2023]. A second orthogonal form of completeness has been studied. The abstract function $f^\sharp$ is a γ-complete approximation of f (or just γ-complete) if $f\gamma = \gamma f^\sharp$ holds. Also, A is called a γ-complete abstraction for f if there exists a γ-complete approximation $f^\sharp : A \to A$ of f. Similarly to α-completeness, A is a γ-complete abstraction for f when there exists a γ-complete approximation $f^\sharp : A \to A$ of f, and, in this case, $f^\sharp$ is indeed the bca, i.e. $f^\sharp = f^A$ necessarily holds. The intuition of γ-completeness (sometimes called exactness [Miné 2017] or forward completeness [Giacobazzi and Quintarelli 2001]) is that $f^\sharp$ defined on A behaves exactly as f, up to the meaning of abstract values provided by the concretization function γ. Local γ-completeness is defined as expected: given a subset of abstract values $S \subseteq A$, $f^\sharp$ is a locally γ-complete on S when for all $a \in S$, $f\gamma(a) = \gamma f^\sharp(a)$.

2.3 Regular Commands

Following O'Hearn [2020]'s programming model for incorrectness logic, in the following the programming language $\mathscr{L}$ is assumed to be the language Reg of *regular commands*:

$$\mathsf{Reg} \ni \mathsf{r} ::= \mathsf{c} \mid \mathsf{r};\mathsf{r} \mid \mathsf{r} \oplus \mathsf{r} \mid \mathsf{r}^{\mathsf{fix}} \mid \mathsf{r}^\star$$

which is general enough to cover deterministic imperative languages as well as nondeterministic and probabilistic programming. The language Reg is parametric on the syntax of basic commands

$\mathsf{c} \in \mathsf{BCmd}$, which can be instantiated with different kinds of instructions such as (deterministic or nondeterministic or parallel) assignments, (Boolean) guards or assumptions, local variable primitives, etc. The term $\mathsf{r}_1;\mathsf{r}_2$ represents sequential composition, $\mathsf{r}_1 \oplus \mathsf{r}_2$ represents a nondeterministic choice command, $\mathsf{r}^\star$ is the Kleene star that executes r any number (possibly zero) of times before exiting. As a shorthand, we write r^n for the composition $\mathsf{r};\ldots;\mathsf{r}$ of n instances of r. Our definition of Reg also includes $\mathsf{r}^{\mathsf{fix}}$, that represents, given a precondition c, the least inductive invariant of r implied by c, that is, the least fixpoint of $\lambda x.c \vee [\![\mathsf{r}]\!]x$. The motivation for including the command $\mathsf{r}^{\mathsf{fix}}$ comes from the fact that while its concrete predicate transformer semantics commonly (i.e., when basic commands have additive semantics) coincides with that of $\mathsf{r}^\star$, the inductive abstract semantics of $\mathsf{r}^{\mathsf{fix}}$ and $\mathsf{r}^\star$ on a given abstract domain may differ, as shown below in Remark 2.2.

We assume that basic commands have a semantics $(\!|\cdot|\!) : \mathsf{BCmd} \to C \to C$ on a complete lattice C such that $(\!|\mathsf{c}|\!)$ is an additive function. This assumption can be done w.l.o.g. in collecting program semantics—a.k.a. predicate transformer semantics—since their basic command semantics on the concrete domain $\langle \wp(\mathbb{D}), \subseteq \rangle$ are always defined as additive lifting of an underlying function f on stores in $\mathbb{D}$, i.e., $\lambda S \in \wp(\mathbb{D}).\{f(\sigma) \in \mathbb{D} \mid \sigma \in S\}$. The concrete semantics $[\![\cdot]\!] : \mathsf{Reg} \to C \to C$ of regular commands is inductively defined as follows:

$$\begin{array}{lll} [\![\mathsf{c}]\!]c \triangleq (\!|\mathsf{c}|\!)c & [\![\mathsf{r}_1;\mathsf{r}_2]\!]c \triangleq [\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]c & [\![\mathsf{r}_1 \oplus \mathsf{r}_2]\!]c \triangleq [\![\mathsf{r}_1]\!]c \vee [\![\mathsf{r}_2]\!]c \\ [\![\mathsf{r}^{\mathsf{fix}}]\!]c \triangleq \mathrm{lfp}(\lambda x \in C.\, c \vee [\![\mathsf{r}]\!]x) & [\![\mathsf{r}^\star]\!]c \triangleq \bigvee\{[\![\mathsf{r}]\!]^k c \mid k \in \mathbb{N}\} & \end{array} \tag{1}$$

2.3.1 Abstract Semantics. The *best inductive abstract semantics* $[\![\cdot]\!]^\sharp_A : \mathsf{Reg} \to A \to A$ of regular commands on an abstract domain $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$ is defined as follows:

$$\begin{array}{lll} [\![\mathsf{c}]\!]^\sharp_A a \triangleq [\![\mathsf{c}]\!]^A a = \alpha[\![\mathsf{c}]\!]\gamma(a) & [\![\mathsf{r}_1;\mathsf{r}_2]\!]^\sharp_A a \triangleq [\![\mathsf{r}_2]\!]^\sharp_A [\![\mathsf{r}_1]\!]^\sharp_A a & [\![\mathsf{r}_1 \oplus \mathsf{r}_2]\!]^\sharp_A a \triangleq [\![\mathsf{r}_1]\!]^\sharp_A a \vee_A [\![\mathsf{r}_2]\!]^\sharp_A a \\ [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A a \triangleq \mathrm{lfp}(\lambda x \in A.\, a \vee_A [\![\mathsf{r}]\!]^\sharp_A x) & [\![\mathsf{r}^\star]\!]^\sharp_A a \triangleq \bigvee_A\{[\![\mathsf{r}]\!]^{\sharp k}_A a \mid k \in \mathbb{N}\} & \end{array} \tag{2}$$

Since we are interested in reasoning on best possible abstract semantics, note that as abstract semantics $[\![\mathsf{c}]\!]^\sharp_A$ of basic commands $\mathsf{c} \in \mathsf{BCmd}$ we always consider the bcas on A of their concrete semantics, i.e., no loss of precision is due to their abstract interpretation. Let us remark that definition (2) is the standard *inductive* abstract interpretation of programs *without widening*, where the abstract transfer functions of basic commands are assumed to be the bcas. It is easy to check, by structural induction, that the abstract semantics in (2) is monotonic—provided that $[\![\mathsf{c}]\!]$ for basic commands are monotone functions—and correct (or sound), i.e., $\alpha[\![\mathsf{r}]\!] \dot{\leq}_A [\![\mathsf{r}]\!]^\sharp_A \alpha$ holds.

Remark 2.1 (On the Lack of Widening). It is worth noting that the use of widening operators could jeopardize the bca property. In fact, there exists no notion of "best" widening [Cousot 2021, Chapter 32], and considering any sound widening is not viable. As a limit case, the following well-known definition (e.g. [Miné 2017, Example 2.16]):

$$a \nabla_{\mathrm{n}} b \triangleq \begin{cases} a & \text{if } b \leq_A a \\ \top_A & \text{otherwise} \end{cases}$$

provides a naïve legal widening $a \nabla_{\mathrm{n}} b$ that may lose all the computed information a. Thus, considering widening operators in the definition (2) of the abstract semantics $[\![\cdot]\!]^\sharp_A$ cannot be meaningful in the context of studying its bca properties. □

Remark 2.2 ($\mathsf{r}^{\mathsf{fix}}$ vs $\mathsf{r}^\star$). By Lemma 2.3 (ii), it turns out that $[\![\mathsf{r}^{\mathsf{fix}}]\!] = [\![\mathsf{r}^\star]\!]$ holds, because the concrete semantics of basic commands $[\![\mathsf{c}]\!]$ is assumed to be additive. However, in general, this is not the case for the abstract semantics, where $[\![\mathsf{r}^\star]\!]^\sharp_A a \leq_A [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A a$ always holds but it may happen that $[\![\mathsf{r}^\star]\!]^\sharp_A a \leq_A [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A a$. To show this, consider the following program:

$$\mathsf{r} \triangleq (x = 2?; x := x + 3) \oplus x := x - 3\,.$$

The abstract semantics of $\mathsf{r}^{\mathsf{fix}}$ and $\mathsf{r}^\star$ on intervals with input $[3,4] \in \mathsf{Int}$ are as follows:

$$\begin{aligned}
[\![\mathsf{r}^\star]\!]^\sharp_{\mathsf{Int}}[3,4] &= \bigvee\nolimits_{\mathsf{Int}} \big\{[3,4],\ [0,1] = [\![\mathsf{r}]\!]^\sharp_{\mathsf{Int}}[3,4],\ [-3,-2] = [\![\mathsf{r}]\!]^\sharp_{\mathsf{Int}}[0,1],\\
&\qquad [-6,-5] = [\![\mathsf{r}]\!]^\sharp_{\mathsf{Int}}[-3,-2],\ [-9,-8] = [\![\mathsf{r}]\!]^\sharp_{\mathsf{Int}}[-6,-5],\ \ldots\big\},\\
[\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_{\mathsf{Int}}[3,4] &= \bigvee\nolimits_{\mathsf{Int}} \big\{\bot_{\mathsf{Int}},\ [3,4] = [3,4] \vee_{\mathsf{Int}} [\![\mathsf{r}]\!]^\sharp_{\mathsf{Int}}\bot_{\mathsf{Int}},\ [0,4] = [3,4] \vee_{\mathsf{Int}} [\![\mathsf{r}]\!]^\sharp_{\mathsf{Int}}[3,4],\\
&\qquad [-3,5] = [3,4] \vee_{\mathsf{Int}} [\![\mathsf{r}]\!]^\sharp_{\mathsf{Int}}[0,4],\ [-6,5] = [3,4] \vee_{\mathsf{Int}} [\![\mathsf{r}]\!]^\sharp_{\mathsf{Int}}[-3,5], \ldots\big\}.
\end{aligned}$$

Hence, it turns out that $[\![\mathsf{r}^\star]\!]^\sharp_{\mathsf{Int}}[3,4] = [-\infty,4] \leq_{\mathsf{Int}} [-\infty,5] = [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_{\mathsf{Int}}[3,4]$. This different behavior depends on the fact that the function $[\![\mathsf{r}^\star]\!]^\sharp_{\mathsf{Int}}$ is not additive, for example:

$$[\![\mathsf{r}^\star]\!]^\sharp_{\mathsf{Int}}[1,1] \vee_{\mathsf{Int}} [\![\mathsf{r}^\star]\!]^\sharp_{\mathsf{Int}}[3,3] = [-2,0] \leq_{\mathsf{Int}} [-2,5] = [\![\mathsf{r}^\star]\!]^\sharp_{\mathsf{Int}}([1,1] \vee_{\mathsf{Int}} [3,3]).$$

It is also worth noticing that $[\![\mathsf{r}^{\mathsf{fix}}]\!]\gamma_{\mathsf{Int}}([3,4]) = [\![\mathsf{r}^\star]\!]\gamma_{\mathsf{Int}}([3,4]) = \{z \in \mathbb{Z} \mid z \leq 4, \forall n \in \mathbb{N}.z \neq 2-3n\}$, so that $\alpha_{\mathsf{Int}}[\![\mathsf{r}^\star]\!]\gamma_{\mathsf{Int}}([3,4]) = [-\infty,4] = [\![\mathsf{r}^\star]\!]^\sharp_{\mathsf{Int}}[3,4]$, i.e., $[\![\mathsf{r}^\star]\!]^\sharp_{\mathsf{Int}}[3,4]$ is the bca, while the bca property does not hold for $\mathsf{r}^{\mathsf{fix}}$ since $\alpha_{\mathsf{Int}}[\![\mathsf{r}^{\mathsf{fix}}]\!]\gamma_{\mathsf{Int}}([3,4]) = [-\infty,4] \neq [-\infty,5] = [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_{\mathsf{Int}}[3,4]$. □

Remark 2.2 can be generalized as follows. An abstract domain $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$ is called *weakly disjunctive* when: $\forall S \subseteq A.\ \vee_A S \neq \top_A \implies \gamma(\vee_A S) = \vee\gamma(S)$. Therefore, in a weakly disjunctive abstract domain A we have that disjunction is either exactly represented in A or A is unable to provide any information on that disjunction. Note that the flat abstract domain for constant propagation [Wegman and Zadeck 1991] is weakly disjunctive, while more structured abstract domains, e.g. the interval domain Int, typically are not weakly disjunctive. In what follows, we characterize the abstract domains A for which $[\![\mathsf{r}^\star]\!]^\sharp_A = [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A$ holds.

Lemma 2.3 (Additivity of Concrete and Abstract Semantics). *Assume that for all basic commands* $\mathsf{c} \in \mathsf{BCmd}$, $[\![\mathsf{c}]\!]$ *is an additive function.*

(i) *For all* $\mathsf{r} \in \mathsf{Reg}$, *the concrete semantics* $[\![\mathsf{r}]\!]$ *is an additive function.*
(ii) *If, additionally,* $A \in \mathrm{Abs}(C)$ *is a disjunctive abstract domain then the abstract semantics* $[\![\mathsf{r}]\!]^\sharp_A$ *is an additive function, and* $[\![\mathsf{r}^\star]\!]^\sharp_A = [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A$ *holds.*

Proof. (i) By structural induction on $\mathsf{r} \in \mathsf{Reg}$. Clear for $\mathsf{r}_1;\mathsf{r}_2$ and $\mathsf{r}_1 \oplus \mathsf{r}_2$. Let us prove it for $\mathsf{r}^{\mathsf{fix}}$. Given $Y \subseteq C$, we have that:

$$\begin{aligned}
\vee\{\mathrm{lfp}(\lambda x \in C.\, y \vee [\![\mathsf{r}]\!]x) \mid y \in Y\} &= && [\text{as } f(\mathrm{lfp}(f)) = \mathrm{lfp}(f)]\\
\vee\{y \vee [\![\mathsf{r}]\!](\mathrm{lfp}(\lambda x \in C.\, y \vee [\![\mathsf{r}]\!]x)) \mid y \in Y\} &= && [\text{as } \vee\{s \vee t \mid s \in S, t \in T\} = (\vee S) \vee (\vee T)]\\
(\vee Y) \vee (\vee\{[\![\mathsf{r}]\!](\mathrm{lfp}(\lambda x \in C.\, y \vee [\![\mathsf{r}]\!]x)) \mid y \in Y\}) &= && [\text{by inductive hypothesis}]\\
(\vee Y) \vee [\![\mathsf{r}]\!](\vee\{\mathrm{lfp}(\lambda x \in C.\, y \vee [\![\mathsf{r}]\!]x) \mid y \in Y\}).
\end{aligned}$$

Hence, $\mathrm{lfp}(\lambda x \in C.\, (\vee Y) \vee [\![\mathsf{r}]\!]x) \leq \vee\{\mathrm{lfp}(\lambda x \in C.\, y \vee [\![\mathsf{r}]\!]x) \mid y \in Y\}$ holds. Conversely, if $y \in Y$, from $\lambda x \in C.\, y \vee [\![\mathsf{r}]\!]x) \dot{\leq} \lambda x \in C.\, (\vee Y) \vee [\![\mathsf{r}]\!]x$ follows that $\vee\{\mathrm{lfp}(\lambda x \in C.\, y \vee [\![\mathsf{r}]\!]x) \mid y \in Y\} \leq \mathrm{lfp}(\lambda x \in C.\, (\vee Y) \vee [\![\mathsf{r}]\!]x)$. Hence, $[\![\mathsf{r}^{\mathsf{fix}}]\!](\vee Y) = \vee\{[\![\mathsf{r}^{\mathsf{fix}}]\!]y \mid y \in Y\}$. Since $[\![\mathsf{r}^{\mathsf{fix}}]\!] = [\![\mathsf{r}^\star]\!]$, additivity follows for $\mathsf{r}^\star$ as well.
(ii) By structural induction on $\mathsf{r} \in \mathsf{Reg}$. For basic commands $\mathsf{c} \in \mathsf{BCmd}$, $[\![\mathsf{c}]\!]^\sharp_A = \alpha[\![\mathsf{c}]\!]\gamma$ is additive being a composition of additive functions. Clear for $\mathsf{r}_1;\mathsf{r}_2$ and $\mathsf{r}_1 \oplus \mathsf{r}_2$. The proof for $\mathsf{r}^{\mathsf{fix}}$ is the same as in (i). For abstract semantics, $[\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A = [\![\mathsf{r}^\star]\!]^\sharp_A$ does not hold in general, therefore additivity of $[\![\mathsf{r}^\star]\!]$ is proved as follows for all $Y \subseteq A$:

$$[\![\mathsf{r}^\star]\!]^\sharp_A(\vee_A Y) = \bigvee\nolimits_A\{[\![\mathsf{r}]\!]^{\sharp\,k}_A(\vee_A Y) \mid k \in \mathbb{N}\} =$$
$$\bigvee\nolimits_A\{\vee_A\{[\![\mathsf{r}]\!]^{\sharp\,k}_A y \mid y \in Y\} \mid k \in \mathbb{N}\} = \bigvee\nolimits_A\{\vee_A\{[\![\mathsf{r}]\!]^{\sharp\,k}_A y \mid k \in \mathbb{N}\} \mid y \in Y\} = \bigvee\nolimits_A\{[\![\mathsf{r}^\star]\!]^\sharp_A y \mid y \in Y\}.$$ □

Theorem 2.4 (Equivalence of Abstract Semantics for $\mathsf{r}^{\mathsf{fix}}$ and $\mathsf{r}^\star$). *Let* $[\![\mathsf{c}]\!]$ *be additive for all* $\mathsf{c} \in \mathsf{BCmd}$ *and* A *be nontrivial. For any* $\mathsf{r} \in \mathsf{Reg}$, $[\![\mathsf{r}^\star]\!]^\sharp_A = [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A$ *iff* A *is weakly disjunctive.*

Proof. ($\Rightarrow$) Assume A nontrivial and not weakly disjunctive. Thus, there exist two elements $a_1, a_2 \in A$ such that $\gamma(a_1) \vee \gamma(a_2) \subsetneq \gamma(a_1 \vee_A a_2) \subsetneq \mathbb{D}$. Let $d \in \gamma(a_2)$, $c \in \gamma(a_1 \vee_A a_2) \setminus \gamma(a_1) \vee \gamma(a_2)$, and $b \in \mathbb{D} \setminus \gamma(a_1 \vee_A a_2)$. Following Remark 2.2 it is enough to consider the following program: $\mathsf{r} \triangleq (x = c?;\ x := b) \oplus x := d$. It is immediate to see that $[\![\mathsf{r}^\star]\!]^\sharp_A a_1 \neq [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A a_1$.
($\Leftarrow$) Assume, by contradiction, that there exist $\mathsf{r} \in \mathsf{Reg}$ and $a \in A$ such that $[\![\mathsf{r}^\star]\!]^\sharp_A a \leq_A [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A a$. Consider the set $S \triangleq \{[\![\mathsf{r}]\!]^{\sharp\, k}_A a \mid k \in \mathbb{N}\} \subseteq A$. It is clear that $\vee_A S \neq \top_A$, otherwise $[\![\mathsf{r}^\star]\!]^\sharp_A a = [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A a = \top_A$. Assume that $\vee\gamma(S) = \gamma(\vee_A S)$. Because, by Lemma 2.3, $[\![\mathsf{c}]\!]$ is additive for all $\mathsf{c} \in \mathsf{BCmd}$ and A is weakly disjunctive, then, for all $\mathsf{r} \in \mathsf{Reg}$, $[\![\mathsf{r}]\!]^\sharp_A$ is additive on S. Therefore, $\bigvee_A S$ is a fixpoint of $\lambda x \in A.\, a \vee_A [\![\mathsf{r}]\!]^\sharp_A x$. Let $d \in A$ be a fixpoint of $\lambda x \in A.\, a \vee_A [\![\mathsf{r}]\!]^\sharp_A x$. By monotonicity of $[\![\mathsf{r}]\!]^\sharp_A$, it is immediate to prove, by induction on k, that for all $k \in \mathbb{N}$, $[\![\mathsf{r}]\!]^{\sharp\, k}_A a \leq_A d$, so that $[\![\mathsf{r}^\star]\!]^\sharp_A a = \bigvee_A S = [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A a$, which is a contradiction, thus proving that $[\![\mathsf{r}^\star]\!]^\sharp_A = [\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_A$. □

In the following, we will denote through $\mathsf{Reg}_{\mathsf{fix}}$ the programs in Reg without Kleene star, i.e., $\mathsf{Reg}_{\mathsf{fix}} \ni \mathsf{r} ::= \mathsf{c} \mid \mathsf{r}; \mathsf{r} \mid \mathsf{r} \oplus \mathsf{r} \mid \mathsf{r}^{\mathsf{fix}}$.

