Static Analysis of Linear Absolute Value Equalities among Variables of a Program

Liqian Chen[a,b,*], Dengping Wei[a,*], Banghu Yin[a], Ji Wang[a,c]

[a] *National University of Defense Technology, Changsha 410073, China*
[b] *Hunan Key Laboratory of Software Engineering for Complex Systems, Changsha 410073, China*
[c] *State Key Laboratory of High Performance Computing, Changsha 410073, China*

Abstract

The classic linear (technically, affine) equality abstract domain, which can infer linear equality relations among variables of a program automatically, is one of the earliest and fundamental abstract domains. However, it cannot express non-convex properties that appear naturally due to the inherent disjunctive behaviors in programs. In this paper, we introduce a new abstract domain, namely *the abstract domain of linear absolute value equalities* (AVE), which generalizes the linear equality abstract domain with absolute value terms of variables. More clearly, we leverage the absolute value function to design the new abstract domain for discovering linear equality relations among values and absolute values of program variables. Moreover, since linear absolute value equalities can only express limited form of inequalities while programs often involve various inequalities, to help the AVE domain, we propose a so-called *signed interval abstract domain* as an extension of the classic interval abstract domain. The key idea is to use two intervals to track respectively the positive part and the negative part of the interval range for each variable. On this basis, we propose to combine the two new abstract domains to improve precision of each other during analysis. Experimental results are encouraging: In practice, the AVE abstract domain (together with the signed interval abstract domain) can find interesting piece-wise linear invariants that are non-convex and out of the expressiveness of the linear equality domain.

Keywords: Abstract interpretation, Abstract domain, Absolute value, Invariant,

*Corresponding author
Email addresses: lqchen@nudt.edu.cn (Liqian Chen), dpwei@nudt.edu.cn (Dengping Wei)

1. Introduction

In 1970s, Karr [1] proposed an efficient algorithm to infer automatically affine relations (conventionally also called linear equality relations) among program variables, in the form of $\sum_k a_k x_k = b$ (where x_k's are program variables and a_k's are coefficients automatically inferred by the algorithm). Karr's algorithm is now understood as an abstract domain of linear equalities (also called Karr's domain) under the framework of abstract interpretation [2].

Linear equality relations are useful in many application contexts, including classical data flow analysis (e.g., constant propagation, definite equalities among variables, etc.), inter-procedural analysis of affine programs [3], analysis of machine code [4], analysis of modern deep learning programs [5], analysis of mobile systems [6], etc. The space complexity of the abstract domain of linear equalities is $O(n^2)$, and the time complexity of its domain operations is $O(n^3)$ (without considering the size of the program) where n is the number of program variables. As a lightweight relational abstract domain, the abstract domain of linear equalities has been widely used in program analysis. However, similarly to most existing abstract domains, the abstract domain of linear equalities (using conjunctions of linear equalities) can only express convex sets, whereas programs often involve non-convex behaviors (due to control-flow joins, disjunctions, etc.).

Absolute value (AV) is a fundamental concept in mathematics, which can express piecewise linear behaviors. In practice, piecewise linear behaviors account for a large class of non-convex behaviors in a program (or after abstracting non-linear behaviors into piecewise linear behaviors, as in the field of hybrid systems). Hence, we could exploit the piecewise linear expressiveness of the AV function to design non-convex abstract domains, to express certain piecewise linear behaviors in a program. Based on this insight, in our previous work [7][8], we have presented an abstract domain of linear AV Inequalities (named AVI, which is of high complexity and thus may have scalability limitations in practice) and an abstract domain of octagonal constraints with absolute

value (which can infer relations of the form $\{\pm x \pm y \leq c, \pm x \pm |y| \leq d, \pm|x| \pm |y| \leq e\}$ over each pair of variables x, y, where $\pm \in \{-1, 0, 1\}$).

In this paper, we leverage absolute value to design a new abstract domain, namely, *the abstract domain of linear Absolute Value Equalities (AVE)*, to discover linear equality relations among values and absolute values of program variables, in the form of $\Sigma_k a_k x_k + \Sigma_k b_k |x_k| = c$ (where x_k's are program variables, and $a_k, b_k, c \in \mathbb{Q}$ are constants automatically inferred by the analysis). Using AVE, we can express certain disjunctions of linear equalities. E.g., $x = -1 \vee x = 1$ can be expressed as $|x| = 1$. Also, using AVE, we can handle many (piece-wise linear) mathematical functions over variables in modern programming languages, such as `abs` (absolute value of an integer), `fabs`(absolute value of a floating-point number), `fmin` (minimum value), `fmax` (maximum value), `fdim` (positive difference) in the C99 standard. E.g., equality $\max(x, y) = z$ can be expressed as $w = x - y \wedge \frac{1}{2}(|w| + x + y) = z$ (by introducing a fresh auxiliary variable w). And the ReLU function in neural network can be expressed as $ReLU(x, 0) = \frac{1}{2}(|x| + x)$. Moreover, even linear inequality $x \geq 0$ can be expressed as AVE $|x| = x$. In other words, we could use AVE to encode certain linear inequality constraints.

The new domain is more expressive than the classic linear equality domain and allows expressing certain non-convex (even unconnected) sets thanks to the expressiveness of absolute value. However, using AV equalities can only encode limited form of linear inequalities (such as $x \geq 0$ which can be encoded as $|x| = x$), but can not exactly capture inequalities such as $x \geq 2$ (which can be only over-approximated as $|x| = x$ in the AVE domain). Hence, we propose to combine the AVE abstract domain with the interval domain. Since the AVE domain fits to capture disjunctive information that encoding the signs of variables, we propose a so-called *signed interval abstract domain* as an extension of the classic interval abstract domain. The main idea of the signed interval abstract domain is to use two intervals to track respectively the positive part and the negative part of the interval range for each variable. Both the AVE domain and the signed interval domain fit for analyzing programs involving disjunctive behaviors, wherein at least some of the program variables contain both positive and negative values. Moreover, the two domains can help each other to improve analysis precision.

The preliminary experimental results of the prototype implementation are promis-

ing on example programs; AVE (together with the signed interval abstract domain) can find piece-wise linear invariants of interest that are non-convex and out of the expressiveness of the conventional linear equality abstract domain in practice.

This paper is an extended version of our TASE 2021 paper [9]. On top of [9], we propose a new abstract domain, named the signed interval abstract domain (Sect. 4), as an extension of the classic interval abstract domain. Furthermore, we propose approaches to combine with the AVE domain and the signed interval domain to improve analysis precision (Sect. 5). We have also conducted more experiments (Sect. 6).

The rest of the paper is organized as follows. Sect. 2 describes some preliminaries. Sect. 3 presents the new proposed abstract domain of linear absolute value equalities. Sect. 4 presents the new proposed signed interval abstract domain. Sect. 5 presents approaches to combine the AVE domain and the signed interval domain. Sect. 6 presents our prototype implementation together with experimental results. Sect. 7 discusses some related work before Sect. 8 concludes.

2. Preliminaries

2.1. System of linear absolute value equalities

Let $|\cdot|$ denote absolute value (AV). Let $x = (x_i)_{i=1}^n$ be a vector and $|x| = (|x_i|)_{i=1}^n$. We consider the following system of linear absolute value equalities (AVE)

$$Ax + B|x| = c \tag{1}$$

where $A, B \in \mathbb{Q}^{m\times n}$ and $c \in \mathbb{Q}^m$.

2.2. Linear complementarity problem and its extensions

Given a matrix $M \in \mathbb{Q}^{n\times n}$ and a vector $q \in \mathbb{Q}^n$, the (standard) linear complementarity problem (LCP) is defined as the problem of finding vectors x^+ and x^- such that

$$x^+ = Mx^- + q \tag{2}$$

$$x^+, x^- \geq 0 \tag{3}$$

$$(x^+)^T x^- = 0. \tag{4}$$

Note that if x^+ and x^- are solutions of the above LCP, then they follow from (3-4) that

$$x_i^+ x_i^- = 0 \qquad \text{for } i = 1, \ldots, n$$

i.e., for each i the following holds: If $x_i^+ > 0$ then $x_i^- = 0$ holds, and if $x_i^- > 0$ then $x_i^+ = 0$ holds. In other words, the zero patterns of x_i^+ and x_i^- are complementary. Thus, condition (4) is called the *complementarity condition* of the above LCP. In the following we introduce two extensions of LCP that are of interest to us.

Given $M, N \in \mathbb{Q}^{m\times n}$ and $q \in \mathbb{Q}^m$, find $x^+, x^- \in \mathbb{Q}^n$ so that

$$Mx^+ + Nx^- \leq q \tag{5}$$

$$x^+, x^- \geq 0 \tag{6}$$

$$(x^+)^T x^- = 0. \tag{7}$$

We call the above problem eXtended Linear Complementary Problem (XLCP), since it can be proved equivalent to eXtended LCP of Mangasarian and Pang [10].

Given $M, N \in \mathbb{Q}^{m\times n}$ and $q \in \mathbb{Q}^m$, find $x^+, x^- \in \mathbb{Q}^n$ so that

$$Mx^+ + Nx^- = q \tag{8}$$

$$x^+, x^- \geq 0 \tag{9}$$

$$(x^+)^T x^- = 0. \tag{10}$$

We call the above problem Horizontal Linear Complementary Problem (HLCP). HLCP can be considered as special case of XLCP. In the literature, HLCP has been shown equivalent to LCP [11] and piece-wise linear system [12].

2.3. Equivalence of AVEs and HLCPs

Let vectors x^+ and x^- be defined by $x^+ = (\max(x_i, 0))_{i=1}^n$ and $x^- = (\max(-x_i, 0))_{i=1}^n$, so that

$$x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0$$

and

$$x = x^+ - x^- \qquad |x| = x^+ + x^- \tag{11}$$

$$x^+ = \frac{1}{2}(x + |x|) \qquad x^- = \frac{1}{2}(|x| - x). \tag{12}$$

According to (11), AVE (1) can be reformulated as the following HLCP:

$$\begin{aligned}(A+B)x^+ + (B-A)x^- &= c\\ x^+, x^- &\geq 0\\ (x^+)^T x^- &= 0.\end{aligned}$$

Similarly, according to (12), HLCP (8-10) can be reformulated as the following AVE:

$$\frac{1}{2}(M-N)x + \frac{1}{2}(M+N)|x| = q.$$

3. An abstract domain of linear absolute value equalities

In this section, we present a new abstract domain, namely the abstract domain of linear absolute value equalities (AVE). The key idea is to use a system of linear absolute value equalities (and equivalent representations) as the domain representation. AVE can be used to infer relationships of the form $\Sigma_k a_k x_k + \Sigma_k b_k |x_k| = c$ over program variables x_k ($k = 1, \ldots, n$), where constants $a_k, b_k, c \in \mathbb{Q}$ are automatically inferred by the analysis.

3.1. Domain Representation

Under the framework of abstract interpretation, when designing numerical abstract domains, we often use a certain type of constraints to represent the abstract elements of the abstract domain. Geometrically, a certain type of constraints correspond to a special shape. E.g., the conjunction of a set of arbitrary linear equalities corresponds to an affine space.

In the AVE domain, to describe an abstract element $\mathbf{P}$, we use an AVE system $Ax + B|x| = c$, where $A, B \in \mathbb{Q}^{m\times n}, c \in \mathbb{Q}^m$, m is the number of linear equalities in the system, and n is the number of variables in the system. It represents the set $\gamma(\mathbf{P}) \stackrel{\text{def}}{=} \{x \in \mathbb{Q}^n \mid Ax + B|x| = c\}$, in which each point $x \in \gamma(\mathbf{P})$ represents a possible program environment (or state), i.e., an assignment of numerical/rational values to program variables. Geometrically, the intersection of each AVE element with each orthant in $\mathbb{Q}^n$ results in an affine space (within the orthant boundary).

3.1.1. Expressiveness lifting

Note that in the AVE domain representation, absolute value $|\cdot|$ applies to only a variable rather than an expression. E.g., equality $\max(x, y) = z$ cannot be directly expressed by linear AVE over x, y, z and their absolute value terms. However, we could introduce a fresh auxiliary variable w to denote $x - y$. Then we could encode

$$\max(x, y) = z$$

as

$$w = x - y \quad \wedge \quad \frac{1}{2}(|w| + x + y) = z$$

which can be expressed by an AVE element (over the dimensions of x, y, z, w).

