Interval Polyhedra: An Abstract Domain to Infer Interval Linear Relationships ⋆

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

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

Abstract. We introduce a new numerical abstract domain, so-called *interval polyhedra (itvPol)*, to infer and propagate interval linear constraints over program variables. *itvPol*, which allows to represent constraints of the form $\sum_k [a_k, b_k]x_k \leq c$, is more expressive than the classic convex polyhedra domain and allows to express certain non-convex (even unconnected) properties. The implementation of *itvPol* can be constructed based on interval linear programming and an interval variant of Fourier-Motzkin elimination. The preliminary experimental results of our prototype are encouraging, especially for programs affected by interval uncertainty, e.g., due to uncertain input data or interval-based abstractions of disjunctive, non-linear, or floating-point expressions. To our knowledge, this is the first application of interval linear algebra to static analysis.

1 Introduction

Abstract interpretation [7] is a theory of semantics approximation. One application is to design computable abstractions and achieve a trade-off between efficiency and precision. Abstract interpretation provides a generic framework for devising static analyses to automatically infer dynamic properties of programs. The notion of an *abstract domain* is a core concept in this framework, and is used to denote a specific kind of computer-representable properties (such as a family of constraints) together with efficient manipulation algorithms to perform abstract operations (such as join, meet, widening, etc.). In particular, *numerical* abstract domains focus on numerical relationships among program variables. There exists a wide variety of numerical abstract domains with different expressiveness and complexity, such as intervals ($a \leq x \leq b$) [6], octagons ($\pm x \pm y \leq c$) [20], convex polyhedra ($\Sigma_k a_k x_k \leq b$) [10].

In the analysis and verification of hardware and software systems, after modeling or abstraction, the given application data may be inexact or affected by uncertainty, that is, they are only known to lie in certain intervals. Particularly, to analyze programs involving non-linear operations (e.g., multiplication or division of two expressions) or floating-point arithmetic, one may resort to a so-called *linearization* technique

⋆ 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.

to abstract non-linear or floating-point expressions into linear expressions with interval coefficients ($\sum_k [a_k, b_k] x_k + [c, d]$) [19, 21]. Furthermore, when analyzing numerical programs using a floating-point implementation (e.g., [4]) of a numerical abstract domain, a real number in the analyzed program might be abstracted as an interval of two neighboring floating-point numbers for soundness. Moreover, many floating-point algorithms can only output safe bounds, even when the input is an exact singleton value (e.g., adding two floating-point numbers in the floating-point interval domain). In other words, intervals appear naturally in practice. Hence, it is useful to have a numerical abstract domain that allows interval linear relationships among numerical program quantities to be maintained during the analysis.

This paper presents a new abstract domain, *interval polyhedra (itvPol)*, to infer relationships of the form $\Sigma_k [a_k, b_k] \times x_k \leq b$ over program variables x_k ($k = 1, \ldots, n$), where constants $a_k, b_k, c \in \mathbb{R}$ are automatically inferred by the analysis. Intuitively, *itvPol* is an interval version of the classic convex polyhedra domain. In general, an interval polyhedron is non-convex (even unconnected); it is the union of a family of convex polyhedra. Thus, *itvPol* can naturally encode certain disjunctive information. In this paper, we propose a method to abstract disjunctions using interval linear constraints. The *itvPol* domain is implemented based on interval linear programming and an interval variant of Fourier-Motzkin variable elimination. The preliminary experimental results of the prototype implementation are promising on benchmark programs; *itvPol* can find more precise invariants than the convex polyhedra domain without too much overhead.

The rest of the paper is organized as follows. Section 2 discusses some related work. Section 3 reviews the basic theory of interval linear systems and interval linear programming. Section 4 defines our numerical abstract domain *itvPol*. Section 5 describes the domain operations of *itvPol*. In Section 6, possible applications of the *itvPol* domain are discussed. Section 7 presents our prototype implementation together with preliminary experimental results before Section 8 concludes.

2 Related Work

Numerical Abstract Domains. Most of the current numerical abstract domains can only represent *convex* properties using a subset of standard linear constraints, which makes the analysis tractable. Examples include intervals [6], octagons [20], convex polyhedra [10], SubPolyhedra [17], etc. Few abstract domains natively allow representing non-convex sets (i.e., that are not disjunctive completion of known convex domains), e.g., congruences [12], max-plus polyhedra [2], domain lifting by max expressions [13]. To deal with disjunctions, a well-known solution is to use *disjunctive completion* [8, 9, 11] or *reduced cardinal power* [8]. Unfortunately, it can be very costly and also the widening operators for such domains are difficult to design (e.g., as discussed in [3]).

The *itvPol* domain that we introduce in this article is closest to the classic domain of convex polyhedra [10] but is strictly more expressive, since the coefficients of variables are generalized to intervals. Our domain is orthogonal to max-plus polyhedra [2] in that *itvPol* generalizes convex polyhedra [10] while max-plus polyhedra generalize octagons [20]. [1] Moreover, *itvPol* can describe even some unconnected sets.

Interval Linear Systems. Solving interval linear systems is a challenging problem in the community of interval analysis and interval linear algebra. This problem was first considered by Oettli and Prager [22] in the middle of the 1960s. And since then, this problem has received much attention [23, 24]. A deep insight of the topological and graph theoretical properties of the solution set was given in [14]. However, both checking the solvability and finding the solution set of an interval linear system are NP-hard [24]. Some algorithms have also been proposed for interval linear programming [5, 15].

In contrast to the above community, we are interested in designing an abstract domain, and thus, need to design new operators tailored to the semantics of programs.

3 Preliminaries

In this section we briefly recall the basic theory and results on interval linear systems, most of which can be found in [23, 24]. We use the following notations. Let $A \in \mathbb{R}^{m\times n}$ be a matrix. Intervals are denoted using boldface letters, such as $\mathbf{x}$, and their bounds are denoted as $\underline{x}$ and $\overline{x}$ so that $\mathbf{x} = [\underline{x}, \overline{x}]$. This notation is extended to linear algebra over intervals. Let $\mathbb{IR}$ be the set of all intervals on $\mathbb{R}$. Throughout the paper, intervals and other interval objects in interval algebra are typeset in boldface letters.

3.1 Interval Linear System

Let $\underline{A}, \overline{A} \in \mathbb{R}^{m\times n}$ be two matrices with $\underline{A} \leq \overline{A}$, where comparison operators are defined element-wise, then the set of matrices

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

is called an *interval matrix*, and the matrices $\underline{A}, \overline{A}$ are called its bounds. Let us define the *center matrix* of $\mathbf{A}$ as $A_c = \frac{1}{2}(\underline{A} + \overline{A})$ and the *radius matrix* as $\triangle_A = \frac{1}{2}(\overline{A} - \underline{A})$. Then, $\mathbf{A} = [\underline{A}, \overline{A}] = [A_c - \triangle_A, A_c + \triangle_A]$. An *interval vector* is a one-column interval matrix $\mathbf{d} = [\underline{d}, \overline{d}] = \{d \in \mathbb{R}^m : \underline{d} \leq d \leq \overline{d}\}$, where $\underline{d}, \overline{d} \in \mathbb{R}^m$ and $\underline{d} \leq \overline{d}$.

Let $\mathbf{A}$ be an $m \times n$ interval matrix and b be a vector of size m. The following system of interval linear inequalities

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

denotes an *interval linear system*, that is, the *family* of all systems of linear inequalities $Ax \leq b$ such that $A \in \mathbf{A}$.

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

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

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

[1] As an example, Fig. 1(1.a) depicts an *itvPol* element which cannot be represented by max-plus polyhedra while Fig. 1(2.b) shows a max-plus polyhedron that is not in *itvPol*.

