An Abstract Domain to Discover Interval Linear Equalities [⋆]

Liqian Chen[1,2], Antoine Miné[1,3], Ji Wang[2], and Patrick Cousot[1,4]

[1] École Normale Supérieure, Paris, France
{chen,mine,cousot}@di.ens.fr
[2] National Laboratory for Parallel and Distributed Processing, Changsha, P.R.China
wj@nudt.edu.cn
[3] CNRS, France
[4] CIMS, New York University, New York, NY, USA

Abstract. We introduce a new abstract domain, namely the domain of *Interval Linear Equalities (itvLinEqs)*, which generalizes the affine equality domain with interval coefficients by leveraging results from interval linear algebra. The representation of *itvLinEqs* is based on a row echelon system of interval linear equalities, which natively allows expressing classical linear relations as well as certain topologically non-convex (even unconnected or non-closed) properties. The row echelon form limits the expressiveness of the domain but yields polynomial-time domain operations. Interval coefficients enable a sound adaptation of *itvLinEqs* to floating-point arithmetic. *itvLinEqs* can be used to infer and propagate interval linear constraints, especially for programs involving uncertain or inexact data. The preliminary experimental results are encouraging: *itvLinEqs* can find a larger range of invariants than the affine equality domain. Moreover, *itvLinEqs* provides an efficient alternative to polyhedra-like domains.

1 Introduction

In 1976, Karr [12] developed a polynomial-time algorithm to discover affine relationships among program variables ($\sum_k a_k x_k = b$). This algorithm is also understood as an abstract domain of affine equalities under the framework of abstract interpretation [5]. The affine equality domain features that the lattice of affine equalities has finite height, thus no widening is needed to ensure termination of the analysis, which makes it suitable for certain analyses, such as precise interprocedural analysis for affine programs [18]. Up to now, the affine equality domain is still one of the most efficient relational numerical abstract domains.

In recent related work [10, 17, 18], one difficulty observed associated with the affine equality domain is that a rational implementation of this domain can lead to exponentially large numbers. To alleviate this problem, in this paper we seek to implement the affine equality domain using floating-point numbers, as we did for the convex polyhedra domain in [2]. However, simply adapting the affine equality domain to floating-point arithmetic, both soundness and precision are difficult to guarantee due to pervasive

[⋆] This work is supported by the INRIA project "Abstraction" common to CNRS and ENS in France, and by the National Natural Science Foundation of China under Grant No.60725206.

rounding errors. E.g., in the floating-point world, when normalizing the coefficient of x to be 1 in the equality $3x + y = 1$, the only way to be sound is to throw this equality away, since the new coefficient $\frac{1}{3}$ is not a representable floating-point number. Thus, a proper and natural way is to extend the affine equality domain with interval coefficients, using intervals to enclose numbers not exactly representable in floating-point.

In the analysis and verification of real-life systems, the application data in the model, especially some physical quantities, may not be known exactly, e.g., elicited by inexact methods or by expert estimation. To handle such uncertainty, the application data are often provided in terms of intervals. Moreover, in program analysis, to cope with non-linearity in programs (such as multiplication/division of expressions, floating-point arithmetic), non-linear expressions may be abstracted into linear expressions with interval coefficients through certain abstraction techniques [16]. Thus, intervals appear naturally in program expressions during static analysis.

This paper introduces an abstract domain of *interval linear equalities (itvLinEqs)*, to infer relationships of the form $\sum_k [a_k, b_k] \times x_k = [c, d]$ over program variables x_k ($k = 1, \ldots, n$), where constants $a_k, b_k, c, d \in \mathbb{R} \cup \{-\infty, +\infty\}$ are automatically inferred by the analysis. Intuitively, *itvLinEqs* is an interval extension of the affine equality domain ($\sum_k a_k x_k = b$) [12] and a restriction to equalities of our previous work on the interval polyhedra domain ($\sum_k [a_k, b_k] x_k \leq c$) [3]. *itvLinEqs* maintains a row echelon system of interval linear equalities and its domain operations can be constructed analogously to those of the affine equality domain. Like the affine equality domain, both the time and space complexity of *itvLinEqs* is polynomial in the number of program variables (respectively quartic and quadratic in the worst case).

We illustrate *itvLinEqs* for invariant generation using a motivating example shown in Fig. 1. Both the affine equality domain [12] and the convex polyhedra domain [6] will obtain no information at ①, while *itvLinEqs* obtains $x + [-2, -1]y = 1$ at ① and proves $y = [-1, -0.5]$ at ②, which indicates that neither overflow nor division by zero happens in the statement $y := 1/y + 1$.

```
real x, y;
if   random()
then    x := y + 1;
else    x := 2y + 1;
endif; ①
assume x == 0; ②
y := 1/y + 1;
```

Loc	Affine equalities/Convex polyhedra	*itvLinEqs*
①	$\top$(no information)	$x + [-2, -1]y = 1$
②	$x = 0$	$x = 0 \wedge y = [-1, -0.5]$

Fig. 1. A motivating example

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 reviews basic theory of interval linear systems. Section 4 presents the representation of *itvLinEqs*. Section 5 describes domain operations of *itvLinEqs*. Section 6 discusses reduction with the interval domain and the floating-point implementation of *itvLinEqs*. Section 7 presents initial experimental results before Section 8 concludes.

2 Related Work

Static Analysis. In the literature, the affine equality domain has been generalized in various ways, such as the domain of convex polyhedra ($\sum_k a_k x_k \leq b$) [6] and the domain of linear congruence equalities ($\sum_k a_k x_k = b$ **mod** c) [9]. Recently, Müller-Olm and Seidl have generalized the analysis of affine relations to polynomial relations of bounded degree [18]. In another direction, Gulwani and Necula [10] presented a polynomial-time randomized algorithm to discover affine equalities using probabilistic techniques.

The idea of using intervals to help the affine equality domain is not new. Feret has used a reduced product between the interval domain and the affine equality domain for the analysis of mobile systems [7]. Recently, the domain of SubPolyhedra [14] has been proposed based on delicate reductions between intervals and the affine equality domain, but allows only the constant term of the equality to be an interval.

More recently, we have proposed to use intervals in our domain of interval polyhedra ($\sum_k [a_k, b_k] x_k \leq c$) [3] which generalizes the convex polyhedra domain by using interval linear inequalities. *itvLinEqs* differs from it in the following respects:

1. *itvLinEqs* limits the constraint system to be in row echelon form, while the interval polyhedra domain has no such limit but restricts interval coefficients to be finite. E.g., $[-\infty, +\infty]x = 1$ (i.e., $x \neq 0$) is only representable in *itvLinEqs* while $\{x + y \leq 1, x + 2y \leq 1\}$ is only representable in the interval polyhedra domain.
2. Concerning the implementation, the interval polyhedra domain relies a lot on linear programming (LP). Since most state-of-the-art LP solvers are implemented using floating-point numbers, both soundness and "numerical instability" issues of floating-point LP should be carefully considered [2]. However, *itvLinEqs* avoids LP and is lightweight.

Interval Linear Algebra. The challenging problem of solving interval linear systems has received much attention in the community of interval analysis [19]. Both checking the solvability and finding the solution of an interval linear system are found to be NP-hard. Different semantics of solutions of an interval linear system have been considered, such as weak and strong solutions.

In contrast to the above community, we are interested in designing an abstract domain. Thus, we mainly focus on designing new operators for manipulating interval linear constraints according to program semantics. In addition, endpoints of interval coefficients are restricted to be finite in the above mentioned work but not in this paper, since infinite interval coefficients may appear naturally after operations such as linearization [16] and widening (Sect. 5.7) in static analysis.

3 Preliminaries

We first briefly recall basic theory and results on standard interval linear systems [19]. We extend interval linear systems with infinite interval coefficients. Let $\mathbf{x} = [\underline{x}, \overline{x}]$ be an interval with its bounds (endpoints) $\underline{x} \leq \overline{x}$. Let $\mathbb{IR}$ be the set of all real intervals $[\underline{a}, \overline{a}]$ where $\underline{a}, \overline{a} \in \mathbb{R}$. Let $\mathbb{IE}$ be the set of all intervals $[\underline{a}, \overline{a}]$ where $\underline{a} \in \mathbb{R} \cup \{-\infty\}, \overline{a} \in \mathbb{R} \cup \{+\infty\}$. Throughout the paper, intervals and other interval objects are typeset in boldface letters.