2.4 While Programs

We consider standard basic commands used in while programs, i.e., no-op, deterministic assignments and Boolean guards: $\mathsf{BCmd} \ni \mathsf{c} ::= \mathbf{skip} \mid x := \mathsf{a} \mid \mathsf{b}?$, where a ranges over deterministic arithmetic expressions on integer values in $\mathbb{Z}$, variables $x \in \mathit{Var}$, and b ranges over deterministic Boolean expressions. A program store $\sigma : V \to \mathbb{Z}$ is a total function from a finite set of variables of interest $V \subseteq \mathit{Var}$ to values, and $\mathbb{D} \triangleq V \to \mathbb{Z}$ denotes the set of stores on the variables ranging in a set V that, for simplicity, is left implicit. The concrete domain is $\langle \wp(\mathbb{D}), \subseteq \rangle$ and the semantics $(\!|\cdot|\!) : \mathsf{BCmd} \to \wp(\mathbb{D}) \to \wp(\mathbb{D})$ of basic expressions is the usual (and additive) one: $(\!|\mathbf{skip}|\!)X \triangleq X$, $(\!|x := \mathsf{a}|\!)X \triangleq \{\sigma[x \mapsto \{\!|\mathsf{a}|\!\}\sigma] \mid \sigma \in X, \{\!|\mathsf{a}|\!\}\sigma\downarrow\}$, $(\!|\mathsf{b}?|\!)X \triangleq \{\sigma \in X \mid \{\!|\mathsf{b}|\!\}\sigma = \mathbf{tt}\}$, where store update $\sigma[x \mapsto v]$ and the semantics of arithmetic expressions $\{\!|\mathsf{a}|\!\} : \mathbb{D} \nrightarrow \mathbb{Z}$ and Boolean expressions $\{\!|\mathsf{b}|\!\} : \mathbb{D} \nrightarrow \{\mathbf{tt}, \mathbf{ff}\}$ are defined as expected. In particular, we assume that for all $\mathsf{r} \in \mathsf{Reg}$ and $\sigma \in \mathbb{D}$, $[\![\mathsf{r}]\!]\{\sigma\} = \{\sigma'\}$ means that in the underlying *deterministic Turing complete* computational (e.g., operational) model for Reg programs, the program r on input σ terminates with final store σ', whereas $[\![\mathsf{r}]\!]\{\sigma\} = \varnothing$ means nontermination. Moreover, by additivity, we have that for all $X \in \wp(\mathbb{D})$, $[\![\mathsf{r}]\!]X = \{\sigma' \in \mathbb{D} \mid \exists \sigma \in X.\, [\![\mathsf{r}]\!]\{\sigma\} = \{\sigma'\}\}$, $[\![\mathsf{r}]\!]\varnothing = \varnothing$, and $[\![\mathsf{r}]\!]\mathbb{D} = S$ means that S is the range of the output stores of r. A deterministic imperative while language can be defined using guarded branching and loop commands as syntactic sugar as follows [Fischer and Ladner 1979; Kozen 1997]: **if** b **then** c_1 **else** $\mathsf{c}_2 \triangleq (\mathsf{b}?; \mathsf{c}_1) \oplus (\neg\mathsf{b}?; \mathsf{c}_2)$ and **while** b **do** $\mathsf{c} \triangleq (\mathsf{b}?; \mathsf{c})^{\mathsf{fix}}; \neg\mathsf{b}?$. For programs with just one variable, $\wp(\mathbb{Z})$ will be used to represent sets of stores.

3 The bca Property

For the sake of generality, we assume that the inductive concrete semantics (1) of programs is given on a generic complete lattice $\langle C, \leq \rangle$. As a notable example, considering a generic complete lattice rather than the collecting predicate transformer domain $\langle \wp(\mathbb{D}), \subseteq \rangle$, allows us to encompass the case of probabilistic programs whose concrete domain is given by (sub)distributions over a carrier set X, an application scenario investigated in Zilberstein et al. [2023]'s outcome logic.

Definition 3.1 (Local and Global bca). Consider $\mathsf{r} \in \mathsf{Reg}$ and $A_{\alpha,\gamma} \in \mathsf{Abs}(C)$.

(i) r satisfies the *local bca* (*lbca*) property on $a \in A$, denoted $\mathsf{bca}_A(\mathsf{r}, a)$, when $[\![\mathsf{r}]\!]^\sharp_A a = \alpha[\![\mathsf{r}]\!]\gamma(a)$.
(ii) r satisfies the *global bca* (*gbca*) property on A, denoted by $\mathsf{bca}_A(\mathsf{r})$, when $[\![\mathsf{r}]\!]^\sharp_A = \alpha[\![\mathsf{r}]\!]\gamma$. □

Lemma 3.2 (Choice Preserves bca). *For all* $a \in A$, *if* $\mathrm{bca}_A(\mathsf{r}_1, a)$ *and* $\mathrm{bca}_A(\mathsf{r}_2, a)$ *then* $\mathrm{bca}_A(\mathsf{r}_1 \oplus \mathsf{r}_1, a)$. *Also, if* $\mathrm{bca}_A(\mathsf{r}_1)$ *and* $\mathrm{bca}_A(\mathsf{r}_2)$ *then* $\mathrm{bca}_A(\mathsf{r}_1 \oplus \mathsf{r}_1)$.

Proof. Given any $a \in A$, we have that:

$$\begin{aligned}
[\![\mathsf{r}_1 \oplus \mathsf{r}_2]\!]^\sharp_A a = & \quad \text{[by definition (2)]}\\
[\![\mathsf{r}_1]\!]^\sharp_A a \vee_A [\![\mathsf{r}_2]\!]^\sharp_A a = & \quad \text{[by } \mathrm{bca}_A(\mathsf{r}_i, a)\text{]}\\
\alpha[\![\mathsf{r}_1]\!]\gamma(a) \vee_A \alpha[\![\mathsf{r}_2]\!]\gamma(a) = & \quad \text{[by additivity of } \alpha\text{]}\\
\alpha([\![\mathsf{r}_1]\!]\gamma(a) \vee [\![\mathsf{r}_2]\!]\gamma(a)) = \alpha[\![\mathsf{r}_1 \oplus \mathsf{r}_2]\!]\gamma(a). & \quad \text{[by definition (1)]}
\end{aligned}$$

This also proves the second implication. □

Thus, the definition of the abstract choice operation as lub of A in (2) does not introduce loss of information hampering the (local or global) bca property. This property of preserving bcas, in general, does not hold for sequential composition, that is, by assuming $\mathrm{bca}_A(\mathsf{r}_1)$ and $\mathrm{bca}_A(\mathsf{r}_2)$, $[\![\mathsf{r}_1; \mathsf{r}_2]\!]^\sharp_A$ is not guaranteed to be the bca.

Example 3.3 (Composition does not Preserve bca). Consider the following program

$$\mathsf{p} \triangleq (z := 2; z := z + 1); z := z - 1 \tag{3}$$

analyzed on the abstract domain Sign of Figure 1. It turns out that $[\![\mathsf{p}]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z} = \mathbb{Z}_{\geq 0}$, whereas $\alpha_{\mathsf{Sign}}[\![\mathsf{p}]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}) = \mathbb{Z}_{>0}$, so that $\mathrm{bca}_{\mathsf{Sign}}(\mathsf{p})$ does not hold. On the other hand, $\mathrm{bca}_{\mathsf{Sign}}(z := 2; z := z+1)$ holds, because for all $a \in \mathsf{Sign} \setminus \{\varnothing\}$, we have that $[\![z := 2; z := z + 1]\!]^\sharp_{\mathsf{Sign}} a = \mathbb{Z}_{>0} = \alpha_{\mathsf{Sign}}(\{3\}) = \alpha_{\mathsf{Sign}}[\![z := 2; z := z + 1]\!]\gamma_{\mathsf{Sign}}(a)$, and $[\![z := 2; z := z + 1]\!]^\sharp_{\mathsf{Sign}}\varnothing = \varnothing = \alpha_{\mathsf{Sign}}[\![z := 2; z := z + 1]\!]\gamma_{\mathsf{Sign}}(\varnothing)$ also holds. Moreover, we have that $\mathrm{bca}_{\mathsf{Sign}}(z := z - 1)$ holds by definition (2), as $z := z - 1$ is a basic command. This shows that composition preserves neither the global nor the local bca property. □

We provide some simple completeness conditions guaranteeing that the global bca property is preserved when sequentially composing programs.

Lemma 3.4 (Gbca for Sequential Composition). *Let* $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$, *and assume that* $\mathrm{bca}_A(\mathsf{r}_1)$ *and* $\mathrm{bca}_A(\mathsf{r}_2)$ *hold.*

(i) *If* A *is* γ*-complete for* r_1 *or* α*-complete for* r_2 *then* $\mathrm{bca}_A(\mathsf{r}_1; \mathsf{r}_2)$ *holds.*
(ii) $\mathrm{bca}_A(\mathsf{r}_1; \mathsf{r}_2)$ *holds iff* r_2 *is locally* α*-complete on* $\{[\![\mathsf{r}_1]\!]\gamma(a) \in C \mid a \in A\}$.

Proof.
(i) We have the following two cases:

$$\begin{aligned}
\alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma = & \quad [A \text{ is } \gamma\text{-complete for } \mathsf{r}_1] \\
\alpha[\![\mathsf{r}_2]\!]\gamma[\![\mathsf{r}_1]\!]^\sharp_A = & \quad [\mathsf{r}_2 \text{ global bca}] \\
[\![\mathsf{r}_2]\!]^\sharp_A[\![\mathsf{r}_1]\!]^\sharp_A &
\end{aligned}
\qquad
\begin{aligned}
\alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma = & \quad [A \text{ is } \alpha\text{-complete for } \mathsf{r}_2] \\
[\![\mathsf{r}_2]\!]^\sharp_A\alpha[\![\mathsf{r}_1]\!]\gamma = & \quad [\mathsf{r}_1 \text{ global bca}] \\
[\![\mathsf{r}_2]\!]^\sharp_A[\![\mathsf{r}_1]\!]^\sharp_A &
\end{aligned}$$

Hence, in both cases, by definitions (1) and (2), we have that $\alpha[\![\mathsf{r}_1; \mathsf{r}_2]\!]\gamma = [\![\mathsf{r}_2]\!]^\sharp_A[\![\mathsf{r}_1]\!]^\sharp_A = [\![\mathsf{r}_1; \mathsf{r}_2]\!]^\sharp_A$.
(ii)

$$\begin{aligned}
\alpha[\![\mathsf{r}_1; \mathsf{r}_2]\!]\gamma = [\![\mathsf{r}_1; \mathsf{r}_2]\!]^\sharp_A \Leftrightarrow & \quad \text{[by definitions (1) and (2)]}\\
\alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma = [\![\mathsf{r}_2]\!]^\sharp_A[\![\mathsf{r}_1]\!]^\sharp_A \Leftrightarrow & \quad [\mathsf{r}_1 \text{ and } \mathsf{r}_2 \text{ global bca}]\\
\alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma = \alpha[\![\mathsf{r}_2]\!]\gamma\alpha[\![\mathsf{r}_1]\!]\gamma \Leftrightarrow & \quad \text{[by definition of local } \alpha\text{-completeness]}\\
\alpha[\![\mathsf{r}_2]\!]([\![\mathsf{r}_1]\!](\gamma(A))) = [\![\mathsf{r}_2]\!]^A\alpha([\![\mathsf{r}_1]\!](\gamma(A))). &
\end{aligned}$$

□

This result will be exploited in Section 7 to define the rule handling sequential compositions in the program logic for inferring the bca property.

It also turns out that Kleene star and fixpoints, in general, do not preserve bcas, that is, by assuming $\mathrm{bca}_A(\mathsf{r})$, both $[\![\mathsf{r}^\star]\!]^\sharp_A$ and $[\![\mathsf{r}^{\mathrm{fix}}]\!]^\sharp_A$ are not guaranteed to be the bca.

Example 3.5 (Fix and Kleene Star do not Preserve bca). Consider as concrete domain the chain of integer numbers $C = \{0,1,2,3\}$, as abstract domain $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$ the subset $A_{\alpha,\gamma} = \{0,2,3\}$, which, being closed under glb's, is an abstraction of C with $\gamma = \lambda x.x$ and $\alpha = \{0 \mapsto 0, 1 \mapsto 2, 2 \mapsto 2, 3 \mapsto 3\}$. We consider a basic command $\mathsf{c} \in \mathrm{BCmd}$ having the following semantics $[\![\mathsf{c}]\!] : C \to C$, which is obviously an additive function: $[\![\mathsf{c}]\!] \triangleq \{0 \mapsto 1, 1 \mapsto 1, 2 \mapsto 3, 3 \mapsto 3\}$. The corresponding bca $[\![\mathsf{c}]\!]^\sharp_A = \alpha[\![\mathsf{c}]\!]\gamma : A \to A$ is therefore as follows: $[\![\mathsf{c}]\!]^\sharp_A = \{0 \mapsto 2, 2 \mapsto 3, 3 \mapsto 3\}$.
Hence, for the input abstract value $0 \in A$, we have the following equalities:

$$[\![\mathsf{c}^\star]\!]\gamma(0) = \bigvee_C \{[\![\mathsf{c}]\!]^n\gamma(0) \mid n \in \mathbb{N}\} = \bigvee\{0,1\} = 1 = \mathrm{lfp}(\lambda x \in C.\, \gamma(0) \vee_C [\![\mathsf{c}]\!]x) = [\![\mathsf{c}^{\mathrm{fix}}]\!]\gamma(0)\,,$$
$$[\![\mathsf{c}^\star]\!]^\sharp_A 0 = \bigvee_A \{[\![\mathsf{c}]\!]^{\sharp n}_A 0 \mid n \in \mathbb{N}\} = \bigvee\{0,2,3\} = 3 = \mathrm{lfp}(\lambda a \in A.\, 0 \vee_A [\![\mathsf{c}]\!]^\sharp_A a) = [\![\mathsf{c}^{\mathrm{fix}}]\!]^\sharp_A 0\,,$$
$$\alpha[\![\mathsf{c}^\star]\!]\gamma(0) = \alpha[\![\mathsf{c}^{\mathrm{fix}}]\!]\gamma(0) = \alpha(1) = 2\,.$$

Thus, it turns out that $\mathrm{bca}_A(\mathsf{c})$ holds—by definition (2) because c is a basic command—while both $\mathrm{bca}_A(\mathsf{c}^\star)$ and $\mathrm{bca}_A(\mathsf{c}^{\mathrm{fix}})$ do not hold. □

It is known [Cousot and Cousot 1979, Theorem 7.1.0.3] that α-completeness is preserved by fixpoints. Here, we show that α- or γ-completeness is a sufficient condition guaranteeing that the Kleene star preserves the bca property, while just α-completeness ensures that the fixpoint command preserves the bca.

Lemma 3.6 (Gbca for Kleene Star and Fix). *Let* $\mathsf{r} \in \mathrm{Reg}$ *and* $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$.

(i) *If A is either α-complete or γ-complete for* r *(consequently,* $\mathrm{bca}_A(\mathsf{r})$ *holds, cf. Section 2.2), then* $\mathrm{bca}_A(\mathsf{r}^\star)$ *holds.*

(ii) *If A is α-complete for* r *(consequently,* $\mathrm{bca}_A(\mathsf{r})$ *holds, cf. Section 2.2), then* $\mathrm{bca}_A(\mathsf{r}^{\mathrm{fix}})$ *holds.*

Proof. (i) We show that

$$\forall n \in \mathbb{N}.\ [\![\mathsf{r}]\!]^{\sharp n}_A a = \alpha[\![\mathsf{r}]\!]^n\gamma(a) \tag{4}$$

Assume that α-completeness holds. For all $a \in A$, we show by induction on $n \in \mathbb{N}$ that $[\![\mathsf{r}]\!]^{\sharp n}_A a = \alpha[\![\mathsf{r}]\!]^n\gamma(a)$. For $n = 0$, $[\![\mathsf{r}]\!]^{\sharp 0}_A a = a = \alpha\gamma a = \alpha[\![\mathsf{r}]\!]^0\gamma(a)$. For $n+1$, we have that:

$$\begin{aligned} [\![\mathsf{r}]\!]^{\sharp n+1}_A a = [\![\mathsf{r}]\!]^\sharp_A [\![\mathsf{r}]\!]^{\sharp n}_A a = \quad & \text{[by inductive hypothesis]} \\ [\![\mathsf{r}]\!]^\sharp_A \alpha[\![\mathsf{r}]\!]^n\gamma(a) = \quad & \text{[by } \alpha\text{-completeness]} \\ \alpha[\![\mathsf{r}]\!][\![\mathsf{r}]\!]^n\gamma(a) = \alpha[\![\mathsf{r}]\!]^{n+1}\gamma(a)\,. & \end{aligned}$$

Assume that γ-completeness holds. For all $a \in A$, we show by induction on $n \in \mathbb{N}$ that $\gamma[\![\mathsf{r}]\!]^{\sharp n}_A a = [\![\mathsf{r}]\!]^n\gamma(a)$. For $n = 0$, $\gamma[\![\mathsf{r}]\!]^{\sharp 0}_A a = \gamma(a) = [\![\mathsf{r}]\!]^0\gamma(a)$. For $n+1$, we have that:

$$\begin{aligned} \gamma[\![\mathsf{r}]\!]^{\sharp n+1}_A a = \gamma[\![\mathsf{r}]\!]^\sharp_A [\![\mathsf{r}]\!]^{\sharp n}_A a = \quad & \text{[by } \gamma\text{-completeness]} \\ [\![\mathsf{r}]\!]\gamma[\![\mathsf{r}]\!]^{\sharp n}_A a = \quad & \text{[by inductive hypothesis]} \\ [\![\mathsf{r}]\!][\![\mathsf{r}]\!]^n\gamma(a) = [\![\mathsf{r}]\!]^{n+1}\gamma(a)\,. & \end{aligned}$$

Hence, also under the assumption of γ-completeness, we obtain that for all $n \in \mathbb{N}$, $[\![\mathsf{r}]\!]^{\sharp n}_A a = \alpha\gamma[\![\mathsf{r}]\!]^{\sharp n}_A a = \alpha[\![\mathsf{r}]\!]^n\gamma(a)$.
Thus, for $\mathsf{r}^\star$ we have that:

$$\begin{aligned} [\![\mathsf{r}^\star]\!]^\sharp_A a = \bigvee\{[\![\mathsf{r}]\!]^{\sharp n}_A a \mid n \in \mathbb{N}\} = \quad & \text{[by (4)]} \\ \bigvee\{\alpha[\![\mathsf{r}]\!]^n\gamma(a) \mid n \in \mathbb{N}\} = \quad & \text{[by additivity of } \alpha\text{]} \\ \alpha(\bigvee\{[\![\mathsf{r}]\!]^n\gamma(a) \mid n \in \mathbb{N}\} = \alpha[\![\mathsf{r}^\star]\!]\gamma(a)\,. & \end{aligned}$$

(ii) First, we have that:

$$\begin{aligned} a \vee_A [\![\mathrm{r}]\!]^\sharp_A(\alpha(\mathrm{lfp}(\lambda x.\gamma(a) \vee_C [\![\mathrm{r}]\!]x))) = & \quad \text{[by } \alpha\text{-completeness]} \\ a \vee_A \alpha[\![\mathrm{r}]\!](\mathrm{lfp}(\lambda x.\gamma(a) \vee_C [\![\mathrm{r}]\!]x)) = & \quad \text{[by additivity of } \alpha \text{ and } \alpha\gamma = id\text{]} \\ \alpha(\gamma(a) \vee_C [\![\mathrm{r}]\!](\mathrm{lfp}(\lambda x.\gamma(a) \vee_C [\![\mathrm{r}]\!]x)) = & \quad \text{[by fixpoint]} \\ \alpha(\mathrm{lfp}(\lambda x.\gamma(a) \vee_C [\![\mathrm{r}]\!]x)) \,. & \end{aligned}$$

Hence, $\mathrm{lfp}(\lambda x \in A.a \vee_A [\![\mathrm{r}]\!]^\sharp_A x) \leq_A \alpha(\mathrm{lfp}(\lambda x.\gamma(a) \vee_C [\![\mathrm{r}]\!]x))$. Conversely, let $b \in A$ such that $a \vee_A [\![\mathrm{r}]\!]^\sharp_A b = b$. Then, $a \leq_A b$ and $[\![\mathrm{r}]\!]^\sharp_A b \leq_A b$, and, in turn, $\gamma(a) \leq_C \gamma(b)$ and $[\![\mathrm{r}]\!]^\sharp_A \alpha\gamma(b) \leq_A b$, so that, by α-completeness, $\alpha[\![\mathrm{r}]\!]\gamma(b) \leq_A b$, and, in turn, $[\![\mathrm{r}]\!]\gamma(b) \leq_C \gamma(b)$. Thus, $\gamma(a) \vee_C [\![\mathrm{r}]\!]\gamma(b) \leq_C \gamma(b)$ holds, therefore entailing that $\mathrm{lfp}(\lambda x.\gamma(a) \vee_C [\![\mathrm{r}]\!]x) \leq_C \gamma(b)$, so that, by Galois connection, $\alpha(\mathrm{lfp}(\lambda x.\gamma(a) \vee_C [\![\mathrm{r}]\!]x)) \leq_A b$. This implies that $\mathrm{lfp}(\lambda x \in A.a \vee_A [\![\mathrm{r}]\!]^\sharp_A x) = \alpha(\mathrm{lfp}(\lambda x.\gamma(a) \vee_C [\![\mathrm{r}]\!]x))$, that is, $[\![\mathrm{r}^{\mathrm{fix}}]\!]^\sharp_A a = \alpha[\![\mathrm{r}^{\mathrm{fix}}]\!]\gamma(a)$. □

4 The Program Class of Best Abstract Interpretations

We study the computability properties of the class of programs satisfying the bca property on a given abstract domain. Fixed $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$, we consider the following two classes of programs:

$$\text{given } a \in A,\ \mathrm{BCA}(A, a) \triangleq \{\mathrm{p} \in \mathrm{Reg} \mid \mathrm{bca}_A(\mathrm{p}, a)\}\,; \qquad \mathrm{BCA}(A) \triangleq \bigcap\nolimits_{a \in A} \mathrm{BCA}(A, a)\,.$$

Firstly, note that by assuming that basic commands include a no-op $\mathbf{skip} \in \mathrm{BCmd}$, it turns out that for all $a \in A$, $\alpha[\![\mathbf{skip}]\!]\gamma(a) = \alpha\gamma(a) = a = [\![\mathbf{skip}]\!]^\sharp_A a$, i.e., $\mathrm{bca}_A(\mathbf{skip})$ holds. As a consequence, for all $a \in A$ and $n \geq 1$ we have that $\mathbf{skip}^n \in \mathrm{BCA}(A, a)$, and, in turn, $\mathbf{skip}^n \in \mathrm{BCA}(A)$. It should be remarked that the assumption $\mathbf{skip} \in \mathrm{BCmd}$ is not restrictive, since a primitive statement having the identity function as input/output semantics is included in any reasonable model of computation. We also assume that Reg is a Turing complete language, and that the domain of values $\mathbb{D}$ includes, when necessary, an encoding of Reg, namely, w.l.o.g., that $\mathrm{Reg} \subseteq \mathbb{D}$.