The weak solution set of an interval linear system is characterized by the following theorem [24].

Theorem 1. *A vector $x \in \mathbb{R}^n$ is a weak solution of* $\mathbf{A}x \leq b$ *iff it satisfies* $A_c x - \triangle_A |x| \leq b$.

In general, the weak solution set can be *non-convex*, and even unconnected (Fig. 1(1)). The non-convexity can be derived from the non-linear factor $|x|$ in Theorem 1. A (closed) *orthant* is one of the 2^n subsets of an n-dimensional Euclidean space defined by constraining each Cartesian coordinate axis to be either nonnegative or nonpositive. Note that, in a given orthant, each component of x keeps a constant sign, so the intersection of the weak solution set with each orthant can be described as a (possibly empty) convex polyhedron. However, not all unions of convex polyhedra with at most one in each orthant can be exactly encoded as interval linear systems (e.g., Fig. 1(2.c)).

The narrowest interval vector $\mathbf{x}^H$ containing the weak solution set $\Sigma_\exists(\mathbf{A}, b)$, is called the *interval hull* of $\Sigma_\exists(\mathbf{A}, b)$, i.e., $\mathbf{x}^H_k = [\underline{x}^H_k, \overline{x}^H_k]$, where $\underline{x}^H_k = \min\{x_k : x \in \Sigma_\exists(\mathbf{A}, b)\}$, $\overline{x}^H_k = \max\{x_k : x \in \Sigma_\exists(\mathbf{A}, b)\}$, for $k = 1, \ldots, n$. Computing the interval hull of the solution set $\Sigma_\exists(\mathbf{A}, b)$ is an NP-hard problem [23].

3.2 Interval Linear Programming

Let $\mathbf{A} \in \mathbb{IR}^{m \times n}$ be an $m \times n$ interval matrix, $b \in \mathbb{R}^m$ be an m-dimensional vector, and $\mathbf{c} \in \mathbb{IR}^n$ be an n-dimensional interval vector. The *family* of linear programming (LP) problems

$$f(A, b, c) = \min\{c^T x : \ Ax \leq b\}$$

with data satisfying

$$A \in \mathbf{A}, c \in \mathbf{c}$$

is called an *interval linear programming (ILP)* problem.

The interval $[\underline{f}(\mathbf{A}, b, \mathbf{c}), \overline{f}(\mathbf{A}, b, \mathbf{c})]$, where $\underline{f}(\mathbf{A}, b, \mathbf{c}) = \inf\{f(A, b, c) : A \in \mathbf{A}, c \in \mathbf{c}\}$, and $\overline{f}(\mathbf{A}, b, \mathbf{c}) = \sup\{f(A, b, c) : A \in \mathbf{A}, c \in \mathbf{c}\}$, is called the *range of the optimal value* of the above ILP problem.

In this paper, we are only interested in computing the lower bound $\underline{f}(\mathbf{A}, b, \mathbf{c})$. However, in general, to compute the exact $\underline{f}(\mathbf{A}, b, \mathbf{c})$, in the worst case up to 2^n LP problems have to be solved, one for each orthant. In practice, [5] proposed an enumerative approach which can considerably reduce the number of LPs in many cases. Recently, Jansson [15] proposed an iterative method to compute a safe lower bound for $\underline{f}(\mathbf{A}, b, \mathbf{c})$ by solving a sequence of midpoint problems, and in many cases, only a small computational effort is required. In the following sections, we use ILP as a black box.

4 The Interval Polyhedra Domain

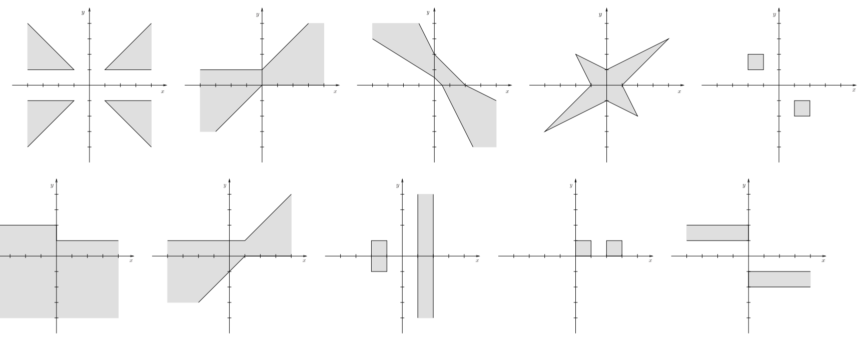
We now introduce the *interval polyhedra* abstract domain (*itvPol*). The main idea is to use interval linear inequality constraints in the representation of the new domain. An important similarity between the *itvPol* domain and most existing numerical abstract domains is that their elements can be defined as the solutions of systems of finitely many constraints from a certain family. To some extent, one may consider the *itvPol* domain as an interval version of the classic convex polyhedra domain which only supports standard (non-interval) linear constraints.

4.1 Representation

An *interval polyhedron* $\mathbf{P}$ is described as an interval linear system $\mathbf{A}x \leq b$, where $\mathbf{A} \in \mathbb{IR}^{m\times n}$ is an interval matrix and $b \in \mathbb{R}^m$ is a plain vector of real numbers. It represents the set $\gamma(\mathbf{P}) = \Sigma_\exists(\mathbf{A}, b)$, and each point $x \in \gamma(\mathbf{P})$ is a possible program environment (or state), i.e., an assignment of numerical/real values to program variables. Note that with respect to the weak solution set, an interval linear equation φ: $\sum_k [\underline{a}_k, \overline{a}_k] \times x_k = [\underline{b}, \overline{b}]$ can be represented as a pair of interval linear inequalities φ': $\sum_k [\underline{a}_k, \overline{a}_k] \times x_k \leq \overline{b}$ and φ'': $\sum_k [-\overline{a}_k, -\underline{a}_k] \times x_k \leq -\underline{b}$. The set of interval polyhedra has the following properties:

- Non-convexity: an interval polyhedron is *non-convex* in general, but its intersection with each orthant in $\mathbb{R}^n$ gives a (possibly empty) convex polyhedron.
- Closed for intersection: the intersection of two interval polyhedra is also an interval polyhedron.
- Non-closed for union: the union of two interval polyhedra may not be an interval polyhedron.

In general, an interval polyhedron has a complicated shape. Fig. 1 shows some examples of interval polyhedra (1) as well as examples that are not interval polyhedra (2). Specifically, (2.a), (2.b), (2.d) are not interval polyhedra because their intersection with some orthant (e.g., the (+,+)-orthant) is not convex. (2.c) and (2.e) are not interval polyhedra as they do not satisfy the topological properties described in [14].

(1)  (2)

(a) (b) (c) (d) (e)

Fig. 1. Examples that are (1) or are not (2) interval polyhedra in two dimensions. The examples (1) correspond to the following interval linear systems: (1.a) $\{[-1, 1]x+y = 0, [-1, 1]y = 1\}$, (1.b) $\{[-1, 0]x+y = [0, 1]\}$, (1.c) $\{[1, 2]x+[1, 2]y = [1, 2]\}$, (1.d) $\{[-1, 1]x+2y = [-2, 2], 2x+[-2, 1]y = [-2, 2]\}$, (1.e) $\{[-1, 1]x = 1, [-1, 1]y = 1, x = [-2, 2], y = [-2, 2], x + y = [-1, 1]\}$.