In general, we can lift the expressiveness of the AVE domain elements by introducing new auxiliary variables to denote those expressions that appear inside the AV function. In fact, a large subclass of piecewise linear functions of practical interest can be represented via AV functions through a so-called canonical (piecewise linear) representation [13]. More clearly, according to [13], a piecewise-linear function f has a canonical piece-wise-linear representation if and only if it possesses the so-called consistent variation property.

3.1.2. Internal domain representation

In Sect. 2, we have shown the equivalence between AVEs and HLCPs, which indicates that we can reuse the method that can solve one of them to solve the other. Inside the implementation, we convert AVEs into HLCPs and maintain the domain representation in the form of HLCPs, such that we can reuse known results in the field of LCP. To this end, we maintain the map between abstract environments over x and abstract environments over x^+, x^- as:

$$\begin{aligned} x &= x^+ - x^-, & |x| &= x^+ + x^- \\ x^+ &= \frac{1}{2}(x + |x|), & x^- &= \frac{1}{2}(|x| - x) \end{aligned}$$

where x^+, x^- satisfy

$$x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0$$

Constraint Normalization. Throughout this paper, we fix a variable ordering $x_1^+ \prec \ldots \prec x_n^+ \prec x_1^- \prec \ldots \prec x_n^-$. For a linear equality $\Sigma_k a^+{}_k x_k^+ + \Sigma_k a^-{}_k x_k^- = b$, the variable $x_i^\pm$ (where $\pm \in \{+, -\}$) with the least index i such that $a^\pm{}_i \neq 0$ is called its *leading variable*. A linear equality φ is said to be *normalized* if the coefficient of its leading variable $x_i^\pm$ satisfies $a_i^\pm$=1. Then, given φ which is not normalized, its normalized form can be obtained by dividing the whole constraint φ by the coefficient $a_i^\pm$ of its leading variable $x_i^\pm$. Note that this normalization operation is exact, i.e., it will cause no precision loss. For convenience sake, we enforce a normalized form on constraints throughout this paper.

Row echelon form. Let $Ax^\pm = b$ be a linear system with $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$. The system $Ax^\pm = b$ is said to be in *row echelon* form if

1) Every row i_0 of A has at least one non-zero entry.

2) Let $x_{j_0}^\pm$ be the leading variable of row i_0 of A. Then for all $i > i_0, j \leq j_0, A_{ij} = 0$.

Furthermore, given a linear system $Ax^\pm = b$ in row echelon form, it is said to be in *reduced row echelon form* if

- Let $x_{j_0}^\pm$ be the leading variable of row i_0 of A. Then for all $i < i_0, A_{ij_0} = 0$.

We use the linear system $Ax^\pm = b$ in reduced row echelon form (together with the standard complementary condition) as the canonical representation of AVE elements.

Constraint reduction via complementary condition. Remember that inside the HLCP constraint representation of AVE elements, besides the linear system part $Ax^\pm = b$, we also have the complementary condition part $x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0$. Note that the complementary condition indicates that at least one of the x_i^+ and x_i^- must be 0. Hence, we can make use of the complementary condition to reduce further the linear system part $Ax^\pm = b$.

- Consider a normalized linear equality $\Sigma_k a^+{}_k x_k^+ + \Sigma_k a^-{}_k x_k^- = b$.
  - If $b = 0$ and $\forall k. a_k^\pm \geq 0$, we know that $x_k^+ = 0$ holds for all k where $a_k^\pm ! = 0$. Hence, the original equality $\Sigma_k a^+{}_k x_k^+ + \Sigma_k a^-{}_k x_k^- = b$ in the linear system part is replaced with a series of one-variable linear equalities $x_k^+ = 0$ (where $a_k^\pm ! = 0$).

- If $b < 0$ and $\forall k.a_k^{\pm} \geq 0$, we know that this AVE element is infeasible and thus becomes bottom.

- Consider a normalized linear equality involving only a pair of complementary variables, i.e., in the form of $x_k^+ + a_k^- x_k^- = b$.
  - If $b > 0$ and $a_k^- < 0$ hold, we know that $x_k^+ = b$ and $x_k^- = 0$. Then, the original equality $x_k^+ + a_k^- x_k^- = b$ in the linear system part is replaced with two one-variable linear equalities $x_k^+ = b$ and $x_k^- = 0$.
  - If $b < 0$ and $a_k^- < 0$ hold, we know that $x_k^+ = 0$ and $x_k^- = b/a_k^-$. Then, the original equality $x_k^+ + a_k^- x_k^- = b$ in the linear system part is replaced with two one-variable linear equalities $x_k^+ = 0$ and $x_k^- = b/a_k^-$.
  - If $b < 0$ and $a_k^- \geq 0$ hold, we know that this AVE element is infeasible and thus becomes bottom.

3.1.3. Generator representation for HLCP

By Minkowski-Weyl theorem [14], the set $P \subseteq \mathbb{Q}^n$ is a polyhedron, iff it is finitely generated, i.e., there exist finite sets $V, R \in \mathbb{Q}^n$ such that P can be generated by (V, R):

$$P = \left\{ \sum_{i=1}^{|V|} \lambda_i V_i + \sum_{j=1}^{|R|} \mu_j R_j \;\middle|\; \forall i, \lambda_i \geq 0, \forall j, \mu_j \geq 0, \sum_{i=1}^{|V|} \lambda_i = 1 \right\}$$

where $|V|, |R|$ denote the cardinality of sets V, R respectively. Elements in V are called *extreme points* (also called vertices), while elements in R are called *extreme rays*. Using the double description method, a convex polyhedron can be represented by either its constraint representation $\{Ax \leq b\}$ or its generator representation (V, R).

In [7], Chen et al. have presented an approach to construct the generator representation for XLCP. Since HLCP is a special case of XLCP, we make use of the approach proposed in [7] to generate the generator representation for an HLCP. For the sake of space, here we only review the main idea of the approach mentioned above and refer the details to [7]. Intuitively, (5-6) of an XLCP describes a convex polyhedron $P = \{x^+ \in \mathbb{Q}^n, x^- \in \mathbb{Q}^n \mid Mx^+ + Nx^- \leq q, x^+ \geq 0, x^- \geq 0\}$, while the complementary condition (7) specifies that $x_i^+ = 0 \vee x_i^- = 0$ holds for all $i = 1, \ldots, n$, which

indicates 2^n complementary patterns. Hence, first, those generators g of P not satisfying the complementary condition are removed. Second, since not all combinations of the complementary generators (i.e., generators satisfying the complementary condition) will result in solutions of XLCP (5-7), generators are grouped according to the complementary patterns such that each group corresponds to a convex polyhedron and any combination of generators in one group will always result in a solution of XLCP (5-7). Fortunately, similarly as the AVI abstract domain [7] , to design the AVE abstract domain, we do not need to group the complementary generators, since no domain operation requires the group information and all domain operations can be implemented based on only a non-redundant set of complementary generators $G^c = (V^c, R^c)$.

For the sake of simplicity, from now on, we assume the AVE element $\mathbf{P}$ corresponds to the following HLCP system:

$$Mx^+ + Nx^- = b$$
$$x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0$$

and we denote its set of complementary generators as

$$G^c_{\mathbf{P}} = (V^c_{\mathbf{P}}, R^c_{\mathbf{P}}).$$

3.2. Domain operations

Now, we describe the implementation of most common domain operations required for static analysis over the AVE domain, most of which require only constraints while few of which require both constraints and complementary generators. Note that we mainly maintain the HLCP constraint representation for abstract elements, and convert it to the complementary generator representation temporally as intermediate representation only when needed.

3.2.1. Lattice operations

- Emptiness test: $\mathbf{P}$ is empty, iff $V^c_{\mathbf{P}} = \emptyset$. However, in practice, when we conduct constraint reduction over the linear system part $Ax^{\pm} = b$ by exploiting the complementary condition, we often get contradictory equalities (implying emptiness), as explained in Sect. 3.1.

In the following, throughout Sect.3.2, we only consider non-empty AVE domain elements and denote them by $\mathbf{P}, \mathbf{P}'$. The operations on empty elements are trivial and therefore we omit for the sake of concision.

- Meet: $\mathbf{P} \sqcap \mathbf{P}'$ is an AVE domain element whose HLCP constraint representation is

$$\begin{aligned} Mx^+ + Nx^- &= b \\ M'x^+ + N'x^- &= b' \\ x^+ \geq 0, x^- \geq 0, (x^+)^T x^- &= 0 \end{aligned}$$

  where $\{Mx^+ + Nx^- = b, M'x^+ + N'x^- = b'\}$ can be converted into reduced row echelon form w.r.t. variable ordering $x_1^+ \prec \ldots \prec x_n^+ \prec x_1^- \prec \ldots \prec x_n^-$ via Gaussian elimination, and thus some redundant constraints are removed or contradictory equalities are found.

- Join: $\mathbf{P} \sqcup \mathbf{P}'$ is the least AVE domain element containing $\mathbf{P}$ and $\mathbf{P}'$, whose set of complementary generators is the union of those of $\mathbf{P}$ and $\mathbf{P}'$: $(V_{\mathbf{P}}^c \cup V_{\mathbf{P}'}^c, R_{\mathbf{P}}^c \cup R_{\mathbf{P}'}^c)$, where $(V_{\mathbf{P}}^c, R_{\mathbf{P}}^c)$, $(V_{\mathbf{P}'}^c, R_{\mathbf{P}'}^c)$ respectively denote the set of complementary generators of $\mathbf{P}$ and that of $\mathbf{P}'$. After we get the complementary generator representation of $\mathbf{P} \sqcup \mathbf{P}'$, we convert it back to HLCP constraint representation. Overall, $\mathbf{P} \sqcup \mathbf{P}'$ is computed by the following steps:

  1. Compute the complementary generator representation $(V_{\mathbf{P}}^c, R_{\mathbf{P}}^c)$, $(V_{\mathbf{P}'}^c, R_{\mathbf{P}'}^c)$ respectively for $\mathbf{P}$ and $\mathbf{P}'$;
  2. Compute $(V_{\mathbf{P}}^c \cup V_{\mathbf{P}'}^c, R_{\mathbf{P}}^c \cup R_{\mathbf{P}'}^c)$, and suppose that $V_{\mathbf{P}}^c \cup V_{\mathbf{P}'}^c$ results in $\{v_1, \ldots, v_p\}$, while $R_{\mathbf{P}}^c \cup R_{\mathbf{P}'}^c$ results in $\{r_1, \ldots, r_q\}$;
  3. Project out variables $\lambda_j (j = 1, \ldots, p), \mu_k (k = 1, \ldots, q)$ (via Gaussian elimination) from the following system:

$$\begin{cases} (x^+ \; x^-)^T = \sum_{j=1}^{p} (\lambda_j v_j) + \sum_{k=1}^{q} (\mu_k r_k) \\ \sum_{j=1}^{p} \lambda_j = 1 \end{cases}$$

     Suppose we get the following system over x^+, x^-:

$$\hat{M}x^+ + \hat{N}x^- = \hat{b}$$

4. Finally, the resulting constraint HLCP representation of $\mathbf{P} \sqcup \mathbf{P}'$ is:

$$\hat{M}x^+ + \hat{N}x^- = \hat{b}$$
$$x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0$$

where $\{\hat{M}x^+ + \hat{N}x^- = \hat{b}\}$ can be converted into reduced row echelon form via Gaussian elimination.

- Inclusion test: $\mathbf{P} \sqsubseteq \mathbf{P}'$ that is $\gamma(\mathbf{P}) \subseteq \gamma(\mathbf{P}')$, iff $\gamma(\mathbf{P} \sqcap \mathbf{P}') = \gamma(\mathbf{P})$. After computing $\mathbf{P} \sqcap \mathbf{P}'$, we maintain the resulting AVE element in reduced row echelon form (after conducting constraint reduction via complementary generators), and compare it syntactically with the HLCP constraint representation of $\mathbf{P}$. If $\mathbf{P} \sqcap \mathbf{P}'$ and $\mathbf{P}$ are syntactically equal, we know $\gamma(\mathbf{P} \sqcap \mathbf{P}') = \gamma(\mathbf{P})$.

3.2.2. Transfer functions

In this paper, for the sake of simplicity, we consider a very simple language with limited program statements including tests, projection, assignments, and loops, similarly as the program statements considered in [15]. However, note that many complex statements can be transformed (or abstracted) into these simple statements we considered in this paper, e.g., through compile frontend.

