# An Abstract Domain to Infer Linear Absolute Value Equalities

Liqian Chen[1], Banghu Yin[2], Dengping Wei[1], and Ji Wang[1,3]
[1] *College of Computer Science, National University of Defense Technology, Changhsha, China*
[2] *College of Systems Engineering, National University of Defense Technology, Changhsha, China*
[3] *State Key Laboratory of High Performance Computing, Changhsha, 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. As a lightweight relational abstract domain, it has been widely used in program analysis. However, it cannot express non-convex properties that appear naturally due to the inherent disjunctive behaviors in a program. In this paper, we introduce a new abstract domain, namely *the abstract domain of linear absolute value equalities*, 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. The new abstract domain can be used to infer piecewise linear behaviors (e.g., due to conditional branches, absolute value function calls, max/min function calls, etc.) in a program. Experimental results of our prototype are encouraging: In practice, the new 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**

## I. 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], 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 [6] [7], 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 using 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. The preliminary experimental results of the prototype implementation are promising on example programs; AVE 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.

*Motivating Example:* 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. At program point ①, the AVE domain can infer (as precise as the AVI domain [6]) that $y == |x|$, which can prove the later assertions, while PolkaEq (an implementation of the linear equality abstract domain) cannot infer any information at ①.

```
     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 | $\top$ (no information) | $y == \|x\| \wedge$ $y == \|y\|$ | $y == \|x\| \wedge$ $y == \|y\|$ |

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

The rest of the paper is organized as follows. Section II describes some preliminaries. Section III presents the new proposed abstract domain of linear absolute value equalities. Section IV presents our prototype implementation together with experimental results. Section V discusses some related work before Section VI concludes.

## II. Preliminaries

### A. 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 $c \in \mathbb{Q}^m$.

### B. 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 it follows from (3-4) that

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

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 the 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 [8].

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 [9] and piece-wise linear system [10].

### C. 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:

$$(A + B)x^+ + (B - A)x^- = c$$
$$x^+, x^- \geq 0$$
$$(x^+)^T x^- = 0.$$

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

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

## III. 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 $x_k$ ($k = 1, \dots, n$), where constants $a_k, b_k, c \in \mathbb{Q}$ are automatically inferred by the analysis.

## *A. 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 $\gamma(\mathbf{P}) = \{x \in \mathbb{Q}^n \mid Ax + B|x| = c\}$, 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).

*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 [11].

*2) Internal domain representation:* In Sect. II, 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:

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

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 $\varphi$ is said to be *normalized* if the coefficient of its leading variable $x_i^\pm$ satisfies $a_i^\pm$=1. Then, given $\varphi$ which is not normalized, its normalized form can be obtained by dividing the whole constraint $\varphi$ 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 $\forall k.a_k^+ >= 0$, 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 $\forall k.a_k^+ >= 0$, 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) Generator representation for HLCP:* By Minkowski-Weyl theorem [12], 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 [6], 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 [6] 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 [6]. 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 [6] , 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:

$$\begin{gathered} Mx^+ + Nx^- = b \\ x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0 \end{gathered}$$

and we denote its set of complementary generators as

$$G^c = (V^c, R^c).$$

## B. 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.

*1) Lattice operations:*

- Emptiness test: $\mathbf{P}$ is empty, iff $V^c = \emptyset$. 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. III-A.

Now, let $\mathbf{P}, \mathbf{P}'$ be two non-empty AVE domain elements.

- Meet: $\mathbf{P} \sqcap \mathbf{P}'$ is an AVE domain element whose HLCP constraint representation is
$$\begin{gathered} Mx^+ + Nx^- = b \\ M'x^+ + N'x^- = b' \\ x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0 \end{gathered}$$
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 $\mathbf{P}'$: $(V^c \cup V'^c, R^c \cup R'^c)$, where $(V^c, R^c)$, $(V'^c, R'^c)$ 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 $(V^c, R^c)$, $(V'^c, R'^c)$ respectively for $\mathbf{P}$ and $\mathbf{P}'$ ;
  2) Compute $(V^c \cup V'^c, R^c \cup R'^c)$, and suppose $V^c \cup V'^c = \{v_1, \ldots, v_p\}, R^c \cup R'^c = \{r_1, \ldots, r_q\}$;
  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:
  $$\begin{gathered} \hat{M}x^+ + \hat{N}x^- = \hat{b} \\ x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0 \end{gathered}$$
  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 $\mathbf{P} \sqcap \mathbf{P}' = \mathbf{P}$. 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 $\mathbf{P} \sqcap \mathbf{P}' = \mathbf{P}$.