An interval linear inequality φ is entailed by an interval polyhedron $\mathbf{P}$, denoted as $\mathbf{P} \models \varphi$, iff $\gamma(\mathbf{P}) \subseteq \gamma(\varphi)$. The order relation $\sqsubseteq$ on interval polyhedra is defined as $\mathbf{P}_1 \sqsubseteq \mathbf{P}_2$ iff $\gamma(\mathbf{P}_1) \subseteq \gamma(\mathbf{P}_2)$, i.e., $\forall\varphi \in \mathbf{P}_2.\mathbf{P}_1 \models \varphi$, which can be implemented using ILP. The inclusion $\mathbf{P}_1 \sqsubseteq \mathbf{P}_2$ holds iff for all $(\sum_k [\underline{a}_k, \overline{a}_k] \times x_k \leq b) \in \mathbf{P}_2$, $\mu \leq b$ holds where μ=max $\sum_k [\underline{a}_k, \overline{a}_k] \times x_k$ subject to $\mathbf{P}_1$. However, $\sqsubseteq$ may be too expensive to compute. We define an approximate order relation $\sqsubseteq_s$ on interval polyhedra based on syntactic representations. Given φ: $\Sigma_k[\underline{a}_k, \overline{a}_k]\times x_k \leq b$ and φ': $\Sigma_k[\underline{a}'_k, \overline{a}'_k]\times x_k \leq b'$, $\varphi \sqsubseteq_s \varphi'$ iff $b \leq b'$ and $\forall k.[\underline{a}_k, \overline{a}_k] \subseteq [\underline{a}'_k, \overline{a}'_k]$. And $\mathbf{P}_1 \sqsubseteq_s \mathbf{P}_2$ iff for all $\varphi_2 \in \mathbf{P}_2$ there exists some $\varphi_1 \in \mathbf{P}_1$ such that $\varphi_1 \sqsubseteq_s \varphi_2$. Then, $\mathbf{P}_1 \sqsubseteq_s \mathbf{P}_2$ implies $\mathbf{P}_1 \sqsubseteq \mathbf{P}_2$, while the converse does not hold. The intersection of $\mathbf{P}_1$ and $\mathbf{P}_2$, denoted as $\mathbf{P}_1 \sqcap \mathbf{P}_2$, is an interval polyhedron whose constraint system is the conjunction of those of $\mathbf{P}_1$ and $\mathbf{P}_2$, thus $\gamma(\mathbf{P}_1 \sqcap \mathbf{P}_2) = \gamma(\mathbf{P}_1) \cap \gamma(\mathbf{P}_2)$. Also, we use the term *bounding box* of an interval polyhedron $\mathbf{P}$, denoted as $BB(\mathbf{P})$, to refer to the interval hull $\mathbf{x}^H$ of $\Sigma_\exists(\mathbf{A}, b)$. $BB(\mathbf{P})$ can be computed by ILP, namely by calculating max(min) x_k subject to $\mathbf{P}$, which is NP-hard (Sect. 3.1). In practice, $BB(\mathbf{P})$ is updated on-the-fly, and an over-approximated bounding box which is sound can be obtained by cheaper methods, as in [4].

As in the classic convex polyhedra abstract domain, the constraint representation of an interval polyhedron is not unique. E.g., the interval linear equation $[-1, 1]x = 1$ and the inequality $[-1, 1]x \leq -1$ have the same weak solution set $\{x \in [-\infty, -1] \cup [1, +\infty]\}$. For efficiency reasons, it is desirable to have as few and simple constraints as possible.

Reduction. According to Theorem 1, an interval linear inequality φ: $\sum_k [\underline{a}_k, \overline{a}_k] \times x_k \leq b$ can be reduced to φ': $\sum_k [\underline{a}'_k, \overline{a}'_k] \times x_k \leq b$, where $x_k \in [\underline{x}^H_k, \overline{x}^H_k]$ and

$$[\underline{a}'_k, \overline{a}'_k] = \begin{cases} [\underline{a}_k, \underline{a}_k] & \text{if } \underline{x}^H_k \geq 0, \\ [\overline{a}_k, \overline{a}_k] & \text{if } \overline{x}^H_k \leq 0, \\ [\underline{a}_k, \overline{a}_k] & \text{otherwise.} \end{cases}$$

The reduction is useful in practice, since φ' will cause less precision loss in the subsequent computations, e.g., in the interval combination operation (see Sect. 5.2.1).

Redundancy Removal. An interval linear inequality $\varphi \in \mathbf{P}$ is said to be *redundant* when φ is entailed by the other constraints in $\mathbf{P}$, that is, $\mathbf{P} \setminus \{\varphi\} \models \varphi$. Given φ: $(\sum_k [\underline{a}_k, \overline{a}_k] \times x_k \leq b) \in \mathbf{P}$, we can check whether φ is redundant by solving the ILP problem: μ= max $\sum_k [\underline{a}_k, \overline{a}_k] \times x_k$ subject to $\mathbf{P} \setminus \{\varphi\}$. If $\mu \leq b$, then φ is redundant and can be eliminated from $\mathbf{P}$. This process is repeated for all inequalities in $\mathbf{P}$. To be more efficient, it is worth using lightweight methods first and resorting to the expensive ILP-based method only when necessary. For example, given $\varphi \in \mathbf{P}$, if there exists another interval linear inequality $\varphi' \in \mathbf{P}$ such that $\varphi' \sqsubseteq_s \varphi$, then φ is redundant in $\mathbf{P}$. Secondly, given φ: $(\sum_k [\underline{a}_k, \overline{a}_k] \times x_k \leq b) \in \mathbf{P}$, if $\forall k.0 \in [\underline{a}_k, \overline{a}_k]$ and $b \geq 0$, then φ defines the universe space and can be removed from $\mathbf{P}$.

4.2 *itvPol* as an (Abstracted) Reduced Cardinal Power of Convex Polyhedra

In this section we consider *itvPol* as a reduced cardinal power (see Sect. 10.2 of [8]) that maps each orthant to a convex polyhedron, by exploiting the fact that the intersection of an interval polyhedron with an orthant is a (possibly empty) convex polyhedron. An n-dimensional interval polyhedron is at worst a set of 2^n convex polyhedra. More precisely, let p be the number of variables in an interval polyhedron that are unrestricted in sign and are associated with at least one (non-singleton) interval coefficient, then the interval polyhedron is partitioned into a set of at most 2^p convex polyhedra.

In the general case, we maintain one (possibly empty) convex polyhedron in each orthant. Each operation on the *itvPol* domain is obtained by "lifting" the corresponding operation from the convex polyhedra domain. E.g., to join two interval polyhedra, one would compute pair-wisely the convex hull of the convex polyhedra in each orthant. The assignment transfer function needs more care, since applying an assignment transfer function on a convex polyhedron in one orthant may cause it to "enter" other orthants. In such a case, the result in each orthant is then updated to be the polyhedral convex hull of the regions which belong to that orthant after the transfer operation. Thus, this domain is not simply equivalent to a finite disjunctive completion of convex polyhedra. In our case, it is perhaps better called an orthant partitioning domain of convex polyhedra.

In order to enjoy the benefits of the compact representation of interval polyhedra, one may abstract further a set of convex polyhedra with at most one in each orthant back to an interval polyhedron after each operation. However, there may not exist an interval polyhedron that exactly defines the union of those convex polyhedra, e.g., by referring to Fig. 1(2.c). And, to our knowledge, up to now there exists no method to compute the smallest interval polyhedron that encloses those convex polyhedra.