Let $\tau[\![\cdot]\!]^\#(\mathbf{P})$ denote the abstract effect of a program statement on an AVE element $\mathbf{P}$.

- Test transfer function: $\tau[\![cx + d|x| = e]\!]^\#(\mathbf{P})$, whose HLCP system is defined as

$$Mx^+ + Nx^- = b$$
$$(c + d)x^+ + (d - c)x^- = e$$
$$x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0$$

where $\{Mx^+ + Nx^- = b, (c + d)x^+ + (d - c)x^- = e\}$ can be converted into reduced row echelon form with respect to the variable ordering $x_1^+ \prec \ldots \prec x_n^+ \prec x_1^- \prec \ldots \prec x_n^-$ via Gaussian elimination, and thus some redundant constraints are removed or contradictory equalities (which implies emptiness) are found.

- Projection: $\tau[\![x_j := random()]\!]^{\#}(\mathbf{P})$ can be implemented by projecting out x_j^+, x_j^- from

$$Mx^+ + Nx^- = b$$

via Gaussian elimination. Suppose we get $M'x^+ + N'x^- = b'$. Then, the resulting HLCP system will be

$$M'x^+ + N'x^- = b'$$
$$x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0$$

where $\{M'x^+ + N'x^- = b'\}$ can be converted into reduced row echelon form via Gaussian elimination.

- Assignment transfer function: $\tau[\![x_j := \Sigma_i a_i x_i + \Sigma_i b_i |x_i| + c]\!]^{\#}(\mathbf{P})$, can be modeled using test transfer function, projection and variable renaming (by temporally introducing a fresh variable x'_j) as follows:

$$\Big(\tau[\![x_j := random()]\!]^{\#} \circ$$

$$\tau[\![\Sigma_i a_i x_i + \Sigma_i b_i |x_i| + c - x'_j = 0]\!]^{\#}(\mathbf{P})\Big)\,[x'_j/x_j]$$

3.2.3. Extrapolations

As we know, since the lattice of linear equalities (in a program) has finite height, we do not need a widening operation for the domain of linear equalities. The intersection of an AVE element with each orthant, results in an affine space (without considering the orthant boundary), that is, an element in the domain of linear equalities. Since the number of the orthants are finite (for a given program), we also do not need a widening operation for the AVE domain. At each widening point, we use the join operator $\sqcup$ instead of the widening operator.

3.3. Example analysis

Example 1. *In Fig. 1, we show a simple typical program that implements the* $fabs()$ *function in C language (noting that we use* $:=$ *to denote the assignment) and then checks the properties over its result. This example involves non-convex (disjunctive) constraints (due to control-flow join), and precise reasoning over these constraints is required to prove these assertions.*

```
    float x, y;
    if (x ≥ 0  /* |x| == x */ ) { y := x; }
    else       /* |x| == −x */ { y := −x; }
①   if (x ≥ 0 /* |x| == x */ ) { assert(y == x); }
    else       /* |x| == −x */ { assert(y == −x); }
    }
```

Loc	PolkaEq	AVI	AVE
①	$\top$	$y == \|x\| \wedge$	$y == \|x\| \wedge$
	(no information)	$y == \|y\|$	$y == \|y\|$

Figure 1: Program *MotivEx* (left) and the generated invariants (right)

In particular, at program point ①*, the analysis needs to perform a join operation over the resulting AVE elements coming from the previous conditional branch:* $\mathbf{P} = \{(x\ y)^T \mid x - y = 0, |x| = x\} = \{(x^+\ x^-\ y^+\ y^-)^T \mid x^+ - y^+ + y^- = 0, x^- = 0, x^+ \geq 0, x^- \geq 0, y^+ \geq 0, y^- \geq 0, x^+x^- = 0, y^+y^- = 0\}$ *(corresponding to the result of the first* then *branch) and* $\mathbf{P'} = \{(x\ y)^T \mid -x - y = 0, |x| = -x\} = \{(x^+\ x^-\ y^+\ y^-)^T \mid x^- - y^+ + y^- = 0, x^+ = 0, x^+ \geq 0, x^- \geq 0, y^+ \geq 0, y^- \geq 0, x^+x^- = 0, y^+y^- = 0\}$ *(corresponding to the result of the first* else *branch).*

The sets of complementary generators for $\mathbf{P}, \mathbf{P'}$ *over* $(x^+, x^-, y^+, y^-)^T$ *are respectively*

$$(V^c_{\mathbf{P}}, R^c_{\mathbf{P}}) = \left\langle\!\!\left\langle \begin{pmatrix} x^+ \\ x^- \\ y^+ \\ y^- \end{pmatrix} : \left\{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}\right\}, \left\{\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}\right\} \right\rangle\!\!\right\rangle$$

$$(V^c_{\mathbf{P}'}, R^c_{\mathbf{P}'}) = \left\langle\!\!\left\langle \begin{pmatrix} x^+ \\ x^- \\ y^+ \\ y^- \end{pmatrix} : \left\{\!\!\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right\}\!\!\right\}, \left\{\!\!\left\{ \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \right\}\!\!\right\} \right\rangle\!\!\right\rangle$$

Hence,

$$(V^c_{\mathbf{P}} \cup V^c_{\mathbf{P}'}, R^c_{\mathbf{P}} \cup R^c_{\mathbf{P}'}) =$$

$$\left\langle\!\!\left\langle \begin{pmatrix} x^+ \\ x^- \\ y^+ \\ y^- \end{pmatrix} : \left\{\!\!\left\{ \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right\}\!\!\right\}, \left\{\!\!\left\{ \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \right\}\!\!\right\} \right\rangle\!\!\right\rangle$$

Projecting out λ_1, μ_1, μ_2 (wherein $\lambda_1 = 1$) from

$$\begin{pmatrix} x^+ \\ x^- \\ y^+ \\ y^- \end{pmatrix} = \lambda_1 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \mu_1 \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \mu_2 \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

will result in

$$x^+ + x^- - y^+ = 0, \quad y^- = 0.$$

Hence, the resulting HLCP constraint representation of $\mathbf{P} \sqcup \mathbf{P}'$ will be

$$\begin{aligned} &x^+ + x^- - y^+ = 0, \quad y^- = 0 \\ &x^+ \geq 0,\ x^- \geq 0,\ (x^+)^T x^- = 0 \\ &y^+ \geq 0,\ y^- \geq 0,\ (y^+)^T y^- = 0 \end{aligned}$$

In other words, we get $\mathbf{P} \sqcup \mathbf{P}' = \{(x\ y)^T \mid y = |x|,\ |y| = y\}$, which is the invariant at program point ① provided by AVE, as shown in Fig. 1.

Note that for the example in Fig. 1, the invariants provided by AVE are as precise as the AVI domain [7], which can prove the later assertions, while PolkaEq (an implementation of the linear equality abstract domain) cannot infer any information at ①.

4. The signed interval abstract domain

The interval abstract domain [16] is one of the basic abstract domains in abstract interpretation, which can infer value range for each program variable. During analysis, each variable x is tracked by an interval, representing its lower bound and the upper bound. Domain operations are implemented via interval arithmetic. Since the height of the lattice of the intervals is infinite, the interval domain employs widening operator to guarantee the convergence of the analysis. Note that the interval abstract domain maintains only one interval for each variable, and thus can not express disjunctive properties. However, in our context, using the AVE domain, we can express certain disjunctive properties depending on different signs of variables. For example, $|x| = 1$ means $x \in [-1, -1] \vee x \in [1, 1]$, while using the classic interval domain, it can only be over-approximated as $[-1, 1]$. To address this problem, one may use finite powersets of intervals to capture the disjunction information, but a proper widening operator for such powerset domain is notoriously hard to design.

In this paper, we propose a so-called *signed interval* abstract domain, as a simple extension of the classic interval domain. The main idea is to maintain two intervals for each variable, to track respectively the non-positive part and non-negative part of the value range of the variable.

4.1. Domain representation

Let $\bot_i$ denote the bottom value of the classic interval domain. Let $I^{\leq 0}$ denote $\bot_i$ or a normal interval $[a, b]$ where $b \leq 0$, and $I^{\geq 0}$ denote $\bot_i$ or a normal interval $[a, b]$ where $a \geq 0$.

As the domain representation for our *signed interval* domain, we use $\langle I^{\leq 0}, I^{\geq 0} \rangle$ to represent the signed interval value of a variable x, which means that $x \in I^{\leq 0} \vee x \in I^{\geq 0}$. In particular, we use $\bot_{si}$ to represent the empty signed interval value, sometimes also denoted as $\langle \bot_i, \bot_i \rangle$. And we use $\top_{si}$ to represent $\langle [-\infty, 0], [0, +\infty] \rangle$, which contains all possible values.

The galois connection between the concrete domain of powerset of real numbers

and the signed interval abstract domain can be defined as $(\alpha_{si}, \gamma_{si})$:

$$(\wp(\mathbb{R}), \subseteq) \underset{\alpha_{si}}{\overset{\gamma_{si}}{\leftrightarrows}} (SgnItvs, \sqsubseteq_{si})$$

where $SgnItvs$ is the set of all signed intervals over $\mathbb{R}$: $\{\langle [a,b],[c,d]\rangle \mid a \in \mathbb{R} \cup \{-\infty\}, b, c \in \mathbb{R}, d \in \mathbb{R} \cup \{+\infty\}, a \le b \le 0, 0 \le c \le d\} \cup \{\langle \bot_i, [c,d]\rangle \mid c \in \mathbb{R}, d \in \mathbb{R} \cup \{+\infty\}, 0 < c \le d\} \cup \{\langle [a,b], \bot_i\rangle \mid a \in \mathbb{R} \cup \{-\infty\}, b \in \mathbb{R}, a \le b < 0\} \cup \{\bot_{si}\}$. $SgnItvs$ forms a complete lattice $(SgnItvs, \sqsubseteq_{si}, \sqcap_{si}, \sqcup_{si}, \bot_{si}, \top_{si})$, where $\top_{si} = \langle [-\infty, 0], [0, +\infty]\rangle$ denotes $\mathbb{R}$, and $\bot_{si}$ denotes empty set. The concretization function $\gamma_{si} \in SgnItvs \to \wp(\mathbb{R})$ is defined as:

$$\gamma_{si}(\mathbf{I}) \overset{\text{def}}{=} \begin{cases} \emptyset & \text{if } \mathbf{I} = \bot_{si} \\ \{x \in \mathbb{R} \mid c \le x \le d\} & \text{else if } \mathbf{I} = \langle \bot_i, [c,d]\rangle \text{ or } \mathbf{I} = \langle [c,d], \bot_i\rangle \\ \{x \in \mathbb{R} \mid a \le x \le b \vee c \le x \le d\} & \text{else if } \mathbf{I} = \langle [a,b], [c,d]\rangle \end{cases}$$

where $\mathbf{I} \in SgnItvs$ represents an abstract element in the signed interval domain.

The abstraction function $\alpha_{si} \in \wp(\mathbb{R}) \to SgnItvs$ is defined as:

$$\alpha_{si}(S) \overset{\text{def}}{=} \begin{cases} \bot_{si} & \text{if } S = \emptyset \\ \langle \bot_i, [\inf S, \sup S]\rangle & \text{else if } \inf S > 0 \\ \langle [\inf S, \sup S], \bot_i\rangle & \text{else if } \sup S < 0 \\ \langle [\inf S, \sup S^{\le 0}], [\inf S^{\ge 0}, \sup S]\rangle & \text{otherwise} \end{cases}$$

where $S \subseteq \mathbb{R}$ is a subset of real numbers, and $S^{\le 0} \overset{\text{def}}{=} \{x \in S \mid x \le 0\}$ while $S^{\ge 0} \overset{\text{def}}{=} \{x \in S \mid x \ge 0\}$.

In general, for the sake of representation, we use $\langle I^{\le 0}, I^{\ge 0}\rangle$ to represent an arbitrary element in $SgnItvs$. Furthermore, we introduce a normalization operator ρ to convert a pair of intervals $(I^{\le 0}, I^{\ge 0})$ into a (legal) element in $SgnItvs$:

$$\rho(I^{\le 0}, I^{\ge 0}) \overset{\text{def}}{=} \langle \dot{I}^{\le 0}, \dot{I}^{\ge 0}\rangle$$

where

$$\dot{I}^{\le 0} \overset{\text{def}}{=} \begin{cases} [0,0] & \text{if } I^{\le 0} = \bot_i \text{ and } 0 \in I^{\ge 0} \\ I^{\le 0} & \text{otherwise} \end{cases}, \quad \dot{I}^{\ge 0} \overset{\text{def}}{=} \begin{cases} [0,0] & \text{if } I^{\ge 0} = \bot_i \text{ and } 0 \in I^{\le 0} \\ I^{\ge 0} & \text{otherwise} \end{cases}.$$

For example, $\rho([-1,0], \bot_i) = \langle [-1,0], [0,0]\rangle$.