4.1 BCA Is Not Straightforward and Cannot Be Achieved by Compilation

We prove that for any nontrivial abstract domain $A \in \mathrm{Abs}(\wp^{\mathrm{re}}(\mathbb{D}))$ and abstract value $a \in A$, there exists a program p^a_A such that $\mathrm{bca}_A(\mathrm{p}^a_A, a)$ does not hold, thus showing that the bca property is not straightforward. This result extends to the weaker bca property a similar theorem proved in [Giacobazzi et al. 2015, Theorem 4.5] for the case of the α-completeness property. First, we need a preliminary lemma.

Lemma 4.1. *Let $A \in \mathrm{Abs}(\wp^{\mathrm{re}}(\mathbb{D}))$, $S \in \wp^{\mathrm{re}}(\mathbb{D})$, and $a \in A \setminus \{\bot_A\}$. Then, there exists $\mathrm{p} \in \mathrm{Reg}$ such that $[\![\mathrm{p}]\!]\gamma(a) = S$.*

Proof. Assume that $A \in \mathrm{Abs}(\wp^{\mathrm{re}}(\mathbb{D}))$, $S \in \wp^{\mathrm{re}}(\mathbb{D})$, and $a \in A \setminus \{\bot_A\}$. Then, we have that $\varnothing \subseteq \gamma(\bot_A) \subsetneq \gamma(a)$, so that $\gamma(a) \neq \varnothing$. Hence, there exists $n \in \gamma(a)$, and we can assume, w.l.o.g., up to an encoding of $\mathbb{D}$ into $\mathbb{N}$, that $n \in \gamma(a) \cap \mathbb{N}$. Consider the program $\mathrm{q} \triangleq (x \geq n;\ x := x + 1)^{\mathrm{fix}}$. Since $[\![\mathrm{q}]\!]\gamma(a) = \mathrm{lfp}(\lambda X \in \wp(\mathbb{N}).\ \gamma(a) \cup [\![x := x + 1]\!][\![x \geq n]\!]X) = \{x \in \mathbb{N} \mid x \geq n\}$, we have that $|[\![\mathrm{q}]\!]\gamma(a)| = |\{x \in \mathbb{N} \mid x \geq n\}| = |\mathbb{D}| = |\omega|$. Thus, there exists a program $\mathrm{i} \in \mathrm{Reg}$ such that $[\![\mathrm{q};\ \mathrm{i}]\!]\gamma(a) = \mathbb{D}$ holds. Because $S \subseteq \mathbb{D}$ is r.e., there exists $\mathrm{s} \in \mathrm{Reg}$ such that $[\![\mathrm{s}]\!]\mathbb{D} = S$. Hence, we have that: $S = [\![\mathrm{s}]\!]\mathbb{D} = [\![\mathrm{s}]\!]([\![\mathrm{q};\ \mathrm{i}]\!]\gamma(a)) = [\![\mathrm{q};\ \mathrm{i};\ \mathrm{s}]\!]\gamma(a)$. It is therefore enough to set $\mathrm{p} \triangleq \mathrm{q};\ \mathrm{i};\ \mathrm{s}$. □

Theorem 4.2 (Local BCA is not Straightforward). *Let $A \in \mathrm{Abs}(\wp^{\mathrm{re}}(\mathbb{D}))$ and $a \in A \setminus \{\bot_A\}$. Then, $\mathrm{BCA}(A, a) = \mathrm{Reg}$ if and only if A is trivial.*

Proof. One implication is straightforward: if $A \in \{id_{\mathrm{Abs}}, \top_{\mathrm{Abs}}\}$ then A is clearly α-complete for all programs and input abstract values, therefore, in both cases, for any $\mathrm{p} \in \mathrm{Reg}$ and $a \in A$,

$\mathrm{bca}_A(\mathsf{p}, a)$ holds. Let $a \in A \setminus \{\bot_A\}$ be such that $\mathrm{BCA}(A, a) = \mathrm{Reg}$. Assume, by contradiction, that A is nontrivial. Then, there exist $S, H \in \wp^{\mathrm{re}}(\mathbb{D})$ such that $S \subsetneq \gamma\alpha(S)$ (as $A \neq id_{\mathrm{Abs}}$) and $\gamma\alpha(H) \subsetneq \mathbb{D}$ (as $A \neq \top_{\mathrm{Abs}}$). Let $b \in \mathbb{D} \setminus \gamma\alpha(H)$ and $c \in \gamma\alpha(S) \setminus S$[1]. Because $S \subseteq \mathbb{D}$ is r.e., $\gamma(a) \subseteq \mathbb{D}$ is r.e., and $a \neq \bot_A$, by Lemma 4.1, there exists $\mathsf{p} \in \mathrm{Reg}$ such that $[\![\mathsf{p}]\!]\gamma(a) = S$. Let $\mathsf{r} \triangleq \mathsf{p}; x = c?; x := b$. We have that:

$$\alpha[\![\mathsf{r}]\!]\gamma(a) = \alpha[\![\mathsf{p}; x = c?; x := b]\!]\gamma(a) = \alpha[\![x := b]\!][\![x = c?]\!][\![\mathsf{p}]\!]\gamma(a) = \quad [\text{as } [\![\mathsf{p}]\!]\gamma(a) = S]$$
$$\alpha[\![x := b]\!][\![x = c?]\!]S = \alpha[\![x := b]\!](\{c\} \cap S) = \alpha[\![x := b]\!]\varnothing = \alpha(\varnothing). \quad [\text{as } c \notin S]$$

Note that $\alpha(\varnothing) \neq \alpha(\{b\})$ holds, otherwise we would have $\{b\} \subseteq \gamma\alpha(\{b\}) = \gamma\alpha(\varnothing) \subseteq \gamma\alpha(H)$, thus contradicting the hypothesis that $b \notin \gamma\alpha(H)$. Moreover, $\alpha(S) = \alpha[\![\mathsf{p}]\!]\gamma(a) \leq_A [\![\mathsf{p}]\!]^\sharp_A a$. Hence:

$$[\![\mathsf{r}]\!]^\sharp_A a = [\![\mathsf{p}; x = c?; x := b]\!]^\sharp_A a = [\![x = c?; x := b]\!]^\sharp_A [\![\mathsf{p}]\!]^\sharp_A a \geq_A \quad [\text{as } [\![\mathsf{p}]\!]^\sharp_A a \geq_A \alpha(S)]$$
$$[\![x = c?; x := b]\!]^\sharp_A \alpha(S) \geq_A \quad [\text{by soundness of } [\![x = c?; x := b]\!]^\sharp_A]$$
$$\alpha[\![x = c?; x := b]\!]\gamma\alpha(S) = \alpha[\![x := b]\!][\![x = c?]\!]\gamma\alpha(S) =$$
$$\alpha[\![x := b]\!](\{c\} \cap \gamma\alpha(S)) = \alpha[\![x := b]\!]\{c\} = \alpha(\{b\}). \quad [\text{as } c \in \gamma\alpha(S)]$$

Since $\alpha(\varnothing) \leq_A \alpha(\{b\})$, we have that $\alpha[\![\mathsf{r}]\!]\gamma(a) = \alpha(\varnothing) \leq_A \alpha(\{b\}) \leq_A [\![\mathsf{r}]\!]^\sharp_A a$, thus proving that $\mathsf{r} \notin \mathrm{BCA}(A, a)$, which contradicts the assumption $\mathrm{BCA}(A, a) = \mathrm{Reg}$. Hence, A must be trivial. □

It is worth remarking that the proof of Theorem 4.2 relies upon the fact that given some $b, c \in \mathbb{D}$, the Boolean test $x = c? \in \mathrm{Reg}$, used for defining the statement $x = c?; x := b \in \mathrm{Reg}$, is a partial function, that is, if $[\![x = c?]\!]\{s\} = \varnothing$ then $x = c?$ does not terminate with input s. This might be counterintuitive, so let us stress that the statement $x = c?; x := b$ corresponds to the following while program: **if** $x = c$ **then** $x := b$ **else** (**while true do skip**) .

As an easy consequence of Theorem 4.2, we also obtain the following characterization of the universal class BCA (this latter result could be also derived as a consequence of Rice's theorem for program analysis as given in [Cousot et al. 2018, Theorem 5.3, Lemma 6.5]).

Corollary 4.3 (Global BCA is not Straightforward). $\mathrm{BCA}(A) = \mathrm{Reg}$ *iff* A *is trivial.*

Proof. By Theorem 4.2, by observing that $\mathrm{BCA}(A) = \mathrm{Reg}$ iff for all $a \in A$, $\mathrm{BCA}(A, a) = \mathrm{Reg}$. □

Next, we prove that for any nontrivial abstract domain A such that for all programs p and abstract values $a, b \in A$, the question $[\![\mathsf{p}]\!]^\sharp_A a =^? b$ is decidable, if we fix $a \in A \setminus \{\top_A\}$, there exists no computable function that compiles p into an equivalent program for the precondition $\gamma(a)$ and which satisfies the bca property on a.

Theorem 4.4 (Impossibility of BCA by Program Transform). *Let* $A \in \mathrm{Abs}(\wp^{\mathrm{re}}(\mathbb{D}))$ *be nontrivial and assume that* $[\![\cdot]\!]^\sharp_A$ *is total recursive. Let* $a \in A \setminus \{\top_A\}$*. Then, there exists no total recursive program transform* $\tau : \mathrm{Reg} \longrightarrow \mathrm{Reg}$ *such that for any* $\mathsf{r} \in \mathrm{Reg}$, $[\![\mathsf{r}]\!]\gamma(a) = [\![\tau(\mathsf{r})]\!]\gamma(a)$ *and* $\tau(\mathsf{r}) \in \mathrm{BCA}(A, a)$.

Proof. Assume by contradiction that $\tau : \mathrm{Reg} \longrightarrow \mathrm{Reg}$ is a total recursive program transform such that for any $\mathsf{r} \in \mathrm{Reg}$, $[\![\mathsf{r}]\!]\gamma(a) = [\![\tau(\mathsf{r})]\!]\gamma(a)$ and $\tau(\mathsf{r}) \in \mathrm{BCA}(A, a)$. Let $b \in A \setminus \{\top_A\}$, and consider the set $\Pi^b_a \triangleq \{\mathsf{p} \in \mathrm{Reg} \mid \alpha[\![\mathsf{p}]\!]\gamma(a) = b\}$. Clearly, Π^b_a is extensional, therefore, by Rice's theorem, Π^b_a is not recursive being nontrivial: in fact, $\Pi^b_a \neq \mathrm{Reg}$ because, obviously, by Lemma 4.1, there exist at least two programs p_1 and p_2 such that $\alpha[\![\mathsf{p}_1]\!]\gamma(a) = b \neq \top_A = \alpha[\![\mathsf{p}_2]\!]\gamma(a)$, and $\Pi^b_a \neq \varnothing$ because $b \in A$, hence $\gamma(b) \in \wp^{\mathrm{re}}(\mathbb{D})$, thus, by Lemma 4.1, there exists $\mathsf{q} \in \mathrm{Reg}$ such that $[\![\mathsf{q}]\!]\gamma(a) = \gamma(b)$, and, in turn, $\alpha[\![\mathsf{q}]\!]\gamma(a) = \alpha\gamma(b) = b$. For all $\mathsf{r} \in \mathrm{Reg}$, we have that:

[1]Observe that in case of recursive abstract domains, it is decidable whether an element $b \in \mathbb{D}$ is such that $b \in \mathbb{D} \setminus \gamma\alpha(H)$, but, in general, even for recursive abstract domains, we cannot decide whether $c \in \gamma\alpha(S) \setminus S$, unless the set S is recursive.

$\mathsf{r} \in \Pi_a^b \Leftrightarrow \alpha[\![\mathsf{r}]\!]\gamma(a) = b \Leftrightarrow \alpha[\![\tau(\mathsf{r})]\!]\gamma(a) = b \Leftrightarrow [\![\tau(\mathsf{r})]\!]_A^\sharp a = b$. Since $[\![\cdot]\!]_A^\sharp$ is a total recursive function, the question $[\![\tau(\mathsf{r})]\!]_A^\sharp a =^? b$ can be decided, thus making Π_a^b recursive, which is a contradiction. □

4.2 Abstract Program Equivalence Is Not Extensional

Similarly to concrete semantics, abstract semantics merges programs into equivalence classes. We define the following concrete and abstract equivalences between programs in Reg as induced, resp., by $[\![\cdot]\!]$ and $[\![\cdot]\!]_A^\sharp$: $\mathsf{p} \sim \mathsf{q} \;\Leftrightarrow\; [\![\mathsf{p}]\!] = [\![\mathsf{q}]\!]$ and $\mathsf{p} \sim_A \mathsf{q} \;\Leftrightarrow\; [\![\mathsf{p}]\!]_A^\sharp = [\![\mathsf{q}]\!]_A^\sharp$.

We prove that there exists a compiler that preserves the concrete semantics but alters the abstract one, and, in particular, the result of this compilation does not satisfy the bca property. A similar result was given in [Bruni et al. 2020, Theorem 31] for the α-completeness property.

Lemma 4.5. *Let $A \in \mathrm{Abs}(\wp^{\mathrm{rec}}(\mathbb{D}))$ be a recursive abstract domain and $a \in A$ such that there exists $S \in \wp^{\mathrm{rec}}(\mathbb{D})$ for which $\alpha(S) \leq_A a$ and $S \subsetneq \gamma\alpha(S)$. Then, there exists a compiler $\tau : \mathrm{Reg} \to \mathrm{Reg}$ such that for any $\mathsf{r} \in \mathrm{Reg}$ with $[\![\mathsf{r}]\!]_A^\sharp a \neq \top_A$:* (i) $\mathsf{r} \sim \tau(\mathsf{r})$, (ii) $\mathsf{r} \nsim_A \tau(\mathsf{r})$, (iii) $\tau(\mathsf{r}) \notin \mathrm{BCA}(A, a)$.

Proof. Let $A \in \mathrm{Abs}(\wp^{\mathrm{rec}}(\mathbb{D}))$ and $a \in A$ be such that there exists $S \in \wp^{\mathrm{rec}}(\mathbb{D})$ for which $\alpha(S) \leq_A a$ and $S \subsetneq \gamma\alpha(S)$. Let $\mathsf{r} \in \mathrm{Reg}$ be such that $[\![\mathsf{r}]\!]_A^\sharp a \neq \top_A$. By monotonicity of $[\![\mathsf{r}]\!]_A^\sharp$, we have that $[\![\mathsf{r}]\!]_A^\sharp \alpha(S) \leq_A [\![\mathsf{r}]\!]_A^\sharp \top_A \leq_A \top_A$, so that, since γ is injective, $\gamma[\![\mathsf{r}]\!]_A^\sharp \alpha(S) \neq \gamma(\top_A) = \mathbb{D}$. Since $\gamma[\![\mathsf{r}]\!]_A^\sharp \alpha(S) \in \wp^{\mathrm{rec}}(\mathbb{D})$ we can find algorithmically an element $b \in \mathbb{D} \setminus \gamma([\![\mathsf{r}]\!]_A^\sharp \alpha(S))$. Since $S \in \wp^{\mathrm{rec}}(\mathbb{D})$, there exist two programs $\mathsf{q}_S, \bar{\mathsf{q}}_S \in \mathrm{Reg}$ such that, for any $X \in \wp^{\mathrm{rec}}(\mathbb{D})$, $[\![\mathsf{q}_S]\!]X = X \cap S$ and $[\![\bar{\mathsf{q}}_S]\!]X = X \cap \neg S$. The program transform $\tau : \mathrm{Reg} \to \mathrm{Reg}$ is defined as follows:

$$\tau(\mathsf{r}) \triangleq (\mathsf{q}_S;\, ((\bar{\mathsf{q}}_S;\, x := b) \oplus (\mathsf{q}_S;\, \mathsf{r}))) \oplus (\bar{\mathsf{q}}_S;\, \mathsf{r})\,.$$

It is clear that $[\![\tau(\mathsf{r})]\!] = [\![(\mathsf{q}_S;\, \mathsf{r}) \oplus (\bar{\mathsf{q}}_S;\, \mathsf{r})]\!] = [\![\mathsf{r}]\!]$ and $\tau(\mathsf{r}) \notin \mathrm{BCA}(A, a)$. It turns out that:

$$\begin{aligned}
[\![\tau(\mathsf{r})]\!]_A^\sharp \alpha(S) &= \\
[\![(\mathsf{q}_S;\, ((\bar{\mathsf{q}}_S;\, x := b) \oplus (\mathsf{q}_S;\, \mathsf{r}))) \oplus (\bar{\mathsf{q}}_S;\, \mathsf{r})]\!]_A^\sharp \alpha(S) &= \\
[\![\mathsf{q}_S;\, ((\bar{\mathsf{q}}_S;\, x := b) \oplus (\mathsf{q}_S;\, \mathsf{r}))]\!]_A^\sharp \alpha(S) \vee_A [\![\bar{\mathsf{q}}_S;\, \mathsf{r}]\!]_A^\sharp \alpha(S) &= \\
[\![\mathsf{q}_S;\, (\bar{\mathsf{q}}_S;\, x := b)]\!]_A^\sharp \alpha(S) \vee_A [\![\mathsf{q}_S;\, \mathsf{q}_S;\, \mathsf{r}]\!]_A^\sharp \alpha(S) \vee_A [\![\bar{\mathsf{q}}_S;\, \mathsf{r}]\!]_A^\sharp \alpha(S) &= \\
[\![\mathsf{q}_S;\, (\bar{\mathsf{q}}_S;\, x := b)]\!]_A^\sharp \alpha(S) \vee_A [\![\mathsf{q}_S;\, \mathsf{r}]\!]_A^\sharp \alpha(S) \vee_A [\![\bar{\mathsf{q}}_S;\, \mathsf{r}]\!]_A^\sharp \alpha(S) &\geq_A \\
&\qquad [\text{as } [\![\mathsf{q}_S]\!]_A^\sharp \alpha(S) \geq_A \alpha(S),\, [\![\bar{\mathsf{q}}_S]\!]_A^\sharp \alpha(S) \geq_A \alpha(\varnothing)] \\
[\![\mathsf{q}_S;\, (\bar{\mathsf{q}}_S;\, x := b)]\!]_A^\sharp \alpha(S) \vee_A [\![\mathsf{r}]\!]_A^\sharp \alpha(S) &= \\
[\![x := b]\!]_A^\sharp [\![\bar{\mathsf{q}}_S]\!]_A^\sharp [\![\mathsf{q}_S]\!]_A^\sharp \alpha(S) \vee_A [\![\mathsf{r}]\!]_A^\sharp \alpha(S) &\geq_A \qquad [\text{as } [\![\mathsf{q}_S]\!]_A^\sharp \alpha(S) \geq_A \alpha(S)] \\
[\![x := b]\!]_A^\sharp [\![\bar{\mathsf{q}}_S]\!]_A^\sharp \alpha(S) \vee_A [\![\mathsf{r}]\!]_A^\sharp \alpha(S) &\geq_A \qquad [\text{as } [\![\mathsf{q}_S]\!]_A^\sharp \dot{\geq}_A \alpha[\![\bar{\mathsf{q}}_S]\!]\gamma] \\
[\![x := b]\!]_A^\sharp \alpha[\![\bar{\mathsf{q}}_S]\!]\gamma\alpha(S) \vee_A [\![\mathsf{r}]\!]_A^\sharp \alpha(S) &\geq_A \qquad [\text{as } [\![x := b]\!]_A^\sharp \dot{\geq}_A \alpha[\![x := b]\!]\gamma] \\
\alpha[\![x := b]\!]\gamma\alpha[\![\bar{\mathsf{q}}_S]\!]\gamma\alpha(S) \vee_A [\![\mathsf{r}]\!]_A^\sharp \alpha(S)\,. &
\end{aligned}$$

Since $S \subsetneq \gamma\alpha(S)$, we have that $[\![\bar{\mathsf{q}}_S]\!]\gamma\alpha(S) \neq \varnothing$. Thus, $\alpha[\![x := b]\!]\gamma\alpha[\![\bar{\mathsf{q}}_S]\!]\gamma\alpha(S) \vee_A [\![\mathsf{r}]\!]_A^\sharp \alpha(S) \geq_A [\![\mathsf{r}]\!]_A^\sharp \alpha(S)$, otherwise we would have that $\alpha[\![x := b]\!]\gamma\alpha[\![\bar{\mathsf{q}}_S]\!]\gamma\alpha(S) \leq_A [\![\mathsf{r}]\!]_A^\sharp \alpha(S)$, which would entail that $b \in \gamma\alpha[\![x := b]\!]\gamma\alpha[\![\bar{\mathsf{q}}_S]\!]\gamma\alpha(S) \subseteq \gamma[\![\mathsf{r}]\!]_A^\sharp \alpha(S)$, which would be a contradiction to the fact that $b \in \mathbb{D} \setminus \gamma([\![\mathsf{r}]\!]_A^\sharp \alpha(S))$. We have therefore shown that $[\![\mathsf{r}]\!]_A^\sharp \alpha(S) \neq [\![\tau(\mathsf{r})]\!]_A^\sharp \alpha(S)$, i.e., $\tau(\mathsf{r}) \nsim_A \mathsf{r}$. □

We call the inductive abstract semantics $[\![\cdot]\!]_A^\sharp$ of definition (2) *non-straightforward* when there exists a program $\mathsf{p} \in \mathrm{Reg}$ such that $[\![\mathsf{p}]\!]_A^\sharp \top_A \neq \top_A$. Note that a nontrivial abstract domain A does not necessarily induce a non-straightforward abstract semantics $[\![\cdot]\!]_A^\sharp$. It is enough to consider a Turing complete language Reg with basic commands $\mathrm{BCmd} = \{x \geq 0?, \mathbf{skip}, x := x + 1, x := 0\}$, and an abstract domain $\mathrm{CP}_{\neq 0}$ defined as the constant propagation abstract domain [Wegman and Zadeck

1991] without the value 0 to make the corresponding abstract semantics $[\![\cdot]\!]^\sharp_{\mathrm{CP}_{\neq 0}}$ straightforward. As a consequence of Lemma 4.5, the only non-straightforward abstract semantics defined on recursive abstract domains inducing an extensional program equivalence is the bca.