We propose an algorithm to calculate a (not necessarily smallest) interval polyhedron that soundly encloses those convex polyhedra. Given a variable ordering, an n-dimensional space is described by a binary tree with variables at internal nodes and convex polyhedra at leaves. Each node represents a variable x, and its left (right) branch specifies the subspace in which $x \leq 0$ ($x \geq 0$). Thus the path from the root to a leaf (involving all variables with respect to the given ordering) defines an orthant, and the convex polyhedron attached at the leaf corresponds to the convex polyhedron in that orthant. At each internal node $Node_x$, an interval polyhedron is constructed in a bottom-up manner, to enclose all the convex polyhedra within the subtree rooted at $Node_x$, using the weak join operation $\sqcup_w$ defined in Sect. 5.2.2. When a variable has a fixed sign, one of its subtrees is empty, which speeds up the computation. Finally, the interval polyhedron at the root of the whole binary tree is an interval polyhedron that encloses all the convex polyhedra in each orthant.

Example 1. Given the interval polyhedron $\mathbf{P} = \{[-1, 0]x+y = [0, 1]\}$ shown in Fig. 1(1.b), after performing the assignment transfer function $[\![x := x + 1]\!]^{\#}$ on $\mathbf{P}$ in the powerset domain of convex polyhedra, we obtain the region shown in Fig. 1(2.b), which cannot be exactly encoded by any interval polyhedron. Then, by computing the polyhedral convex hull in each orthant, we get the convex polyhedra $\{x \geq 0, y \geq 0, -x + y \leq 1\}$ in the $(+, +)$-orthant and $\{x \geq 0, -1 \leq y \leq 0\}$ in the $(+, -)$-orthant. Finally, we obtain the interval polyhedron $\{[-1, 0]x + y = [-1, 1]\}$ using the above algorithm.

As an abstracted reduced cardinal power of convex polyhedra, *itvPol* is at worst exponentially more complex than the convex polyhedra domain. However, in some real-life applications, many of the variables do not change their signs, that is, the weak solution set intersects only a few orthants. In such situations, *itvPol* as an abstracted reduced cardinal power of convex polyhedra has a reasonable complexity.

5 *itvPol* as a New Abstract Domain

In general, *itvPol* as an abstracted reduced cardinal power of convex polyhedra may be too complex to be applied to program analysis. To solve this problem, we present an alternative construction based on faster approximate algorithms. Similarly to the constraint-based convex polyhedra abstract domain [4], this construction is based on two main operations: projection and (interval) linear programming. We will now briefly describe the implementation of the most common domain operations.

5.1 Projection

The projection operation is used to remove all information pertaining to a variable x_i while preserving the relational information between other variables. It can be computed by eliminating all occurrences of x_i in the constraints defining $\mathbf{P}$, using an Interval Fourier-Motzkin Elimination (IFME) algorithm defined below, which is an adaptation of the classic Fourier-Motzkin elimination algorithm to interval arithmetic.

Let $\mathbf{P} = \{\mathbf{A}x \leq b\}$ be an interval polyhedron and x_i be a variable to be eliminated. If all non-zero interval coefficients of x_i in $\mathbf{P}$ do not contain zero, the classic Fourier-Motzkin elimination algorithm can be easily adapted to interval arithmetic. However, in general, the constraint φ: $(\sum_k [\underline{a}_k, \overline{a}_k] x_k \leq b) \in P$ in which $[\underline{a}_i, \overline{a}_i] \neq [0,0]$ but $0 \in [\underline{a}_i, \overline{a}_i]$, will break the algorithm due to division by an interval containing zero. To avoid this, we apply the following linearization operator $\zeta(\varphi, x_i)$ beforehand.

Definition 2 (Linearization operator). *Let* φ: $\sum_k [\underline{a}_k, \overline{a}_k] \times x_k \leq b$ *be an interval linear inequality and* $x_i \in [\underline{x}_i^H, \overline{x}_i^H]$.

$$\zeta(\varphi, x_i) \stackrel{\text{def}}{=} \begin{cases} \underline{a}_i \times x_i + \sum\limits_{k \neq i} [\underline{a}_k, \overline{a}_k] \times x_k \leq b & \text{if } \underline{x}_i^H \geq 0 \\ \overline{a}_i \times x_i + \sum\limits_{k \neq i} [\underline{a}_k, \overline{a}_k] \times x_k \leq b & \text{if } \overline{x}_i^H \leq 0 \\ c \times x_i + \sum\limits_{k \neq i} [\underline{a}_k, \overline{a}_k] \times x_k \leq \sup(b - [\underline{a}_i - c, \overline{a}_i - c] \times [\underline{x}_i^H, \overline{x}_i^H]) & \text{otherwise} \end{cases}$$

where c can be any real number.

In practice, we often choose $c = (\underline{a}_i + \overline{a}_i)/2$ that is the midpoint of the interval $[\underline{a}_i, \overline{a}_i]$, which causes the least loss of precision (minimizing $\sup(b - [\underline{a}_i - c, \overline{a}_i - c] \times [\underline{x}_i^H, \overline{x}_i^H])$).

Example 2. Consider the interval linear inequality $[0,2]x + y \leq 2$ with respect to the bounds $x, y \in [-2, 4]$. If we choose the midpoint of $[0,2]$ as c, $\zeta(\varphi, x)$ will give $x + y \leq 6$. Note that some loss of precision happens here, e.g., the point (0,4) satisfies the result inequality $x + y \leq 6$ but does not satisfy the original interval inequality $[0,2]x + y \leq 2$.

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

By falling back to the above linearization technique, IFME can be applied to general interval polyhedra. Given an inequality φ: $\sum_k [\underline{a}_k, \overline{a}_k] x_k \leq b$, we define $\iota(\varphi, x_i)$ as

$$\iota(\varphi, x_i) \stackrel{\text{def}}{=} \begin{cases} \zeta(\varphi, x_i) & \text{if } 0 \in [\underline{a}_i, \overline{a}_i] \wedge [\underline{a}_i, \overline{a}_i] \neq [0,0], \\ \varphi & \text{otherwise.} \end{cases}$$

Then, the Interval Fourier-Motzkin Elimination algorithm can be defined as

$$\begin{aligned} \mathrm{IFME}(\mathbf{P}, x_i) \stackrel{\text{def}}{=} & \ \{ (\textstyle\sum_k [\underline{a}_k, \overline{a}_k] x_k \leq b) \in \mathbf{P}' \mid [\underline{a}_i, \overline{a}_i] = [0,0] \} \\ & \cup \left\{ \textstyle\sum_{k \neq i} \left(\dfrac{[\underline{a}_k^+, \overline{a}_k^+]}{[\underline{a}_i^+, \overline{a}_i^+]} + \dfrac{[\underline{a}_k^-, \overline{a}_k^-]}{[-\overline{a}_i^-, -\underline{a}_i^-]} \right) x_k \leq b' \ \middle| \begin{array}{l} (\sum_k [\underline{a}_k^+, \overline{a}_k^+] x_k \leq b^+) \in \mathbf{P}', \underline{a}_i^+ > 0 \\ (\sum_k [\underline{a}_k^-, \overline{a}_k^-] x_k \leq b^-) \in \mathbf{P}', \overline{a}_i^- < 0 \end{array} \right\} \end{aligned}$$

where $\mathbf{P}' = \{\iota(\varphi, x_i) \mid \varphi \in \mathbf{P}\}$ and $b' = \sup\left(\dfrac{b^+}{[\underline{a}_i^+, \overline{a}_i^+]} + \dfrac{b^-}{[-\overline{a}_i^-, -\underline{a}_i^-]} \right)$.

Theorem 3 (Soundness of the Interval Fourier-Motzkin Elimination). *Given an interval polyhedron* $\mathbf{P}$ *and a variable* x_i*, any point satisfying* $\mathbf{P}$ *also satisfies* $\mathrm{IFME}(\mathbf{P}, x_i)$*, that is,* $\forall x \in \gamma(\mathbf{P}) \Rightarrow x \in \gamma(\mathrm{IFME}(\mathbf{P}, x_i))$.