Let $\underline{A} \in (\mathbb{R} \cup \{-\infty\})^{m\times n}, \overline{A} \in (\mathbb{R} \cup \{+\infty\})^{m\times n}$ be two matrices with $\underline{A} \le \overline{A}$ where the order is defined element-wise. Then the set of matrices

$$\mathbf{A} = [\underline{A}, \overline{A}] = \{A \in \mathbb{R}^{m\times n}: \underline{A} \le A \le \overline{A}\}$$

is called an (extended) *interval matrix*, and the matrices $\underline{A}, \overline{A}$ are called its bounds. An *interval vector* is a one-column interval matrix $\mathbf{b} = \{b \in \mathbb{R}^m: \underline{b} \le b \le \overline{b}\}$, where $\underline{b} \in (\mathbb{R} \cup \{-\infty\})^m$, $\overline{b} \in (\mathbb{R} \cup \{+\infty\})^m$ and $\underline{b} \le \overline{b}$.

Let $\mathbf{A}$ be an interval matrix of size $m \times n$, $\mathbf{b}$ be an interval vector of size m, and x be a vector of variables in $\mathbb{R}^n$. The following system of interval linear equalities

$$\mathbf{A}x = \mathbf{b}$$

denotes an (extended) *interval linear system*, that is the family of all systems of linear equalities $Ax = b$ with data satisfying $A \in \mathbf{A}$, $b \in \mathbf{b}$.

Definition 1 (Weak solution). *A vector $x \in \mathbb{R}^n$ is called a* weak solution *of the interval linear system $\mathbf{A}x = \mathbf{b}$, if it satisfies $Ax = b$ for some $A \in \mathbf{A}$, $b \in \mathbf{b}$. And the set*

$$\Sigma_{\exists\exists}(\mathbf{A}, \mathbf{b}) = \{x \in \mathbb{R}^n : \exists A \in \mathbf{A}, \exists b \in \mathbf{b}. Ax = b\}$$

is said to be the weak solution set *of the system $\mathbf{A}x = \mathbf{b}$.*

The weak solution set $\Sigma_{\exists\exists}(\mathbf{A}, \mathbf{b})$ can be characterized by the following theorem.

Theorem 1. *Let $\Sigma_{j=1}^n [\underline{A}_{ij}, \overline{A}_{ij}]x_j = [\underline{b}_i, \overline{b}_i]$ be the i-th row of $\mathbf{A}x = \mathbf{b}$. Then a vector $x \in \mathbb{R}^n$ is a weak solution of $\mathbf{A}x = \mathbf{b}$ iff both linear inequalities*

$$\begin{cases} \sum_{j=1}^n A'_{ij}x_j \le \overline{b}_i \\ -\sum_{j=1}^n A''_{ij}x_j \le -\underline{b}_i \end{cases}$$

hold for all $i = 1, \ldots, m$ where A'_{ij}, A''_{ij} are defined through

$$A'_{ij} = \begin{cases} \underline{A}_{ij} & \text{if } x_j > 0 \\ 0 & \text{if } x_j = 0 \\ \overline{A}_{ij} & \text{if } x_j < 0 \end{cases} \qquad A''_{ij} = \begin{cases} \overline{A}_{ij} & \text{if } x_j > 0 \\ 0 & \text{if } x_j = 0 \\ \underline{A}_{ij} & \text{if } x_j < 0 \end{cases}$$

Theorem 1 can be derived from Theorem 2.11 in [19] that we extended to the case of infinite interval coefficients. Note that for the linear inequality $\sum_{j=1}^n A'_{ij}x_j \le \overline{b}_i$ in Theorem 1, each term $A'_{ij}x_j$ will never result in $+\infty$, since $A'_{ij} = -\infty$ may hold only when $x_j > 0$ and $A'_{ij} = +\infty$ may hold only when $x_j < 0$. Whenever one term $A'_{ij}x_j$ results in $-\infty$, the linear inequality $\sum_{j=1}^n A'_{ij}x_j \le \overline{b}_i$ defines the universal space and can be omitted from the system. The same argument holds for $-\sum_{j=1}^n A''_{ij}x_j \le -\underline{b}_i$.

Recall that a (closed) *orthant* is one of the 2^n subsets of an n-dimensional Euclidean space defined by constraining each Cartesian coordinate to be either nonnegative or nonpositive. In a given orthant, each component x_j of x keeps a constant sign, so the intersection of the weak solution set $\Sigma_{\exists\exists}(\mathbf{A}, \mathbf{b})$ with each orthant can be described as a not necessarily closed convex polyhedron. In fact, the possible non-closeness happens in a restricted way so that making it closed will add only a set of points satisfying $x_j = 0$ for some x_j. Particularly, if $\mathbf{A} \in \mathbb{IR}^{m\times n}$, the region in each closed orthant is a closed convex polyhedron [3]. In general, $\Sigma_{\exists\exists}(\mathbf{A}, \mathbf{b})$ can be *non-convex* and even *unconnected*, e.g., $[-1, 1]x = 1$ describes the set $\{x : x \in [-\infty, -1] \cup [1, +\infty]\}$.

Example 1. Given $[-\infty, +\infty]x = 1$, according to Theorem 1,

$$[-\infty, +\infty]x = 1 \Leftrightarrow \begin{cases} \{(-\infty)x \leq 1, -(+\infty)x \leq -1\} \Leftrightarrow \{-\infty \leq 1, -\infty \leq -1\} & \text{if } x > 0 \\ \{(+\infty)x \leq 1, -(-\infty)x \leq -1\} \Leftrightarrow \{-\infty \leq 1, -\infty \leq -1\} & \text{if } x < 0 \\ \{0x \leq 1, -0x \leq -1\} \Leftrightarrow \{0 \leq 1, 0 \leq -1\} & \text{if } x = 0 \end{cases}$$

To sum up, $[-\infty, +\infty]x = 1$ means $x \neq 0$ (since in the case of $x = 0$ the corresponding constraint system of $[-\infty, +\infty]x = 1$ is infeasible).

4 Representation

Now, we introduce the abstract domain of *interval linear equalities (itvLinEqs)*. The main idea of *itvLinEqs* is to use a row echelon system of interval linear equalities as its representation. The concretization of each element in *itvLinEqs* is defined as the weak solution set of the corresponding constraint system.

Constraint Normalization. Throughout this paper, we fix a variable ordering $x_1 \prec x_2 \prec \ldots \prec x_n$. $\Sigma_k[\underline{a}_k, \overline{a}_k]x_k = [\underline{b}, \overline{b}]$ is a *universal constraint* if $[\underline{b}, \overline{b}] = [-\infty, +\infty]$ or $0 \in [\underline{b}, \overline{b}] \wedge \forall k. 0 \in [\underline{a}_k, \overline{a}_k]$. We use $\Sigma_k[0, 0]x_k = [0, 0]$ as a normalized form for universal constraints. Let φ be a non-universal constraint $\Sigma_k[\underline{a}_k, \overline{a}_k]x_k = [\underline{b}, \overline{b}]$. Its *leading variable* x_i is the variable with the least index i such that $[\underline{a}_i, \overline{a}_i] \neq [0, 0]$. φ is said to be *normalized* if the interval coefficient of its leading variable x_i satisfies $[\underline{a}_i, \overline{a}_i] \in \{[0, 1], [0, +\infty], [1, c], [-1, c'], [-\infty, +\infty]\}$ where $c, c' \in \mathbb{R} \cup \{+\infty\}, c \geq 1, c' > 0$. Then, given φ which is not normalized, its normalized form can be obtained by dividing the whole constraint φ by either -1 (if $[\underline{a}_i, \overline{a}_i] = [-\infty, 0]$), $\pm\underline{a}_i$ (if $\underline{a}_i \notin \{0, -\infty\}$), or $\pm\overline{a}_i$ (if $\overline{a}_i \notin \{0, +\infty\}$). 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 $\mathbf{A}x = \mathbf{b}$ be an interval linear system with $\mathbf{A} \in \mathbb{IE}^{m \times n}$ and $\mathbf{b} \in \mathbb{IE}^m$. The system $\mathbf{A}x = \mathbf{b}$ is said to be in *row echelon* form if

1) $m = n$, and
2) Either x_i is the leading variable of the i-th row, or the i-th row is filled with zeros.