4.2. Domain operation

We design the domain operations for signed interval abstract domain, on top of that in the classic interval domain. The time and space complexity of domain operations for the signed interval abstract domain is also $O(n)$. In the following, throughout this subsection, we assume that $\langle I^{\leq 0}, I^{\geq 0}\rangle$ and $\langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle$ are (legal) elements from *SgnItvs*, and we also guarantee that the resulting elements of the defined domain operations are also (legal) elements in *SgnItvs*.

1) Lattice operations: Let $\sqsubseteq_i, \sqcap_i, \sqcup_i$ respectively denote the abstract inclusion, meet, join operation in the classic interval domain.

   - Inclusion test $\sqsubseteq_{si}$: $\langle I^{\leq 0}, I^{\geq 0}\rangle \sqsubseteq_{si} \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle$ if and only if $I^{\leq 0} \sqsubseteq_i \dot{I}^{\leq 0} \wedge I^{\geq 0} \sqsubseteq_i \dot{I}^{\geq 0}$.
   - Meet $\sqcap_{si}$: $\langle I^{\leq 0}, I^{\geq 0}\rangle \sqcap_{si} \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle \stackrel{\text{def}}{=} \langle \ddot{I}^{\leq 0}, \ddot{I}^{\geq 0}\rangle$ where

     $$\begin{cases} \ddot{I}^{\leq 0} \stackrel{\text{def}}{=} & (I^{\leq 0} \sqcap_i \dot{I}^{\leq 0}) \sqcup_i (I^{\leq 0} \sqcap_i \dot{I}^{\geq 0}) \sqcup_i (I^{\geq 0} \sqcap_i \dot{I}^{\leq 0}) \\ \ddot{I}^{\geq 0} \stackrel{\text{def}}{=} & (I^{\geq 0} \sqcap_i \dot{I}^{\geq 0}) \sqcup_i (I^{\leq 0} \sqcap_i \dot{I}^{\geq 0}) \sqcup_i (I^{\geq 0} \sqcap_i \dot{I}^{\leq 0}) \end{cases}$$

     Note that here 0 may lie in different components (i.e., the non-positive component or non-negative component) of $\langle I^{\leq 0}, I^{\geq 0}\rangle$ and $\langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle$, and thus we need consider different combinations. For example, $\langle [-5,-3], [0,5]\rangle \sqcap_{si} \langle [-2,0], [1,2]\rangle$ results in $\langle [0,0], [0,2]\rangle$.
   - Join $\sqcup_{si}$: $\langle I^{\leq 0}, I^{\geq 0}\rangle \sqcup_{si} \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle \stackrel{\text{def}}{=} \langle I^{\leq 0} \sqcup_i \dot{I}^{\leq 0}, I^{\geq 0} \sqcup_i \dot{I}^{\geq 0}\rangle$.

2) Signed interval arithmetics: Let $+_i, -_i, \times_i, /_i$ respectively denote the abstract addition, subtraction (negation), multiplication and division operation in the classic interval domain.

   - $-_{si}\langle I^{\leq 0}, I^{\geq 0}\rangle \stackrel{\text{def}}{=} \langle -_i I^{\geq 0}, -_i I^{\leq 0}\rangle$.
   - $\langle I^{\leq 0}, I^{\geq 0}\rangle +_{si} \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle \stackrel{\text{def}}{=} \langle \ddot{I}^{\leq 0}, \ddot{I}^{\geq 0}\rangle$ where

     $$\begin{cases} \ddot{I}^{\leq 0} \stackrel{\text{def}}{=} & (I^{\leq 0} +_i \dot{I}^{\leq 0}) \ \sqcup_i \ ((I^{\leq 0} +_i \dot{I}^{\geq 0}) \sqcap_i [-\infty, 0]) \sqcup_i \ ((I^{\geq 0} +_i \dot{I}^{\leq 0}) \sqcap_i [-\infty, 0]) \\ \ddot{I}^{\geq 0} \stackrel{\text{def}}{=} & (I^{\geq 0} +_i \dot{I}^{\geq 0}) \sqcup_i \ ((I^{\leq 0} +_i \dot{I}^{\geq 0}) \sqcap_i [0, +\infty]) \ \sqcup_i \ ((I^{\geq 0} +_i \dot{I}^{\leq 0}) \sqcap_i [0, +\infty]) \end{cases}$$

   - $\langle I^{\leq 0}, I^{\geq 0}\rangle -_{si} \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle \stackrel{\text{def}}{=} \langle I^{\leq 0}, I^{\geq 0}\rangle +_{si} (-_{si}(\langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle))$

- $\langle I^{\leq 0}, I^{\geq 0}\rangle \times_{si} \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle \stackrel{\text{def}}{=} \langle \ddot{I}^{\leq 0}, \ddot{I}^{\geq 0}\rangle$ where

$$\begin{cases} \ddot{I}^{\leq 0} \stackrel{\text{def}}{=} & (I^{\leq 0} \times_i \dot{I}^{\geq 0}) \sqcup_i (I^{\geq 0} \times_i \dot{I}^{\leq 0}) \\ \ddot{I}^{\geq 0} \stackrel{\text{def}}{=} & (I^{\leq 0} \times_i \dot{I}^{\leq 0}) \sqcup_i (I^{\geq 0} \times_i \dot{I}^{\geq 0}) \end{cases}$$

- $\langle I^{\leq 0}, I^{\geq 0}\rangle /_{si} \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle \stackrel{\text{def}}{=} \begin{cases} \top^{si} & if\ 0 \in \dot{I}^{\leq 0}\ or\ 0 \in \dot{I}^{\geq 0} \\ \langle \ddot{I}^{\leq 0}, \ddot{I}^{\geq 0}\rangle & otherwise \end{cases}$ where

$$\begin{cases} \ddot{I}^{\leq 0} \stackrel{\text{def}}{=} & (I^{\leq 0} /_i \dot{I}^{\geq 0}) \sqcup_i (I^{\geq 0} /_i \dot{I}^{\leq 0}) \\ \ddot{I}^{\geq 0} \stackrel{\text{def}}{=} & (I^{\leq 0} /_i \dot{I}^{\leq 0}) \sqcup_i (I^{\geq 0} /_i \dot{I}^{\geq 0}) \end{cases}$$

- $abs_{si}(\langle I^{\leq 0}, I^{\geq 0}\rangle \stackrel{\text{def}}{=} \langle \bot_i, \ddot{I}^{\geq 0}\rangle)$ where

$$\ddot{I}^{\geq 0} \stackrel{\text{def}}{=} \begin{cases} I^{\geq 0} & \text{if } I^{\leq 0} = \bot_i \\ -_i I^{\leq 0} & \text{if } I^{\geq 0} = \bot_i \\ (-_i I^{\leq 0}) \sqcup_i I^{\geq 0} & \text{otherwise} \end{cases}$$

3) Transfer functions: Let $\tau[\![\cdot]\!]^{\#}(\mathbf{B})$ denote the abstract effect of a program statement on $\mathbf{B}$ where $\mathbf{B} \in SgnItvs^n$ (wherein n denotes the number of program variables), and each $\mathbf{B}_i$ is a signed interval. Let $\bot_{sb}$ denote the bottom element in $SgnItvs^n$.

- Assignment transfer function:

$$\tau[\![x_j := e]\!]^{\#}(\mathbf{B}) \stackrel{\text{def}}{=} \begin{cases} \bot_{sb} & \text{if} \quad \mathbf{B} = \bot_{sb} \\ \mathbf{B}[x_j \leftarrow \langle I^{\leq 0}, I^{\geq 0}\rangle] & \text{otherwise} \end{cases}$$

where $\langle I^{\leq 0}, I^{\geq 0}\rangle$ is the resulting signed interval of evaluating expression e, denoted as $E^{\sharp}[\![e]\!]^{\#}(\mathbf{B})$. $E^{\sharp}[\![e]\!]^{\#}(\mathbf{B})$ can be defined based on the signed

interval arithmetic as follows by structural induction:

$$\begin{cases} E^\sharp[\![\langle I^{\leq 0}, I^{\geq 0}\rangle]\!]^\#(\mathbf{B}) \stackrel{\text{def}}{=} \langle I^{\leq 0}, I^{\geq 0}\rangle \\ E^\sharp[\![x_k]\!]^\#(\mathbf{B}) \stackrel{\text{def}}{=} \mathbf{B}[x_k] \\ E^\sharp[\![-e]\!]^\#(\mathbf{B}) \stackrel{\text{def}}{=} -^\sharp_{si} E^\sharp[\![e]\!]^\#(\mathbf{B}) \\ E^\sharp[\![e_1 + e_2]\!]^\#(\mathbf{B}) \stackrel{\text{def}}{=} E^\sharp[\![e_1]\!]^\#(\mathbf{B}) +^\sharp_{si} E^\sharp[\![e_2]\!]^\#(\mathbf{B}) \\ E^\sharp[\![e_1 - e_2]\!]^\#(\mathbf{B}) \stackrel{\text{def}}{=} E^\sharp[\![e_1]\!]^\#(\mathbf{B}) -^\sharp_{si} E^\sharp[\![e_2]\!]^\#(\mathbf{B}) \\ E^\sharp[\![e_1 \times e_2]\!]^\#(\mathbf{B}) \stackrel{\text{def}}{=} E^\sharp[\![e_1]\!]^\#(\mathbf{B}) \times^\sharp_{si} E^\sharp[\![e_2]\!]^\#(\mathbf{B}) \\ E^\sharp[\![e_1/e_2]\!]^\#(\mathbf{B}) \stackrel{\text{def}}{=} E^\sharp[\![e_1]\!]^\#(\mathbf{B})/^\sharp_{si} E^\sharp[\![e_2]\!]^\#(\mathbf{B}) \\ E^\sharp[\![\ |e|\ ]\!]^\#(\mathbf{B}) \stackrel{\text{def}}{=} abs^\sharp_{si}(E^\sharp[\![e]\!]^\#(\mathbf{B})) \end{cases}$$

- Test transfer function: Let $\langle \mathbf{x}_j^{\leq 0}, \mathbf{x}_j^{\geq 0}\rangle$ denote the signed interval for x_j in the current abstract element of the signed interval abstract domain. We consider only the following forms tests while other forms of tests can be abstracted as the following ones based on (signed) interval arithmetic, similarly to the handling of tests in the classic interval abstract domain.

$$\tau[\![x_j - c \leq 0]\!]^\#(\mathbf{B}) \stackrel{\text{def}}{=} \begin{cases} \bot_{sb} & \text{if } \mathbf{B} = \bot_{sb} \\ \bot_{sb} & \text{else if } \mathbf{x}_j^{\leq 0} = \bot_i \text{ and } \mathbf{x}_j^{\geq 0} = \bot_i \\ \bot_{sb} & \text{else if } \mathbf{x}_j^{\leq 0} \neq \bot_i \text{ and } c < \underline{\mathbf{x}_j^{\leq 0}} \\ \mathbf{B}[x_j \leftarrow \langle [\underline{\mathbf{x}_j^{\leq 0}}, c], \bot_i\rangle] & \text{else if } \mathbf{x}_j^{\leq 0} \neq \bot_i \text{ and } \underline{\mathbf{x}_j^{\leq 0}} \leq c < \overline{\mathbf{x}_j^{\leq 0}} \\ \mathbf{B}[x_j \leftarrow \langle \mathbf{x}_j^{\leq 0}, \bot_i\rangle] & \text{else if } \mathbf{x}_j^{\geq 0} \neq \bot_i \text{ and } c < \underline{\mathbf{x}_j^{\geq 0}} \\ \mathbf{B}[x_j \leftarrow \langle \mathbf{x}_j^{\leq 0}, [\underline{\mathbf{x}_j^{\geq 0}}, \min(c, \overline{\mathbf{x}_j^{\geq 0}})]\rangle] & \text{else if } \mathbf{x}_j^{\geq 0} \neq \bot_i \text{ and } c \geq \underline{\mathbf{x}_j^{\geq 0}} \\ \mathbf{B} & \text{otherwise} \end{cases}$$

$$\tau[\![-x_j + c \leq 0]\!]^\#(\mathbf{B}) \stackrel{\text{def}}{=} \begin{cases} \bot_{sb} & \text{if } \mathbf{B} = \bot_{sb} \\ \bot_{sb} & \text{else if } \mathbf{x}_j^{\leq 0} = \bot_i \text{ and } \mathbf{x}_j^{\geq 0} = \bot_i \\ \bot_{sb} & \text{else if } \mathbf{x}_j^{\geq 0} \neq \bot_i \text{ and } c > \overline{\mathbf{x}_j^{\geq 0}} \\ \mathbf{B}[x_j \leftarrow \langle \bot_i, [c, \overline{\mathbf{x}_j^{\geq 0}}]\rangle] & \text{else if } \mathbf{x}_j^{\geq 0} \neq \bot_i \text{ and } \overline{\mathbf{x}_j^{\geq 0}} \geq c > \underline{\mathbf{x}_j^{\geq 0}} \\ \mathbf{B}[x_j \leftarrow \langle \bot_i, \mathbf{x}_j^{\geq 0}\rangle] & \text{else if } \mathbf{x}_j^{\leq 0} \neq \bot_i \text{ and } c > \overline{\mathbf{x}_j^{\leq 0}} \\ \mathbf{B}[x_j \leftarrow \langle [\max(c, \underline{\mathbf{x}_j^{\leq 0}}), \overline{\mathbf{x}_j^{\leq 0}}], \mathbf{x}_j^{\geq 0}\rangle & \text{else if } \mathbf{x}_j^{\leq 0} \neq \bot_i \text{ and } c \leq \overline{\mathbf{x}_j^{\leq 0}} \\ \mathbf{B} & \text{otherwise} \end{cases}$$