5.2 Join

In order to abstract the control-flow join, we need to compute the union of environments of program variables. However, to our knowledge, no existing method is available to compute the smallest interval polyhedron enclosing this union. We propose to compute an overapproximation of this union cheaply using an operation that we call *weak join*.

The main idea is as follows. We first define an operation $\uplus$ on constraints that over-approximates the set union $\cup$ such that $\gamma(\varphi') \cup \gamma(\varphi'') \subseteq \gamma(\varphi' \uplus \varphi'')$. Given two interval polyhedra $\mathbf{P}'$ and $\mathbf{P}''$, by distributivity

$$\gamma(\mathbf{P}') \cup \gamma(\mathbf{P}'') = (\bigcap_{\varphi' \in \mathbf{P}'} \gamma(\varphi')) \cup (\bigcap_{\varphi'' \in \mathbf{P}''} \gamma(\varphi'')) = \bigcap_{\substack{\varphi' \in \mathbf{P}' \\ \varphi'' \in \mathbf{P}''}} (\gamma(\varphi') \cup \gamma(\varphi'')) \subseteq \bigcap_{\substack{\varphi' \in \mathbf{P}' \\ \varphi'' \in \mathbf{P}''}} (\gamma(\varphi' \uplus \varphi'')) .$$

Our weak join can be constructed basically by pairwise combinations of inequalities from $\mathbf{P}_1$ with those from $\mathbf{P}_2$ using the $\uplus$ operation.

5.2.1 Interval Combination

Definition 3 (Interval Combination). *Given two interval linear inequalities* φ': $\sum_k [\underline{a}'_k, \overline{a}'_k] \times x_k \leq b'$ *and* φ'': $\sum_k [\underline{a}''_k, \overline{a}''_k] \times x_k \leq b''$*, the* interval combination *of* φ' *and* φ'' *is defined as*

$$\varphi' \uplus \varphi'' \stackrel{\text{def}}{=} \left(\sum_k [\underline{a}_k, \overline{a}_k] \times x_k \leq b\right),$$

where $b = \max(b', b'')$ *and* $[\underline{a}_k, \overline{a}_k] = [\min(\underline{a}'_k, \underline{a}''_k), \max(\overline{a}'_k, \overline{a}''_k)]$.

This definition straightforwardly lifts to interval polyhedra. Given two interval polyhedra $\mathbf{P}'$ and $\mathbf{P}''$, $\mathbf{P}' \uplus \mathbf{P}'' \stackrel{\text{def}}{=} \{\varphi' \uplus \varphi'' \mid \varphi' \in \mathbf{P}' \wedge \varphi'' \in \mathbf{P}''\}$.

Example 3. Consider two interval polyhedra $\mathbf{P}' = \{y \leq 1, -y \leq -1\}$ and $\mathbf{P}'' = \{-x + y \leq 0, x - y \leq 0\}$. By interval combination, we obtain $\mathbf{P} = \mathbf{P}' \uplus \mathbf{P}'' = \{[-1, 0]x + y = [0, 1]\}$, whose weak solution set is depicted in Fig. 1(1.b). Note that some loss of precision happens here, e.g., the point (1,0) satisfies the result $\mathbf{P}$ but satisfies neither $\mathbf{P}'$ nor $\mathbf{P}''$.

Theorem 4 (Soundness of the interval combination). *Given two interval linear inequalities* φ' *and* φ''*, their interval combination* $\varphi' \uplus \varphi''$ *soundly over-approximates the union of* φ' *and* φ''*, that is,* $\forall x.(x \in \gamma(\varphi') \vee x \in \gamma(\varphi'')) \Rightarrow x \in \gamma(\varphi' \uplus \varphi'')$.

The above theorem implies the soundness of $\uplus$ on interval polyhedra, i.e., $\forall x.(x \in \gamma(\mathbf{P}') \vee x \in \gamma(\mathbf{P}'')) \Rightarrow x \in \gamma(\mathbf{P}' \uplus \mathbf{P}'')$. However, the result of $\varphi' \uplus \varphi''$ may not be the tightest interval linear inequality whose weak solution set encloses the union of the weak solution sets of φ' and φ''. Moreover, the precision of the interval combination depends on the representation of the input. The tighter the input coefficient intervals are, the more precise the result will be. Hence, the reduction operation (in Sect. 4.1) is often used before performing interval combinations.

In some cases, the interval combination can be improved. Given φ': $\sum_k [\underline{a}'_k, \overline{a}'_k] \times x_k \leq b'$ and φ'': $\sum_k [\underline{a}''_k, \overline{a}''_k] \times x_k \leq b''$, if there exists a positive *multiplier* λ such that $\lambda \times [\underline{a}'_i, \overline{a}'_i] = [\underline{a}''_i, \overline{a}''_i]$ (i.e., $\lambda \underline{a}'_i = \underline{a}''_i$ and $\lambda \overline{a}'_i = \overline{a}''_i$) for some i, then the interval combination of φ' and φ'' can be computed as $\varphi' \uplus \varphi'' = (\sum_k [\underline{a}_k, \overline{a}_k] \times x_k \leq b)$, where $b = \max(\lambda b', b'')$ and $[\underline{a}_k, \overline{a}_k] = [\min(\lambda \underline{a}'_k, \underline{a}''_k), \max(\lambda \overline{a}'_k, \overline{a}''_k)]$. In most cases, the interval combination with multiplier is more precise than the general one. E.g., given φ_1: $x + y \leq 2$ and φ_2: $-x + 2y \leq 2$, $\varphi_1 \uplus \varphi_2$ gives φ: $[-1, 1]x + [1, 2]y \leq 2$. However, if we use a version with multiplier (i.e., rewrite φ_1: $x + y \leq 2$ as φ'_1: $2x + 2y \leq 4$), $\varphi'_1 \uplus \varphi_2$ will give φ': $[-1, 2]x + 2y \leq 4$, and the result φ' is more precise than the previous one φ.

5.2.2 Weak Join

Definition 4 (Envelope). *Given two interval polyhedra* $\mathbf{P}_1$ *and* $\mathbf{P}_2$*, the* envelope *of* $\mathbf{P}_1$ *and* $\mathbf{P}_2$ *is defined as*

$$env(\mathbf{P}_1, \mathbf{P}_2) \stackrel{\text{def}}{=} \mathcal{S}_1 \cup \mathcal{S}_2$$

where $\mathcal{S}_1 = \{ \varphi_1 \in \mathbf{P}_1 \mid \mathbf{P}_2 \models \varphi_1 \}, \mathcal{S}_2 = \{ \varphi_2 \in \mathbf{P}_2 \mid \mathbf{P}_1 \models \varphi_2 \}$.

Let $i \in \{1, 2\}$, for any $\varphi \in \mathbf{P}_i$, if $\varphi \in env(\mathbf{P}_1, \mathbf{P}_2)$, we say that φ is an *envelope constraint* in $\mathbf{P}_i$, otherwise φ is a *nonenvelope constraint* in $\mathbf{P}_i$. We denote the set of nonenvelope constraints in $\mathbf{P}_i$ as $\overline{env}(\mathbf{P}_i)$. Given two boxes $\mathbf{B}' = [\underline{b}', \overline{b}']$ and $\mathbf{B}'' = [\underline{b}'', \overline{b}'']$, their join in the interval abstract domain is defined as $\mathbf{B}' \sqcup_I \mathbf{B}'' = [\min(\underline{b}', \underline{b}''), \max(\overline{b}', \overline{b}'')]$. Note that $BB(\gamma(\mathbf{P}_1) \cup \gamma(\mathbf{P}_2)) = BB(\mathbf{P}_1) \sqcup_I BB(\mathbf{P}_2)$.