***itvLinEqs* Elements.** Each domain element $\mathbf{P}$ in *itvLinEqs* is described as an interval linear system $\mathbf{A}x = \mathbf{b}$ in row echelon form, where $\mathbf{A} \in \mathbb{IE}^{n \times n}$ and $\mathbf{b} \in \mathbb{IE}^n$. It represents the set $\gamma(\mathbf{P}) = \Sigma_{\exists\exists}(\mathbf{A}, \mathbf{b}) = \{x \in \mathbb{R}^n : \exists A \in \mathbf{A}, \exists b \in \mathbf{b}. Ax = b\}$ where each point x is a possible environment (or state), i.e., an assignment of real values to abstract variables. Some examples of *itvLinEqs* elements are shown in Fig. 2.

Row Echelon Abstraction. A system of affine equalities can be equivalently converted into row echelon form via elementary matrix transformations. Unfortunately, not all systems of interval linear equalities can be exactly expressed in row echelon form. Let $\mathbf{P}$ be an arbitrary system of interval linear equalities. We seek a system in row echelon form $\rho(\mathbf{P})$ such that $\gamma(\mathbf{P}) \subseteq \gamma(\rho(\mathbf{P}))$. Unfortunately, row echelon abstraction $\rho(\mathbf{P})$ may not be uniquely defined and the best abstraction may not exist. A row echelon

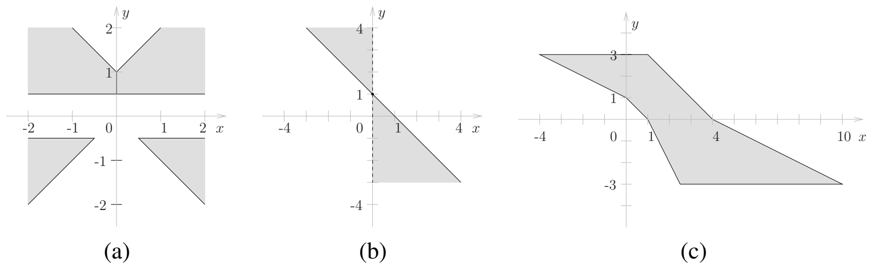

Fig. 2. Examples of *itvLinEqs* elements in 2 dimensions: (a) $\{[-1,1]x+y=[0,1],[-1,1]y=0.5\}$; (b) $\{[1,+\infty]x+y=1\}$; (c) $\{[1,2]x+[1,2]y=[2,4], y=[-3,3]\}$.

abstraction $\rho(\mathbf{P})$ for $\mathbf{P}$ can be constructed based on constraint addition (Sect. 5.3), by "adding" the constraints in $\mathbf{P}$ one by one to a row echelon system initially filled with 0.

Although row echelon abstraction may cause some loss of precision, we enforce row echelon form in *itvLinEqs* since it yields polynomial-time domain operations and avoids the "exponential growth" problem (i.e., producing exponential output) [21]. Furthermore, row echelon form can still represent exactly any affine space. Finally, row echelon form also makes it easier for us to construct our new domain by following an analogous framework to the already known domain of affine equalities.

5 Domain Operations

In this section, we discuss the implementation of most common domain operations required for static analysis over *itvLinEqs*.

5.1 Constraint Comparison

To enforce a row echelon form, we often need to compare several candidate constraints and choose the best one to take the place of the *i*-th row of the system. We first use some heuristic metrics to estimate the precision of the information contained in a normalized constraint φ.

Definition 2. *Let $\varphi : (\sum_k[\underline{a}_k,\overline{a}_k]x_k = [\underline{b},\overline{b}])$ be a normalized constraint and $[\underline{x}_k,\overline{x}_k]$ be the bounds of x_k. Then metrics $f_{weight}(\varphi), f_{width}(\varphi) \in \mathbb{R}\cup\{+\infty\}$, $f_{mark}(\varphi)\in\mathbb{R}$ are defined as:*
$1) f_{weight}(\varphi) \overset{\text{def}}{=} \sum_k(\overline{a}_k-\underline{a}_k)\times(\overline{x}_k-\underline{x}_k)+(\overline{b}-\underline{b})$,
$2) f_{width}(\varphi) \overset{\text{def}}{=} \sum_k(\overline{a}_k-\underline{a}_k)+(\overline{b}-\underline{b})$,
$3) f_{mark}(\varphi) \overset{\text{def}}{=} \sum_k \delta(\underline{a}_k,\overline{a}_k)+\delta(\underline{b},\overline{b})$, where

$$\delta(\underline{d},\overline{d}) \overset{\text{def}}{=} \begin{cases} -1 & \text{if } \underline{d}=\overline{d}, \\ +200 & \text{else if } \underline{d}=-\infty \text{ and } \overline{d}=+\infty, \\ +100 & \text{else if } \underline{d}=-\infty \text{ or } \overline{d}=+\infty, \\ 0 & \text{otherwise.} \end{cases}$$

Definition 3 (Constraint comparison). *Given two normalized constraints φ and φ', we write $\varphi \preceq \varphi'$ if $(f_{weight}(\varphi), f_{width}(\varphi), f_{mark}(\varphi)) \leq (f_{weight}(\varphi'), f_{width}(\varphi'), f_{mark}(\varphi'))$ holds in the sense of lexicographic order.*

Specifically, f_{weight} takes into account variable bounds information; f_{width} considers only the width information of interval coefficients; f_{mark} gives marks to those constraints having infinite interval coefficients. In this sense, it is guaranteed that an affine equality is always smaller for $\preceq$ than other kinds of constraints. E.g., $(x + y = 1) \preceq (x + y = [1, 2]) \preceq (x + y = [1, +\infty])$. For convenience, we say that a non-universal constraint φ is *better* than φ' if $\varphi \preceq \varphi'$ or φ' is a universal constraint. (If $\varphi \preceq \varphi'$ and $\varphi' \preceq \varphi$, we choose the best one according to the context.) Constraint comparison requires $O(n)$ time.

5.2 Projection

In program analysis, projection is an important primitive to construct assignment transfer functions and interprocedural analysis. In *itvLinEqs*, it is also useful for constraint addition and join. We use $\{\boxplus, \boxminus, \boxtimes, \boxslash\}$ for interval arithmetic operations.

We first introduce a *partial linearization* operator $\zeta(\varphi, x_j)$ to linearize the interval coefficient of x_j in φ to be a scalar.

Definition 4 (Partial linearization). *Let $\varphi\colon (\sum_k [\underline{a}_k, \overline{a}_k] \times x_k = [\underline{b}, \overline{b}])$ be an interval linear equality and $[\underline{x}_j, \overline{x}_j]$ be the bounds of x_j.*

$$\zeta(\varphi, x_j) \overset{\text{def}}{=} \left(c \times x_j + \textstyle\sum_{k \neq j} [\underline{a}_k, \overline{a}_k] \times x_k = [\underline{b}, \overline{b}] \boxminus [\underline{a}_j - c, \overline{a}_j - c] \boxtimes [\underline{x}_j, \overline{x}_j]\right)$$

where c can be any real number.

A good choice of c that causes less precision loss depends on the values of $\underline{a}_j, \overline{a}_j, \underline{x}_j, \overline{x}_j$. In practice, when $[\underline{a}_j, \overline{a}_j]$ is finite, we often choose $c = (\underline{a}_j + \overline{a}_j)/2$ that is the midpoint of $[\underline{a}_j, \overline{a}_j]$, which gives good results in most cases. If one endpoint of the interval $[\underline{a}_j, \overline{a}_j]$ is infinite, we choose the other endpoint as c. When x_j has infinite bounds, we choose the best one w.r.t. $\preceq$ between resulting constraints given by $c = \underline{a}_j$ and by $c = \overline{a}_j$.

Theorem 2 (Soundness of the partial linearization operator). *Given an interval linear equality φ and a variable $x_j \in [\underline{x}_j, \overline{x}_j]$, $\zeta(\varphi, x_j)$ soundly over-approximates φ, that is, $\forall x. (x_j \in [\underline{x}_j, \overline{x}_j] \wedge x \in \gamma(\varphi)) \Rightarrow x \in \gamma(\zeta(\varphi, x_j))$.*

Now, we consider the problem of projecting out a variable from one constraint and from a pair of constraints.