*2) Transfer functions :* 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
$$\begin{array}{c} Mx^+ + Nx^- = b \\ (c+d)x^+ + (d-c)x^- = e \\ x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0 \end{array}$$
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
$$\begin{array}{c} M'x^+ + N'x^- = b' \\ x^+ \geq 0, x^- \geq 0, (x^+)^T x^- = 0 \end{array}$$
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) 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.

**Example 1.** *Consider the motivating example shown in Fig. 1. 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*
$$(V^c_{\mathbf{P}}, R^c_{\mathbf{P}}) = \left(\begin{pmatrix} x^+ \\ x^- \\ y^+ \\ y^- \end{pmatrix} : \left\{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}\right\}, \left\{\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}\right\}\right)$$
$$(V^c_{\mathbf{P}'}, R^c_{\mathbf{P}'}) = \left(\begin{pmatrix} x^+ \\ x^- \\ y^+ \\ y^- \end{pmatrix} : \left\{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}\right\}, \left\{\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}\right\}\right)$$
*Hence,*
$$(V^c_{\mathbf{P}} \cup V^c_{\mathbf{P}'}, R^c_{\mathbf{P}} \cup R^c_{\mathbf{P}'}) =$$
$$\left(\begin{pmatrix} x^+ \\ x^- \\ y^+ \\ y^- \end{pmatrix} : \left\{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}\right\}, \left\{\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}\right\}\right)$$
*Projecting out* $\lambda_1, \mu_1, \mu_2$ *(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*
$$\begin{array}{l} x^+ + x^- - y^+ = 0, \quad y^- = 0 \\ x^+ \geq 0,\ x^- \geq 0,\ (x^+)^T x^- = 0 \\ y^+ \geq 0,\ y^- \geq 0,\ (y^+)^T y^- = 0 \end{array}$$
*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.*

## IV. Implementation and experimental results

Our prototype domain, rAVE, is developed based on Sect. III using multi-precision rational numbers. It utilizes GMP library [13] to conduct exact arithmetics. rAVE is interfaced to the Apron numerical abstract domain library [14]. Our experiments were conducted using the Interproc [15] static analyzer. To assess the precision and efficiency of rAVE, we compare the obtained invariants and performance of rAVE with PolkaEq (which infers linear equalities as the linear equality domain, provided in Apron [14]) as well as our previous work *rAVI* which is a rational implementation of the domain of *linear absolute value inequalities* [6].

To demonstrate the expressiveness of rAVE, two simple programs are shown in Figs. 2-3, together with the generated invariants. The program *AVtest1* shown in Fig. 2 comes from [6]. In *AVtest1*, the initial state consists of four points that are respectively from 4 different orthants over the $x$-$y$ plane: $(1,1), (-1,1), (-1,-1), (1,-1)$. 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 $x \geq 0$) in the branch conditions by linear AV equalities (such as $|x| == x$), such that rAVE can recognize. At program

```
real $x, y$;
assume $x = 1$ or $x = -1$;
assume $y = 1$ or $y = -1$;
while ($true$) {
① if ($x \geq 0$ /* $|x|$ == $x$ */ ) { $x := x + 1$; }
   else      /* $|x|$ == $-x$ */  { $x := x - 1$; }
   if ($y \geq 0$ /* $|y|$ == $y$ */ ) { $y := y + 1$; }
   else      /* $|y|$ == $-y$ */   { $y := y - 1$; }
}
```

| Loc | PolkaEq | rAVE | rAVI |
|---|---|---|---|
| ① | $\top$ (no information) | $\lvert x\rvert = \lvert y\rvert$ | $\lvert x\rvert = \lvert y\rvert \land \lvert x\rvert \geq 1$ |

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

point ①, rAVE can prove that $|y| = |x|$, rAVI can prove that $|y| = |x| \land |x| \geq 1$, while PolkaEq obtains no information.

The program Synergy1 shown in Figure 3 comes from [16], but we modified it a bit, by using $lock = -1$ ( rather than the normal $lock = 0$) to denote $Unlocked$ state. Moreover, we introduce four new variables $t, s, tm1, sm1$ to denote $x - y$, $lock - 1$, $|t| - 1$(i.e., $|x - y| - 1$) and $|s| - 1$ (i.e., $|lock - 1| - 1$) respectively. Note that, as shown in Fig. 3, we also replace disequalities (such as $t \neq 0$) in the branch (or loop) conditions by linear AV equalities (such as $|tm1|$ == $tm1$), such that rAVE can recognize. Both rAVE and rAVI can prove $lock = 1$ after the loop, and finally prove that the error at program point ① is unreachable (which implies that the program is correct). However, PolkaEq cannot prove that ① is unreachable.