Definition 5 (Weak Join). *Given two interval polyhedra* $\mathbf{P}_1$ *and* $\mathbf{P}_2$*, we define a* weak join *operation for the itvPol domain as*

$$\mathbf{P}_1 \sqcup_w \mathbf{P}_2 \stackrel{\text{def}}{=} env(\mathbf{P}_1, \mathbf{P}_2) \sqcap (\overline{env}(\mathbf{P}_1) \uplus \overline{env}(\mathbf{P}_2)) \sqcap (BB(\mathbf{P}_1) \sqcup_I BB(\mathbf{P}_2)).$$

Note that the weak join operation may introduce redundant constraints in the result, but most of them can be eliminated by syntactic means (see Sect. 4.1).

Example 4. Consider two interval polyhedra $\mathbf{P}_1 = \{[-1, 1]x+2y \leq 2, -2x-y \leq 2, x-y \leq 1, -y \leq 0\}$ (i.e., the region above the x-axis in Fig. 1(1.d)) and $\mathbf{P}_2 = \{[-1, 1]x - 2y \leq 2, 2x + y \leq 2, -x + y \leq 1, y \leq 0\}$ (i.e., the region below the x-axis in Fig. 1(1.d)). $env(\mathbf{P}_1, \mathbf{P}_2) = \{[-1, 1]x + 2y \leq 2, [-1, 1]x - 2y \leq 2\} = \{[-1, 1]x + 2y = [-2, 2]\}$. $\overline{env}(\mathbf{P}_1) \uplus \overline{env}(\mathbf{P}_2) = \{-2x - y \leq 2, x - y \leq 1, -y \leq 0\} \uplus \{2x + y \leq 2, -x + y \leq 1, y \leq 0\} = \{2x + [-2, 1]y = [-2, 2]\}$. Thus, $\mathbf{P}_1 \sqcup_w \mathbf{P}_2 = \{[-1, 1]x + 2y = [-2, 2], 2x + [-2, 1]y = [-2, 2]\}$, whose weak solution set is depicted in Fig. 1(1.d).

Theorem 5 (Soundness of the Weak Join). *Given two interval polyhedra* $\mathbf{P}_1$ *and* $\mathbf{P}_2$*, the weak join* $\mathbf{P}_1 \sqcup_w \mathbf{P}_2$ *is an overapproximation of both* $\mathbf{P}_1$ *and* $\mathbf{P}_2$*, that is,* $\forall x.(x \in \gamma(\mathbf{P}_1) \vee x \in \gamma(\mathbf{P}_2)) \Rightarrow x \in \gamma(\mathbf{P}_1 \sqcup_w \mathbf{P}_2)$.

The above weak join can construct constraints that are satisfied by the set-union of the input interval polyhedra but not satisfied by their convex hull (e.g., $\gamma(\mathbf{P}_1 \sqcup_w \mathbf{P}_2) = \gamma(\mathbf{P}_1) \cup \gamma(\mathbf{P}_2)$ holds in Example 4). However, when both interval polyhedra $\mathbf{P}_1$ and $\mathbf{P}_2$ are convex polyhedra and in the same orthant, $\mathbf{P}_1 \sqcup_w \mathbf{P}_2$ is less precise than their polyhedral convex hull. In such a case, one may use the polyhedral convex hull instead. E.g., given two points $(0, 0)$ and $(1, 1)$ in the $(+,+)$-orthant of the x-y plane, $\sqcup_w$ can only give $\{0 \leq x \leq 1, 0 \leq y \leq 1\}$, which is less precise than the result of their polyhedral convex hull (i.e., $\{0 \leq x \leq 1, y = x\}$).

5.3 Emptiness Test

An interval polyhedron $\mathbf{P}$ is *empty* iff its constraint set is not weakly solvable, that is, $\gamma(\mathbf{P}) = \emptyset$. Theorem 1 shows that checking weak solvability of interval linear systems can be in principle performed by checking solvability of one linear system per orthant, by some finite procedure (e.g., linear programming). During program analysis, constraints are often added one by one. Thus, the emptiness test can be done incrementally. When adding a new constraint $\sum_k [a_k, b_k] \times x_k \leq b$ to a nonempty interval polyhedron $\mathbf{P}$, we solve the ILP problem: $\mu = \min \sum_k [a_k, b_k] \times x_k$ subject to $\mathbf{P}$. If $\mu > b$, the new interval polyhedron is indeed empty.

5.4 Transfer Functions

Test Transfer Function. The result of a test $[\![\sum_k [\underline{a}_k, \overline{a}_k] \times x_k \leq b]\!]^\#(\mathbf{P})$ is simply the interval polyhedron $\mathbf{P}$ with the constraint $\sum_k [\underline{a}_k, \overline{a}_k] \times x_k \leq b$ added, where $[\![\cdot]\!]^\#(\mathbf{P})$ denotes the effect of a program statement on the interval polyhedron $\mathbf{P}$. More complicated cases, such as tests involving disjunctive, non-linear or floating-point expressions can be soundly abstracted to the form $\sum_k [\underline{a}_k, \overline{a}_k] \times x_k \leq b$, as discussed in Sect. 6.

Assignment Transfer Function. The assignment of a certain expression e to the variable x_j can be modeled using test, projection and variable renaming as follows:

$$[\![x_j := e]\!]^\#(\mathbf{P}) \stackrel{\text{def}}{=} (IFME([\![x'_j - e = 0]\!]^\#(\mathbf{P}), x_j))[x'_j/x_j]\,. \tag{1}$$

The fresh variable x'_j, introduced to hold the value of the expression e, is necessary when x_j also appears on the right hand of the assignment, e.g., $x := [-1, 2]x + [2, 3]$.

Alternatively, the assignment transfer function can be implemented by substitution, when the coefficient of x_j in e does not contain zero. Let $e = \sum_k [\underline{d}_k, \overline{d}_k]x_k + [\underline{c}, \overline{c}]$, where $0 \notin [\underline{d}_j, \overline{d}_j]$. Then the assignment transfer function by substitution is defined as

$$[\![x_j := e]\!]^\#(\mathbf{P}) \stackrel{\text{def}}{=} \left\{[\underline{a}'_j, \overline{a}'_j]x_j + \textstyle\sum_{k\neq j}[\underline{a}'_k, \overline{a}'_k]x_k \leq b' \mid (\sum_k[\underline{a}_k, \overline{a}_k]x_k \leq b) \in \mathbf{P}\right\} \tag{2}$$

where $[\underline{a}'_j, \overline{a}'_j] = \dfrac{[\underline{a}_j, \overline{a}_j]}{[\underline{d}_j, \overline{d}_j]}$, $[\underline{a}'_k, \overline{a}'_k] = \left([\underline{a}_k, \overline{a}_k] - \dfrac{[\underline{d}_k, \overline{d}_k]}{[\underline{d}_j, \overline{d}_j]}\right)$ and $b' = sup\left(b + \dfrac{[\underline{c}, \overline{c}]}{[\underline{d}_j, \overline{d}_j]}\right)$.

Note that unlike the case of convex polyhedra, neither (1) nor (2) is an exact or best abstraction for interval polyhedra. In most cases, (2) gives more precise results than (1). E.g., given an interval polyhedron $\mathbf{P} = \{[-1, 1]x \leq -1\}$, for the assignment $x := -x$, (1) will give the whole space as the result while (2) will result in $\mathbf{P}' = \{[-1, 1]x \leq -1\}$ which is exact.