Projection by Bounds. To project out x_j from one constraint $\varphi\colon (\sum_k [\underline{a}_k, \overline{a}_k] x_k = [\underline{b}, \overline{b}])$, we simply choose $c = 0$ in $\zeta(\varphi, x_j)$. Then $\sum_{k \neq j} [\underline{a}_k, \overline{a}_k] x_k = [\underline{b}, \overline{b}] \boxminus [\underline{a}_j, \overline{a}_j] \boxtimes [\underline{x}_j, \overline{x}_j]$ will be an overapproximation of φ which does not involve x_j any more.

Projection by Combination. Let $\varphi\colon (\sum_k [\underline{a}_k, \overline{a}_k] x_k = [\underline{b}, \overline{b}])$ and $\varphi'\colon (\sum_k [\underline{a}'_k, \overline{a}'_k] x_k = [\underline{b}', \overline{b}'])$ be two constraints satisfying $[\underline{a}_j, \overline{a}_j] \neq [0, 0]$ and $[\underline{a}'_j, \overline{a}'_j] \neq [0, 0]$. To compute a constraint ϕ that does not involve x_j and satisfies $\gamma(\varphi) \cup \gamma(\varphi') \subseteq \gamma(\phi)$, we follow a similar way to Gaussian elimination. First, we convert the interval coefficient of x_j in φ to 1 (e.g., by $\zeta(\varphi, x_j)$ with $c = 1$). Assume that we get $x_j + \sum_{k \neq j} [\underline{a}''_k, \overline{a}''_k] x_k = [\underline{b}'', \overline{b}'']$.

Algorithm 1 PROJECT($\mathbf{P}, x_j$)

```
Input:   P : an itvLinEqs element $\mathbf{A}x = \mathbf{b}$;
         $x_j$ : a variable to be projected out;
Output:  P′: an itvLinEqs element that does not involve $x_j$ and satisfies $\gamma(\mathbf{P}) \subseteq \gamma(\mathbf{P}')$.

1:  $\mathbf{P}' \leftarrow \mathbf{P}$
2:  for $i = 1$ to $j - 1$ do
3:      if $([\underline{A}_{ij}, \overline{A}_{ij}] \neq [0, 0])$ then
4:          $\varphi \leftarrow \zeta(\mathbf{P}'_i, x_j)$ with $c = 0$        // projection by bounds
5:          for $k = i + 1$ to $j$ do
6:              if $([\underline{A}_{kj}, \overline{A}_{kj}] \neq [0, 0])$ then
7:                  let $\varphi'$ be the resulting constraint by combining $\mathbf{P}'_i$ and $\mathbf{P}'_k$ to project out $x_j$
8:                  if $(\varphi' \preceq \varphi)$ then $\varphi \leftarrow \varphi'$
9:          $\mathbf{P}'_i \leftarrow \varphi$   // $\varphi$ is the best constraint with leading variable $x_i$ that does not involve $x_j$
10: $\mathbf{P}'_j \leftarrow [0, 0]^{1\times(n+1)}$
11: return $\mathbf{P}'$
```

Then by substituting x_j with $([\underline{b}'', \overline{b}''] - \sum_{k\neq j}[\underline{a}''_k, \overline{a}''_k]x_k)$ in φ', the combination of φ and φ' to eliminate x_j can be achieved as

$$\phi : \left(\textstyle\sum_{k\neq j}([\underline{a}'_k, \overline{a}'_k] \boxminus [\underline{a}'_j, \overline{a}'_j] \boxtimes [\underline{a}''_k, \overline{a}''_k])x_k = [\underline{b}', \overline{b}'] \boxminus [\underline{a}'_j, \overline{a}'_j] \boxtimes [\underline{b}'', \overline{b}'']\right).$$

Particularly, when $0 \notin [\underline{a}_j, \overline{a}_j]$, converting $[\underline{a}_j, \overline{a}_j]$ to 1 in φ can be also achieved by the following theorem.

Theorem 3. *Let φ be $\sum_k[\underline{a}_k, \overline{a}_k]x_k = [\underline{b}, \overline{b}]$ with $0 \notin [\underline{a}_j, \overline{a}_j]$. Then*

$$\varphi'' : \left(x_j + \textstyle\sum_{k\neq j}([\underline{a}_k, \overline{a}_k] \boxslash [\underline{a}_j, \overline{a}_j])x_k = [\underline{b}, \overline{b}] \boxslash [\underline{a}_j, \overline{a}_j]\right)$$

is an overapproximation of φ, that is, $\gamma(\varphi) \subseteq \gamma(\varphi'')$.

To convert $[\underline{a}_j, \overline{a}_j]$ to 1 in φ, the "division" method in Theorem 3 does not depend on the bounds of x_j but requires that $0 \notin [\underline{a}_j, \overline{a}_j]$, while $\zeta(\varphi, x_j)$ with $c = 1$ is more general but depends on the bounds of x_j. Both methods may cause some loss of precision. In practice, in most cases Theorem 3 gives more precise results, especially when the bounds of x_j are coarse or even infinite. E.g., given $\varphi : ([1, 2]x + y = 2)$ with no bounds information, converting the coefficient of x to be 1, $\zeta(\varphi, x_j)$ with $c = 1$ will give a universal constraint while Theorem 3 will result in $\varphi'' : (x + [0.5, 1]y = [1, 2])$. Note that some loss of precision happens here, e.g., point (0,1) satisfies φ'' but not φ.

Projection in *itvLinEqs*. We denote as $\mathbf{P}_i$ the i-th row of $\mathbf{P}$. Based on the above projection operations on constraints, we now propose Algorithm 1 to project out x_j from an *itvLinEqs* element $\mathbf{P}$, denoted as PROJECT($\mathbf{P}, x_j$). For each row, we try various elimination methods (by bounds and by combining with other constraints) and keep the best resulting constraint w.r.t. $\preceq$. E.g., given $\mathbf{P} = \{x - y = 0, [-1, 1]y = [2, +\infty]\}$, PROJECT($\mathbf{P}, y$) results in $\{[-1, 1]x = [2, +\infty]\}$. Note that the affine space of the result of PROJECT($\mathbf{P}, x_j$) is always as precise as that given by projecting out x_j from the affine space of $\mathbf{P}$ via Gaussian elimination. The worst-case complexity of Algorithm 1 is $O(n^3)$.

5.3 Constraint Addition

We now consider the problem of "adding" a new constraint $\varphi : (\sum_k [\underline{a}_k, \overline{a}_k] x_k = [\underline{b}, \overline{b}])$ to an *itvLinEqs* element $\mathbf{P}$, denoted as $[\![\varphi]\!]^{\#}(\mathbf{P})$, i.e., to derive a row echelon abstraction $\mathbf{P}'$ such that $\gamma(\mathbf{P}) \cap \gamma(\varphi) \subseteq \gamma(\mathbf{P}')$. Let x_i be the leading variable of φ. We first initialize $\mathbf{P}'$ as $\mathbf{P}$. Then, we compare φ with the i-th row φ' of $\mathbf{P}'$ (i.e., $\varphi' = \mathbf{P}'_i$).

1) If φ is parallel to φ', i.e., $\forall k.[\underline{a}_k, \overline{a}_k] = [\underline{a}'_k, \overline{a}'_k]$, $\mathbf{P}'_i$ will be updated as $\sum_k [\underline{a}'_k, \overline{a}'_k] x_k = [\max\{\underline{b}, \underline{b}'\}, \min\{\overline{b}, \overline{b}'\}]$. If $\max\{\underline{b}, \underline{b}'\} > \min\{\overline{b}, \overline{b}'\}$, then $\mathbf{P}'$ is infeasible.
2) Otherwise, we choose the best one for $\preceq$ between φ and φ' to replace $\mathbf{P}'_i$. Next, we combine φ with φ' to eliminate x_i (Sect.5.2) and recursively "add" the resulting constraint φ'' to the updated $\mathbf{P}'$. As the index of the leading variable of the constraint to add increases strictly, this process terminates when φ'' becomes universal.

Unfortunately, in general neither $\gamma(\mathbf{P}') \subseteq \gamma(\mathbf{P})$ nor $\gamma(\mathbf{P}') \subseteq \gamma(\varphi)$ holds.