4) Extrapolations: Similarly to the classic interval abstract domain, the lattice of signed intervals is of infinite height, and thus we need a widening and a narrowing operation for the domain of signed intervals.

- Widening $\triangledown_{si}$: $\bot_{si} \triangledown_{si} \langle I^{\leq 0}, I^{\geq 0}\rangle \stackrel{\text{def}}{=} \langle I^{\leq 0}, I^{\geq 0}\rangle$, $\langle I^{\leq 0}, I^{\geq 0}\rangle \triangledown_{si} \bot_{si} \stackrel{\text{def}}{=} \langle I^{\leq 0}, I^{\geq 0}\rangle$. Now, we assume that $\langle I^{\leq 0}, I^{\geq 0}\rangle \neq \bot_{si}, \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle \neq \bot_{si}$, and $\langle I^{\leq 0}, I^{\geq 0}\rangle \sqsubseteq_{si} \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle$. Then,

$$\langle I^{\leq 0}, I^{\geq 0}\rangle \triangledown_{si} \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle \stackrel{\text{def}}{=} \rho(\ddot{I}^{\leq 0}, \ddot{I}^{\geq 0})$$

where

$$\ddot{I}^{\leq 0} \stackrel{\text{def}}{=} \begin{cases} I^{\leq 0} \sqcup_i \dot{I}^{\leq 0} & \text{if } I^{\leq 0} = \bot_i \text{ or } \dot{I}^{\leq 0} = \bot_i \\ [(\underline{I^{\leq 0}} \leq \underline{\dot{I}^{\leq 0}}) \,?\, \underline{I^{\leq 0}} : -\infty,\ \ (\overline{I^{\leq 0}} \geq \overline{\dot{I}^{\leq 0}}) \,?\, \overline{I^{\leq 0}} : 0] & \text{otherwise} \end{cases}$$

$$\ddot{I}^{\geq 0} \stackrel{\text{def}}{=} \begin{cases} I^{\geq 0} \sqcup_i \dot{I}^{\geq 0} & \text{if } I^{\geq 0} = \bot_i \text{ or } \dot{I}^{\geq 0} = \bot_i \\ [(\underline{I^{\geq 0}} \leq \underline{\dot{I}^{\geq 0}}) \,?\, \underline{I^{\geq 0}} : 0,\ \ (\overline{I^{\geq 0}} \geq \overline{\dot{I}^{\geq 0}}) \,?\, \overline{I^{\geq 0}} : +\infty] & \text{otherwise} \end{cases}$$

Note that here we require the normalization operation ρ to obtain a legal resulting signed interval. For example, $\langle \bot_i, [2,3]\rangle \triangledown_{si} \langle \bot_i, [1,3]\rangle = \rho(\bot_i, [0,3]) = \langle [0,0], [0,3]\rangle$.

- Narrowing $\vartriangle_{si}$: $\bot_{si} \vartriangle_{si} \langle I^{\leq 0}, I^{\geq 0}\rangle \stackrel{\text{def}}{=} \langle I^{\leq 0}, I^{\geq 0}\rangle$, $\langle I^{\leq 0}, I^{\geq 0}\rangle \vartriangle_{si} \bot_{si} \stackrel{\text{def}}{=} \langle I^{\leq 0}, I^{\geq 0}\rangle$. Now, we assume that $\langle I^{\leq 0}, I^{\geq 0}\rangle \neq \bot_{si}, \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle \neq \bot_{si}, \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle \sqsubseteq_{si} \langle I^{\leq 0}, I^{\geq 0}\rangle$. Then,

$$\langle I^{\leq 0}, I^{\geq 0}\rangle \vartriangle_{si} \langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0}\rangle \stackrel{\text{def}}{=} \langle \ddot{I}^{\leq 0}, \ddot{I}^{\geq 0}\rangle$$

where

$$\ddot{I}^{\leq 0} \stackrel{\text{def}}{=} \begin{cases} \bot_i & \text{if } I^{\leq 0} = \bot_i \text{ or } \dot{I}^{\leq 0} = \bot_i \\ [\underline{I^{\leq 0}} = -\infty \,?\, \underline{\dot{I}^{\leq 0}} : \underline{I^{\leq 0}},\ \ \overline{I^{\leq 0}} = 0 \,?\, \overline{\dot{I}^{\leq 0}} : \overline{I^{\leq 0}}] & \text{otherwise} \end{cases}$$

$$\ddot{I}^{\geq 0} \stackrel{\text{def}}{=} \begin{cases} \bot_i & \text{if } I^{\geq 0} = \bot_i \text{ or } \dot{I}^{\geq 0} = \bot_i \\ [\underline{I^{\geq 0}} = 0 \,?\, \underline{\dot{I}^{\geq 0}} : \underline{I^{\geq 0}},\ \ \overline{I^{\geq 0}} = +\infty \,?\, \overline{\dot{I}^{\geq 0}} : \overline{I^{\geq 0}}] & \text{otherwise} \end{cases}$$

For example, $\langle [-\infty, 0], [0,5]\rangle \vartriangle_{si} \langle [-4,-1], [1,4]\rangle = \langle [-4,-1], [1,5]\rangle$.

5. Combining the AVE domain with the Signed Interval domain

The AVE abstract domain has limited expressiveness in expressing inequalities. It can only encode inequalities such as $x \geq 0$ (or $x \leq 0$) as $|x| - x = 0$ (or $|x| + x =$

0). However, when we analyze programs, programs often involve inequalities, most of which are out of the expressiveness of the AVE domain. E.g., the AVE domain can not handle inequalities $x \leq -2$ (without introducing auxiliary variables), while the information provided in $x \leq -2$ is very useful for the AVE domain, since we know variable x becomes *negative* and thus the absolute value term for x could be removed. To alleviate this problem, one natural idea is to combine the AVE domain with the classic interval domain, but the disjunction information (encoding the sign information of varaibles) captured by the AVE domain may be lost when converting to a classic interval. Hence, we propose to combine the AVE domain with the signed interval domain that we propose in Sect. 4, which will provide more precise results. For example, for $|x| = 1$, the classic interval is $[-1, 1]$, while the signed interval is $\langle[-1, -1], [1, 1]\rangle$. In this section, we describe the details on how to combine the AVE domain with the signed interval domain to perform analysis.

5.1. Using the signed interval domain to improve the precision of AVE domain

We use the signed interval domain to infer the value range for each variable at each program point. Let $\langle \mathbf{x}_j^{\leq 0}, \mathbf{x}_j^{\geq 0} \rangle$ denote the signed interval range for variable x_j. When $\mathbf{x}_j^{\leq 0} \sqsubseteq_i [0, 0]$ or $\mathbf{x}_j^{\geq 0} \sqsubseteq_i [0, 0]$, we could use the signed interval range information given by the signed interval domain, to improve the precision of the AVE domain.

- When $\mathbf{x}_j^{\leq 0} \sqsubseteq_i [0, 0]$, we know that variable x is non-negative, which indicate that $|x| = x$. Thus, we can use the non-negative information to improve the AVE element (at the same program point). In the internal HLCP representation, we add $x^- = 0$ into the HLCP system, and then use Gaussian elimination to convert the resulting system into reduced row echelon form. Furthermore, if $\underline{\mathbf{x}_j^{\geq 0}} = \overline{\mathbf{x}_j^{\geq 0}} = c$ wherein c is non-negative constant, we will add both $x^- = 0$ and $\underline{x^+} = c$ into the HLCP system.

- Similarly, when $\mathbf{x}_j^{\geq 0} \sqsubseteq_i [0, 0]$, we know that variable x is non-positive, which indicate that $|x| = -x$. Then, in the internal HLCP representation, we add $x^+ = 0$ into the HLCP system, and then use Gaussian elimination to convert the resulting system into reduced row echelon form. Furthermore, if $\underline{\mathbf{x}_j^{\leq 0}} = \overline{\mathbf{x}_j^{\leq 0}} = c$ wherein

c is non-positive constant, we will add both $x^+ = 0$ and $x^- = -c$ into the HLCP system.

5.2. Using the AVE domain to improve the precision of the signed interval domain

The signed interval domain is non-relation domain, which can only express individual signed interval range for each separate variable. Since the AVE domain can infer AVE relations among variables, we could make the AVE relations to tighten the signed interval ranges in the signed interval domain.

Absolute value programming. In recent years, much research attention has been paid to solving AVE (1) in the field of optimization. It has been shown that solving AVE (1) is an NP-hard problem [17]. Intensive iterative methods have been proposed to solve AVE (1). Meanwhile, various methods have been proposed for absolute value programming problems where absolute values appear in the constraints and the objective function [17]. Particularly, Hu et al. [18] presented several approaches for solving Linear Programming problems with Complementarity Constraints (LPCC), which is a mathematical programming problem wherein the objective function and all constraints are linear, except for the complementarity conditions. Recently, Hladík [19] proposed several outer approximations of the solution set of AVE (1). Since either LPCC or AV programming is an NP-hard non-convex non-linear programming problem, finding a global optimum is non-trivial and of high complexity.

Relaxation and linear programming. In our context, we would like to make use of AVE to help tightening variable bounds, and thus we require outer approximations of solutions while not necessarily optimum solution. We use the following approach which needs to know the upper bound of $|x|$. Assume that the signed interval for each variable x_j is not $\perp_{si}$. Note that the upper bound u of $|x|$ can be obtained from the signed interval domain:

$$u_j \stackrel{\text{def}}{=} \begin{cases} \overline{\mathbf{x}_j^{\geq 0}} & \text{if } \mathbf{x}_j^{\leq 0} = \perp_i \\ -\underline{\mathbf{x}_j^{\leq 0}} & \text{else if } \mathbf{x}_j^{\geq 0} = \perp_i \\ max(-\underline{\mathbf{x}_j^{\leq 0}}, \overline{\mathbf{x}_j^{\geq 0}}) & \text{otherwise} \end{cases}$$

First, we review the following theorem from [7].