5.5 Widening

itvPol does not satisfy the ascending chain condition. Thus, a widening [7] operator is needed to ensure convergence of fixpoint computations (used to analyze loops). We define the widening in the *itvPol* domain as follows:

Definition 6 (Widening). *Given a threshold k and two interval polyhedra* $\mathbf{P}_1 \sqsubseteq \mathbf{P}_2$ *in the i-th iteration, we define the* widening *in the i-th iteration as*

$$\mathbf{P}_1 \triangledown_i^{[k]} \mathbf{P}_2 \stackrel{\text{def}}{=} \begin{cases} \mathcal{S}_1 \cup \mathcal{S}_2 & \text{if } i \leq k \\ \mathcal{S}_1 & \text{otherwise} \end{cases}$$

where $\mathcal{S}_1 = \{\varphi_1 \in \mathbf{P}_1 \mid \mathbf{P}_2 \models \varphi_1\}, \mathcal{S}_2 = \{\varphi_2 \in \mathbf{P}_2 \mid \exists \varphi_1 \in \mathbf{P}_1, \gamma(\mathbf{P}_1) = \gamma((\mathbf{P}_1 \setminus \{\varphi_1\}) \cup \{\varphi_2\})\}$.

$\mathcal{S}_1$ keeps stable inequalities from $\mathbf{P}_1$. $\mathcal{S}_2$ recovers precision by adding those inequalities from $\mathbf{P}_2$ that are mutually redundant with an inequality of $\mathbf{P}_1$ with respect to $\mathbf{P}_1$. Unlike the classic convex polyhedra domain where the widening is defined as $\mathcal{S}_1 \cup \mathcal{S}_2$ in all cases, *itvPol* needs a threshold k to guarantee convergence of the above widening by disabling $\mathcal{S}_2$ after the k-th iteration. Given a chain $(X_i)_{i\in\mathbb{N}}$, the increasing chain $(Y_i)_{i\in\mathbb{N}}$ defined by $Y_0{=}X_0$ and $Y_{i+1}{=}Y_i \triangledown_i^{[k]} X_{i+1}$, is stable after a finite time, since after the k-th iteration, the set of constraints in Y_{j+1} is a subset of the constraints in Y_j $(j > k)$.

6 Applications of the Interval Polyhedra Domain

6.1 Handling Disjunctions

We propose to apply the technique of *interval combination* to abstract disjunctions of linear constraints by interval linear inequalities. In general, given a DNF (Disjunctive Normal Form) formula, each DNF-term can be considered as a convex polyhedron. The disjunction of those convex polyhedra can be abstracted as an interval polyhedron using the join operation of *itvPol* (Sect. 5.2). On the other hand, given a CNF (Conjunctive Normal Form) formula, each CNF-term can be abstracted as one interval linear inequality by interval combination (Sect. 5.2.1), thus the whole CNF formula can be abstracted as an interval polyhedron.

For example, consider the program in Fig. 2. At ②, the negation of $-1 \leq x \leq 1$ on integers gives $x \leq -2 \vee -x \leq -2$, which can be exactly encoded as an interval linear inequality $[-1, 1]x \leq -2$. And, using *itvPol*, we can obtain the exact information at ⑤, i.e., $y = -1$, which implies $y \neq 0$. However, the convex polyhedra domain can only obtain $-1 \leq y \leq 0$ at ⑤ which fails to prove the assertion $y \neq 0$.

```
int x, y;
if ( x ≥ −1 and x ≤ 1 ) then
      y := x − 1; ①
else ②
      y := x; ③
endif; ④
if( x == 0 ) then
⑤    assert(y ≠ 0);
endif;
```

Loc	Convex Polyhedra	Interval Polyhedra
①	$x - y = 1 \wedge -1 \leq x \leq 1$	$x - y = 1 \wedge -1 \leq x \leq 1$
②	$\top$ (no information)	$[-1, 1]x \leq -2$
③	$y = x$	$y = x \wedge [-1, 1]x \leq -2$
④	$0 \leq x - y \leq 1$	$0 \leq x - y \leq 1$ $\wedge[-1, 1]x + [0, 1]y \leq -1$ $\wedge x + [-1, 0]y \leq 1$
⑤	$x = 0 \wedge -1 \leq y \leq 0$	$x = 0 \wedge y = -1$

Fig. 2. Example program1 (left) and the generated invariants (right).

6.2 Handling Non-Linear Expressions

Miné has proposed a so-called *linearization* algorithm able to abstract arbitrary expressions into interval linear form $\Sigma_k[\underline{a}_k, \overline{a}_k] \times x_k + [\underline{c}, \overline{c}]$ [21]. However, most current numerical abstract domains, such as the convex polyhedra domain, cannot deal with interval linear forms directly. Thus, one has to employ a so-called *quasi-linearization* technique to convert the interval linear form $\Sigma_k[\underline{a}_k, \overline{a}_k] \times x_k + [\underline{c}, \overline{c}]$ into quasi-linear form $\Sigma_k a'_k \times x_k + [\underline{c}', \overline{c}']$ [19]. The quasi-linearization process may cause precision loss. However, using *itvPol*, one can avoid (at least delay) such precision loss, since *itvPol* directly supports the representation of interval linear forms.

Given the program in Fig. 3, after the linearization of the non-linear expression $z \times x + 1$, we obtain $[-5, 5]x + y = 1$ at ①. When using the convex polyhedra domain, we have to apply quasi-linearization to $[-5, 5]x + y = 1$. And the best quasi-linearization will be $-5x + y \leq 21 \wedge -5x - y \leq 19$. Note that some precision loss happens here, e.g., the point $(0, 0)$ satisfies $-5x + y \leq 21 \wedge -5x - y \leq 19$ but does not satisfy $[-5, 5]x + y = 1$. Finally, the convex polyhedra domain can only obtain $x \geq -1$ at ② while using *itvPol* we can prove $x \geq 3$ at ②.

int x, y, z;
assume $-5 \le z \le 5$;
assume $x \ge -2$;
$y := z \times x + 1$; ①
assume $y == -14$; ②

Loc	Convex Polyhedra	Interval Polyhedra
①	$-5 \le z \le 5 \wedge x \ge -2$	$-5 \le z \le 5 \wedge x \ge -2$
	$\wedge -5x + y \le 21 \wedge -5x - y \le 19$	$\wedge [-5, 5]x + y = 1$
②	$y = -14 \wedge -5 \le z \le 5$	$y = -14 \wedge -5 \le z \le 5$
	$\wedge x \ge -1$	$\wedge x \ge 3$

Fig. 3. Example program2 (left) and the generated invariants (right).

6.3 Handling Floating-Point Arithmetic

Real-life programming languages do not manipulate rationals or reals, but floating-point numbers, which are much more difficult to abstract. One solution is to approximate floating-point expressions as linear expressions in the real field with interval coefficients by making rounding explicit [19]. Rounding is highly non-linear but can be abstracted using intervals. For instance, $X + Y$ in the floating-point world can be abstracted into $[1 - p, 1 + p] \times X + [1 - p, 1 + p] \times Y + [-mf, mf]$ with the relative error p and the absolute error mf (the smallest non-zero positive value in the floating-point format), e.g., $p = 2^{-23}$ and $mf = 2^{-149}$ in the single precision floating-point format. This fits the linearization framework which can be extended to treat floating-point arithmetic soundly. Thus, floating-point programs can be directly analyzed using *itvPol* after applying floating-point abstractions.