Constraint addition is used as a primitive for operations such as transfer functions, row echelon abstraction, intersection. The intersection of two *itvLinEqs* elements $\mathbf{P}$ and $\mathbf{P}'$, denoted as $\mathbf{P} \sqcap_w \mathbf{P}'$, can be implemented via "adding" the constraints from $\mathbf{P}'$ to $\mathbf{P}$ one by one, from the first row to the last. Note that $\gamma(\mathbf{P}) \cap \gamma(\mathbf{P}') \subseteq \gamma(\mathbf{P} \sqcap_w \mathbf{P}')$, but the converse may not hold. Also, $\sqcap_w$ is not commutative. Constraint addition can be computed in time $O(n^2)$ and $\mathbf{P} \sqcap_w \mathbf{P}'$ in time $O(n^3)$.

5.4 Join

In order to abstract the control-flow join, we need to design a *join* operation that returns an *itvLinEqs* element which geometrically encloses the two input *itvLinEqs* elements. However, in general, there is no best join available for *itvLinEqs* that computes the smallest *itvLinEqs* element enclosing the input arguments. In this paper, we propose a cheap join operation that we call *weak join*, which can compute the exact affine hull of the affine spaces of the input arguments (i.e., no affine relation is missed) and performs well but without precision guarantee on general interval linear constraints.

5.4.1 Approximate Convex Combination

In the affine equality domain, the join of two affine spaces can be computed via affine hull that is based on affine combination.[1] In the convex polyhedra domain, the join of two convex polyhedra can be computed via polyhedral convex hull that is based on convex combination [21].[2] Following the same idea, we seek to construct a join operation for *itvLinEqs* based on an approximate convex combination.

Given two *itvLinEqs* elements $\gamma(\mathbf{P}) = \{x \mid \mathbf{A}x = \mathbf{b}\}$ and $\gamma(\mathbf{P}') = \{x \mid \mathbf{A}'x = \mathbf{b}'\}$, based on the convex combination of points respectively from $\mathbf{P}$ and $\mathbf{P}'$ we define a set of points

$$\gamma(\mathbf{P}) \uplus \gamma(\mathbf{P}') = \left\{ x \in \mathbb{R}^n \;\middle|\; \begin{array}{c} \exists \sigma_1, \sigma_2 \in \mathbb{R}, z, z' \in \mathbb{R}^n. \\ x = \sigma_1 z + \sigma_2 z' \wedge \sigma_1 + \sigma_2 = 1 \wedge \sigma_1 \geq 0 \wedge \\ \mathbf{A}z = \mathbf{b} \quad \wedge \quad \mathbf{A}'z' = \mathbf{b}' \quad \wedge \sigma_2 \geq 0 \end{array} \right\}$$

[1] An *affine combination* of vectors $x_1, \ldots, x_n$ is a vector of the form $\Sigma_{i=1}^n \lambda_i x_i$ with $\Sigma_{i=1}^n \lambda_i = 1$.

[2] A *convex combination* of vectors $x_1, \ldots, x_n$ is a vector of the form $\Sigma_{i=1}^n \lambda_i x_i$ with $\Sigma_{i=1}^n \lambda_i = 1$ and $\forall i. \lambda_i \geq 0$.

It is obvious that $\gamma(\mathbf{P}) \Cup \gamma(\mathbf{P}')$ is an overapproximation of the union of $\gamma(\mathbf{P})$ (when $\sigma_1 = 1$) and $\gamma(\mathbf{P}')$ (when $\sigma_2 = 1$). To avoid the non-linear terms $\sigma_1 z$ and $\sigma_2 z'$, we introduce $y = \sigma_1 z$ as well as $y' = \sigma_2 z'$ and relax the system into

$$\left\{ x \in \mathbb{R}^n \;\middle|\; \begin{array}{l} \exists \sigma_1, \sigma_2 \in \mathbb{R}, y, y' \in \mathbb{R}^n. \\ x = y + y' \ \wedge \sigma_1 + \sigma_2 = 1 \ \wedge \sigma_1 \geq 0 \wedge \\ \mathbf{A}y = \sigma_1 \mathbf{b} \ \wedge \ \mathbf{A}'y' = \sigma_2 \mathbf{b}' \ \wedge \sigma_2 \geq 0 \end{array} \right\}$$

which can be rewritten as

$$\left\{ x \in \mathbb{R}^n \;\middle|\; \begin{array}{rl} \exists \sigma_1 \in \mathbb{R}, y \in \mathbb{R}^n. & \\ \mathbf{A}'x - \mathbf{A}'y + \mathbf{b}'\sigma_1 = \mathbf{b}' & \wedge \\ \mathbf{A}\,y \ - \mathbf{b}\,\sigma_1 = 0 & \wedge \\ \sigma_1 = [0, 1] & \end{array} \right\} \tag{1}$$

which is in row echelon form with respect to the variable ordering $x_1 \prec \ldots \prec x_n \prec y_1 \prec \ldots \prec y_n \prec \sigma_1$. Projecting out y, σ_1 from (1) in sequence (i.e., $y_1, \ldots, y_n, \sigma_1$) via the projection operation in *itvLinEqs* (see Algorithm 1) yields an *itvLinEqs* element, denoted as $\mathbf{P} \Cup_w \mathbf{P}'$. Then we have

$$\gamma(\mathbf{P}) \cup \gamma(\mathbf{P}') \subseteq \gamma(\mathbf{P}) \Cup \gamma(\mathbf{P}') \subseteq \gamma(\mathbf{P} \Cup_w \mathbf{P}').$$

Note that $\mathbf{P} \Cup_w \mathbf{P}'$ which is computed via the projection operation in *itvLinEqs* is definitely an *itvLinEqs* element, while $\gamma(\mathbf{P}) \Cup \gamma(\mathbf{P}')$ which is a point set defined via exact existential quantifiers may not be exactly representable in *itvLinEqs*.

$\mathbf{P} \Cup_w \mathbf{P}'$ will not miss any affine equality that the affine equality domain will generate through affine combination, since an affine equality is always kept when compared with other non-affine equalities according to the definition of constraint comparison $\preceq$. Moreover, $\mathbf{P} \Cup_w \mathbf{P}'$ can also generate other kinds of interesting interval linear constraints, such as linear stripes (of the form $\sum_k a_k x_k = [\underline{b}, \overline{b}]$), to take the place of those rows where no affine relation holds anymore after the join operation. In contrast to polyhedral convex hull which is of exponential time in the worst case and the result of which is always convex, $\mathbf{P} \Cup_w \mathbf{P}'$ can be achieved in polynomial time $O(n^4)$ and can generate non-convex constraints (although it may miss some linear inequalities).

5.4.2 Interval Combination

Definition 5 (Interval combination). *Given two constraints* φ': $(\sum_k [\underline{a}'_k, \overline{a}'_k] \times x_k = [\underline{b}', \overline{b}'])$ *and* φ'': $(\sum_k [\underline{a}''_k, \overline{a}''_k] \times x_k = [\underline{b}'', \overline{b}''])$, *the* interval combination *of* φ' *and* φ'' *is defined as*

$$\varphi' \uplus \varphi'' : \left(\textstyle\sum_k [\min(\underline{a}'_k, \underline{a}''_k), \max(\overline{a}'_k, \overline{a}''_k)] \times x_k = [\min(\underline{b}', \underline{b}''), \max(\overline{b}', \overline{b}'')]\right).$$

This definition straightforwardly lifts to *itvLinEqs* elements. Given two elements in *itvLinEqs* $\mathbf{P}'$ and $\mathbf{P}''$, we define $\mathbf{P}' \uplus \mathbf{P}''$ as $\mathbf{P}$ such that $\mathbf{P}_i = \mathbf{P}'_i \uplus \mathbf{P}''_i$ for all $i = 1, \ldots, n$.

Theorem 4 (Soundness of the interval combination). *Given two interval linear equalities φ' and φ'', their interval combination $\varphi' \uplus \varphi''$ soundly over-approximates the union of φ' and φ'', that is, $\gamma(\varphi') \cup \gamma(\varphi'') \subseteq \gamma(\varphi' \uplus \varphi'')$.*

Theorem 4 implies the soundness of $\uplus$ on *itvLinEqs*, i.e., $\gamma(\mathbf{P}') \cup \gamma(\mathbf{P}'') \subseteq \gamma(\mathbf{P}' \uplus \mathbf{P}'')$. $\mathbf{P} \uplus \mathbf{P}'$ can be computed in time $O(n^2)$.