Theorem 1. *Any AV inequality*

$$\sum_i a_i x_i + \sum_{i \neq p} b_i |x_i| + b_p |x_p| \leq c$$

where $b_p > 0$, can be equivalently reformulated as a conjunction of the following two AV inequalities

$$\begin{cases} \sum_i a_i x_i + \sum_{i \neq p} b_i |x_i| + b_p x_p \leq c \\ \sum_i a_i x_i + \sum_{i \neq p} b_i |x_i| - b_p x_p \leq c \end{cases}$$

Consider AVE (1) , i.e., $Ax + B|x| = c$. Let us split B componentwisely into 2 nonnegative matrices $B^+, B^- \geq 0$ such that $B = B^+ - B^-$ wherein $B^+_{ij} = 0 \vee B^+_{ij} = B_{ij}$ and similarly $B^-_{ij} = 0 \vee B^-_{ij} = -B_{ij}$, for $i, j \in [0, n]$ wherein n is the size of x. Now, the AVE $Ax + B|x| = c$ can be reformulated into

$$Ax + B^+|x| - B^-|x| = c$$

that is,

$$\begin{cases} Ax + B^+|x| - B^-|x| \leq c \\ -Ax - B^+|x| + B^-|x| \leq -c \end{cases}$$

According to Theorem 1, the above AV inequality system can be reformulated as the following form

$$A'x - B'|x| \leq c'$$

where $B' \geq 0$. In other words, in the above AV inequality system, the coefficients of $|x|$ are all non-positive. E.g., $\{|x| + |y| = 1\}$, i.e., $\{|x| + |y| \leq 1, -|x| - |y| \leq -1\}$, is equivalent to $\{x + y \leq 1, x - y \leq 1, -x + y \leq 1, -x - y \leq 1, -|x| - |y| \leq -1\}$.

Since $|x| \leq u$, the above AV inequality system can be relaxed as

$$A'x \leq c + B'u$$

which is a linear system. Then, we can obtain tightener non-negative interval bound for x_i (assuming non empty), i.e., $[\underline{\mathbf{x}_i^{\geq 0}}, \overline{\mathbf{x}_i^{\geq 0}}]$, by solving the following linear programming problem

$$\begin{array}{ll} \min \,/\, \max & x_i \\ s.t. & \begin{cases} A'x \leq c + B'u \\ x \in \mathbf{x}^{\leq 0} \sqcup_i \mathbf{x}^{\geq 0} \\ 0 \leq x_i \end{cases} \end{array}$$

where $\mathbf{x}^{\leq 0} \sqcup_i \mathbf{x}^{\geq 0}$ obtains the normal interval ranges for variables x (based on the signed intervals provided by the signed interval domain), and $u = max(-\underline{x^{\leq 0}}, \overline{x_i^{\geq 0}})$. And for n variables, we need solve $2n$ linear programming problems.

Note that since we relax $B'|x|$ into $B'u$, the bounds found by LP may be not the optimum bounds for variables. Moreover, when the bounds of some variables are very large or even lost after widening, the resulting bounds may be too conservative. In addition, it is costly to run $2n$ linear programs after every operation on a program involving n variables. Thus we need some alternative lightweight methods for bound tightening.

Bound Propagation. Bound propagation is a kind of constraint propagation widely used in constraint programming. Each AV equality in the AVE domain element can be used to tighten the bounds for those variables occurring in it. Consider an AV equality $\Sigma_i a_i x_i + \Sigma_i b_i |x_i| = c$ and assume that $a_k \neq 0$ or $b_k \neq 0$. Assume that from the signed interval abstract domain, we know $x_k \in \langle \mathbf{x}_k^{\leq 0}, \mathbf{x}_k^{\geq 0} \rangle$. Now we partition x_k into the following two sub-cases:

- The subcase assuming that $x_k \geq 0$: $\Sigma_i a_i x_i + \Sigma_i b_i |x_i| = c$ becomes $(a_k + b_k)x_k + \Sigma_{i \neq k} a_i x_i + \Sigma_{i \neq k} b_i |x_i| = c$, i.e.,

$$(a_k + b_k)x_k = c - (\Sigma_{i \neq k} a_i x_i + \Sigma_{i \neq k} b_i |x_i|). \tag{13}$$

  Compute the right-hand expression via the signed interval arithmetic, by substituting x_i by $\langle \mathbf{x}_i^{\leq 0}, \mathbf{x}_i^{\geq 0} \rangle$ and $|x_i|$ by $abs_{si}(\langle \mathbf{x}_i^{\leq 0}, \mathbf{x}_i^{\geq 0} \rangle)$. Suppose that this results in a signed interval $\langle I^{\leq 0}, I^{\geq 0} \rangle$ for the right-hand expression. Then,

  - if $a_k + b_k = 0$, we derive a new candidate signed interval bound

$$\begin{cases} \langle \perp_i, \perp_i \rangle & \text{if } 0 \notin \langle I^{\leq 0}, I^{\geq 0} \rangle \\ \langle \perp_i, [0, +\infty] \rangle & \text{otherwise} \end{cases}$$

  - if $a_k + b_k \neq 0$, we know that

$$x_k = \frac{\langle I^{\leq 0}, I^{\geq 0} \rangle}{a_k + b_k}$$

from which suppose that we derive a new candidate signed interval bound $\langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0} \rangle$ for variable x_k. Since we are in the case of $x_k \geq 0$, it can be further reduced to $\langle \dot{I}^{\leq 0}, \dot{I}^{\geq 0} \rangle \sqcap_{si} \langle [0,0], [0,+\infty] \rangle$.

Overall, without abuse of notations, we use the following to denote the resulting candidate signed interval bound for the case of $x_k \geq 0$:

$$\rho(\bot_k, [\underline{x'^{\geq 0}_k}, \overline{x'^{\geq 0}_k}])$$

Note that formula (13) can be also reformulated as

$$(a_k + b_k)x_k = c - (\Sigma_{i \neq k}(a_i + b_i)x_i^+ + \Sigma_{i \neq k}(-a_i + b_i)x_i^-) \tag{14}$$

Since $x_i^+ = \max(x_i, 0)$ and $x_i^- = \max(-x_i, 0)$ (as described in Sect. 2.3), the right-hand expression of 14 can be computed via the signed interval arithmetic, by substituting x_i^+ by

$$\begin{cases} \langle \bot_i, [\underline{\mathbf{x}_i^{\geq 0}}, \overline{\mathbf{x}_i^{\geq 0}}] \rangle & \text{if } \mathbf{x}_i^{\leq 0} = \bot_i \\ \langle [0,0], [\underline{\mathbf{x}_i^{\geq 0}}, \overline{\mathbf{x}_i^{\geq 0}}] \rangle & \text{otherwise} \end{cases}$$

and x_i^- by

$$\begin{cases} \langle \bot_i, [-\overline{\mathbf{x}_i^{\leq 0}}, \underline{-\mathbf{x}_i^{\leq 0}}] \rangle & \text{if } \mathbf{x}_i^{\geq 0} = \bot_i \\ \langle [0,0], [-\overline{\mathbf{x}_i^{\leq 0}}, \underline{-\mathbf{x}_i^{\leq 0}}] \rangle & \text{otherwise} \end{cases}$$

Similarly, as for (13), by separately considering the case of $a_k + b_k = 0$ and the case of $a_k + b_k \neq 0$, we can derive another resulting candidate signed interval bound for the case of $x_k \geq 0$:

$$\rho(\bot_k, [\underline{\mathbf{x}''^{\geq 0}_k}, \overline{\mathbf{x}''^{\geq 0}_k}])$$

Note that computing the new bounds for x_k, the results derived from formula (13) and (14) may be incomparable. For example, to derive a new candidate bound for y according to $y = 2x - |x|$ where $x \in \langle [-5,-2], [1,8] \rangle$, using (13) can derive $\langle [-18, 0], [0, 15] \rangle$. while using (14) can derive $\langle [-15, 0], [0, 8] \rangle$ (and using orthant enumeration over x can derive $\langle [-15,-6], [1,8] \rangle$, but which may need enumerate 2^{n-1} orthants when the right-hand expression involve $n-1$ variables).

Hence, overall, we take the following as the final resulting signed interval for x_k in the case of $x_k \geq 0$:

$$\rho(\bot_i, [\underline{\mathbf{x}_k'^{\geq 0}}, \overline{\mathbf{x}_k'^{\geq 0}}]) \sqcap_{si} \rho(\bot_i, [\underline{\mathbf{x}_k''^{\geq 0}}, \overline{\mathbf{x}_k''^{\geq 0}}])$$

- The subcase assuming that $x_k \leq 0$: $\Sigma_i a_i x_i + \Sigma_i b_i |x_i| = c$ becomes $(a_k - b_k)x_k + \Sigma_{i \neq k} a_i x_i + \Sigma_{i \neq k} b_i |x_i| = c$, which implies that

$$(a_k - b_k)x_k = c - (\Sigma_{i \neq k} a_i x_i + \Sigma_{i \neq k} b_i |x_i|) \tag{15}$$

and

$$(a_k - b_k)x_k = c - (\Sigma_{i \neq k}(a_i + b_i)x_i^+ + \Sigma_{i \neq k}(-a_i + b_i)x_i^-) \tag{16}$$

Similarly as the subcase of $x_k \geq 0$, by separately considering the case of $a_k + b_k = 0$ and the case of $a_k + b_k \neq 0$ for both (15) and (16), we can derive a candidate signed interval bound for the case of $x_k \leq 0$

$$\rho([\underline{\mathbf{x}_k'^{\leq 0}}, \overline{\mathbf{x}_k'^{\leq 0}}], \bot_i) \sqcap_{si} \rho([\underline{\mathbf{x}_k''^{\leq 0}}, \overline{\mathbf{x}_k''^{\leq 0}}], \bot_i)$$

Overall, after using AV equality $\underline{\Sigma_i a_i x_i} + \overline{\Sigma_i b_i |x_i|} = c$ to tighten the signed interval range for x_k (originally $x_k \in \langle [\underline{\mathbf{x}_k^{\leq 0}}, \overline{\mathbf{x}_k^{\leq 0}}], [\underline{\mathbf{x}_k^{\geq 0}}, \overline{\mathbf{x}_k^{\geq 0}}] \rangle$), we will derive a new bound for x_k

$$\begin{cases} \rho(\bot_i, [\underline{\mathbf{x}_k^{\geq 0}}, \overline{\mathbf{x}_k^{\geq 0}}] \sqcap_i [\underline{\mathbf{x}_k'^{\geq 0}}, \overline{\mathbf{x}_k'^{\geq 0}}] \sqcap_i [\underline{\mathbf{x}_k''^{\geq 0}}, \overline{\mathbf{x}_k''^{\geq 0}}]) & \text{if } \underline{\mathbf{x}_k} \geq 0 \\ \rho([\underline{\mathbf{x}_k^{\leq 0}}, \overline{\mathbf{x}_k^{\leq 0}}] \sqcap_i [\underline{\mathbf{x}_k'^{\leq 0}}, \overline{\mathbf{x}_k'^{\leq 0}}] \sqcap_i [\underline{\mathbf{x}_k''^{\leq 0}}, \overline{\mathbf{x}_k''^{\leq 0}}], \bot_i) & \text{if } \overline{\mathbf{x}_k} \leq 0 \\ \rho([\underline{\mathbf{x}_k^{\leq 0}}, \overline{\mathbf{x}_k^{\leq 0}}] \sqcap_i [\underline{\mathbf{x}_k'^{\leq 0}}, \overline{\mathbf{x}_k'^{\leq 0}}] \sqcap_i [\underline{\mathbf{x}_k''^{\leq 0}}, \overline{\mathbf{x}_k''^{\leq 0}}], [\underline{\mathbf{x}_k^{\geq 0}}, \overline{\mathbf{x}_k^{\geq 0}}] \sqcap_i [\underline{\mathbf{x}_k'^{\geq 0}}, \overline{\mathbf{x}_k'^{\geq 0}}] \sqcap_i [\underline{\mathbf{x}_k''^{\geq 0}}, \overline{\mathbf{x}_k''^{\geq 0}}]) & \text{otherwise} \end{cases}$$

The above process can be repeated with each variable in that AV equality and with each AV equality in the system. This process has a cubic time cost in the worst case. However, it does not necessarily provide optimum bounds. More precision can be achieved, at greater cost, by iterating the process.

Note that, in this paper, we use reduced row echelon form as the constraint representation of the AVE domain. However, via Gaussian elimination, we can combine affine equalities to obtain new affine equalities on which performing bound propagation may bring further tightening. However, generating all possible such combinations possibly requires exponential time. In [6], Feret proposes a polynomial algorithm to

tighten interval bounds via affine equalities. They use a positive representation of systems of affine equalities, as well as a triangular systems of constraints of the form $a_1x_1 + \ldots + a_nx_n \in I$ where I is an interval. Here, we may also leverage this idea from [6] to obtain such constraint forms and then perform further bound propagation (by replacing interval arithmetic used in [6] with signed interval arithmetic), but with extract costs.

5.3. Discussion on combination

To combine abstract domains, *reduced product* [20] is a commonly-used operator under the framework of abstract interpretation. Reduced product first applies the domain operations separately in each abstract domain, and then employs a so-called *reduction* step to exchange information and tighten the resulting abstract elements by using the results from the other domain. However, to find the tightest element, the overall cost of a reduction operator could be rather expensive and may even require iterations to compute a fixpoint. Hence, in practice, we often use a reduction operator that refines only partially the information tracked by the two domains [21].