Corollary 4.6 (Abstract Program Equivalence is not Extensional). *Let $A \in \mathrm{Abs}(\wp^{\mathrm{rec}}(\mathbb{D}))$ and $[\![\cdot]\!]^\sharp_A$ be non-straightforward. Then, $\sim_A$ is extensional iff $\mathrm{BCA}(A) = \mathrm{Reg}$ iff $A = id_{\mathrm{Abs}}$.*

Proof. If $A = id_{\mathrm{Abs}}$ then $\sim_{id_{\mathrm{Abs}}} = \sim$. Let A be nontrivial and $[\![\cdot]\!]^\sharp_A$ be non-straightforward. Therefore, there exists $\mathsf{r} \in \mathrm{Reg}$ such that $[\![\mathsf{r}]\!]^\sharp_A \top_A \neq \top_A$. By Lemma 4.5, $\mathsf{r} \sim \tau(\mathsf{r})$ and $\tau(\mathsf{r}) \not\sim_A \mathsf{r}$. □

4.3 BCA Is Not Recursive

We conclude our study by proving that $\mathrm{BCA}(A, a)$ and $\mathrm{BCA}(A)$ are, in general, nonrecursive sets of programs, so that the questions "$\mathrm{bca}_A(\mathsf{p}, a)$?" and "$\mathrm{bca}_A(\mathsf{p})$?" are undecidable. Let us remark that the non-extensionality of the abstract equivalence $\sim_A$, given by Corollary 4.6, prevents the application of Rice's theorem for proving the nonrecursivity of $\mathrm{BCA}(A)$. On the other hand, similarly to the proof of Rice's theorem, this result can be anyway established through a diagonal argument obtained by applying Kleene's fixpoint theorem, both for $\mathrm{BCA}(A)$ and $\mathrm{BCA}(A, a)$. We assume that the abstract elements $a \in A$, being representations of recursive enumerable sets $\gamma(a) \in \wp^{\mathrm{re}}(\mathbb{D})$, have an encoding in Reg, so that for each $a \in A$ we can always construct a program $\mathsf{p}_a \in \mathrm{Reg}$ such that $\gamma(a) = \mathrm{rng}([\![\mathsf{p}_a]\!])$.

Theorem 4.7 (Local BCA is not Recursive). *Let $A \in \mathrm{Abs}(\wp^{\mathrm{rec}}(\mathbb{D}))$ be nontrivial, $a \in A \setminus \{\bot_A\}$, and $[\![\cdot]\!]^\sharp_A$ be a total recursive function. Then, $\mathrm{BCA}(A, a)$ is not recursive.*

Proof. Let $A \in \mathrm{Abs}(\wp^{\mathrm{rec}}(\mathbb{D}))$ be a nontrivial recursive abstract domain. Because the abstract semantics $[\![\cdot]\!]^\sharp_A : \mathrm{Reg} \times A \longrightarrow A$ is a total recursive function and Reg is Turing complete, there exists a program $\mathsf{u} \in \mathrm{Reg}$ such that $[\![\mathsf{u}]\!] : \mathrm{Reg} \times \mathbb{D} \longrightarrow \mathbb{D}$ is a binary total recursive function such that for all $\mathsf{p} \in \mathrm{Reg}$ and $a \in A$, $\gamma([\![\mathsf{p}]\!]^\sharp_A a) = [\![\mathsf{u}]\!](\mathsf{p}, \gamma(a))$. By s-m-n theorem, there exists a total recursive function $\sharp : \mathrm{Reg} \times \mathrm{Reg} \longrightarrow \mathrm{Reg}$ such that for all $\mathsf{p} \in \mathrm{Reg}$ and $a \in A$, $\gamma([\![\mathsf{p}]\!]^\sharp_A a) = [\![\sharp(\mathsf{u}, \mathsf{p})]\!]\gamma(a)$ holds, thus entailing that $[\![\mathsf{p}]\!]^\sharp_A a = \alpha\gamma([\![\mathsf{p}]\!]^\sharp_A a) = \alpha[\![\sharp(\mathsf{u}, \mathsf{p})]\!]\gamma(a)$. Let $a \in A$ such that $a \neq \bot_A$. Because $a \neq \bot_A$, we have that $\gamma(a) \neq \varnothing$. Moreover, $\gamma(a) \in \wp^{\mathrm{rec}}(\mathbb{D})$, hence, by Lemma 4.1, there exists a program $\mathsf{c}_a \in \mathrm{Reg}$ such that $\alpha[\![\mathsf{c}_a]\!]\gamma(a) = a$. Since A is nontrivial there exists a program $\mathsf{c}_{\neq a} \in \mathrm{Reg}$ such that $\alpha[\![\mathsf{c}_{\neq a}]\!]\gamma(a) \neq a$.

Assume, by contradiction, that $\mathrm{BCA}(A, a)$ is recursive, so that we can define the following function $\mathsf{h} : \mathrm{Reg} \longrightarrow \mathrm{Reg}$: for all $\mathsf{p} \in \mathrm{Reg}$,

$$\mathsf{h}(\mathsf{p}) \triangleq \begin{cases} \mathsf{c}_a & \text{if } \mathsf{p} \in \mathrm{BCA}(A, a) \text{ and } [\![\mathsf{p}]\!]^\sharp_A a \neq a \\ \mathsf{c}_{\neq a} & \text{if } \mathsf{p} \in \mathrm{BCA}(A, a) \text{ and } [\![\mathsf{p}]\!]^\sharp_A a = a \\ \sharp(\mathsf{u}, \mathsf{p}) & \text{if } \mathsf{p} \notin \mathrm{BCA}(A, a) \end{cases} \tag{5}$$

Since $\mathrm{BCA}(A, a)$ is recursive and $[\![\cdot]\!]^\sharp_A$ is a total recursive function, we have that $[\![\mathsf{p}]\!]^\sharp_A a =^? a$ is a recursive predicate on Reg, so that h turns out to be a total recursive function. By Kleene's fixpoint theorem applied to h, there exists $\mathsf{e} \in \mathrm{Reg}$ such that $[\![\mathsf{e}]\!] = [\![\mathsf{h}(\mathsf{e})]\!]$.

Assume that $\mathsf{e} \in \mathrm{BCA}(A, a)$. We distinguish two cases.

(1) If $[\![\mathsf{e}]\!]^\sharp_A a \neq a$ then we have that:

$$\begin{aligned} [\![\mathsf{e}]\!]^\sharp_A a = &\quad [\text{as } \mathsf{e} \in \mathrm{BCA}(A, a)] \\ \alpha[\![\mathsf{e}]\!]\gamma(a) = &\quad [\text{as } \mathsf{e} \sim \mathsf{h}(\mathsf{e})] \\ \alpha[\![\mathsf{h}(\mathsf{e})]\!]\gamma(a) = &\quad [\text{by definition (5) of h}] \\ \alpha[\![\mathsf{c}_a]\!]\gamma(a) = a &\quad [\text{as shown above}] \end{aligned}$$

Hence, this contradicts the hypothesis that $[\![\mathsf{e}]\!]^\sharp_A a \neq a$.
(2) If $[\![\mathsf{e}]\!]^\sharp_A a = a$ then:

$$\begin{aligned} a = & \quad [\text{by assumption}] \\ [\![\mathsf{e}]\!]^\sharp_A a = & \quad [\text{as } \mathsf{e} \in \mathrm{BCA}(A, a)] \\ \alpha[\![\mathsf{e}]\!]\gamma(a) = & \quad [\text{as } \mathsf{e} \sim \mathsf{h}(\mathsf{e})] \\ \alpha[\![\mathsf{h}(\mathsf{e})]\!]\gamma(a) = & \quad [\text{by definition (5) of h}] \\ \alpha[\![\mathsf{c}_{\neq a}]\!]\gamma(a) \neq a & \end{aligned}$$

which is again a contradiction. To conclude, let us assume that $\mathsf{e} \notin \mathrm{BCA}(A, a)$. Then, we have that:

$$\begin{aligned} \alpha[\![\mathsf{e}]\!]\gamma(a) = & \quad [\text{as } \mathsf{e} \sim \mathsf{h}(\mathsf{e})] \\ \alpha[\![\mathsf{h}(\mathsf{e})]\!]\gamma(a) = & \quad [\text{by definition (5) of h}] \\ \alpha[\![\sharp(\mathsf{u}, \mathsf{e})]\!]\gamma(a) = [\![\mathsf{e}]\!]^\sharp_A a & \quad [\text{by definition of } \sharp] \end{aligned}$$

hence proving that $e \in \mathrm{BCA}(A, a)$, which is a contradiction. □

As a consequence, we also obtain the nonrecursivity of $\mathrm{BCA}(A)$.

Corollary 4.8 (Global BCA is not Recursive). *Let $A \in \mathrm{Abs}(\wp^{\mathrm{rec}}(\mathbb{D}))$ be nontrivial and $[\![\cdot]\!]^\sharp_A$ be a total recursive function. Then, $\mathrm{BCA}(A)$ is not recursive.*

Proof. The proof follows immediately by Theorem 4.7 because the predicate $\mathrm{BCA}(A, a)$ is not recursive and, by definition, $\mathsf{p} \in \mathrm{BCA}(A) \Leftrightarrow \forall a \in A.\ \mathsf{p} \in \mathrm{BCA}(A, a)$. □

5 Impossibility of Minimal Abstraction Refinement or Simplification to Achieve Bca

We proved in Theorem 4.4 that, in general, it is impossible to compile a program into an equivalent one, even for a specific abstract precondition, that satisfies the bca property on a given abstraction A. It is, therefore, natural to consider whether the bca property can be achieved by modifying the abstract domain A. Given a program r and an abstract domain A, we address the following two basic questions that arise for the bca property:

- (Q_1) does there exist the *least domain refinement* A_r of A such that r satisfies the local/global bca property on A_r?
- (Q_2) does there exist the *greatest domain abstraction* A_a of A such that r satisfies the local/global bca property on A_a?

Least refinements and greatest abstractions of domains in abstract interpretation have been studied within a general framework in [Filé et al. 1996; Giacobazzi and Ranzato 1997]. Following [Giacobazzi and Ranzato 1997], given a property $\mathcal{P}$ of abstractions of a concrete domain C, namely $\mathcal{P} \subseteq \mathrm{Abs}(C)$, the *core* of an abstract domain $A \in \mathrm{Abs}(C)$, when it exists, is the most concrete abstraction of A in the complete lattice $\langle \mathrm{Abs}(C), \sqsubseteq \rangle$ that satisfies $\mathcal{P}$. Dually, the *shell* of A, when it exists, is the most abstract refinement of A that satisfies $\mathcal{P}$. Notable examples include the completeness core and shell investigated by Giacobazzi et al. [2000], the disjunctive shell (a.k.a. disjunctive completion) [Cousot and Cousot 1979; Filé and Ranzato 1999; Giacobazzi and Ranzato 1996], and the condensing shell [Giacobazzi et al. 2005] to make goal-driven and goal-independent analyses agree.

In the following, we provide negative answers to both questions (Q_1) and (Q_2) above, thus showing the impossibility of attaining the (global or local) bca property through minimal abstraction refinements or simplifications.

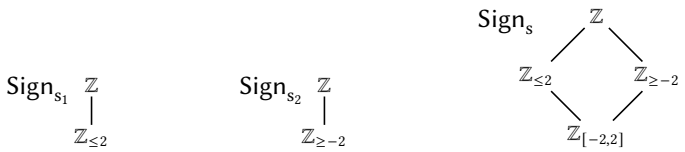

Fig. 2. Abstract Domains for Sign Analysis.

5.1 Global and Local bca Shells Do Not Exist

Consider the program $\mathsf{p} \triangleq (z := 2; z := z + 1); z := z - 1$ already defined in (3) of Example 3.3 and its Sign analysis $[\![\mathsf{p}]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z} = \mathbb{Z}_{\geq 0}$. We observed in Example 3.3 that the analysis of $z := 2; z := z + 1$ is as precise as possible, i.e., $[\![z := 2; z := z + 1]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z} = \mathbb{Z}_{>0}$, whereas the bca property is lost when composing with the analysis of the third assignment as $[\![z := z - 1]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{>0} = \mathbb{Z}_{\geq 0}$. Hence, p does not satisfy the gbca property on Sign, and the lbca property on any abstract input in Sign different from $\varnothing$. We observe that $\mathsf{Sign}_{\mathrm{r}_1} \triangleq \mathsf{Sign} \cup \{\mathbb{Z}_{>1}, \mathbb{Z}_{>2}\}$ and $\mathsf{Sign}_{\mathrm{r}_2} \triangleq \mathsf{Sign} \cup \{\mathbb{Z}_{=2}, \mathbb{Z}_{=3}\}$ are two abstract domains that refine Sign and such that the analysis of p on both of them satisfies the gbca property. To check this, it is enough to consider the input abstract value $\mathbb{Z} \in \mathsf{Sign}$:

$$\begin{aligned}
[\![\mathsf{p}]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_1}}\mathbb{Z} &= [\![z := z - 1]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_1}}[\![z := z + 1]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_1}}[\![z := 2]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_1}}\mathbb{Z} \\
&= [\![z := z - 1]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_1}}[\![z := z + 1]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_1}}\mathbb{Z}_{>1} \\
&= [\![z := z - 1]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_1}}\mathbb{Z}_{>2} \;=\; \mathbb{Z}_{>1} \;=\; \alpha_{\mathsf{Sign}_{\mathrm{r}_1}}[\![\mathsf{p}]\!]\gamma_{\mathsf{Sign}_{\mathrm{r}_1}}(\mathbb{Z})\,,
\end{aligned}$$

$$\begin{aligned}
[\![\mathsf{p}]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_2}}\mathbb{Z} &= [\![z := z - 1]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_2}}[\![z := z + 1]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_2}}[\![z := 2]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_2}}\mathbb{Z} \\
&= [\![z := z - 1]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_2}}[\![z := z + 1]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_2}}\mathbb{Z}_{=2} \\
&= [\![z := z - 1]\!]^\sharp_{\mathsf{Sign}_{\mathrm{r}_2}}\mathbb{Z}_{=3} \;=\; \mathbb{Z}_{=2} \;=\; \alpha_{\mathsf{Sign}_{\mathrm{r}_2}}[\![\mathsf{p}]\!]\gamma_{\mathsf{Sign}_{\mathrm{r}_2}}(\mathbb{Z})\,.
\end{aligned}$$

However, it turns out that the lub in $\mathsf{Abs}(\wp(\mathbb{Z}))$ of $\mathsf{Sign}_{\mathrm{r}_1}$ and $\mathsf{Sign}_{\mathrm{r}_2}$ is Sign, i.e. $\mathsf{Sign}_{\mathrm{r}_1} \sqcup \mathsf{Sign}_{\mathrm{r}_2} = \mathsf{Sign}$, and this entails that the least domain refinement $\mathsf{Sign}_{\mathrm{r}}$ of Sign such that p satisfies the gbca property on $\mathsf{Sign}_{\mathrm{r}}$ does not exist, because:

$$\mathsf{Sign} = \mathsf{Sign}_{\mathrm{r}_1} \sqcup \mathsf{Sign}_{\mathrm{r}_2} \sqsubseteq \sqcup\{A \in \mathsf{Abs}(\wp(\mathbb{Z})) \mid A \sqsubseteq \mathsf{Sign},\ \mathrm{bca}_A(\mathsf{p})\} \sqsubseteq \mathsf{Sign}\,.$$

This same argument also shows that the least domain refinement $\mathsf{Sign}_{\mathrm{r}}$ of Sign such that $\mathrm{bca}_{\mathsf{Sign}_{\mathrm{r}}}(\mathsf{p}, \mathbb{Z})$ holds, does not exist. We have thus shown that the question (Q_1), in general, has a negative answer.

5.2 Global and Local bca Cores Do Not Exist

Consider the program $\mathsf{p} \triangleq x := -x/2;\ x := x + 1$, where n/m denotes integer division (i.e., the fractional part is discarded), and its sign analysis on the domain Sign_s depicted in Figure 2, whose abstract values clearly define the corresponding abstraction map α_{Sign_s}:

$$[\![\mathsf{p}]\!]^\sharp_{\mathsf{Sign}_s}\mathbb{Z}_{[-2,2]} = [\![x := x + 1]\!]^\sharp_{\mathsf{Sign}_s}[\![x := -x/2]\!]^\sharp_{\mathsf{Sign}_s}\mathbb{Z}_{[-2,2]} = [\![x := x + 1]\!]^\sharp_{\mathsf{Sign}_s}\mathbb{Z}_{[-2,2]} \;=\; \mathbb{Z}_{\geq -2}\,.$$

This is not the best analysis on Sign_s, because:

$$\begin{aligned}
\alpha_{\mathsf{Sign}_s}[\![x := x + 1]\!][\![x := -x/2]\!]\gamma_{\mathsf{Sign}_s}(\mathbb{Z}_{[-2,2]}) &= \alpha_{\mathsf{Sign}_s}[\![x := x + 1]\!]\{-1, 0, 1\} \\
&= \alpha_{\mathsf{Sign}_s}(\{0, 1, 2\}) = \mathbb{Z}_{[-2,2]}\,.
\end{aligned}$$

On the other hand, we observe that Sign_{s_1} and Sign_{s_2} in Figure 2, which are further abstractions of Sign_s, both satisfy the gbca property because:

$$\begin{aligned}\llbracket \mathsf{p} \rrbracket^\sharp_{\mathsf{Sign}_{s_1}} \mathbb{Z}_{\leq 2} &= \llbracket x := x+1 \rrbracket^\sharp_{\mathsf{Sign}_{s_1}} \llbracket x := -x/2 \rrbracket^\sharp_{\mathsf{Sign}_{s_1}} \mathbb{Z}_{\leq 2} \\ &= \llbracket x := x+1 \rrbracket^\sharp_{\mathsf{Sign}_{s_1}} \mathbb{Z} = \mathbb{Z} = \alpha_{\mathsf{Sign}_{s_1}} \llbracket \mathsf{p} \rrbracket \gamma_{\mathsf{Sign}_{s_1}} (\mathbb{Z}_{\leq 2}) \,, \\ \llbracket \mathsf{p} \rrbracket^\sharp_{\mathsf{Sign}_{s_2}} \mathbb{Z}_{\geq -2} &= \llbracket x := x+1 \rrbracket^\sharp_{\mathsf{Sign}_{s_2}} \llbracket x := -x/2 \rrbracket^\sharp_{\mathsf{Sign}_{s_2}} \mathbb{Z}_{\geq -2} \\ &= \llbracket x := x+1 \rrbracket^\sharp_{\mathsf{Sign}_{s_2}} \mathbb{Z} = \mathbb{Z} = \alpha_{\mathsf{Sign}_{s_2}} \llbracket \mathsf{p} \rrbracket \gamma_{\mathsf{Sign}_{s_2}} (\mathbb{Z}_{\geq -2}) \,. \end{aligned}$$

Moreover, for the abstract value $\mathbb{Z}$, we can easily check that $\llbracket \mathsf{p} \rrbracket^\sharp_{\mathsf{Sign}_{s_1}} \mathbb{Z} = \mathbb{Z} = \alpha_{\mathsf{Sign}_{s_1}} \llbracket \mathsf{p} \rrbracket \gamma_{\mathsf{Sign}_{s_1}} (\mathbb{Z})$, and $\llbracket \mathsf{p} \rrbracket^\sharp_{\mathsf{Sign}_{s_2}} \mathbb{Z} = \mathbb{Z} = \alpha_{\mathsf{Sign}_{s_2}} \llbracket \mathsf{p} \rrbracket \gamma_{\mathsf{Sign}_{s_2}} (\mathbb{Z})$.