5.4.3 Weak Join

Definition 6 (Weak join). *Given two itvLinEqs elements* $\mathbf{P}$ *and* $\mathbf{P}'$*, we define a* weak join *operation for the itvLinEqs domain as*

$$\mathbf{P} \sqcup_w \mathbf{P}' \overset{\text{def}}{=} (\mathbf{P} \Cup_w \mathbf{P}') \sqcap_w (\mathbf{P} \uplus \mathbf{P}').$$

Intuitively, the part $\mathbf{P} \Cup_w \mathbf{P}'$ follows a similar way as the polyhedral convex hull of the convex polyhedra domain and thus can construct some important convex constraints (such as affine equalities and linear stripes). Especially, $\mathbf{P} \Cup_w \mathbf{P}'$ can calculate the exact affine hull of the affine spaces of the input. However, for non-affine relations, in general $\mathbf{P} \Cup_w \mathbf{P}'$ has no precision guarantee, since it is implemented based on a series of projections which often depend on the bounds of variables. Thus, we use the other part $\mathbf{P} \uplus \mathbf{P}'$ to recover some precision by generating non-convex constraints based on syntactic heuristics. $\mathbf{P} \uplus \mathbf{P}'$ does not depend on the bounds of variables and can be easily implemented via the join of the interval domain. $\mathbf{P} \sqcup_w \mathbf{P}'$ can be computed in time $O(n^4)$.

Example 2. Given two *itvLinEqs* elements $\mathbf{P} = \{I = 2, J - K = 5, [-1,1]K = 1\}$ and $\mathbf{P}' = \{I = 3, J - K = 8, [-1,4]K = 2\}$. $\mathbf{P} \Cup_w \mathbf{P}' = \{3I - J + K = 1, J - K = [5,8]\}$. $\mathbf{P} \uplus \mathbf{P}' = \{I = [2,3], J - K = [5,8], [-1,4]K = [1,2]\}$. Thus, $\mathbf{P} \sqcup_w \mathbf{P}' = \{3I - J + K = 1, J - K = [5,8], [-1,4]K = [1,2]\}$. Whereas, when considering only the join of their affine spaces $\{I = 2, J - K = 5\}$ and $\{I = 3, J - K = 8\}$, affine hull gives $\{3I - J + K = 1\}$ in the affine equality domain and polyhedral convex hull gives $\{3I - J + K = 1, J - K = [5,8]\}$ in the convex polyhedra domain.

Theorem 5 (Soundness of the weak join). *Given two itvLinEqs elements* $\mathbf{P}$ *and* $\mathbf{P}'$*, the weak join* $\mathbf{P} \sqcup_w \mathbf{P}'$ *overapproximates both* $\mathbf{P}$ *and* $\mathbf{P}'$*, i.e.,* $\gamma(\mathbf{P}) \cup \gamma(\mathbf{P}') \subseteq \gamma(\mathbf{P} \sqcup_w \mathbf{P}')$.

5.5 Assignment Transfer Function

The assignment of an interval linear expression e to a variable x_j can be modeled using constraint addition, projection and variable renaming as follows:

$$[\![x_j := e]\!]^{\#}(\mathbf{P}) \overset{\text{def}}{=} (\textsc{Project}([\![x'_j - e = 0]\!]^{\#}(\mathbf{P}), x_j))[x'_j / x_j]\,.$$

The fresh variable x'_j, introduced to hold the value of the expression e, is necessary when x_j appears in e, e.g., $x := [-1,1]x + 1$. The assignment transfer function $[\![x_j := e]\!]^{\#}(\mathbf{P})$ can be computed in time $O(n^3)$ and its soundness is obvious.

5.6 Inclusion Test

The best order relation $\sqsubseteq$ on *itvLinEqs* is defined as $\mathbf{P} \sqsubseteq \mathbf{P}'$ iff $\gamma(\mathbf{P}) \subseteq \gamma(\mathbf{P}')$. Theorem 1 shows that $\sqsubseteq$ can be in principle checked by checking the inclusion in each orthant in the convex polyhedra domain. However, it may be too expensive to compute (an exponential number of linear programs). To solve this problem, we introduce an approximate order relation $\sqsubseteq_s$ on *itvLinEqs*. Given $\varphi : (\Sigma_k [\underline{a}_k, \overline{a}_k] x_k = [\underline{b}, \overline{b}])$ and $\varphi' : (\Sigma_k [\underline{a}'_k, \overline{a}'_k] x_k = [\underline{b}', \overline{b}'])$, $\varphi \sqsubseteq_s \varphi'$ iff $[\underline{b}, \overline{b}] \subseteq [\underline{b}', \overline{b}']$ and $\forall k. [\underline{a}_k, \overline{a}_k] \subseteq [\underline{a}'_k, \overline{a}'_k]$. Given two *itvLinEqs* elements $\mathbf{P}$ and $\mathbf{P}'$, $\mathbf{P} \sqsubseteq_s \mathbf{P}'$ iff for each row $\mathbf{P}'_i$ of $\mathbf{P}'$, either $\mathbf{P}'_i$ is a universal constraint or $\mathbf{P}_i \sqsubseteq_s \mathbf{P}'_i$. Then, $\mathbf{P} \sqsubseteq_s \mathbf{P}'$ implies $\mathbf{P} \sqsubseteq \mathbf{P}'$, while the converse may not hold. Checking $\mathbf{P} \sqsubseteq_s \mathbf{P}'$ requires $O(n^2)$ time.

5.7 Widening

Unlike the affine equality domain, *itvLinEqs* does not satisfy the ascending chain condition. Thus, to cope with loops, a widening [5] operator is needed to ensure the convergence of fixpoint computations.

Definition 7. *Given two interval linear equalities* $\varphi' : (\sum_k [\underline{a}'_k, \overline{a}'_k] x_k = [\underline{b}', \overline{b}'])$ *and* $\varphi'' : (\sum_k [\underline{a}''_k, \overline{a}''_k] x_k = [\underline{b}'', \overline{b}''])$*, we define the* widening *on constraints* φ' *and* φ'' *as*

$$\varphi' \bigtriangledown_{row} \varphi'' : \left(\sum_k ([\underline{a}'_k, \overline{a}'_k] \bigtriangledown_{itv} [\underline{a}''_k, \overline{a}''_k]) x_k = ([\underline{b}', \overline{b}'] \bigtriangledown_{itv} [\underline{b}'', \overline{b}''])\right)$$

where $\bigtriangledown_{itv}$ *is any widening of the interval domain [4], such as:*

$$[\underline{a}, \overline{a}] \bigtriangledown_{itv} [\underline{b}, \overline{b}] = [\underline{a} \leq \underline{b} \,?\, \underline{a} : -\infty,\ \overline{a} \geq \overline{b} \,?\, \overline{a} : +\infty]$$

Then we define the widening in the *itvLinEqs* domain as follows:

Definition 8 (Widening of *itvLinEqs*). *Given two itvLinEqs elements* $\mathbf{P}' \sqsubseteq \mathbf{P}''$*, we define the* widening *as* $\mathbf{P}' \bigtriangledown_{ile} \mathbf{P}'' \overset{\text{def}}{=} \mathbf{P}$ *where*

$$\mathbf{P}_i = \begin{cases} \mathbf{P}''_i & \text{if } \mathbf{P}''_i \text{ is an affine equality} \\ \mathbf{P}'_i \bigtriangledown_{row} \mathbf{P}''_i & \text{otherwise} \end{cases}$$

Note that if $\mathbf{P}' \sqsubseteq \mathbf{P}''$ does not hold, we use $\mathbf{P}' \bigtriangledown_{ile} (\mathbf{P}' \sqcup_w \mathbf{P}'')$ instead. The widening $\bigtriangledown_{ile}$ keeps all affine equalities from $\mathbf{P}''$, thus will not cause any precision loss on affine relations. When no affine relation holds at the i-th row, $\mathbf{P}'_i \bigtriangledown_{row} \mathbf{P}''_i$ recovers precision by capturing the stable information between a pair of evolving constraints $\mathbf{P}'_i$ and $\mathbf{P}''_i$. It is easy to check that the widening $\bigtriangledown_{ile}$ satisfies $\mathbf{P}' \sqsubseteq (\mathbf{P}' \bigtriangledown_{ile} \mathbf{P}'')$ and $\mathbf{P}'' \sqsubseteq (\mathbf{P}' \bigtriangledown_{ile} \mathbf{P}'')$. And the convergence of the widening $\bigtriangledown_{ile}$ can be guaranteed by the following two facts: 1) The lattice of affine equalities has finite height, and the number of affine equalities in $\mathbf{P}''$ is decreasing until it reaches the dimension of the affine space in the program; 2) The number of interval coefficients (including both variable coefficients and constant coefficients) in an *itvLinEq* element is at most $\frac{1}{2}n(n+3)$, and the interval widening $\bigtriangledown_{itv}$ at each position of these interval coefficients will guarantee the convergence of the non-affine part. The complexity of the widening $\bigtriangledown_{ile}$ is $O(n^2)$.