In this paper, when using *AVE* to tighten *SgnItvs*, if we use absolute value programming (or linear programming with complementarity constraints), we can get the optimum tight bounds for *SgnItvs* elements, but the process is NP-hard and thus too costly. On the other hand, the approach based on relaxation with linear programming, and the approach based on bound propagation provide sound bounds but not necessarily optimum bounds. In fact, the latter two methods for bound tightening are generally incomparable to each other. Each of them may find tighter bounds than the other one in some cases. Compared with the other two approaches, the approach based on bound propagation (which can stop at any time during the iteration process) is more light-weight, and does not rely on other extra solvers (such as LP solvers). Hence, for implementing the combination of *AVE* and *SgnItvs*, we choose to use the approach based on bound propagation in an iterative manner, and strike a balance between cost and precision (by limiting the times of iterations).

Example 2. *Consider an AVE element* $\{|x| + z = 5, y - |z| = 0\}$ *with the initial interval bounds* $x, y, z \in [-\infty, +\infty]$. *In an if-branch* $if(-2 \leq x \leq 5)$, *we know* $x \in [-2, 5]$.

And we could use the technique of bound propagation to derive the tighter bounds $y \in [0,5], z \in [0,5]$ *(while using the approach based on relaxation with linear programming can only give* $y \in [0,+\infty], z \in [0,5]$ *). Moreover, since we know* $z \in [0,5]$*, the absolute value term* $|z|$ *reduces to* z*. Hence, the AVE element in the if-branch becomes* $\{|x| + z = 5, y - z = 0\}$.

6. Implementation and experimental results

We have developed a prototype domain for the AVE domain, named *rAVE*, based on Sect. 3, and a prototype domain for the signed interval domain, named *rSgnItvs*, based on Sect. 4. Both prototype domains utilize GMP library [22] to conduct exact arithmetic, using multi-precision rational numbers. *rAVE* and *rSgnItvs* are interfaced to the Apron numerical abstract domain library [23]. Our experiments were conducted using the Interproc [24] static analyzer. To assess the precision and efficiency of *rAVE* (together with *rSgnItvs*, denoted as *rAVE+rSgnItvs*), we compare the obtained invariants and performance of *rAVE* (as well as *rAVE+rSgnItvs*) with *PolkaEq* (which infers linear equalities as the linear equality domain, provided in Apron [23]) as well as our previous work *rAVI* which is a rational implementation of the domain of *linear absolute value inequalities* [7].

To demonstrate the expressiveness of *rAVE* (as well as *rAVE+rSgnItvs*), two simple programs are shown in Figs. 2-3, together with the generated invariants. The program *AVtest1* shown in Fig. 2 comes from[7]. In *AVtest1*, the initial state consists of four points that are respectively from 4 different orthants over the x-y plane: $(2,2), (-2,2), (-2,-2), (2,-2)$. The loop increases outward the values of x and y in each orthant simultaneously, along the direction $y = x$ and $y = -x$ respectively. Note that, as shown in Fig. 2, we encode linear inequalities (such as $x >= 0$) in the branch conditions by linear AV equalities (such as $|x| == x$), such that *rAVE* can recognize. At program point ①, *rAVE* can prove that $|y| = |x|$, while combining *rAVE* and rSgnItvs can infer further $|y| = |x| \wedge x, y \in \langle[-\infty,-2],[2,+\infty]\rangle$, which is equivalent to the result by *rAVI* (i.e., $|y| = |x| \wedge |x| \geq 2$). However, at program point ①, *PolkaEq* obtains no information.

real x, y;

assume $x = 2$ or $x = -2$;

assume $y = 2$ or $y = -2$;

while (*true*) {

① if ($x >= 0$ /* $|x| == x$ */) { $x := x + 1$; }

else /* $|x| == -x$ */ { $x := x - 1$; }

if ($y >= 0$ /* $|y| == y$ */) { $y := y + 1$; }

else /* $|y| == -y$ */ { $y := y - 1$; }

}

Loc	PolkaEq	rAVE	rAVE+rSgnItv	rAVI
①	$\top$ (no information)	$\|x\| = \|y\|$	$\|x\| = \|y\| \wedge$ $x, y \in \langle[-\infty, -2], [2, +\infty]\rangle$	$\|x\| = \|y\| \wedge$ $\|x\| \geq 2$

Figure 2: Program *AVtest1* (left) and the generated invariants (right)

The program *AVtest2* shown in Fig. 3 is adapted from program *AVtest1*. In *AVtest2*, the linear inequalities (such as $x < 2$) in the else-branch conditions can not be syntactically rewritten as linear AV equalities without the help of the semantics of the program. Hence, using only the AVE domain, the inferred linear AV equalities information will be lost after handling the else-branches. Finally, at program point ①, using only *rAVE* obtains no information. However, *rAVE+rSgnItvs* can infer $x \leq -2$ and $y \leq -2$ respectively after the else-branch conditions. Note that $x \leq -2$ and $y \leq -2$ implies that $|x| == -x$ and $|y| == -y$, which will be tracked by the AVE domain in the following process. Finally, at program point ①,*rAVE+rSgnItvs* can infer that $|y| = |x| \wedge x, y \in \langle[-\infty, -2], [2, +\infty]\rangle$, while *rAVI* can also prove that $|y| = |x| \wedge |x| \geq 2$, but *PolkaEq* obtains no information.

Overall, Table 1 shows the comparison of performance and resulting invariants for a selection of small examples. Programs *MotivEx*, *AVtest1*, *AVtest2* respectively correspond to those programs shown in Figs. 1-3. Other programs in Table 1 are mostly collected (or adapted) from existing work [25] [26][27][28][29], which are used for analyzing programs involving disjunctive program behaviors. More clearly, programs

real x, y;
assume $x = 2$ or $x = -2$;
assume $y = 2$ or $y = -2$;
while (*true*) {
① if ($x >= 2$) { $x := x + 1$; }
  else { $x := x - 1$; }
  if ($y >= 2$) { $y := y + 1$; }
  else { $y := y - 1$; }
}

Loc	PolkaEq	rAVE	rAVE+rSgnItv	rAVI
①	$\top$ (no information)	$\top$ (no information)	$\lvert x\rvert = \lvert y\rvert \wedge$ $x, y \in \langle[-\infty, -2], [2, +\infty]\rangle$	$\lvert x\rvert = \lvert y\rvert \wedge$ $\lvert x\rvert \geq 2$

Figure 3: Program *AVtest2* (left) and the generated invariants (right)

MotivEx, *MotivExV2*, *AVtest1*, *AVtest2*, *Synergy*, *Goc*, *Hola1* involve AV functions, *Speed2*, *Speed3*, *Speed4* involve max functions, while *SimRelu*, *Reverse*, *Recwhile*, *Speed1* involve other kinds of disjunctive behaviors (e.g., due to branch conditions) [1].

The column "#var" gives the number of variables in the program. As experimental setup, for each program, the value of the widening delay parameter for Interproc is set to 1.

Invariants. The column "#eq" for *PolkaEq* gives the number of affine equalities discovered. The column "#AVeq" for *rAVE*, *rAVE+rSgnItvs*, *rAVI* respectively gives the number of linear AV equalities discovered by the corresponding domain. The column "#FBnd" for *rAVE+rSgnItvs* gives the number of meaningful finite bounds inferred for program variables. The column "#Ineq" for *rAVI* gives the number of linear inequalities discovered.

The column "Cmp (vs. *rAVE+rSgnItvs*)" compares the invariants obtained by

[1] Note that for the sake of simplicity, in this paper, we use branch statements to encode both the AV and max functions in the benchmark programs.

Table 1: Experimental results for benchmark examples

Program		PolkaEq		rAVE		rAVE+rSgnItvs			rAVI			Cmp (vs. rAVE+rSgnItvs)		
name	#var	t(ms)	#eq	t(ms)	#AVeq	t(ms)	#AVeq	#FBnd	t(ms)	#AVeq	#Ineq	PolkaEq	rAVE	rAVI
MotivEx	2	0	0	12	2	16	2	3	8	2	0	$\sqsupset$	=	=
MotivExV2	4	0	0	16	4	16	4	4	12	4	3	$\sqsupset$	=	=
AVtest1	2	0	0	32	1	28	1	4	68	1	2	$\sqsupset$	$\sqsupset$	=
AVtest2	2	0	0	32	0	40	1	4	52	1	2	$\sqsupset$	$\sqsupset$	=
SimRelu	3	4	1	16	2	12	3	3	8	3	3	$\sqsupset$	$\sqsupset$	=
Synergy	10	8	1	60	3	80	3	6	44	4	1	$\sqsupset$	=	$\sqsubset$
Reverse	2	20	0	8	1	52	1	4	20	2	1	$\sqsupset$	=	=
Recwhile	2	40	0	20	1	64	1	4	144	2	3	$\sqsupset$	$\sqsupset$	=
Split	3	8	1	40	1	40	3	4	56	4	0	$\sqsupset$	$\sqsupset$	=
Goc	2	0	0	12	2	8	2	3	12	4	2	$\sqsupset$	$\sqsupset$	=
Hola1	3	4	0	36	3	64	3	8	164	3	4	$\sqsupset$	$\sqsupset$	=
Speed1	6	0	1	28	3	24	5	4	56	6	3	$\sqsupset$	$\sqsupset$	$\sqsubset$
Speed2	6	8	3	92	6	112	6	2	10	4	3	$\sqsupset$	=	=
Speed3	7	28	3	52	9	68	10	6	128	9	2	$\sqsupset$	$\sqsupset$	=
Speed4	6	8	1	48	3	44	5	6	112	6	7	$\sqsupset$	$\sqsupset$	$\sqsubset$

PolkaEq, *rAVE*, *rAVI* respectively with *rAVE+rSgnItvs*. "$\sqsubset$" (or "$\sqsupset$") indicates that the compared domain outputs stronger (or less precise) invariants than *rAVE+ rSgnItvs*, while "=" indicates that the generated invariants are equivalent. The results in Table 1 show that *rAVE* outputs stronger invariants than *PolkaEq* for all these examples. *rAVE* outputs 1∼9 linear AV equality invariants for each of these examples at the loop head. Note that traditional convex abstract domains (including *PolkaEq*) are not fit for the benchmark examples shown in Table 1, since these programs involve non-convex behaviors (such as absolute value functions, max functions, disjunctions, etc.) that are out of the expressiveness of convex domains. Futhermore, *rAVE+rSgnItvs* infers the same set of linear AV equalities as *rAVE* for most of the benchmark programs, but also infers more AV equalities than *rAVE* for 6 programs including AVtest2 shown in Fig. 3. Moreover, *rAVE+rSgnItvs* also infer meaningful finite bounds of signed interval range over program variables (most of which are out of the expressiveness of using only *rAVE*), as shown in the column "#FBnd".

Compared with *rAVE+rSgnItvs*, *rAVI* infers the same set of linear AV equality invariants as *rAVE+rSgnItvs* for most of the programs, but also infers more AV equalities

as well as linear inequalities than *rAVE+rSgnItvs* for some programs. Overall, *rAVI* offers more precise invariants than *rAVE+rSgnItvs* for 3 programs. For *Synergy*, *rAVI* infers more AV equalities. For *Speed1* and *Speed4*, *rAVI* infers AV inequality relations involving multiple variables, which are out of the expressiveness of *rAVE+rSgnItvs*.

Performance. All experiments are carried out on a virtual machine (using VirtualBox), with a guest OS of Ubuntu 14.04 (2GB Memory), host OS of MacOS 10.12, 16GB RAM and a Intel(R) Core(TM) i7 CPU 2.7 GHz. The column "t(ms)" presents the analysis times in milliseconds. Experimental time for each program is obtained by taking the average time of ten runnings. From Table 1, we can see that *rAVE* is less efficient than *PolkaEq*, because the time complexity of domain operations in the linear equality domain is lower than that of *rAVE*. Moreover, for these examples, *PolkaEq* does not find any interesting linear equalities, and thus its domain operations perform even faster. Similarly, without surprise, we can see that in most cases *rAVE* is more efficient than *rAVE+rSgnItvs*, and *rAVE+rSgnItvs* is more efficient than *rAVI*.