We observe that $\mathsf{Sign}_s = \mathsf{Sign}_{s_1} \sqcap \mathsf{Sign}_{s_2}$ holds, where $\sqcap$ is the glb in $\mathrm{Abs}(\wp(\mathbb{Z}))$. Hence, this implies that the least domain abstraction Sign_a of Sign_s such that p satisfies the global bca property on Sign_a does not exist, as

$$\mathsf{Sign}_s \sqsubseteq \sqcap \{ A \in \mathrm{Abs}(\wp(\mathbb{Z})) \mid \mathsf{Sign}_s \sqsubseteq A,\, \mathrm{bca}_A(\mathsf{p}) \} \sqsubseteq \mathsf{Sign}_{s_1} \sqcap \mathsf{Sign}_{s_2} = \mathsf{Sign}_s \,.$$

This example therefore shows that the question (Q_2) on the existence of the core for the gbca property has, in general, a negative answer.

For the local bca property, the question (Q_2) needs to be clarified. In fact, if a given domain A is not lbca on some abstract input $a \in A$, and we consider a further abstraction $A' \sqsupseteq A$, it may happen that $\gamma(a) \notin \gamma(A')$, so that we cannot ask whether A' satisfies the local bca property on a. Thus, the question needs to be reformulated by leveraging concrete values as follows: A program $\mathsf{r} \in \mathrm{Reg}$ is defined to satisfy the lbca property on some concrete value $c \in C$ for the abstraction A when $\llbracket \mathsf{r} \rrbracket^\sharp_A \alpha_A(c) = \alpha_A \llbracket \mathsf{r} \rrbracket \gamma_A \alpha_A(c)$ holds. Of course, this definition includes the case of Definition 3.1 (i), as any abstract value is the abstraction of some concrete value, i.e., for all $a \in A$ there exists $c_a \in C$ such that $\alpha_A(c_a) = a$. Then, according to this more general definition of lbca, the example above shows that for p, the lbca core of Sign_s for the concrete value $\{-2, -1, 0, 1, 2\} \in \wp(\mathbb{Z})$ does not exist.

5.3 Bca vs Completeness

It is known that *global* α- and γ-complete shells and cores exist with very mild hypotheses [Giacobazzi et al. 1998, 2000; Ranzato 2013]. Thus, in this respect, global completeness is in sharp contrast w.r.t. the global bca property. On the other hand, local α-complete shells do not exist in general, and this led to investigating a notion of *pointed locally* α*-complete* shells, where the domain refinement of an abstraction A consists in adding to A the greatest (i.e., less precise) concrete value that makes the refined abstraction locally α-complete on some given input [Bruni et al. 2022, Section 4]. The lack of existence of local α-complete shells is therefore in accordance with the negative answer to (Q_1) for the local bca property shown above in Section 5.1. On the other hand, in contrast to the negative answer to (Q_2) for global and local bca cores, for the sake of completeness, we show that locally α-complete cores unconditionally exist.

Lemma 5.1 (Existence of Locally α-Complete Cores). *Let $f : C \longrightarrow C$ be a monotone function, $\{A_i \in \mathrm{Abs}(C) \mid i \in I\}$ be a family of abstract domains, and $c \in C$. If, for all $i \in I$, A_i is locally α-complete for f on c then $\sqcap_{i \in I} A_i$ is locally α-complete for f on c.*
As a consequence, locally α-complete cores always exist, i.e., given $A \in \mathrm{Abs}(C)$, the abstract domain $\sqcap \{A_a \in \mathrm{Abs}(C) \mid A_a \sqsubseteq A,\, A_a$ locally α-complete for f on $c\}$ is locally α-complete for f on c.

Proof. Let $\mu_i \triangleq \gamma_{A_i} \circ \alpha_{A_i}$ be the upper closure operator on C induced by the abstraction A_i, so that the local α-completeness equation for A_i boils down to $\mu_i f \mu_i(c) = \mu_i f(c)$. We observe that for all $j \in I$, $(\sqcap_i \mu_i) f (\sqcap_i \mu_i)(c) = (\sqcap_i \mu_i) f (\wedge_i \mu_i(c)) \leq (\sqcap_i \mu_i) f(\mu_j(c)) = \wedge_i \mu_i (f(\mu_j(c))) \leq \mu_j(f(\mu_j(c))) =$

$\mu_j(f(c))$. Thus, it turns out that $(\sqcap_i\mu_i)f(\sqcap_i\mu_i)(c) \leq \wedge_{i\in I}\mu_i(f(c)) = (\sqcap_i\mu_i)f(c)$, and, in turn, we have that $(\sqcap_i\mu_i)f(\sqcap_i\mu_i)(c) = (\sqcap_i\mu_i)f(c)$.
Therefore, $\sqcap\{A_{\mathrm{a}} \in \mathrm{Abs}(C) \mid A \sqsubseteq A_{\mathrm{a}},\ A_{\mathrm{a}} \text{ locally } \alpha\text{-complete for } f \text{ on } c\}$ is an abstraction which is locally α-complete for f on c, namely, it is the locally α-complete core of A for f on c. □

6 Bca Triples

Given a program r and an abstraction A, we formalize the intuitive idea of a logical triple $[a]_A\ \mathsf{r}\ [b]_A$ where the precondition is any abstract value $a \in A$ and the postcondition $b \in A$ is the inductive abstract semantics of r in A and this is the best possible one, i.e., b coincides with the best correct approximation $\alpha_A[\![\mathsf{r}]\!]\gamma_A(a)$.

Definition 6.1 (Validity of bca Triples). Let $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$ be any abstract domain. A *bca triple* is $[a]_A\ \mathsf{r}\ [b]_A$ where $\mathsf{r} \in \mathrm{Reg}$ and $a, b \in A$. A bca triple $[a]_A\ \mathsf{r}\ [b]_A$ is *valid*, also denoted by $\models [a]_A\ \mathsf{r}\ [b]_A$, when:

(i) $[\![\mathsf{r}]\!]^\sharp_A a = b$, i.e., b is the inductive analysis of r in A on input a;
(ii) $\alpha[\![\mathsf{r}]\!]\gamma(a) = b$, i.e., b is the bca of r in A on input $\gamma(a)$. □

Let us remark that, given $a \in A$, there exists $b \in A$ such that $\models [a]_A\ \mathsf{r}\ [b]_A$ iff $\mathsf{r} \in \mathrm{BCA}(A, a)$, that is, the inductive analysis of r in A is the best possible one.

Remark 6.2 (Naïve bca Triples are not Valid). Firstly, it is worth observing that, in general, for all $a \in A$, the naïve bca triples having as postcondition the bca or the analysis of the program r, in general, are not valid, namely,

$$(1)\ [a]_A\ \mathsf{r}\ [\alpha[\![\mathsf{r}]\!]\gamma(a)]_A \quad \text{and} \quad (2)\ [a]_A\ \mathsf{r}\ [[\![\mathsf{r}]\!]^\sharp_A a]_A \quad \text{are not, in general, valid triples.}$$

In fact, by choosing $\mathsf{p} \triangleq x := x + 1;\ x := x - 1$, $A = \mathsf{Sign}$, and $a = \mathbb{Z}_{>0} \in A$, we can show both (1) and (2) because $[\![\mathsf{p}]\!]^\sharp_{\mathsf{Sign}}(\mathbb{Z}_{>0}) = \mathbb{Z}_{\geq 0} \neq \mathbb{Z}_{>0} = \alpha_{\mathsf{Sign}}[\![\mathsf{p}]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}_{>0})$, so that both $[\mathbb{Z}_{>0}]_{\mathsf{Sign}}\ \mathsf{p}\ [\mathbb{Z}_{>0}]_{\mathsf{Sign}}$ and $[\mathbb{Z}_{>0}]_{\mathsf{Sign}}\ \mathsf{p}\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}$ are not valid triples. □

Remark 6.3 (Local α-Completeness and bca Validity). Secondly, we note that local α-completeness is a sufficient condition to have a valid bca triple, i.e., if $[\![\mathsf{r}]\!]^\sharp_A\alpha(c) = \alpha[\![\mathsf{r}]\!]c$ holds, i.e. r is locally α-complete in A on $c \in C$, then $[\alpha(c)]_A\ \mathsf{r}\ [[\![\mathsf{r}]\!]^\sharp_A\alpha(c)]_A$ is a valid bca triple. In fact, we have that:

$$\begin{aligned}
[\![\mathsf{r}]\!]^\sharp_A\alpha(c) &= && \text{[by local } \alpha\text{-completeness]}\\
\alpha[\![\mathsf{r}]\!]c &\leq && \text{[as } c \leq \gamma\alpha(c) \text{ and by monotonicity of } \alpha[\![\mathsf{r}]\!]\text{]}\\
\alpha[\![\mathsf{r}]\!]\gamma\alpha(c) &\leq [\![\mathsf{r}]\!]^\sharp_A\alpha(c) && \text{[by soundness of } [\![\mathsf{r}]\!]^\sharp_A\text{]}
\end{aligned}$$

so that $[\![\mathsf{r}]\!]^\sharp_A\alpha(c) = \alpha[\![\mathsf{r}]\!]\gamma\alpha(c)$ holds, that is, $[\alpha(c)]_A\ \mathsf{r}\ [[\![\mathsf{r}]\!]^\sharp_A\alpha(c)]_A$ is valid.

The vice versa does not hold, namely, it may happen that $[\alpha(c)]_A\ \mathsf{r}\ [[\![\mathsf{r}]\!]^\sharp_A\alpha(c)]_A$ is a valid bca triple whereas $[\![\mathsf{r}]\!]^\sharp_A\alpha(c) \neq \alpha[\![\mathsf{r}]\!]c$. An example showing this can be given by considering the program $\mathsf{p} \triangleq x := x+2;\ x := x-1$, the Sign abstract domain, and $c = \{-1, 0\} \in \wp(\mathbb{Z})$ such that $\alpha_{\mathsf{Sign}}(c) = \mathbb{Z}_{\leq 0}$. In fact, we have that:

$$\begin{aligned}
[\![\mathsf{p}]\!]^\sharp_{\mathsf{Sign}}\alpha_{\mathsf{Sign}}(c) &= [\![x := x - 1]\!]^\sharp_{\mathsf{Sign}}[\![x := x + 2]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{\leq 0}\\
&= [\![x := x - 1]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z} = \mathbb{Z} \neq \mathbb{Z}_{\geq 0} = \alpha_{\mathsf{Sign}}(\{0, 1\}) = \alpha_{\mathsf{Sign}}[\![\mathsf{p}]\!]c\,,
\end{aligned}$$

namely, local α-completeness does not hold. On the other hand, the bca triple $[\mathbb{Z}_{\leq 0}]_A\ \mathsf{p}\ [[\![\mathsf{p}]\!]^\sharp_A\mathbb{Z}_{\leq 0}]_A \equiv [\mathbb{Z}_{\leq 0}]_A\ \mathsf{p}\ [\mathbb{Z}]_A$ is valid because $[\![\mathsf{p}]\!]^\sharp_A\mathbb{Z}_{\leq 0} = \mathbb{Z} = \alpha_{\mathsf{Sign}}(\mathbb{Z}_{\leq 1}) = \alpha_{\mathsf{Sign}}[\![\mathsf{p}]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}_{\leq 0})$. □

Remark 6.4 (On the Weakest Abstract Precondition). Given a program r and an abstract postcondition $b \in A$, an expected question to settle is about the existence of the weakest abstract

$$\frac{}{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r} \, [\alpha[\![\mathsf{r}]\!]\gamma(a)]_A} \text{ (basic)} \qquad \frac{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r}_1 \, [b_1]_A \quad \vdash_{\mathrm{bca}} [a]_A \, \mathsf{r}_2 \, [b_2]_A}{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r}_1 \oplus \mathsf{r}_2 \, [b_1 \vee_A b_2]_A} \text{ (choice)}$$

$$\frac{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r}_1 \, [c]_A \quad \vdash_{\mathrm{bca}} [c]_A \, \mathsf{r}_2 \, [b]_A \quad b = \alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a)}{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r}_1; \mathsf{r}_2 \, [b]_A} \text{ (seq)}$$

$$\frac{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r} \, [b]_A \quad b \leq_A a}{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r}^{\mathrm{fix}} \, [a]_A} \text{ (abs-inv)} \qquad \frac{\exists n \geq 1. \ \vdash_{\mathrm{bca}} [a]_A \, \mathsf{r}^n \, [b]_A \quad \vdash_{\mathrm{bca}} [a \vee_A b]_A \, \mathsf{r}^{\mathrm{fix}} \, [a \vee_A b]_A}{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r}^{\mathrm{fix}} \, [a \vee_A b]_A} \text{ (rec)}$$

$$\frac{\vdash_{\mathrm{bca}} [a']_A \, \mathsf{r} \, [b]_A \quad a' \leq_A a \quad [\![\mathsf{r}]\!]^\sharp_A a \leq_A b}{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r} \, [b]_A} \text{ (weaken}_{\mathrm{pre}}\text{)}$$

Fig. 3. The Program Logic $\vdash_{\mathrm{bca}}$.

input $a \in A$ such that $[a]_A \, \mathsf{r} \, [b]_A$ is a valid bca triple. By defining

$$\mathrm{wp}_{\mathsf{r}}(b) \triangleq \vee_A \{a \in A \mid \models [a]_A \, \mathsf{r} \, [b]_A\} \tag{6}$$

this boils down to the validity of the triple $[\mathrm{wp}_{\mathsf{r}}(b)]_A \, \mathsf{r} \, [b]_A$. The answer can be easily shown to be negative. In fact, it is enough to consider the interval $b = [1, 1] \in \mathsf{Int}$ for the assignment $x := x * x$. In this case, both triples $[[1, 1]]_{\mathsf{Int}} \, x := x * x \, [[1, 1]]_{\mathsf{Int}}$ and $[[-1, -1]]_{\mathsf{Int}} \, x := x * x \, [[1, 1]]_{\mathsf{Int}}$ are valid, while for their interval lub $[-1, 1] = [-1, -1] \vee_{\mathsf{Int}} [1, 1]$, it turns out that the bca triple $[[-1, 1]]_{\mathsf{Int}} \, x := x * x \, [[1, 1]]_{\mathsf{Int}}$ is not valid.

On the other hand, if the abstract domain A is disjunctive and the concrete semantics of basic commands is assumed to be additive—as it is the case for the while programs in Section 2.4—then the weakest abstract precondition $\mathrm{wp}_{\mathsf{r}}(b)$ exists. This is a consequence of Lemma 2.3:

$$\begin{aligned}
\alpha[\![\mathsf{r}]\!]\gamma(\mathrm{wp}_{\mathsf{r}}(b)) = \quad & \text{[by additivity of } \gamma, \alpha, \text{ and, by Lemma 2.3, of } [\![\mathsf{r}]\!]] \\
\vee_A \{\alpha[\![\mathsf{r}]\!]\gamma(a) \in A \mid \models [a]_A \, \mathsf{r} \, [b]_A\} = \quad & \text{[as } \models [a]_A \, \mathsf{r} \, [b]_A] \\
b = \quad & \text{[as } \models [a]_A \, \mathsf{r} \, [b]_A] \\
\vee_A \{[\![\mathsf{r}]\!]^\sharp_A a \in A \mid \models [a]_A \, \mathsf{r} \, [b]_A\} = \quad & \text{[by Lemma 2.3, } [\![\mathsf{r}]\!]^\sharp_A \text{ is additive]} \\
[\![\mathsf{r}]\!]^\sharp_A (\vee_A \{a \in A \mid \models [a]_A \, \mathsf{r} \, [b]_A\}) = \quad & \text{[by definition (6)]} \\
[\![\mathsf{r}]\!]^\sharp_A \mathrm{wp}_{\mathsf{r}}(b) \, . &
\end{aligned}$$

For example, this is the case of the Sign abstraction in Figure 1, which is disjunctive. □

7 A Program Logic of Best Correct Approximations

Consider a given abstract domain A and assume that the abstract interpretation of all the basic commands is their best correct approximations—otherwise, it would not make sense to prove inductively the bca property. We have shown in Section 3 that bcas are preserved by the nondeterministic choice (cf. Lemma 3.2), whereas sequential compositions, Kleene star and fixpoint do not preserve the bca (cf. Examples 3.3 and 3.5). Thus, if we aim at designing a proof system for proving bca triples $[a]_A \, \mathsf{r} \, [b]_A$, then the crucial rules are concerned with sequential compositions, Kleene star and fixpoint commands, and these rules have to single out specific proof principles for them. Moreover, this proof system may include a consequence-like rule for the precondition a only, since the abstract postcondition b obviously cannot be altered (neither weakened nor strengthened).

Let us remark that since, by Theorem 4.7, if $[\![\cdot]\!]^\sharp_A$ is computable then $\mathrm{BCA}(A, a)$ is not recursive, *there exists no computable sound proof system which is able to prove all the valid bca triples*, due to the equivalence $\exists b.\ \models [a]_A\ \mathsf{r}\ [b]_A \Leftrightarrow \mathsf{r} \in \mathrm{BCA}(A, a)$.

Definition 7.1 (Program Logic $\vdash_{\mathrm{bca}}$). Given an abstract domain $A_{\alpha,\gamma} \in \mathrm{Abs}(C)$, the *bca program logic* $\vdash_{\mathrm{bca}}$, also denoted by $\vdash_{\mathrm{bca}_A}$ to highlight the abstraction A, for deriving triples $[a]_A\ \mathsf{r}\ [b]_A$ for regular programs $\mathsf{r} \in \mathrm{Reg}_{\mathrm{fix}}$ is defined in Figure 3. □

7.1 Discussion

The *inductive* rule for propagating best correct approximations through sequential compositions is a crucial principle of our proof system (more in general, of any inductive proof system for proving bcas).

Lemma 7.2 (Justification of the Rule (seq)**).** *Assume that* $\models [a]_A\ \mathsf{r}_1\ [c]_A$ *and* $\models [c]_A\ \mathsf{r}_2\ [b]_A$. *Then,* $\models [a]_A\ \mathsf{r}_1; \mathsf{r}_2\ [b]_A$ *iff* $b = \alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a)$.

Proof. We have that:

$$\begin{aligned}
&\models [a]_A\ \mathsf{r}_1; \mathsf{r}_2\ [b]_A \Leftrightarrow && \text{[by Definition 6.1]}\\
&[\![\mathsf{r}_2]\!]^\sharp_A[\![\mathsf{r}_1]\!]^\sharp_A a = b\ \&\ b = \alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a) \Leftrightarrow && \text{[by hypothesis, } \models [a]_A\ \mathsf{r}_1\ [c]_A\text{]}\\
&[\![\mathsf{r}_2]\!]^\sharp_A c = b\ \&\ b = \alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a) \Leftrightarrow && \text{[by hypothesis, } \models [c]_A\ \mathsf{r}_2\ [b]_A\text{]}\\
&b = \alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a)\,. &&
\end{aligned}$$

□

In the rule (seq), from the inductive validity of two triples $[a]_A\ \mathsf{r}_1\ [c]_A$ and $[c]_A\ \mathsf{r}_2\ [b]_A$, we want to derive that the triple $[a]_A\ \mathsf{r}_1; \mathsf{r}_2\ [b]_A$ is valid. Lemma 7.2 states that the hypothesis $b = \alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a)$ of the rule (seq) *cannot be avoided* in any inductive logic that aims at inferring that sequential compositions preserve bcas. It is worth remarking that $b = \alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a)$ is equivalent to

$$[\![\mathsf{r}_2]\!]^\sharp_A\alpha\big([\![\mathsf{r}_1]\!]\gamma(a)\big) = \alpha[\![\mathsf{r}_2]\!]\big([\![\mathsf{r}_1]\!]\gamma(a)\big) \tag{7}$$

as $b = [\![\mathsf{r}_2]\!]^\sharp_A c = [\![\mathsf{r}_2]\!]^\sharp_A\alpha[\![\mathsf{r}_1]\!]\gamma(a)$. Then, it turns out that

$$\begin{aligned}
&\alpha[\![\mathsf{r}_1; \mathsf{r}_2]\!]\gamma(a) = b \Leftrightarrow && \text{[by definition (1)]}\\
&\alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a) = b \Leftrightarrow && \text{[as } \models [c]_A\ \mathsf{r}_2\ [b]_A\text{]}\\
&\alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a) = \alpha[\![\mathsf{r}_2]\!]\gamma(c) \Leftrightarrow && \text{[as } \models [a]_A\ \mathsf{r}_1\ [c]_A\text{]}\\
&\alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a) = \alpha[\![\mathsf{r}_2]\!]\gamma(\alpha[\![\mathsf{r}_1]\!]\gamma(a)) \Leftrightarrow && \text{[as } \alpha[\![\mathsf{r}_2]\!]\gamma(\alpha[\![\mathsf{r}_1]\!]\gamma(a)) = [\![\mathsf{r}_2]\!]^\sharp_A(\alpha[\![\mathsf{r}_1]\!]\gamma(a))\text{]}\\
&(7) \text{ holds.} &&
\end{aligned}$$

This condition (7) highlights that in the rule (seq) we are indeed requiring that r_2 is locally α-complete on $[\![\mathsf{r}_1]\!]\gamma(a)$.

Let us draw some further remarks on the program logic $\vdash_{\mathrm{bca}}$.