```
int $x, y, lock, t, s, tm1, sm1$;
$lock := -1$;    /* -1 means Unlocked */
do {
    $lock := 1$;    /* 1 means Locked */
    $x := y$;
    if (brandom) { $lock := -1$;   $y := y + 1$; }
    $t = x - y$;
    $tm1 = |t| - 1$;
} while ($t \neq 0$  /* $|tm1|$ == $tm1$ */   );
$s = lock - 1$;
$sm1 = |s| - 1$;
if ($s \neq 0$  /* $|sm1|$ == $sm1$ */  )
{   ①: error;   }
```

Figure 3. Program *Synergy1*

Overall, Table I shows the comparison of performance and resulting invariants for a selection of small examples. Programs *MotivEx*, *AVtest1*, *Synergy1* respectively correspond to those programs shown in Figs. 1-3. Programs *Complexity_cav08* and *Speed_popl09* respectively come from [17] and [18], which are used for analyzing time complexity of programs. Programs *Reverse* and *Recwhile* come from [19]. *Reverse* is a loop that reverses the sign of variable $x$ at each iteration, and *Recwhile* consists of two stages, increasing $y$ in the inner loop first and then increasing $x$ in the outer loop. And for each program, the value of the widening delay parameter for INTERPROC is set to 1. "#iter." gives the number of increasing iterations during the analysis.

**Invariants.** The column "Invariant" compares the invariants obtained. The left sub-column compares rAVE with PolkaEq while the right sub-column compares rAVI with rAVE. A "$\sqsupset$" indicates the domain on the right side outputs stronger (i.e., more precise) invariants than the domain on the left side, while a "=" indicates that the generated invariants are equivalent. The results in Table I show that rAVE outputs stronger invariants than PolkaEq for all these examples. rAVE outputs 1~6 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 I, 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.

Compared with rAVI, rAVE infers equivalent invariants as rAVI for three programs. For four programs, rAVI outputs stronger invariants than rAVE. However, we observe that the set of found linear absolute value equalities by rAVI and that found by rAVE are the same. In addition to linear absolute value equalities, rAVI also infers certain linear inequalities and linear absolute value inequalities, which are of the expressiveness of rAVE (when without introducing auxiliary variables). E.g., for *AVtest1*, rAVI can infer a linear absolute value inequality ($|x| \geq 1$) which is out of the expressiveness of rAVE (when using only the variables $x, y$ in the program).

**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 Windows 10, 16GB RAM and a Intel(R) Core(TM) i7-10710U CPU @ 1.10GHz 1.61GHz. 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 I, 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 rAVE is more efficient than rAVI.

## V. 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 [20] give a simplified algorithm of Karr's algorithm for computing all affine relations in

Table I
Experimental results for benchmark examples

| Program | PolkaEq | | rAVE | | rAVI | | Invariant | |
|---|---|---|---|---|---|---|---|---|
| | #iter. | t(ms) | #iter. | t(ms) | #iter. | t(ms) | PolkaEq vs. rAVE | rAVE vs. rAVI |
| MotivEx | 1 | 3.3 | 1 | 4.0 | 1 | 5.1 | $\sqsupset$ | = |
| AVtest1 | 3 | 7.1 | 4 | 11.2 | 3 | 12.3 | $\sqsupset$ | $\sqsupset$ |
| Complexity_cav08 | 3 | 4.4 | 4 | 14.4 | 4 | 20.7 | $\sqsupset$ | $\sqsupset$ |
| Synergy1 | 3 | 5.3 | 3 | 17.3 | 4 | 30.9 | $\sqsupset$ | = |
| Reverse | 3 | 4.0 | 3 | 5.6 | 4 | 8.7 | $\sqsupset$ | = |
| Recwhile | 3 | 3.6 | 7 | 24.5 | 7 | 31.2 | $\sqsupset$ | $\sqsupset$ |
| Speed_popl09 | 3 | 5.5 | 4 | 25.0 | 4 | 30.3 | $\sqsupset$ | $\sqsupset$ |

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 [21] 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 ($\sum_k a_k x_k \leq b$) [22] and the domain of linear congruence equalities ($\sum_k a_k x_k = b \textbf{ mod } c$) [23]. 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 [24] generalized affine relation analysis to work for modular arithmetic. King and S$\phi$ndergaard [25] 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 [24] and King/S$\phi$ndergaard [25] 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 [26], max-plus polyhedra [27], domain lifting by max expressions [17], interval polyhedra [19]. In our previous work [28], 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 [6] and the domain of octagonal constraint with absolute value [7]. 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.