Widening with Thresholds. Widening with thresholds [1] $\bigtriangledown^T$ is a widening parameterized by a finite set of threshold values T, including $-\infty$ and $+\infty$. Widening with thresholds for the interval domain is defined as:

$$\begin{aligned}[\underline{a}, \overline{a}] \bigtriangledown^T_{itv} [\underline{b}, \overline{b}] = [&\underline{a} \leq \underline{b} \,?\, \underline{a} : \max\{\ell \in T \mid \ell \leq \underline{b}\},\\ &\overline{a} \geq \overline{b} \,?\, \overline{a} : \min\{h \in T \mid h \geq \overline{b}\}]\end{aligned}$$

By replacing $\bigtriangledown_{itv}$ with $\bigtriangledown^T_{itv}$ in $\bigtriangledown_{row}$, our widening with thresholds $\bigtriangledown^T_{row}$ lifts the interval widening with thresholds from individual variables to multiple variables in a natural way. Quite interestingly, it can guess not only the lower and upper bounds of the constant term (like augmenting the template polyhedra domain [20] with thresholds on the constant term), but also the shape (i.e., the $\sum_k [\underline{a}_k, \overline{a}_k] x_k$ part) of the stable invariants.

Example 3.

```
real $x, y$;
$x := 0.75 * y + 1$;
while $true$ do
① if $random()$
    then    $x := y + 1$;
    else    $x := 0.25 * x + 0.5 * y + 1$;
done;
```

Given the above program, after the first iteration, the input arguments of the widening at ① are $\varphi : ([1,1]x + [-0.75, -0.75]y = [1,1])$ and $\varphi' : ([1,1]x + [-1, -0.6875]y = [1, 1.25])$. $\varphi \nabla_{row} \varphi'$ results in $[1,1]x + [-\infty, +\infty]y = [1, +\infty]$. However, if we use $\pm n \pm 0.5$ ($n \in \mathbb{N}$ and $n \le 2$) together with $+\infty$ and $-\infty$ as the threshold set T, $\varphi \nabla^{T}_{row} \varphi'$ will result in $[1,1]x + [-1, -0.5]y = [1, 1.5]$, which will be stable in the subsequent iterations.

6 Implementation

Reduction with the Interval Domain. Variable bounds play a very important role in our domain. E.g., both partial linearization (Def. 4) and constraint comparison (in Sect. 5.1) rely on variable bounds. However, *itvLinEqs* itself has limited ability to infer bounds information. Thus we employ the interval domain to maintain such information.

To avoid the well-known convergence problem of interaction between reduction and widening [15], we perform reduction between the interval domain and *itvLinEqs* only in one direction, i.e., from *itvLinEqs* to the interval domain. After certain domain operations (such as test/assignment transfer functions, meet), we propagate the information from *itvLinEqs* to the interval domain to tighten the bounds. Such bound tightening is performed through constraint propagation techniques, as in [2], by exploiting the fact that each constraint can be used to tighten the bounds of those variables involved.

Floating-Point Implementation. Up to now, the whole domain of *itvLinEqs* was considered in exact arithmetic. Now, we consider the problem of implementing *itvLinEqs* using floating-point numbers, since floating-point numbers are time and memory efficient. *itvLinEqs* is mainly based on interval arithmetic, which can be easily implemented soundly via interval arithmetic with outward rounding (i.e., rounding upper bounds upward and lower bounds downward). And this is sufficient to guarantee that all domain operations implemented in floating-point in this way are sound.

However, a floating-point implementation of *itvLinEqs* may also cause other issues. First, floating-point *itvLinEqs* may miss some affine equalities due to rounding errors, that is to say, floating-point *itvLinEqs* is not necessarily strictly more powerful than the exact (rational) affine equality domain. Normalizing an interval linear equality may not be exact any more in the floating-point world, e.g., normalizing $3x + y = 1$. Also, the analysis based on floating-point *itvLinEqs* may suffer from the known stabilization problem of floating-point iterations [1]. However, the widening with thresholds can partly alleviate this problem. E.g., we can choose thresholds like $\pm 2^{\pm n} (n \in \mathbb{N})$, as the division and multiplication by these threshold values are simply shifting binary bits and are exact in most cases.

Program	Analyzer	*FP-itvLinEqs*				*polkaeq*			Result
name(#vars)	#∇delay	#iter.	#=	#≃	time(ms)	#iter.	#=	time(ms)	Invar.
Karr1(3)	1	4	1	1	13	4	1	8	>
Karr2(4)	1	1	2	1	10	1	2	7	>
GS1(4)	1	1	2	3	19	1	2	13	>
GS2(4)	1	1	2	0	9	1	2	7	=
MOS1(6)	1	8	1	1	66	8	1	33	>
MOS2(1)	1	1	1	0	3	1	1	5	=
policy1(2)	1	4	1	1	12	4	1	10	>
Karr1_f(3)	1	5	0	2	19	3	0	9	>
Deadcode(2)	1	1	1	1	4	1	0	11	>

Fig. 3. Experimental results comparing *FP-itvLinEqs* with a domain for affine equalities.

7 Experiments

We have developed a prototype domain, *FP-itvLinEqs*, using double precision floating-point numbers. *FP-itvLinEqs* is interfaced to the Apron numerical abstract domain library [11]. Our experiments were conducted using the Interproc [13] static analyzer. In order to assess the precision and efficiency of *FP-itvLinEqs*, we compare the obtained invariants and the performance of *FP-itvLinEqs* with *polkaeq* [11] which is an implementation in exact arithmetic to infer affine equalities,[3] NewPolka [11] which is an implementation in exact arithmetic of the convex polyhedra domain, as well as *itvPol* [3] which is a sound floating-point implementation of our interval polyhedra domain.

We evaluated *FP-itvLinEqs* on three sets of examples. The results are summarized in Figs. 3-5. The column "#∇delay" specifies the value of the widening delay parameter for Interproc (i.e., the number of loop iterations performed before applying the widening operator). "#iter." gives the number of increasing iterations during the analysis. "Result Invar." compares as a whole the invariants obtained. A ">" ("<", "≠") indicates that the left analysis outputs stronger (weaker, incomparable) invariants than the right analysis. "time" presents the analysis times (where ">1h" indicates a timeout) when the analyzer is run on a 1.6GHz PC with 768MB of RAM running Fedora 9.

Comparison with a Domain for Affine Equalities. We first compare *FP-itvLinEqs* with *polkaeq* on a collection of small examples for discovering affine equalities, which were obtained from [12, 18, 10, 8]. Fig. 3 summarizes the results on these examples. The number of discovered invariants is given by "#=" for affine equalities and "#≃" for other kinds of constraints. For these programs, *FP-itvLinEqs* can find all the affine relations that *polkaeq* finds, since indeed such programs involve only small integer values, thus the floating-point computation causes little or even no precision loss. *FP-itvLinEqs* also finds additional non-affine constraints. For the program Karr1_f which is the floating-point version of Karr1, the affine equalities that hold in Karr1 do not hold in Karr1_f any more, but *FP-itvLinEqs* can still find an interval linear invariant that involves 3

[3] In fact, *polkaeq* is implemented on top of NewPolka convex polyhedra rather than Karr's algorithm [12], but *polkaeq* is as expressive as Karr's algorithm.