(a) We can also consider the following rule for sequential composition:

$$\frac{\vdash_{\mathrm{bca}}\ [a]_A\ \mathsf{r}_1\ [c]_A \quad \vdash_{\mathrm{bca}}\ [c]_A\ \mathsf{r}_2\ [b]_A \quad \gamma(c) = [\![\mathsf{r}_1]\!]\gamma(a)}{\vdash_{\mathrm{bca}}\ [a]_A\ \mathsf{r}_1; \mathsf{r}_2\ [b]_A}\ (\mathrm{seq}_\gamma)$$

having as premise the local γ-completeness of r_1 on a, that is, $\gamma(c) = \gamma[\![\mathsf{r}_1]\!]^\sharp_A a = [\![\mathsf{r}_1]\!]\gamma(a)$. This rule is sound because if $\gamma(c) = [\![\mathsf{r}_1]\!]\gamma(a)$ holds then the premise of (seq) is satisfied:

$$\begin{aligned}
&\gamma(c) = [\![\mathsf{r}_1]\!]\gamma(a) \Rightarrow && \text{[by applying } \alpha[\![\mathsf{r}_2]\!] \text{ to both sides]}\\
&\alpha[\![\mathsf{r}_2]\!]\gamma(c) = \alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a) \Leftrightarrow && \text{[as } b = \alpha[\![\mathsf{r}_2]\!]\gamma(c)\text{]}\\
&b = \alpha[\![\mathsf{r}_2]\!][\![\mathsf{r}_1]\!]\gamma(a)\,. &&
\end{aligned}$$

Hence, it turns out that (seq_γ) is a weaker rule than (seq), and we will freely use (seq_γ) in our examples. We will show in Example 8.1 that (seq_γ) is *strictly weaker* than (seq).

(b) While the logic $\vdash_{\mathrm{bca}}$ includes a rule for weakening the abstract precondition a of a triple $[a]_A\ \mathsf{r}\ [b]_A$, as expected no rule for altering (weakening or strengthening) the abstract postcondition can be available, since the validity of $[a]_A\ \mathsf{r}\ [b]_A$ entails an equality to b, so that the abstract postcondition b cannot be altered.

(c) Let us point out that when A coincides with the concrete domain, i.e. $A = id_{\mathrm{Abs}}$, we do not obtain Hoare logic as an instance of the bca logic. In fact, by considering $A = id_{\mathrm{Abs}} = C = \wp(\mathbb{D})$, a bca triple $[X]_{id_{\mathrm{Abs}}}\ \mathsf{r}\ [Y]_{id_{\mathrm{Abs}}}$, with $X, Y \in \wp(\mathbb{D})$, is valid when $[\![\mathsf{r}]\!]X = Y$ holds, while validity of a Hoare logic triple $\{X\}\ \mathsf{r}\ \{Y\}$ means that $[\![\mathsf{r}]\!]X \subseteq Y$ holds.

Remark 7.3 (Infinitary Rule for fix is Unsound). As an alternative to the recursive rule (rec), one could guess the following infinitary rule:

$$\frac{\forall n \in \mathbb{N}.\ \vdash_{\mathrm{bca}} [a_n]_A\ \mathsf{r}\ [a_{n+1}]_A}{\vdash_{\mathrm{bca}} [a_0]_A\ \mathsf{r}^{\mathsf{fix}}\ [\bigvee_{n\in\mathbb{N}} a_n]_A}\ (\mathrm{iterate}_{\mathsf{fix}})$$

A similar rule has been used both in incorrectness logic [O'Hearn 2020] and in local completeness logic [Bruni et al. 2021, 2023] for the Kleene star, namely with $\mathsf{r}^{\mathsf{fix}}$ replaced by $\mathsf{r}^\star$. We recall that incorrectness logic deals with concrete semantics, while local completeness logic, which is parametric on an abstraction A, considers the Kleene star $\mathsf{r}^\star$ but does not include the fixpoint command $\mathsf{r}^{\mathsf{fix}}$, and we showed in Remark 2.2 and Theorem 2.4 that the abstract semantics of $\mathsf{r}^\star$ and $\mathsf{r}^{\mathsf{fix}}$ may differ. It turns out that this infinitary rule $(\mathrm{iterate}_{\mathsf{fix}})$ is *unsound*. This unsoundness can be shown by resorting to the program $\mathsf{r}^{\mathsf{fix}}$ with $\mathsf{r} \triangleq (x = 2; x := x + 3) \oplus x := x - 3$ already used in Remark 2.2. In fact, we can prove that for all intervals $[k, k+1] \in \mathsf{Int}$, for $k \in \mathbb{Z}$, such that $2 \notin [k, k+1]$, we can derive $\vdash_{\mathrm{bca}} [[k, k+1]]_{\mathsf{Int}}\ \mathsf{r}\ [[k-3, k-2]]_{\mathsf{Int}}$ as follows:

$$\dfrac{\dfrac{\dfrac{\text{hypothesis } 2 \notin [k,k+1]}{\vdash_{\mathrm{bca}} [[k,k+1]]_{\mathsf{Int}}\ x = 2\ [\bot_{\mathsf{Int}}]_{\mathsf{Int}}}\ (\text{basic}) \quad \dfrac{}{\vdash_{\mathrm{bca}} [\bot_{\mathsf{Int}}]_{\mathsf{Int}}\ x := x+3\ [\bot_{\mathsf{Int}}]_{\mathsf{Int}}}\ (\text{basic})}{\vdash_{\mathrm{bca}} [[k,k+1]]_{\mathsf{Int}}\ x = 2; x := x+3\ [\bot_{\mathsf{Int}}]_{\mathsf{Int}}}\ (\mathrm{seq}_\gamma) \quad \vdash_{\mathrm{bca}} [[k,k+1]]_{\mathsf{Int}}\ x := x-3\ [[k-3,k-2]]_{\mathsf{Int}}}{\vdash_{\mathrm{bca}} [[k,k+1]]_{\mathsf{Int}}\ \mathsf{r}\ [[k-3,k-2]]_{\mathsf{Int}}}\ (\text{choice})$$

Hence, with abstract input $a_0 = [3, 4] \in \mathsf{Int}$, we can infer the following denumerable sequence:

$$\vdash_{\mathrm{bca}} [[3,4]]_{\mathsf{Int}}\ \mathsf{r}\ [[0,1]]_{\mathsf{Int}} \quad \vdash_{\mathrm{bca}} [[0,1]]_{\mathsf{Int}}\ \mathsf{r}\ [[-3,-2]]_{\mathsf{Int}} \quad \vdash_{\mathrm{bca}} [[-3,-2]]_{\mathsf{Int}}\ \mathsf{r}\ [[-6,-5]]_{\mathsf{Int}} \quad \cdots$$

By applying $(\mathrm{iterate}_{\mathsf{fix}})$, since $[-\infty, 4] = \vee_{\mathsf{Int}}\{[3,4],\ [0,1],\ [-3,-2],\ [-6,-5], \ldots\}$, we would obtain that the triple $[[3,4]]_{\mathsf{Int}}\ \mathsf{r}^{\mathsf{fix}}\ [[-\infty,4]]_{\mathsf{Int}}$ is valid, while, as noticed in Remark 2.2, it turns out that $[\![\mathsf{r}^{\mathsf{fix}}]\!]^\sharp_{\mathsf{Int}}[3,4] = [-\infty, 5]$.

In summary, we have shown that the logical reasoning underpinning the *abstract bca* semantics of loops defined as $\mathsf{r}^{\mathsf{fix}}$ is fundamentally different to the principles for handling loops expressed as $\mathsf{r}^\star$ in incorrectness and local completeness logics (see also the discussion in Section 7.3). □

7.2 Soundness and Incompleteness

It turns out that the program logic $\vdash_{\mathrm{bca}}$ is sound.

Theorem 7.4 (Soundness of $\vdash_{\mathrm{bca}}$). *For all* $\mathsf{r} \in \mathrm{Reg}_{\mathsf{fix}}$, $A \in \mathrm{Abs}(C)$, $a, b \in A$, *if* $\vdash_{\mathrm{bca}} [a]_A\ \mathsf{r}\ [b]_A$ *then* $\models [a]_A\ \mathsf{r}\ [b]_A$.

Proof. The proof proceeds by structural induction on $\mathsf{r} \in \mathrm{Reg}_{\mathsf{fix}}$ and by induction on the derivation tree of $\vdash_{\mathrm{bca}} [a]_A\ \mathsf{r}\ [b]_A$.
(basic). Clear, as $[\![\mathsf{c}]\!]^\sharp_A a = \alpha[\![\mathsf{c}]\!]\gamma(a)$ by definition (2).

(choice).

$$\begin{aligned}
\llbracket \mathrm{r}_1 \oplus \mathrm{r}_2 \rrbracket^\sharp_A a = & \quad \text{[by definition (2)]}\\
\llbracket \mathrm{r}_1 \rrbracket^\sharp_A a \vee_A \llbracket \mathrm{r}_2 \rrbracket^\sharp_A a = b_1 \vee_A b_2 = & \quad \text{[by tree induction]}\\
\alpha\llbracket \mathrm{r}_1 \rrbracket\gamma(a) \vee_A \alpha\llbracket \mathrm{r}_2 \rrbracket\gamma(a) = & \quad \text{[by additivity of } \alpha\text{]}\\
\alpha(\llbracket \mathrm{r}_1 \rrbracket\gamma(a) \vee_C \llbracket \mathrm{r}_2 \rrbracket\gamma(a)) = \alpha\llbracket \mathrm{r}_1 \oplus \mathrm{r}_2 \rrbracket\gamma(a)\,. & \quad \text{[by definition (1)]}
\end{aligned}$$

(seq).

$$\begin{aligned}
\llbracket \mathrm{r}_1; \mathrm{r}_2 \rrbracket^\sharp_A a = & \quad \text{[by definition (2)]}\\
\llbracket \mathrm{r}_2 \rrbracket^\sharp_A \llbracket \mathrm{r}_1 \rrbracket^\sharp_A a = \llbracket \mathrm{r}_2 \rrbracket^\sharp_A c = b = & \quad \text{[by tree induction]}\\
\alpha\llbracket \mathrm{r}_2 \rrbracket\gamma(c) = & \quad \text{[by tree induction]}\\
\alpha\llbracket \mathrm{r}_2 \rrbracket\gamma(\alpha\llbracket \mathrm{r}_1 \rrbracket\gamma(a)) = & \quad \text{[by premise's rule]}\\
\alpha\llbracket \mathrm{r}_2 \rrbracket\llbracket \mathrm{r}_1 \rrbracket\gamma(a) = \alpha\llbracket \mathrm{r}_1; \mathrm{r}_2 \rrbracket\gamma(a)\,. & \quad \text{[by definition (1)]}
\end{aligned}$$

(abs-inv). (i) We have that $a \vee_A \llbracket \mathrm{r} \rrbracket^\sharp_A a = a \vee_A b = a$, i.e., a is a fixpoint of $\lambda x \in A.\, a \vee_A \llbracket \mathrm{r} \rrbracket^\sharp_A x$. Then, if $z \in A$ is a fixpoint, i.e., $a \vee_A \llbracket \mathrm{r} \rrbracket^\sharp_A z = z$, then $a \leq_A z$, so that $a = \mathrm{lfp}(\lambda x \in A.\, a \vee_A \llbracket \mathrm{r} \rrbracket^\sharp_A x) = \llbracket \mathrm{r}^{\mathrm{fix}} \rrbracket^\sharp_A a$ holds. (ii) By tree induction, we have that $\alpha\llbracket \mathrm{r} \rrbracket\gamma(a) = b \leq_A a$, so that, by Galois connection, $\llbracket \mathrm{r} \rrbracket\gamma(a) \leq_C \gamma(a)$, and, in turn, $\gamma(a) \vee_C \llbracket \mathrm{r} \rrbracket\gamma(a) = \gamma(a)$, i.e., $\gamma(a)$ is a fixpoint of $\lambda x \in C.\, \gamma(a) \vee_C \llbracket \mathrm{r} \rrbracket x$. Then, if $z \in C$ is a fixpoint, i.e., $z = \gamma(a) \vee_C \llbracket \mathrm{r} \rrbracket z$, then $\gamma(a) \leq z$, so that $\gamma(a) = \mathrm{lfp}(\lambda x \in C.\, \gamma(a) \vee_C \llbracket \mathrm{r} \rrbracket x)$. As a consequence, $a = \alpha\gamma(a) = \alpha(\mathrm{lfp}(\lambda x \in C.\, \gamma(a) \vee_C \llbracket \mathrm{r} \rrbracket x)) = \alpha\llbracket \mathrm{r}^{\mathrm{fix}} \rrbracket\gamma(a)$.

(rec). (i) First, we have that $\mathrm{lfp}(\lambda x \in A.\, a \vee_A \llbracket \mathrm{r} \rrbracket^\sharp_A x) \leq_A \mathrm{lfp}(\lambda x \in A.\, a \vee_A b \vee_A \llbracket \mathrm{r} \rrbracket^\sharp_A x) = a \vee_A b$. On the other hand, for all $z \in A$, if $a \vee_A \llbracket \mathrm{r} \rrbracket^\sharp_A z = z$, i.e. z is a fixpoint, then $a \leq_A z$ and $\llbracket \mathrm{r} \rrbracket^\sharp_A z \leq_A z$, so that $\llbracket \mathrm{r}^n \rrbracket^\sharp_A z = (\llbracket \mathrm{r} \rrbracket^\sharp_A)^n z \leq_A z$ holds, and, in turn, $b = \llbracket \mathrm{r}^n \rrbracket^\sharp_A a \leq_A \llbracket \mathrm{r}^n \rrbracket^\sharp_A z \leq_A z$. Hence, $a \vee_A b \leq_A z$ holds. As a consequence, $\llbracket \mathrm{r}^{\mathrm{fix}} \rrbracket^\sharp_A a = \mathrm{lfp}(\lambda x \in A.\, a \vee_A \llbracket \mathrm{r} \rrbracket^\sharp_A x) = a \vee_A b$. (ii) We have that $\mathrm{lfp}(\lambda x \in C.\, \gamma(a) \vee_C \llbracket \mathrm{r} \rrbracket x) \leq_C \mathrm{lfp}(\lambda x \in C.\, \gamma(a \vee_A b) \vee_C \llbracket \mathrm{r} \rrbracket x)$, hence $\alpha(\mathrm{lfp}(\lambda x \in C.\, \gamma(a) \vee_C \llbracket \mathrm{r} \rrbracket x)) \leq_A \alpha(\mathrm{lfp}(\lambda x \in C.\, \gamma(a \vee_A b) \vee_C \llbracket \mathrm{r} \rrbracket x)) = a \vee_A b$. For the converse inequality, let $C \ni g \triangleq \mathrm{lfp}(\lambda x \in C.\, \gamma(a) \vee_C \llbracket \mathrm{r} \rrbracket x))$, so that $g = \gamma(a) \vee_C \llbracket \mathrm{r} \rrbracket g$, and, in turn, $\gamma(a) \leq g$ and $\llbracket \mathrm{r} \rrbracket g \leq g$. Thus, we obtain that for all $k \in \mathbb{N}$, $\llbracket \mathrm{r} \rrbracket^k \gamma(a) \leq g$. Hence, from $\gamma(a) \leq g$, by GC, we obtain $a = \alpha\gamma(a) \leq \alpha(g)$, and from $\llbracket \mathrm{r} \rrbracket^n \gamma(a) \leq g$ we derive, by GC and by $\models [a]_A\, \mathrm{r}^n\, [b]_A$, that $b = \alpha\llbracket \mathrm{r}^n \rrbracket\gamma(a) = \alpha\llbracket \mathrm{r} \rrbracket^n \gamma(a) \leq_A \alpha(g)$. As a consequence, $a \vee_A b \leq_A \alpha(g) = \alpha(\mathrm{lfp}(\lambda x \in C.\, \gamma(a) \vee_C \llbracket \mathrm{r} \rrbracket x)) = \alpha\llbracket \mathrm{r}^{\mathrm{fix}} \rrbracket\gamma(a)$.

($\mathsf{weaken}_{\mathrm{pre}}$).

$$\begin{aligned}
\llbracket \mathrm{r} \rrbracket^\sharp_A a \geq_A & \quad \text{[by } a' \leq_A a \text{ and monotonicity]}\\
\llbracket \mathrm{r} \rrbracket^\sharp_A a' = & \quad \text{[by tree induction]}\\
b \geq & \quad \text{[by } \llbracket \mathrm{r} \rrbracket^\sharp_A a \leq_A b\text{]}\\
\llbracket \mathrm{r} \rrbracket^\sharp_A a \geq_A & \quad \text{[by soundness of } \llbracket \mathrm{r} \rrbracket^\sharp_A\text{]}\\
\alpha\llbracket \mathrm{r} \rrbracket\gamma(a) \geq_A & \quad \text{[by } a' \leq_A a \text{ and monotonicity]}\\
\alpha\llbracket \mathrm{r} \rrbracket\gamma(a') = & \quad \text{[by tree induction]}\\
b \geq_A \llbracket \mathrm{r} \rrbracket^\sharp_A a & \quad \text{[by } \llbracket \mathrm{r} \rrbracket^\sharp_A a \leq_A b\text{]}
\end{aligned}$$

thus showing that $\llbracket \mathrm{r} \rrbracket^\sharp_A a = b = \alpha\llbracket \mathrm{r} \rrbracket\gamma(a)$. □

On the other hand, the proof system $\vdash_{\mathrm{bca}}$ is logically incomplete, i.e., the converse of the soundness Theorem 7.4 does not hold.

$$\frac{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r} \, [b]_A \quad b \leq_A a}{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r}^\star \, [a]_A} \ (\text{abs-inv}_\star) \qquad \frac{\exists n \geq 1.\ \vdash_{\mathrm{bca}} [a]_A \, \mathsf{r}^n \, [b]_A \qquad \vdash_{\mathrm{bca}} [a \vee_A b]_A \, \mathsf{r}^\star \, [a \vee_A b]_A}{\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r}^\star \, [a \vee_A b]_A} \ (\text{rec}_\star)$$

Fig. 4. The bca Rules for Kleene Star.

Theorem 7.5 (Incompleteness of $\vdash_{\mathrm{bca}}$). *There exist* $\mathsf{r} \in \mathrm{Reg}_{\mathsf{fix}}$, $A \in \mathrm{Abs}(C)$ *and* $a, b \in A$ *such that* $\models [a]_A \, \mathsf{r} \, [b]_A$ *but* $\nvdash_{\mathrm{bca}} [a]_A \, \mathsf{r} \, [b]_A$.

Proof. Let $\mathsf{r}_1 \triangleq x := x + 1; x := x - 1$, $\mathsf{r}_2 \triangleq x := x + 1$, and consider the sign abstraction Sign. We have that $[\mathbb{Z}_{>0}]_{\mathrm{Sign}} \, \mathsf{r}_1; \mathsf{r}_2 \, [\mathbb{Z}_{>0}]_{\mathrm{Sign}}$ is a valid bca triple, as

$$\begin{aligned}
[\![\mathsf{r}_1; \mathsf{r}_2]\!]^\sharp_{\mathrm{Sign}} \mathbb{Z}_{>0} &= [\![x := x + 1]\!]^\sharp_{\mathrm{Sign}} [\![x := x - 1]\!]^\sharp_{\mathrm{Sign}} [\![x := x + 1]\!]^\sharp_{\mathrm{Sign}} \mathbb{Z}_{>0} \\
&= [\![x := x + 1]\!]^\sharp_{\mathrm{Sign}} [\![x := x - 1]\!]^\sharp_{\mathrm{Sign}} \mathbb{Z}_{>0} = [\![x := x + 1]\!]^\sharp_{\mathrm{Sign}} \mathbb{Z}_{\geq 0} \\
&= \mathbb{Z}_{>0} = \alpha_{\mathrm{Sign}}(\mathbb{Z}_{>1}) = \alpha_{\mathrm{Sign}} [\![\mathsf{r}_1; \mathsf{r}_2]\!] \gamma_{\mathrm{Sign}}(\mathbb{Z}_{>0}) \, .
\end{aligned}$$

However, this triple $[\mathbb{Z}_{>0}]_{\mathrm{Sign}} \, \mathsf{r}_1; \mathsf{r}_2 \, [\mathbb{Z}_{>0}]_{\mathrm{Sign}}$ cannot be inferred in $\vdash_{\mathrm{bca}}$. This is because an attempt to derive $[\mathbb{Z}_{>0}]_{\mathrm{Sign}} \, \mathsf{r}_1; \mathsf{r}_2 \, [\mathbb{Z}_{>0}]_{\mathrm{Sign}}$ would need to derive the triples $[\mathbb{Z}_{>0}]_{\mathrm{Sign}} \, \mathsf{r}_1 \, [c]_{\mathrm{Sign}}$ and $[c]_{\mathrm{Sign}} \, \mathsf{r}_2 \, [\mathbb{Z}_{>0}]_{\mathrm{Sign}}$, for some $c \in \mathrm{Sign}$, and then to apply (seq), if its noninductive premise holds. Moreover, the triple $[\mathbb{Z}_{>0}]_{\mathrm{Sign}} \, \mathsf{r}_1 \, [c]_{\mathrm{Sign}}$ should be valid, thus entailing that $c = \alpha_{\mathrm{Sign}} [\![\mathsf{r}_1]\!] \gamma_{\mathrm{Sign}}(\mathbb{Z}_{>0}) = \alpha_{\mathrm{Sign}}(\mathbb{Z}_{>0}) = \mathbb{Z}_{>0}$. However, the triple $[\mathbb{Z}_{>0}]_{\mathrm{Sign}} \, \mathsf{r}_1 \, [\mathbb{Z}_{>0}]_{\mathrm{Sign}}$ cannot be valid as

$$[\![\mathsf{r}]\!]^\sharp_{\mathrm{Sign}} \mathbb{Z}_{>0} = [\![x := x - 1]\!]^\sharp_{\mathrm{Sign}} [\![x := x + 1]\!]^\sharp_{\mathrm{Sign}} \mathbb{Z}_{>0} = [\![x := x - 1]\!]^\sharp_{\mathrm{Sign}} \mathbb{Z}_{>0} = \mathbb{Z}_{\geq 0} \, .$$

Hence, $[\mathbb{Z}_{>0}]_{\mathrm{Sign}} \, \mathsf{r}_1 \, [\mathbb{Z}_{>0}]_{\mathrm{Sign}}$ cannot be inferred, thus proving that $\nvdash_{\mathrm{bca}} [a]_A \, \mathsf{r} \, [b]_A$. Let us also remark that the rule $(\mathrm{weaken}_{\mathrm{pre}})$ cannot help, because we cannot modify the input precondition $\mathbb{Z}_{>0}$.
On the other hand, it is worth observing that if we change the input precondition to $\mathbb{Z}_{\geq 0}$ then we could easily derive the valid triple $[\mathbb{Z}_{\geq 0}]_{\mathrm{Sign}} \, \mathsf{r} \, [\mathbb{Z}_{\geq 0}]_{\mathrm{Sign}}$. □

7.3 Rules for Kleene Star

The two bca rules for $\mathsf{r}^{\mathsf{fix}}$ can be also used for the bca reasoning on $\mathsf{r}^\star$: the corresponding rules are given in Figure 4 and turn out to be sound.