## VI. 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 . 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 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. Another direction of work is to consider the combination of the AVE abstract domain with the interval abstract domain, similarly to the common combination use of the linear equality abstract domain with the interval abstract domain in practice.

## Acknowledgment

We thank Antoine Miné and Patrick Cousot for their discussions on this work. This work is supported by the National Key R&D Program of China (No. 2017YFB1001802), and the National Natural Science Foundation of China (No. 61872445).

## References

[1] M. Karr, "Affine relationships among variables of a program," *Acta Inf.*, vol. 6, pp. 133–151, 1976.

[2] P. Cousot and 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 and 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, and T. W. Reps, "Abstract domains of affine relations," *ACM Trans. Program. Lang. Syst.*, vol. 36, no. 4, pp. 11:1–11:73, 2014.

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

[6] L. Chen, A. Miné, J. Wang, and P. Cousot, "Linear absolute value relation analysis," in *ESOP'11*, ser. LNCS, vol. 6602. Springer, 2011, pp. 156–175.

[7] L. Chen, J. Liu, A. Miné, D. Kapur, and J. Wang, "An abstract domain to infer octagonal constraints with absolute value," in *SAS'14*, ser. LNCS, vol. 8723. Springer, 2014, pp. 101–117.

[8] O. L. Mangasarian and J. S. Pang, "The extended linear complementarity problem," *SIAM J. Matrix Anal. Appl.*, vol. 16, no. 2, pp. 359–368, 1995.

[9] M. Anitescu, G. Lesaja, and F. Potra, "Equivaence between different formulations of the linear complementarity promblem," *Optimization Methods and Software*, vol. 7, no. 3, pp. 265–290, 1997.

[10] B. Eaves and C. Lemke, "Equivalence of lcp and pls," *MATHEMATICS OF OPERATIONS RESEARCH*, vol. 6, no. 4, pp. 475–484, 1981.

[11] L. Chua and A.-C. Deng, "Canonical piecewise-linear representation," *IEEE Trans. on Circuits and Systems*, vol. 35, no. 1, pp. 101–111, 1988.

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

[13] "Gnu multiple precision arithmetic library," http://gmplib.org/.

[14] B. Jeannet and A. Miné, "Apron: A library of numerical abstract domains for static analysis," in *CAV'09*, ser. LNCS, vol. 5643. Springer, 2009, pp. 661–667.

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

[16] B. S. Gulavani, T. A. Henzinger, Y. Kannan, A. V. Nori, and S. K. Rajamani, "Synergy: a new algorithm for property checking," in *SIGSOFT FSE'06*. ACM Press, 2006, pp. 117–127.

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

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

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

[20] M. Müller-Olm and H. Seidl, "A note on Karr's algorithm," in *ICALP'04*, ser. LNCS, vol. 3142. Springer, 2004, pp. 1016–1028.

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

[22] P. Cousot and N. Halbwachs, "Automatic discovery of linear restraints among variables of a program," in *ACM POPL'78*. ACM Press, 1978, pp. 84–96.

[23] P. Granger, "Static analysis of linear congruence equalities among variables of a program," in *TAPSOFT'91*, ser. LNCS, vol. 493. Springer-Verlag, 1991, pp. 169–192.

[24] M. Müller-Olm and H. Seidl, "Analysis of modular arithmetic," *ACM Trans. Program. Lang. Syst.*, vol. 29, no. 5, p. 29, 2007.

[25] A. King and H. Søndergaard, "Inferring congruence equations using SAT," in *CAV'08*, ser. LNCS, vol. 5123. Springer, 2008, pp. 281–293.

[26] P. Granger, "Static analysis of arithmetical congruences," *International Journal of Computer Mathematics*, pp. 165–199, 1989.

[27] X. Allamigeon, S. Gaubert, and E. Goubault, "Inferring min and max invariants using max-plus polyhedra," in *SAS'08*, ser. LNCS, vol. 5079. Springer Verlag, 2008, pp. 189–204.

[28] L. Chen, A. Miné, J. Wang, and P. Cousot, "An abstract domain to discover interval linear equalities," in *VMCAI'10*, ser. LNCS, vol. 5944. Springer, 2010, pp. 112–128.