Program	Analyzer	*FP-itvLinEqs*				NewPolka			*itvPol*				Result	
name(#vars)	#∇delay	#iter.	#≤	#≃	time	#iter.	#≤	time	#iter.	#≤	#≃	time	Invar.	
policy2(2)	1	5	3	1	20ms	6	2	22ms	5	3	0	46ms	$>$	$>$
policy3(2)	1	5	2	2	18ms	6	2	20ms	5	2	2	49ms	$>$	$<$
policy4(2)	1	5	3	1	19ms	7	1	24ms	6	2	1	59ms	$>$	$\neq$
bubblesort(4)	1	3	3	3	87ms	8	2	58ms	8	1	3	123ms	$>$	$\neq$
symmetricalstairs(2)	1	6	3	0	33ms	6	3	31ms	5	2	0	45ms	$<$	$>$
maccarthy91(3)	1	5	1	2	28ms	4	2	15ms	4	2	3	83ms	$\neq$	$<$
incdec(32)	3	8	26	12	32s	×	×	>1h	×	×	×	>1h	$>$	$>$
mesh2X2(32)	5	8	24	18	20s	×	×	>1h	7	5	3	190s	$>$	$\neq$
bigjava(44)	3	7	18	16	43s	×	×	>1h	6	6	4	1206s	$>$	$\neq$

Fig. 4. Experimental results comparing *FP-itvLinEqs* with domains for inequalities.

variables. For the program Deadcode (whose source code is $\{x := [0, 1];$ if $(x{==}2)$ then $y := 1;$ else $y := x;\})$, at the end of the program, *FP-itvLinEqs* proves $y = x$ whereas *polkaeq* can not find any affine equality.

Comparison with Domains for Inequalities. The second set of examples obtained from [8, 2, 20] is for discovering inequalities, as shown in Fig. 4. The number of discovered invariants is given by "#≤" for linear inequalities (including affine equalities and linear stripes, each of which is counted as two linear inequalities), and "#≃" for other kinds of constraints. The left sub-column of "Result Invar." compares *FP-itvLinEqs* with NewPolka while the right sub-column compares *FP-itvLinEqs* with *itvPol*. Compared with NewPolka, in most cases *FP-itvLinEqs* gives more precise results, since *FP-itvLinEqs* finds some non-convex interval linear invariants which make the overall feasible space of the invariants found (at each program point) smaller than that by NewPolka. Particularly, for large-dimension examples, NewPolka fails to complete the analysis in $1h$, while *FP-itvLinEqs* works well. Compared with *itvPol*, *FP-itvLinEqs* seems rather efficient. In fact, the efficiency difference becomes increasingly prominent when the number of variables increases. Besides, *FP-itvLinEqs* generates some invariants with infinite interval coefficients (e.g., 2 such constraints for policy3 and bubblesort, 5 for incdec and mesh2X2, 12 for bigjava) out of the reach of *itvPol*.

Widening with Thresholds. In Fig. 5, we compare *FP-itvLinEqs* using widening with thresholds and without thresholds (while we use only widening without thresholds in Figs. 3-4). Example3 corresponds to the previous example in Sect. 5.7. *ratelimiter_f* is a floating-point program extracted from a real-life system [2]. *nonlinear* is an example

Program	Analyzer	*FP-itvLinEqs*					Result
		without thresholds		with thresholds			Invar.
name(#vars)	#∇delay	#iter.	time(ms)	#iter.	#newinv.	time(ms)	
Example3(2)	1	4	12	4	1	18	$<$
ratelimiter_f(5)	2	5	88	5	2	91	$<$
nonlinear(3)	1	5	29	7	1	56	$<$

Fig. 5. Experimental results for widening with thresholds.

involving nonlinear expressions. When using widening with thresholds ($\{\pm n \pm 0.5 \mid n \in \mathbb{N}, n \leq 150\} \cup \{-\infty, +\infty\}$), *FP-itvLinEqs* finds tighter or new invariants, the number of which is given by "#*newinv*." in Fig. 5.

8 Conclusion

We have presented an abstract domain of *interval linear equalities (itvLinEqs)*, which extends the affine equality domain with interval coefficients. *itvLinEqs* can represent and manipulate interval linear constraints, which natively allows expressing classical linear relations as well as certain non-convex properties. *itvLinEqs* enforces a row echelon form of the constraint system, which enables a polynomial-time implementation. We have shown through experiments that *itvLinEqs* can find interesting interval linear invariants in practice, including commonly used affine equalities, linear stripes, linear inequalities. *itvLinEqs* provides a time and space efficient alternative to polyhedra-like domains.

Future work will consider the variable ordering in *itvLinEqs*, since it has an impact on the precision of the overall analysis. In order to choose a proper variable ordering, data dependencies between variables need to be considered. It is also possible to maintain dynamic variable ordering, e.g., different orderings in different loops. It would be also interesting to consider other heuristic strategies to choose which constraint to keep and which to drop to maintain a row echelon form, e.g., to keep those constraints appearing syntactically in the program. We also plan to improve the prototype implementation (e.g., using a sparse representation for the constraint matrix) and use *itvLinEqs* for analyzing large realistic programs. Another direction of the work is to relax the row echelon form and allow several constraints per leading variable.

Acknowledgements. We would like to thank Axel Simon for useful discussions, and the reviewers for their helpful comments.

References

1. B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *ACM PLDI'03*, pages 196–207. ACM Press, 2003.
2. L. Chen, A. Miné, and P. Cousot. A sound floating-point polyhedra abstract domain. In *APLAS'08*, volume 5356 of *LNCS*, pages 3–18. Springer Verlag, 2008.
3. L. Chen, A. Miné, J. Wang, and P. Cousot. Interval polyhedra: An abstract domain to infer interval linear relationships. In *SAS'09*, volume 5673 of *LNCS*, pages 309–325. Springer Verlag, 2009.
4. P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proc. of the 2nd International Symposium on Programming*, pages 106–130. Dunod, Paris, 1976.
5. 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*, pages 238–252. ACM Press, 1977.
6. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *ACM POPL'78*, pages 84–96. ACM Press, 1978.

7. J. Feret. Occurrence counting analysis for the pi-calculus. In *GETCO'00*, volume 39(2) of *Electr. Notes Theor. Comput. Sci.*, pages 1–18. Elsevier, 2001.
8. S. Gaubert, E. Goubault, A. Taly, and S. Zennou. Static analysis by policy iteration on relational domains. In *ESOP'07*, volume 4421 of *LNCS*, pages 237–252. Springer, 2007.
9. P. Granger. Static analysis of linear congruence equalities among variables of a program. In *TAPSOFT'91*, volume 493 of *LNCS*, pages 169–192. Springer-Verlag, 1991.
10. S. Gulwani and G. Necula. Discovering affine equalities using random interpretation. In *ACM POPL'03*, pages 74–84. ACM Press, 2003.
11. B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *CAV'09*, volume 5643 of *LNCS*, pages 661–667. Springer, 2009.
12. M. Karr. Affine relationships among variables of a program. *Acta Inf.*, 6:133–151, 1976.
13. G. Lalire, M. Argoud, and B. Jeannet. Interproc. http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc/.
14. V. Laviron and F. Logozzo. Subpolyhedra: A (more) scalable approach to infer linear inequalities. In *VMCAI'09*, volume 5403 of *LNCS*, pages 229–244. Springer, 2009.
15. A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
16. A. Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *VMCAI'06*, volume 3855 of *LNCS*, pages 348–363. Springer, 2006.
17. M. Müller-Olm and H. Seidl. A note on Karr's algorithm. In *ICALP'04*, volume 3142 of *LNCS*, pages 1016–1028. Springer, 2004.
18. M. Müller-Olm and H. Seidl. Precise interprocedural analysis through linear algebra. In *ACM POPL'04*, pages 330–341. ACM Press, 2004.
19. J. Rohn. Solvability of systems of interval linear equations and inequalities. In *Linear Optimization Problems with Inexact Data*, pages 35–77. Springer, 2006.
20. S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI'05*, volume 3385 of *LNCS*, pages 25–41. Springer Verlag, 2005.
21. A. Simon and A. King. Exploiting sparsity in polyhedral analysis. In *SAS'05*, volume 3672 of *LNCS*, pages 336–351. Springer Verlag, 2005.