Theorem 7.6 (Soundness of Kleene Star Rules). *The program logic* $\vdash_{\mathrm{bca}}$ *augmented with the rules in Figure 4 is sound. Thus, for all* $\mathsf{r} \in \mathrm{Reg}$, $A \in \mathrm{Abs}(C)$, $a, b \in A$, *if* $\vdash_{\mathrm{bca}} [a]_A \, \mathsf{r} \, [b]_A$ *then* $\models [a]_A \, \mathsf{r} \, [b]_A$.

Proof. $(\text{abs-inv}_\star)$. We have that $[\![\mathsf{r}]\!]^{\sharp\, 0}_A a = a$, and, by an easy induction, for all $k \geq 1$, $[\![\mathsf{r}]\!]^{\sharp\, k}_A a \leq_A a$. Thus, $[\![\mathsf{r}^\star]\!]^\sharp_A a = \bigvee_A \{ [\![\mathsf{r}]\!]^{\sharp\, k}_A a \mid k \in \mathbb{N} \} = a$ holds. Moreover, we show that for all $k \geq 1$, $\alpha [\![\mathsf{r}]\!]^k \gamma(a) \leq_A a$. In fact, $\alpha [\![\mathsf{r}]\!]^1 \gamma(a) = \alpha [\![\mathsf{r}]\!] \gamma(a) = b \leq_A a$ as $\models [a]_A \, \mathsf{r} \, [b]_A$. Then,

$$\begin{aligned}
\alpha [\![\mathsf{r}]\!]^{k+1} \gamma(a) = \alpha [\![\mathsf{r}]\!] [\![\mathsf{r}]\!]^k \gamma(a) &\leq_A \quad [\text{as } id \mathrel{\dot{\leq}} \gamma\alpha] \\
\alpha [\![\mathsf{r}]\!] \gamma\alpha [\![\mathsf{r}]\!]^k \gamma(a) &\leq_A \quad [\text{by induction}] \\
\alpha [\![\mathsf{r}]\!] \gamma(b) &\leq_A \quad [\text{as } b \leq_A a] \\
\alpha [\![\mathsf{r}]\!] \gamma(a) &= \quad [\text{as } \models [a]_A \, \mathsf{r} \, [b]_A] \\
b &\leq_A a \quad [\text{as } b \leq_A a]
\end{aligned}$$

Hence,

$$\begin{aligned}
\alpha(\llbracket \mathsf{r}^\star \rrbracket \gamma(a)) = &\quad \text{[by definition (1)]}\\
\alpha(\bigvee\{\llbracket \mathsf{r} \rrbracket^k \gamma(a) \mid k \in \mathbb{N}\}) = &\quad \text{[by additivity of } \alpha\text{]}\\
\alpha\gamma(a) \vee \bigvee\{\alpha\llbracket \mathsf{r} \rrbracket^k \gamma(a) \mid k \geq 1\} = &\quad \text{[by } \alpha\gamma = id\text{]}\\
a \vee \bigvee\{\alpha\llbracket \mathsf{r} \rrbracket^k \gamma(a) \mid k \geq 1\} = a &\quad \text{[by the observation above]}
\end{aligned}$$

$(\mathsf{rec}_\star)$. First, we have that $\llbracket \mathsf{r}^\star \rrbracket^\sharp_A a = \bigvee_A\{\llbracket \mathsf{r} \rrbracket^{\sharp\, k}_A a \mid k \in \mathbb{N}\} \leq_A \bigvee_A\{\llbracket \mathsf{r} \rrbracket^{\sharp\, k}_A a \vee_A b \mid k \in \mathbb{N}\} = a \vee_A b$. Conversely, since $\llbracket \mathsf{r} \rrbracket^{\sharp\, 0}_A a = a$ and $\llbracket \mathsf{r} \rrbracket^{\sharp\, n}_A a = \llbracket \mathsf{r}^n \rrbracket^\sharp_A a = b$, we have that $\bigvee_A\{\llbracket \mathsf{r} \rrbracket^{\sharp\, k}_A a \mid k \in \mathbb{N}\} \geq a \vee_A b$, so that $\llbracket \mathsf{r}^\star \rrbracket^\sharp_A a = a \vee_A b$ follows.
Furthermore, we also have that $\alpha\llbracket \mathsf{r} \rrbracket^0 \gamma(a) = \alpha\gamma(a) = a$, and, $\alpha\llbracket \mathsf{r} \rrbracket^n \gamma(a) = \alpha\llbracket \mathsf{r}^n \rrbracket\gamma(a) = b$, so that $\alpha(\bigvee\{\llbracket \mathsf{r} \rrbracket^k \gamma(a) \mid k \in \mathbb{N}\}) = \bigvee_A\{\alpha\llbracket \mathsf{r} \rrbracket^k \gamma(a) \mid k \in \mathbb{N}\} \geq a \vee_A b$ holds. Also,

$$\begin{aligned}
\alpha(\llbracket \mathsf{r}^\star \rrbracket \gamma(a)) \leq_A &\quad \text{[as } a \leq_A a \vee_A b\text{]}\\
\alpha(\llbracket \mathsf{r}^\star \rrbracket \gamma(a \vee_A b)) = a \vee_A b &\quad \text{[as } \models [a \vee_A b]_A\, \mathsf{r}^\star\, [a \vee_A b]_A\text{]}
\end{aligned}$$

Hence, $\alpha(\llbracket \mathsf{r}^\star \rrbracket \gamma(a)) = \alpha(\bigvee_C\{\llbracket \mathsf{r} \rrbracket^k \gamma(a) \mid k \in \mathbb{N}\}) = a \vee_A b$. □

Analogously to the discussion in Remark 7.3, it turns out that the following infinitary rule $(\mathsf{iterate}_\star)$, which could be guessed as an alternative to the rule $(\mathsf{rec}_\star)$, is unsound:

$$\frac{\forall n \in \mathbb{N}.\ \vdash_{\mathsf{bca}} [a_n]_A\, \mathsf{r}\, [a_{n+1}]_A}{\vdash_{\mathsf{bca}} [a_0]_A\, \mathsf{r}^\star\, [\bigvee_{n\in\mathbb{N}} a_n]_A}\ (\mathsf{iterate}_\star)$$

Exactly this same rule has been used both in incorrectness logic [O'Hearn 2020] for proving its logical completeness and has been proved to be sound also in local completeness logic [Bruni et al. 2021, 2023]. We prove here the unsoundness of $(\mathsf{iterate}_\star)$ through the following program:

$$\mathsf{r} \triangleq x > 0; (x := x - 3 \oplus x := x - 1)$$

on the interval abstraction Int. In fact, by defining $a_0 \triangleq [3, 3] \in \mathsf{Int}$, we have that:

$$\vdash_{\mathsf{bca}} [[3,3]]_{\mathsf{Int}}\, \mathsf{r}\, [[0,2]]_{\mathsf{Int}} \qquad \vdash_{\mathsf{bca}} [[0,2]]_{\mathsf{Int}}\, \mathsf{r}\, [[-2,1]]_{\mathsf{Int}} \qquad \vdash_{\mathsf{bca}} [[-2,1]]_{\mathsf{Int}}\, \mathsf{r}\, [[-2,0]]_{\mathsf{Int}}$$
$$\vdash_{\mathsf{bca}} [[-2,0]]_{\mathsf{Int}}\, \mathsf{r}\, [\bot_{\mathsf{Int}}]_{\mathsf{Int}} \qquad \vdash_{\mathsf{bca}} [\bot_{\mathsf{Int}}]_{\mathsf{Int}}\, \mathsf{r}\, [\bot_{\mathsf{Int}}]_{\mathsf{Int}}$$

These bca triples can all be derived by leveraging (seq_γ), because the test $x > 0$ is globally γ-complete, i.e., for any interval $I \in \mathsf{Int}$, $\llbracket x > 0 \rrbracket\gamma(I) = \gamma([1, +\infty) \wedge_{\mathsf{Int}} I) \in \gamma(\mathsf{Int})$. Hence, by applying the rule $(\mathsf{iterate}_\star)$, we would derive the triple $[[3,3]]_{\mathsf{Int}}\, \mathsf{r}^\star\, [[-2,3]]_{\mathsf{Int}}$, which is not valid. In fact, $\alpha_{\mathsf{Int}}\llbracket \mathsf{r}^\star \rrbracket\gamma_{\mathsf{Int}}([3,3]) = \alpha_{\mathsf{Int}}(\cup\{\{3\}, \{0,2\}, \{-1,1\}, \varnothing\}) = \alpha_{\mathsf{Int}}(\{-1,0,1,2,3\}) = [-1,3]$, meaning that $[-1,3]$ is the best possible postcondition, and this is not the interval postcondition $[-2,3]$.

8 Illustrative Examples using the Logic $\vdash_{\mathsf{bca}}$

We provide some simple examples to illustrate how to use the logic $\vdash_{\mathsf{bca}}$ to prove bca triples.

Example 8.1 (Sequential Composition). Consider the program $\mathsf{p}_1 \triangleq x := x + 2;\ x := x - 1$ and the Sign abstract domain. The analysis of p_1 with abstract input $\mathbb{Z}_{\leq 0}$ is as follows:

$$\llbracket \mathsf{p}_1 \rrbracket^\sharp_{\mathsf{Sign}} \mathbb{Z}_{\leq 0} = \llbracket x := x - 1 \rrbracket^\sharp_{\mathsf{Sign}} \llbracket x := x + 2 \rrbracket^\sharp_{\mathsf{Sign}} \mathbb{Z}_{\leq 0} = \llbracket x := x - 1 \rrbracket^\sharp_{\mathsf{Sign}} \mathbb{Z} = \mathbb{Z}\,.$$

This Sign analysis is the bca, as $\alpha_{\mathsf{Sign}}\llbracket \mathsf{p}_1 \rrbracket\gamma_{\mathsf{Sign}}(\mathbb{Z}_{\leq 0}) = \alpha_{\mathsf{Sign}}(\mathbb{Z}_{\leq 1}) = \mathbb{Z}$. We can easily derive the triple $\vdash_{\mathsf{bca}} [\mathbb{Z}_{\leq 0}]_{\mathsf{Sign}}\, \mathsf{p}_1\, [\mathbb{Z}]_{\mathsf{Sign}}$ as follows:

$$\frac{\dfrac{}{\vdash_{\mathsf{bca}} [\mathbb{Z}_{\leq 0}]_{\mathsf{Sign}}\, x := x + 2\, [\mathbb{Z}]_{\mathsf{Sign}}}\ (\mathsf{basic}) \qquad \dfrac{}{\vdash_{\mathsf{bca}} [\mathbb{Z}]_{\mathsf{Sign}}\, x := x - 1\, [\mathbb{Z}]_{\mathsf{Sign}}}\ (\mathsf{basic}) \qquad \mathbb{Z} = \alpha_{\mathsf{Sign}}\llbracket x := x - 1 \rrbracket\llbracket x := x + 2 \rrbracket\gamma_{\mathsf{Sign}}(\mathbb{Z}_{\leq 0})}{\vdash_{\mathsf{bca}} [\mathbb{Z}_{\leq 0}]_{\mathsf{Sign}}\, \mathsf{p}_1\, [\mathbb{Z}]_{\mathsf{Sign}}}\ (\mathsf{seq})$$

Notice that we applied the (seq) rule by leveraging the local α-completeness of $x := x - 1$ on $[\![x := x + 2]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}_{\leq 0}) = \mathbb{Z}_{\leq 2}$. Here, (seq_γ) cannot be applied as $x := x + 2$ is not locally γ-complete on $\mathbb{Z}_{\leq 0}$: in fact, $\gamma_{\mathsf{Sign}}[\![x := x + 2]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{\leq 0} = \mathbb{Z} \neq \mathbb{Z}_{\leq 2} = [\![x := x + 2]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}_{\leq 0})$.

Then, consider the input property $\mathbb{Z}_{\geq 0} \in \mathsf{Sign}$, so that the analysis is as follows:

$$[\![\mathsf{p}_1]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{\geq 0} = [\![x := x - 1]\!]^\sharp_{\mathsf{Sign}}[\![x := x + 2]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{\geq 0} = [\![x := x - 1]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{>0} = \mathbb{Z}_{\geq 0}\,.$$

In this case, this analysis is not the bca, as $\alpha_{\mathsf{Sign}}[\![\mathsf{p}_1]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}_{\geq 0}) = \alpha_{\mathsf{Sign}}(\mathbb{Z}_{>0}) = \mathbb{Z}_{>0}$, so that both triples $[\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}\ \mathsf{p}_1\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}$ and $[\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}\ \mathsf{p}_1\ [\mathbb{Z}_{>0}]_{\mathsf{Sign}}$ are not valid.

Consider now the program $\mathsf{p}_2 \triangleq x := x + 1;\ x := x - 2$, whose Sign analysis with input $\mathbb{Z}_{\geq 0}$ is:

$$[\![\mathsf{p}_2]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{\geq 0} = [\![x := x - 2]\!]^\sharp_{\mathsf{Sign}}[\![x := x + 1]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{\geq 0} = [\![x := x - 2]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{>0} = \mathbb{Z}\,,$$

and this is the bca as $\alpha_{\mathsf{Sign}}[\![\mathsf{p}_2]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}_{\geq 0}) = \alpha_{\mathsf{Sign}}(\mathbb{Z}_{\geq -1}) = \mathbb{Z}$. Here, the (seq_γ) rule can be applied as the command $x := x + 1$ is locally γ-complete on $\mathbb{Z}_{\geq 0}$ as $\gamma_{\mathsf{Sign}}(\mathbb{Z}_{>0}) = \mathbb{Z}_{>0} = [\![x := x + 1]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}_{\geq 0})$. Thus, we can prove the bca triple $[\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}\ \mathsf{p}_2\ [\mathbb{Z}]_{\mathsf{Sign}}$ as follows:

$$\dfrac{\dfrac{}{\vdash_{\mathsf{bca}}\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}\ x := x + 1\ [\mathbb{Z}_{>0}]_{\mathsf{Sign}}}\ (\mathsf{basic}) \quad \dfrac{}{\vdash_{\mathsf{bca}}\ [\mathbb{Z}_{>0}]_{\mathsf{Sign}}\ x := x - 2\ [\mathbb{Z}]_{\mathsf{Sign}}}\ (\mathsf{basic}) \quad \gamma_{\mathsf{Sign}}(\mathbb{Z}_{>0}) = [\![x := x + 1]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}_{\geq 0})}{\vdash_{\mathsf{bca}}\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}\ \mathsf{p}_2\ [\mathbb{Z}]_{\mathsf{Sign}}}\ (\mathsf{seq}_\gamma)$$

□

Example 8.2 (Loop). Consider the analysis on Sign of the following program

$$\mathsf{p}_3 \triangleq (x > 0?;\ x := x - 1)^{\mathsf{fix}}; x \leq 0?$$

corresponding to the while program **while** $x > 0$ **do** $x := x - 1$.

Let $B \triangleq (x > 0?;\ x := x - 1)^{\mathsf{fix}}$. The Sign analysis of p_3 with input $\mathbb{Z}_{>0} \in \mathsf{Sign}$ is:

$$[\![\mathsf{p}_3]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{>0} = [\![x \leq 0?]\!]^\sharp_{\mathsf{Sign}}(\mathsf{lfp}(\lambda x \in \mathsf{Sign}.\, \mathbb{Z}_{>0} \vee_{\mathsf{Sign}} [\![B]\!]^\sharp_{\mathsf{Sign}}x)) = [\![x \leq 0?]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{\geq 0} = \mathbb{Z}_{=0}\,.$$

The bca triple $[\mathbb{Z}_{>0}]_{\mathsf{Sign}}\ \mathsf{p}_3\ [\mathbb{Z}_{=0}]_{\mathsf{Sign}}$ is valid as $\mathbb{Z}_{=0}$ is the bca, i.e., $\alpha_{\mathsf{Sign}}[\![\mathsf{p}_3]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}_{>0}) = \alpha_{\mathsf{Sign}}(\mathbb{Z}_{=0}) = \mathbb{Z}_{=0}$. We can prove this triple in $\vdash_{\mathsf{bca}}$ as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdash_{\mathsf{bca}}\ [\mathbb{Z}_{>0}]_{\mathsf{Sign}}\ x > 0?\ [\mathbb{Z}_{>0}]_{\mathsf{Sign}}}\ (\mathsf{basic}) \quad \dfrac{}{\vdash_{\mathsf{bca}}\ [\mathbb{Z}_{>0}]_{\mathsf{Sign}}\ x := x - 1\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}}\ (\mathsf{basic}) \quad \gamma_{\mathsf{Sign}}(\mathbb{Z}_{>0}) = [\![x > 0?]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}_{>0})}{\vdash_{\mathsf{bca}}\ [\mathbb{Z}_{>0}]_{\mathsf{Sign}}\ B\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}}\ (\mathsf{seq}_\gamma) \quad \mathbb{Z}_{>0} \leq_{\mathsf{Sign}} \mathbb{Z}_{\geq 0} \quad [\![B]\!]^\sharp_{\mathsf{Sign}}\mathbb{Z}_{\geq 0} \leq \mathbb{Z}_{\geq 0}}{\vdash_{\mathsf{bca}}\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}\ B\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}}\ (\mathsf{weaken}_{\mathsf{pre}})}{(\ddagger)}$$

$$\dfrac{\dfrac{(\ddagger)}{\vdash_{\mathsf{bca}}\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}\ B^{\mathsf{fix}}\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}}\ (\mathsf{abs\text{-}inv}) \quad \dfrac{}{\vdash_{\mathsf{bca}}\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}\ x \leq 0?\ [\mathbb{Z}_{=0}]_{\mathsf{Sign}}}\ (\mathsf{basic}) \quad \gamma_{\mathsf{Sign}}(\mathbb{Z}_{\geq 0}) = [\![B^{\mathsf{fix}}]\!]\gamma_{\mathsf{Sign}}(\mathbb{Z}_{\geq 0})}{\vdash_{\mathsf{bca}}\ [\mathbb{Z}_{\geq 0}]_{\mathsf{Sign}}\ \mathsf{p}_3\ [\mathbb{Z}_{=0}]_{\mathsf{Sign}}}\ (\mathsf{seq}_\gamma)$$

□

Example 8.3 (Loop). Finally, let us consider an interval analysis for the program

$$\mathsf{p}_4 \triangleq (x < 0?;\ x := x + 2)^{\mathsf{fix}}; x \geq 0?$$

corresponding to the while program **while** $x < 0$ **do** $x := x + 2$.

Let $B \triangleq (x < 0?;\ x := x + 2)^{\mathsf{fix}}$. The interval analysis of p_4 with input $[-4, -4] \in \mathsf{Int}$ is:

$$[\![\mathsf{p}_4]\!]^\sharp_{\mathsf{Int}}[-4, -4] = [\![x \geq 0?]\!]^\sharp_{\mathsf{Int}}(\mathsf{lfp}(\lambda x \in \mathsf{Int}.\, [-4, -4] \vee_{\mathsf{Int}} [\![B]\!]^\sharp_{\mathsf{Int}}x)) = [\![x \geq 0?]\!]^\sharp_{\mathsf{Int}}[-4, 1] = [0, 1]\,.$$

This Int analysis is not the bca as $\alpha_{\mathsf{Int}}[\![\mathsf{p}_4]\!]\gamma_{\mathsf{Int}}([-4, -4]) = \alpha_{\mathsf{Sign}}(\{0\}) = [0, 0]$. Hence, both bca triples $[[-4, -4]]_{\mathsf{Int}}\ \mathsf{p}_4\ [[0, 1]]_{\mathsf{Int}}$ and $[[-4, -4]]_{\mathsf{Int}}\ \mathsf{p}_4\ [[0, 0]]_{\mathsf{Int}}$ are not valid.