7. Related work

The original work on inferring linear equality relations among program variables was due to Karr [1] in 1970s, which is now understood as the abstract domain of linear equalities in abstract interpretation. In the recent two decades, Müller-Olm and Seidl [30] give a simplified algorithm of Karr's algorithm for computing all affine relations in affine programs, and the time complexity (for analyzing the whole program) goes down to $O(nk^3)$ where n is the program size and k is the number of program variables. Gulwani and Necula [31] introduced the technique of random interpretation and presented a polynomial-time randomized algorithm to discover linear equalities using probabilistic techniques.

In the literature, the linear equality domain has been generalized in various ways, such as the domain of convex polyhedra ($\sum_k a_k x_k \leq b$) [32] and the domain of linear congruence equalities ($\sum_k a_k x_k = b$ **mod** c) [33]. Müller-Olm and Seidl have generalized the analysis of affine relations to polynomial relations of bounded degree [3]. In another direction, Müller-Olm and Seidl [34] generalized affine relation analysis to

work for modular arithmetic. King and Sϕndergaard [35] proposed an approach for deriving invariants of congruence equations where the modulo is a power of 2. Elder et al. [4] studied the relations among several known abstract domains for affine relation analysis over variables that hold machine integers, found that the domains of Müller-Olm/Seidl [34] and King/Sϕndergaard [35] are, in general, incomparable, and provided sound interconversion methods between these domains.

This paper aims at generalizing the linear equality domain to handle certain disjunction behaviors in a program. Like most existing numerical abstract domains, the linear equality domain uses conjunctions of convex constraints as the domain representation, and thus can only represent convex sets. Until now, few existing abstract domains natively allow representing non-convex sets, e.g., congruences [36], max-plus polyhedra [37], domain lifting by max expressions [25], interval polyhedra [27], circular linear progressions[38], bit-vector-sound finite-disjunctive domains[39]. In our previous work [40], we have proposed an abstract domain of interval linear equalities, which generalizes the linear equality domain with interval coefficients (over variables). In the domain of interval linear equalities, the intersection of a domain element with each orthant gives a not-necessarily closed convex polyhedron, while in the domain of AVE, the intersection of a domain element with each orthant gives an affine space (without considering the orthant constraints, such as $x \geq 0$). However, in general, in the environment of the same set of program variables, the expressiveness of the abstract domain of AVE and that of the abstract domain of interval linear equalities (which also uses row echelon form) is incomparable. Moreover, the AVE domain enjoy optimal abstractions over domain operations, while the domain of interval linear equalities does not have optimal abstractions for most domain operations (such as the join operation).

The idea of using absolute value to design non-convex abstract domains is not new. In our previous work, we have proposed to leverage the linear constraints with absolute value to design abstract domains, such as the domain of linear absolute value inequalities [7] and the domain of octagonal constraint with absolute value [8]. The abstract domain of linear AV inequalities (AVI) is more expressive than the domain of AVE. However, an AVI abstract element has the potential to include exponential number of constraints, while the number of linear AV equalities inside an AVE abstract element

is bounded by $2n$ where n is the number of program variables. Moreover, the domain operations of the AVI domain are much more costly and requires widening to ensure the termination of fixpoint iterations. The domain of octagonal constraint with absolute value can infer only the relations of the form $\{\pm x \pm y \leq c, \pm x \pm |y| \leq d, \pm|x| \pm |y| \leq e\}$ over each pair of variables x, y, where $\pm \in \{-1, 0, 1\}$), but cannot express linear AV equalities involving coefficients that are not in $\{-1, 0, 1\}$ or involving more than two variables in an equality. In general, the expressiveness of the AVE domain and that of the domain of octagonal constraint with absolute value is incomparable.

8. Conclusion

In this paper, we propose a new abstract domain, namely *the abstract domain of linear Absolute Value Equalities (AVE)*, to infer linear equality relations among values and absolute values of program variables in a program (in the form of $\Sigma_k a_k x_k + \Sigma_k b_k |x_k| = c$), which generalizes the classic linear (technically, affine) equality abstract domain ($\Sigma_k a_k x_k = c$) [1]. The key idea behind is to employ absolute value (AV) to capture certain piecewise linear relations in the program, as a mean to deal with non-convex behaviors in the program. First, we show the equivalence between linear AV equality systems and horizontal linear complementarity problem (HLCP) systems. Then, we present the domain representation (including HLCP constraint representation and complementary generator representation) as well as domain operations (that are required for static analysis, such as meet, join, etc.) designed for AVE . Moreover, we propose a so-called *signed interval abstract domain* as an extension of the classic interval abstract domain. The main idea of the signed interval abstract domain is to use two intervals to track respectively the positive part and the negative part of the interval range for each variable. Then we propose to combine the AVE domain with the signed interval domain, to help each other to improve analysis precision. On this basis, we develop a prototype for the AVE domain using rational numbers and interface it to the Apron numerical abstract domain library. Experimental results are encouraging: The AVE domain (together with the signed interval domain) can discover interesting piecewise linear invariants (that are non-convex and out of the expressiveness of the

conventional linear equality abstract domain).

It remains for future work to test AVE on large realistic programs, and consider automatic methods to introduce auxiliary variables on the fly that can be used inside the AV function to improve the precision of AVE-based analysis.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Nos. 61872445, 62032024, 62102432).

References

[1] M. Karr, Affine relationships among variables of a program, Acta Inf. 6 (1976) 133–151.

[2] P. Cousot, R. Cousot, Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: ACM POPL'77, ACM Press, 1977, pp. 238–252.

[3] M. Müller-Olm, H. Seidl, Precise interprocedural analysis through linear algebra, in: ACM POPL'04, ACM Press, 2004, pp. 330–341.

[4] M. Elder, J. Lim, T. Sharma, T. Andersen, T. W. Reps, Abstract domains of affine relations, ACM Trans. Program. Lang. Syst. 36 (4) (2014) 11:1–11:73.

[5] Y. Zhang, L. Ren, L. Chen, Y. Xiong, S. Cheung, T. Xie, Detecting numerical bugs in neural network architectures, in: ESEC/FSE'20, ACM, 2020, pp. 826–837.

[6] J. Feret, Occurrence counting analysis for the pi-calculus, in: GETCO'00, Vol. 39(2) of Electr. Notes Theor. Comput. Sci., Elsevier, 2001, pp. 1–18.

[7] L. Chen, A. Miné, J. Wang, P. Cousot, Linear absolute value relation analysis, in: ESOP'11, Vol. 6602 of LNCS, Springer, 2011, pp. 156–175.

[8] L. Chen, J. Liu, A. Miné, D. Kapur, J. Wang, An abstract domain to infer octagonal constraints with absolute value, in: SAS'14, Vol. 8723 of LNCS, Springer, 2014, pp. 101–117.

[9] L. Chen, B. Yin, D. Wei, J. Wang, An abstract domain to infer linear absolute value equalities, in: International Symposium on Theoretical Aspects of Software Engineering, TASE 2021, Shanghai, China, August 25-27, 2021, IEEE, 2021, pp. 47–54.

[10] O. L. Mangasarian, J. S. Pang, The extended linear complementarity problem, SIAM J. Matrix Anal. Appl. 16 (2) (1995) 359–368.

[11] M. Anitescu, G. Lesaja, F. Potra, Equivaence between different formulations of the linear complementarity promblem, Optimization Methods and Software 7 (3) (1997) 265–290.

[12] B. Eaves, C. Lemke, Equivalence of lcp and pls, MATHEMATICS OF OPERATIONS RESEARCH 6 (4) (1981) 475–484.

[13] L. Chua, A.-C. Deng, Canonical piecewise-linear representation, IEEE Trans. on Circuits and Systems 35 (1) (1988) 101–111.

[14] A. Schrijver, Theory of linear and integer programming, John Wiley & Sons, Inc., 1986.

[15] A. Miné, Tutorial on static inference of numeric invariants by abstract interpretation, Found. Trends Program. Lang. 4 (3–4) (2017) 120–372.

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

[17] O. L. Mangasarian, Absolute value programming, Computational Optimization and Applications 36 (1) (2007) 43–53.

[18] J. Hu, J. Mitchell, J.-S. Pang, B. Yu, On linear programs with linear complementarity constraints, Journal of Global Optimization 53 (2012) 29—-51.

[19] M. Hladík, Bounds for the solutions of absolute value equations, Computational Optimization and Applications 69 (1) (2018) 243—-266.

[20] P. Cousot, R. Cousot, Systematic design of program analysis frameworks, in: ACM POPL'79, ACM Press, New York, 1979, pp. 269–282.

[21] A. Cortesi, G. Costantini, P. Ferrara, A survey on product operators in abstract interpretation, in: A. Banerjee, O. Danvy, K. Doh, J. Hatcliff (Eds.), Semantics, Abstract Interpretation, and Reasoning about Programs: Essays Dedicated to David A. Schmidt on the Occasion of his Sixtieth Birthday, Manhattan, Kansas, USA, 19-20th September 2013, Vol. 129 of EPTCS, 2013, pp. 325–336.

[22] Gnu multiple precision arithmetic library, `http://gmplib.org/`.

[23] B. Jeannet, A. Miné, Apron: A library of numerical abstract domains for static analysis, in: CAV'09, Vol. 5643 of LNCS, Springer, 2009, pp. 661–667.

[24] G. Lalire, M. Argoud, B. Jeannet, Interproc, `http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc/`.

[25] B. S. Gulavani, S. Gulwani, A numerical abstract domain based on expression abstraction and max operator with application in timing analysis, in: CAV'08, Vol. 5123 of LNCS, Springer-Verlag, 2008, pp. 370–384.

[26] S. Gulwani, K. K. Mehra, T. M. Chilimbi, SPEED: precise and efficient static estimation of program computational complexity, in: POPL'09, ACM, 2009, pp. 127–139.

[27] L. Chen, A. Miné, J. Wang, P. Cousot, Interval polyhedra: An abstract domain to infer interval linear relationships, in: SAS'09, Vol. 5673 of LNCS, Springer Verlag, 2009, pp. 309–325.

[28] I. Dillig, T. Dillig, B. Li, K. L. McMillan, Inductive invariant generation via abductive inference, in: OOPSLA, ACM, 2013, pp. 443–456.

[29] R. Sharma, I. Dillig, T. Dillig, A. Aiken, Simplifying loop invariant generation using splitter predicates, in: G. Gopalakrishnan, S. Qadeer (Eds.), CAV 2011, Vol. 6806 of Lecture Notes in Computer Science, Springer, 2011, pp. 703–719.

[30] M. Müller-Olm, H. Seidl, A note on Karr's algorithm, in: ICALP'04, Vol. 3142 of LNCS, Springer, 2004, pp. 1016–1028.

[31] S. Gulwani, G. Necula, Discovering affine equalities using random interpretation, in: ACM POPL'03, ACM Press, 2003, pp. 74–84.

[32] P. Cousot, N. Halbwachs, Automatic discovery of linear restraints among variables of a program, in: ACM POPL'78, ACM Press, New York, 1978, pp. 84–96.

[33] P. Granger, Static analysis of linear congruence equalities among variables of a program, in: TAPSOFT'91, Vol. 493 of LNCS, Springer-Verlag, 1991, pp. 169–192.

[34] M. Müller-Olm, H. Seidl, Analysis of modular arithmetic, ACM Trans. Program. Lang. Syst. 29 (5) (2007) 29.

[35] A. King, H. Søndergaard, Inferring congruence equations using SAT, in: CAV'08, Vol. 5123 of LNCS, Springer, 2008, pp. 281–293.

[36] P. Granger, Static analysis of arithmetical congruences, International Journal of Computer Mathematics (1989) 165–199.

[37] X. Allamigeon, S. Gaubert, E. Goubault, Inferring min and max invariants using max-plus polyhedra, in: SAS'08, Vol. 5079 of LNCS, Springer Verlag, 2008, pp. 189–204.

[38] R. Sen, Y. N. Srikant, Executable analysis using abstract interpretation with circular linear progressions, in: MEMOCODE 2007, IEEE Computer Society, 2007, pp. 39–48.

[39] T. Sharma, T. W. Reps, Sound bit-precise numerical domains, in: VMCAI 2017, Vol. 10145 of Lecture Notes in Computer Science, Springer, 2017, pp. 500–520.

[40] L. Chen, A. Miné, J. Wang, P. Cousot, An abstract domain to discover interval linear equalities, in: VMCAI'10, Vol. 5944 of LNCS, Springer, 2010, pp. 112–128.