Consider now the input property $[-4, -3] \in \mathsf{Int}$, whose corresponding Int analysis is:

$$[\![\mathsf{p}_4]\!]^\sharp_{\mathsf{Int}}[-4, -3] = [\![x \geq 0?]\!]^\sharp_{\mathsf{Int}}(\mathsf{lfp}(\lambda x \in \mathsf{Int}.\, [-4, -3] \vee_{\mathsf{Int}} [\![B]\!]^\sharp_{\mathsf{Int}}x)) = [\![x \geq 0?]\!]^\sharp_{\mathsf{Int}}[-4, 1] = [0, 1]\,.$$

In this case, this is the best possible analysis, because $\alpha_{\mathsf{Int}}[\![\mathsf{p}_4]\!]\gamma_{\mathsf{Int}}([-4, -3]) = \alpha_{\mathsf{Sign}}(\{0, 1\}) = [0, 1]$. We can derive the bca triple $[[-4, -3]]_{\mathsf{Sign}}\ \mathsf{p}_4\ [[0, 1]]_{\mathsf{Sign}}$ as follows:

$$\dfrac{\dfrac{}{\vdash_{\mathrm{bca}} [[-4,-3]]_{\mathsf{Int}}\, x < 0?\, [[-4,-3]]_{\mathsf{Int}}}\,(\mathrm{basic}) \quad \dfrac{}{\vdash_{\mathrm{bca}} [[-4,-3]]_{\mathsf{Int}}\, x := x+2\, [[-2,-1]]_{\mathsf{Int}}}\,(\mathrm{basic}) \quad \gamma_{\mathsf{Int}}([-4,-3]) = [\![x < 0?]\!]\gamma_{\mathsf{Sign}}([-4,-3])}{\dfrac{\vdash_{\mathrm{bca}} [[-4,-3]]_{\mathsf{Int}}\, B\, [[-2,-1]]_{\mathsf{Int}}}{\text{(a)}}}\,(\mathrm{seq}_\gamma)$$

$$\dfrac{\dfrac{}{\vdash_{\mathrm{bca}} [[-2,-1]]_{\mathsf{Int}}\, x < 0?\, [[-2,-1]]_{\mathsf{Int}}}\,(\mathrm{basic}) \quad \dfrac{}{\vdash_{\mathrm{bca}} [[-2,-1]]_{\mathsf{Int}}\, x := x+2\, [[0,1]]_{\mathsf{Int}}}\,(\mathrm{basic}) \quad \gamma_{\mathsf{Int}}([-2,-1]) = [\![x < 0?]\!]\gamma_{\mathsf{Sign}}([-2,-1])}{\dfrac{\vdash_{\mathrm{bca}} [[-2,-1]]_{\mathsf{Int}}\, B\, [[0,1]]_{\mathsf{Int}}}{\text{(b)}}}\,(\mathrm{seq}_\gamma)$$

$$\dfrac{\dfrac{\dfrac{}{\vdash_{\mathrm{bca}} [[-4,-1]]_{\mathsf{Int}}\, x < 0?\, [[-4,1]]_{\mathsf{Int}}}\,(\mathrm{basic}) \quad \dfrac{}{\vdash_{\mathrm{bca}} [[-4,-1]]_{\mathsf{Int}}\, x := x+2\, [[-2,1]]_{\mathsf{Int}}}\,(\mathrm{basic}) \quad \gamma_{\mathsf{Int}}([-4,1]) = [\![x < 0?]\!]\gamma_{\mathsf{Sign}}([-4,-1])}{\vdash_{\mathrm{bca}} [[-4,1]]_{\mathsf{Int}}\, B\, [[-2,1]]_{\mathsf{Int}}}\,(\mathrm{seq}_\gamma) \quad [-2,1] \leq_{\mathsf{Int}} [-4,1]}{\dfrac{\vdash_{\mathrm{bca}} [[-4,1]]_{\mathsf{Int}}\, B^{\mathrm{fix}}\, [[-4,1]]_{\mathsf{Int}}}{\text{(c)}}}\,(\mathrm{abs\text{-}inv})$$

$$\dfrac{\dfrac{\text{(a)}}{\vdash_{\mathrm{bca}} [[-4,-3]]_{\mathsf{Int}}\, B\, [[-2,-1]]_{\mathsf{Int}}}\,(\mathrm{seq}_\gamma) \quad \dfrac{\text{(b)}}{\vdash_{\mathrm{bca}} [[-2,-1]]_{\mathsf{Int}}\, B\, [[0,1]]_{\mathsf{Int}}}\,(\mathrm{seq}_\gamma) \quad \gamma_{\mathsf{Int}}([-2,-1]) = [\![B]\!]\gamma_{\mathsf{Sign}}([-4,-3])}{\dfrac{\vdash_{\mathrm{bca}} [[-4,-3]]_{\mathsf{Int}}\, B^2\, [[0,1]]_{\mathsf{Int}}}{\text{(d)}}}\,(\mathrm{seq}_\gamma)$$

$$\dfrac{\dfrac{\text{(d)}}{\vdash_{\mathrm{bca}} [[-4,-3]]_{\mathsf{Int}}\, B^2\, [[0,1]]_{\mathsf{Int}}}\,(\mathrm{seq}_\gamma) \quad \dfrac{\text{(c)}}{\vdash_{\mathrm{bca}} [[-4,1]]_{\mathsf{Int}}\, B^{\mathrm{fix}}\, [[-4,1]]_{\mathsf{Int}}}\,(\mathrm{abs\text{-}inv}) \quad [-4,1] = [-4,-3] \vee_{\mathsf{Int}} [0,1]}{\dfrac{\vdash_{\mathrm{bca}} [[-4,-3]]_{\mathsf{Int}}\, B^{\mathrm{fix}}\, [[-4,1]]_{\mathsf{Int}}}{\text{(e)}}}\,(\mathrm{rec})$$

$$\dfrac{\dfrac{\text{(e)}}{\vdash_{\mathrm{bca}} [[-4,-3]]_{\mathsf{Int}}\, B^{\mathrm{fix}}\, [[-4,1]]_{\mathsf{Int}}}\,(\mathrm{rec}) \quad \vdash_{\mathrm{bca}} [[-4,1]]_{\mathsf{Int}}\, x \geq 0\, [[0,1]]_{\mathsf{Int}} \quad \gamma_{\mathsf{Int}}([-4,-1]) = [\![B^{\mathrm{fix}}]\!]\gamma_{\mathsf{Sign}}([-4,-3])}{\vdash_{\mathrm{bca}} [[-4,-3]]_{\mathsf{Int}}\, \mathsf{p}_4\, [[0,1]]_{\mathsf{Int}}}\,(\mathrm{seq}_\gamma)$$

Here, the derivation of the triple $[[-4,-3]]_{\mathsf{Int}}\, \mathsf{p}_4\, [[0,1]]_{\mathsf{Int}}$ reconstructs the least fixpoint computation of $\lambda x \in \mathsf{Int}.\, [-4,-3] \vee_{\mathsf{Int}} [\![B]\!]^\sharp_A x$, yet proving that $[-4,1]$ is the best possible analysis in Int. □

9 Related Work

While the notion of best correct approximation is well established [Cousot 2021, Sections 27.5-27.6], and abstract transfer functions are often designed with a proof of optimality (see, e.g., [Miné 2017] for numerical abstractions such as affine equalities [Karr 1976] and octagons [Miné 2006]), the property of being the best abstract interpretation has not been thoroughly explored.

The most related works are [Reps et al. 2004] and [Thakur and Reps 2012]. Reps et al. [2004] put forward an algorithm to attain, under some hypotheses, a symbolic abstract transfer function which is the bca on a given abstraction, in particular for the predicate abstraction domain of logical formulae [Jhala et al. 2018]. [Reps et al. 2004] shows how to implement symbolically the abstraction function α through operations on logical formulae and, in particular, relies on having an algorithm for satisfiability checking of a logical formula. This first result has been later generalized and improved in [Thakur and Reps 2012], where the approach leverages Stålmarck's algorithm [Sheeran and Stålmarck 2000] as satisfiability checker of formulae. A summary of these two works is described in the survey [Reps and Thakur 2016, Section 5] on automating abstract interpretation.

A second line of work focussing on best correct approximations is [Giacobazzi and Ranzato 2010, 2014], which investigates a notion of *correctness kernel*. Given the concrete semantics $[\![\mathsf{p}]\!]$ of some program p and an abstraction A, the correctness kernel of A for p is defined to be the most abstract domain $\mathcal{K}_{\mathsf{p}}(A)$ that induces the same bca of $[\![\mathsf{p}]\!]$ as A does, i.e., the most abstract domain $A_{\mathrm{m}} \in \mathrm{Abs}(C)$ such that $\alpha_{A_{\mathrm{m}}}[\![\mathsf{p}]\!]\gamma_{A_{\mathrm{m}}} = \alpha_A[\![\mathsf{p}]\!]\gamma_A$ holds. Giacobazzi and Ranzato [2010, 2014] show that this correctness kernel $\mathcal{K}_{\mathsf{p}}(A)$ exists under mild conditions, provide a constructive method to build $\mathcal{K}_{\mathsf{p}}(A)$, and apply this approach to the well-known CEGAR method for abstract model checking [Clarke et al. 2000, 2003] to define a more accurate abstraction refinement heuristic, which is a crucial component of the CEGAR method.

Finally, our logic $\vdash_{\mathrm{bca}}$ can be viewed as an *abstract program logic*, meaning that the reasoning is parametrized and relies on a given abstract domain $A_{\alpha,\gamma}$. This feature is shared with other program logics, such as the algebraic Hoare logic by Cousot et al. [2012] and the local completeness logic by Bruni et al. [2021, 2023]. The algebraic Hoare logic [Cousot et al. 2012] aims at proving triples $\{a\}\mathrm{p}\{b\}$, where $a, b \in A$ are abstract assertions, whose validity is defined by $[\![\mathrm{p}]\!]\gamma(a) \subseteq \gamma(b)$. The local completeness logic [Bruni et al. 2021, 2023] proves triples $[c]\mathrm{p}[d]$, where $c, d \in C$ are concrete assertions, whose validity means that both $d \leq_C [\![\mathrm{p}]\!]c$ and $[\![\mathrm{p}]\!]^{\sharp}_{A}\alpha(c) = \alpha(d) = \alpha([\![\mathrm{p}]\!]c)$ hold, thus entailing that d is an under-approximation of the strongest postcondition $[\![\mathrm{p}]\!]c$, as in incorrectness logic [O'Hearn 2020], and, besides, the analysis of p is locally α-complete on the precondition c.

10 Conclusion and Future Work

We studied the properties of the best correct approximations of a concrete semantics by abstract interpretation from three complementary and interrelated perspectives. First, we studied the computability attributes of the class of programs admitting the best possible abstract interpretation in A, by showing its nonrecursivity; then, we have shown the impossibility of achieving the bca property through either program compilation or minimal abstraction refinements or simplifications of the domain A; finally, we designed a program logic parametrized on the abstraction A to infer Hoare-like triples $[a]_A\ \mathrm{p}\ [b]_A$ encoding that the abstract interpretation of a program p on A with a given abstract input $a \in A$ provides an abstract output $b \in A$ which is the best possible one in A.

One might ask whether breaking a program p into suitable subprograms could induce optimal abstract transfer functions for each subprogram, possibly resulting in the best overall abstraction for p. This question is challenging for future work but it is too general without constraints on the choice of subprograms. For example, the program p in Example 3.3 works for proving that composition does not preserve the bca property when it is considered as the composition $(z := 2; z := z + 1); z := z - 1$. However, if p is viewed as $z := 2; (z := z + 1; z := z - 1)$, then Example 3.3 would fail because the bca property in Sign does not hold for the subprogram $z := z + 1; z := z - 1$.

We envisage a further stimulating direction of future work. We proved that the bca is the only possible *sound and extensional abstract program semantics*. Equivalently, any abstract interpretation which is not the best, turns out to be *intrinsically intensional*. This phenomenon is rooted into the noncompositionality of abstract interpretation: whenever we inductively compose abstract transfer functions we may lose the bca property. In this context, we plan to investigate a notion of *abstract Kleene algebra with tests* (aKAT), based on an abstract rather than concrete semantics. The observation in Section 2.3.1 that the Kleene star $\mathrm{r}^{\star}$ and fixpoint operations $\mathrm{r}^{\mathrm{fix}}$ behave differently in the abstract semantics suggests that the well-known axiomatization of KAT [Kozen 1997] for reasoning on program behaviours as modeled by concrete semantics does not work for abstract semantics. Hence, a notion of abstract KAT should have a different algebraic structure, compliant with the specific fixpoint strategy adopted. The goal is to explore KAT-like axiomatizations of program behaviors defined by *abstract semantics*, where loop invariants are modeled by $\mathrm{r}^{\mathrm{fix}}$ instead of $\mathrm{r}^{\star}$. This would enable equational reasoning on programs through the lens of abstract interpretation, making aKAT for abstract interpretation what KAT is for concrete interpretation.

Acknowledgments

Roberto Giacobazzi and Francesco Ranzato were partially funded by a *WhatsApp Research Award* on "Privacy-aware Program Analysis" and by an *Amazon Research Award* for *AWS Automated Reasoning* on "Implicit Program Analysis". Roberto Giacobazzi was supported by the *Air Force Office of Scientific Research* under award number FA9550-23-1-0544. Francesco Ranzato was partially funded by the *Italian MUR*, under the PRIN 2022 PNRR project no. P2022HXNSC.

References

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

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

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. 2022. Abstract interpretation repair. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI'22*. ACM, 426–441. https://doi.org/10.1145/3519939.3523453

Roberto Bruni, Roberto Giacobazzi, Roberta Gori, and Francesco Ranzato. 2023. A Correctness and Incorrectness Program Logic. *J. ACM* 70, 2 (2023), 15:1–15:45. https://doi.org/10.1145/3582267

Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2000. Counterexample-guided abstraction refinement. In *Proc. of the 12th Internat. Conf. on Computer Aided Verification* (*CAV '00*) *(Lecture Notes in Computer Science, Vol. 1855)*. Springer-Verlag, 154–169. https://doi.org/10.1007/10722167_15

Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2003. Counterexample-guided abstraction refinement for symbolic model checking. *J. ACM* 50, 5 (2003), 752–794. https://doi.org/10.1145/876638.876643

Patrick Cousot. 2021. *Principles of Abstract Interpretation.* MIT Press.

Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. ACM POPL'77*. ACM, 238–252. https://doi.org/10.1145/512950.512973

Patrick Cousot and Radhia Cousot. 1979. Systematic design of program analysis frameworks. In *Proc. ACM POPL'79*. ACM, 269–282. https://doi.org/10.1145/567752.567778

Patrick Cousot, Radhia Cousot, Francesco Logozzo, and Michael Barnett. 2012. An abstract interpretation framework for refactoring with application to extract methods with contracts. In *Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012*. ACM, 213–232. https://doi.org/10.1145/2384616.2384633

Patrick Cousot, Roberto Giacobazzi, and Francesco Ranzato. 2018. Program analysis is harder than verification: A computability perspective. In *Proc. of the 30th International Conference on Computer Aided Verification, CAV 2018 (Lecture Notes in Computer Science, Vol. 10982)*. Springer, 75–95. https://doi.org/10.1007/978-3-319-96142-2_8

Patrick Cousot, Roberto Giacobazzi, and Francesco Ranzato. 2019. A^2I: Abstract^2 interpretation. *Proc. ACM Program. Lang.* 3, POPL (2019), 42:1–42:31. https://doi.org/10.1145/3290355

Gilberto Filé, Roberto Giacobazzi, and Francesco Ranzato. 1996. A unifying view of abstract domain design. *ACM Comput. Surv.* 28, 2 (1996), 333–336. https://doi.org/10.1145/234528.234742

Gilberto Filé and Francesco Ranzato. 1999. The powerset operator on abstract interpretations. *Theoretical Computer Science* 222, 1-2 (1999), 77–111. https://doi.org/10.1016/S0304-3975(98)00007-3

Michael J. Fischer and Richard E. Ladner. 1979. Propositional Dynamic Logic of Regular Programs. *J. Comput. Syst. Sci.* 18, 2 (1979), 194–211. https://doi.org/10.1016/0022-0000(79)90046-1

Roberto Giacobazzi, Francesco Logozzo, and Francesco Ranzato. 2015. Analyzing Program Analyses. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015*. ACM, 261–273. https://doi.org/10.1145/2676726.2676987

Roberto Giacobazzi and Elisa Quintarelli. 2001. Incompleteness, counterexamples and refinements in abstract model-checking. In *Proc. of the 8th Int. Static Analysis Symp. (SAS'01) (Lecture Notes in Computer Science, Vol. 2126)*. Springer, 356–373. https://doi.org/10.1007/3-540-47764-0_20

Roberto Giacobazzi and Francesco Ranzato. 1996. Compositional Optimization of Disjunctive Abstract Interpretations. In *Proceedings of the 6th European Symposium on Programming, ESOP'96 (Lecture Notes in Computer Science, Vol. 1058)*. Springer, 141–155. https://doi.org/10.1007/3-540-61055-3_34

Roberto Giacobazzi and Francesco Ranzato. 1997. Refining and compressing abstract domains. In *Proc. of the 24th Internat. Colloq. on Automata, Languages and Programming* (*ICALP '97*) *(Lecture Notes in Computer Science, Vol. 1256)*. Springer-Verlag, 771–781. https://doi.org/10.1007/3-540-63165-8_230

Roberto Giacobazzi and Francesco Ranzato. 2010. Example-Guided Abstraction Simplification. In *Proceedings 37th International Colloquium on Automata, Languages and Programming, ICALP 2010 (Lecture Notes in Computer Science, Vol. 6199)*. Springer, 211–222. https://doi.org/10.1007/978-3-642-14162-1_18

Roberto Giacobazzi and Francesco Ranzato. 2014. Correctness kernels of abstract interpretations. *Inf. Comput.* 237 (2014), 187–203. https://doi.org/10.1016/J.IC.2014.02.003

Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. 1998. Complete abstract interpretations made constructive. In *Proc. of the 23rd Internat. Symp. on Mathematical Foundations of Computer Science* (*MFCS '98*) *(Lecture Notes in Computer Science, Vol. 1450)*. Springer-Verlag, 366–377. https://doi.org/10.1007/BFb0055786

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

Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. 2005. Making Abstract Domains Condensing. *ACM Transactions on Computational Logic* 6, 1 (2005), 33–60. https://doi.org/10.1145/1042038.1042040

Ranjit Jhala, Andreas Podelski, and Andrey Rybalchenko. 2018. Predicate Abstraction for Program Verification. In *Handbook of Model Checking*. Springer, 447–491. https://doi.org/10.1007/978-3-319-10575-8_15

Michael Karr. 1976. Affine relationships among variables of a program. *Acta Informatica* 6 (1976), 133–151. https://doi.org/10.1007/BF00268497

Dexter Kozen. 1997. Kleene Algebra with Tests. *ACM Trans. Program. Lang. Syst.* 19, 3 (May 1997), 427–443. https://doi.org/10.1145/256167.256195

Antoine Miné. 2006. The octagon abstract domain. *High. Order Symb. Comput.* 19, 1 (2006), 31–100. https://doi.org/10.1007/s10990-006-8609-1

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

Peter W. O'Hearn. 2020. Incorrectness logic. *Proc. ACM Program. Lang.* 4, POPL (2020), 10:1–10:32. https://doi.org/10.1145/3371078

Francesco Ranzato. 2013. Complete Abstractions Everywhere. In *Proceedings of the 14th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2013 (Lecture Notes in Computer Science, Vol. 7737)*. Springer, 15–26. https://doi.org/10.1007/978-3-642-35873-9_3

Thomas W. Reps, Shmuel Sagiv, and Greta Yorsh. 2004. Symbolic Implementation of the Best Transformer. In *Proceedings 5th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2004 (Lecture Notes in Computer Science, Vol. 2937)*. Springer, 252–266. https://doi.org/10.1007/978-3-540-24622-0_21

Thomas W. Reps and Aditya V. Thakur. 2016. Automating Abstract Interpretation. In *Proceedings 17th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2016 (Lecture Notes in Computer Science, Vol. 9583)*. Springer, 3–40. https://doi.org/10.1007/978-3-662-49122-5_1

Henry G. Rice. 1953. Classes of Recursively Enumerable Sets and Their Decision Problems. *Trans. Amer. Math. Soc.* 74 (1953), 358–366. https://doi.org/10.2307/1990888

Hartley Rogers. 1987. *Theory of recursive functions and effective computability (Reprint from 1967)*. MIT Press.

Mary Sheeran and Gunnar Stålmarck. 2000. A Tutorial on Stålmarck's Proof Procedure for Propositional Logic. *Formal Methods Syst. Des.* 16, 1 (2000), 23–58. https://doi.org/10.1023/A:1008725524946

Robert I. Soare. 1980. *Recursively Enumerable Sets and Degrees*. Springer-Verlag.

Aditya V. Thakur and Thomas W. Reps. 2012. A Method for Symbolic Computation of Abstract Operations. In *Proceedings 24th International Conference on Computer Aided Verification, CAV 2012 (Lecture Notes in Computer Science, Vol. 7358)*. Springer, 174–192. https://doi.org/10.1007/978-3-642-31424-7_17

Mark N. Wegman and F. Kenneth Zadeck. 1991. Constant propagation with conditional branches. *ACM Transactions on Programming Languages and Systems* 13, 2 (1991), 181–210. https://doi.org/10.1145/103135.103136

Noam Zilberstein, Derek Dreyer, and Alexandra Silva. 2023. Outcome Logic: A Unifying Foundation for Correctness and Incorrectness Reasoning. *Proc. ACM Program. Lang.* 7, OOPSLA1 (2023), 522–550. https://doi.org/10.1145/3586045

Received 2024-07-10; accepted 2024-11-07