Let us consider the program in Fig. 4. The quasi-linearization of both floating-point assignments $y := -2 \otimes_r x \oplus_r 1$ and $y := -x \oplus_r 1$ will give $y \leftarrow [-\infty, +\infty]$, since x is unbounded. Thus, the convex polyhedra domain will obtain no useful information, while *itvPol* can prove $0.4999998 \le x \le 1.0000002$ at ②, which indicates that x is bounded and thus there is no overflow exception in the last statement (i.e., $x := x \oplus_r 1$).

real x, y;
if random() then
  $y := -2 \otimes_r x \oplus_r 1$;
else
  $y := -x \oplus_r 1$;
endif; ①
assume $y == 0$; ②
$x := x \oplus_r 1$;

Loc	Convex Polyhedra	Interval Polyhedra
①	$\top$(no information)	$[0.9999999, 2.0000005]x + y \le 1.0000001$
		$\wedge [0.9999999, 2.0000005]x + y \ge 0.9999999$
②	$y = 0$	$y = 0 \wedge 0.4999998 \le x \le 1.0000002$

Fig. 4. Example program3 (left) and the generated invariants (right). $\odot_r$ denotes single precision floating-point semantics with arbitrary rounding mode ($\odot \in \{\otimes, \oplus\}, r \in \{+\infty, -\infty\}$).

7 Implementation and Experimental Results

Our prototype domain, *itvPol*, is developed based on Sect. 5 using double precision floating-point numbers. It makes use of GLPK (GNU Linear Programming Kit) [18] which implements the simplex algorithm for linear programming. We implemented an interval linear programming solver based on GLPK following the methods from [5, 15]. The soundness of the floating-point LP/ILP solver is guaranteed by rigorous linear programming [4, 15]. The whole *itvPol* domain is implemented based on interval arithmetic with outward rounding (i.e., rounding upper bounds upward and lower bounds downward), which guarantees the soundness of the floating-point implementation.

```
real x, y;
x := −1;
while ( true ) do
① x := −x;
done;
```

Loc	Convex Polyhedra	Interval Polyhedra
①	$-1 \le x \le 1$	$-1 \le x \le 1 \wedge [-1,1]x \le -1$

Fig. 5. program4 (left) and the generated invariants (right).

itvPol is interfaced to the APRON [1] numerical abstract domain library. Our experiments were conducted using the Interproc [16] static analyzer. We extended Interproc to support input data with intervals (such as expressions and constraints with interval coefficients). In order to assess the precision and efficiency of *itvPol*, we compare the obtained invariants as well as the performance of *itvPol* with our previous work FPPol [4] which is a sound floating-point implementation of the convex polyhedra domain.

To demonstrate the expressiveness of *itvPol*, two simple typical loops are shown in Fig. 5 and Fig. 6, together with the invariants generated by the analyzer. *program4* is a loop that reverses the sign of variable x at each iteration, and *program5* consists of two stages, increasing y in the inner loop first and then increasing x in the outer loop. For *program4* in Fig. 5, *itvPol* can prove that $x = -1 \vee x = 1$ at ①, which is exact and more precise than the invariant $-1 \le x \le 1$ given by FPPol. For *program5* in Fig. 6, *itvPol* can prove that $-20 \le y \le -10 \vee y \ge 10$ at ②, while FPPol can only prove $y \ge -20$.

```
int x, y;
x := 1;
y := −20;
while (x ≤ 9) do
①     x := x + 1;
      while (y ≤ 9) do
            y := y + 1;
      done;
done; ②
```

Loc	Convex Polyhedra	Interval Polyhedra
①	$y \ge -20$ $\wedge 1 \le x \le 9$	$y \ge -20$ $\wedge 1 \le x \le 9$ $\wedge -x + [0,1]y \le -2$ $\wedge [-1,1]y \le -10$
②	$y \ge -20 \wedge x \ge 10$	$y \ge -20 \wedge x \ge 10$ $\wedge [-1,1]y \le -10$

Fig. 6. program5 (left) and the generated invariants (right).

Fig. 7 shows the comparison of performance and result invariants for a selection of benchmark examples.[2] The first set of benchmark programs, program1-5, corresponds to examples shown in Fig. 2-6. The second set of examples is reused from our previous work [4], most of which come from Interproc. For each program, "#vars" indicates the total number of program variables, and "#±" indicates the number of variables which have unrestricted sign. 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 with the fixed threshold $k = 10$ in Def. 6). "#iterat." gives the number of increasing iterations during the analysis.

Invariants. The column "Result Invar." compares the invariants obtained. A ">" ("<") indicates that *itvPol* outputs stronger (weaker) invariants than FPPol. For pro-

[2] We also analyzed the benchmark programs using NewPolka which is implemented in exact arithmetic, and the result invariants are almost the same as those by FPPol. *itvPol* performs 2 times faster than NewPolka on ratelimiter_f and at worst 4 times slower on other programs.

Program		Analyzer	*itvPol*			FPPol			Result
name	#vars(#±)	#∇delay	#iterat.	#lp	*time*(*ms*)	#iterat.	#lp	*time*(*ms*)	Invar.
program1	2(2)	1	1	256	31	1	138	24	>
program2	3(3)	1	1	78	12	1	54	11	>
program3	2(2)	1	1	68	13	1	0	6	>
program4	1(1)	3	4	19	10	4	8	7	>
program5	2(1)	1	5	270	49	6	187	35	>
sequencewhiles	3(1)	1	9	368	61	9	237	46	>
ratelimiter_f	5(4)	4	4	5846	792	5	2966	1425	>
bubblesort	4(4)	1	8	845	123	8	646	101	>
maccarthy91	3(2)	1	4	609	83	4	442	63	>
heapsort	7(7)	1	4	1534	273	4	1929	374	<
symmetricalstairs	2(1)	1	5	245	45	6	469	78	<
ackerman	4(2)	1	4	883	127	6	1477	298	<

Fig. 7. Experimental results for benchmark examples.

grams involving variables unrestricted in sign, *itvPol* can often find some interesting non-convex invariants. When all variables in the program are restricted in sign, in most cases *itvPol* generates no better invariants than FPPol, since *itvPol* uses the weak join $\sqcup_w$ which is weaker than the polyhedral convex hull of FPPol in such a case.

Performance. "time(ms)" presents the analysis times in milliseconds when the analyzer is run on a 1.6GHz PC with 768MB of RAM running Fedora 9. Fig. 7 shows that the overall computation cost of *itvPol* is not so high compared with FPPol. The reason can be derived mainly from the fact that *itvPol* uses the weak join $\sqcup_w$. In some cases, e.g., ratelimiter_f and heapsort, *itvPol* even outperforms FPPol. "#lp" shows the number of LP queries issued to GLPK. During our experiments, we found that the time spent in the LP solver frequently takes at least half of the total analysis time when using *itvPol*.

8 Conclusion

In this paper, a new numerical abstract domain called *interval polyhedra (itvPol)* was presented, which introduces interval linear algebra to static analysis. This domain can represent and manipulate linear constraints with interval coefficients. *itvPol* has some attractive features in that it natively allows expressing certain non-convex (even unconnected) properties without any explicit disjunctive representations. The domain operations can be constructed by interval linear programming and interval Fourier-Motzkin elimination. Possible applications of *itvPol* are described, e.g., to handle programs involving disjunctive, non-linear, or floating-point expressions. *itvPol* can discover interesting non-convex properties for programs involving variables unrestricted in sign.

It remains for future work to design more precise or even optimal abstractions for the join of the *itvPol* domain, and to test *itvPol* on large realistic programs. We also plan to improve the efficiency of *itvPol*, e.g., by reducing the number of LP queries.

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

References

1. APRON numerical abstract domain library. http://apron.cri.ensmp.fr/library/.
2. X. Allamigeon, S. Gaubert, and E. Goubault. Inferring min and max invariants using max-plus polyhedra. In *SAS'08*, volume 5079 of *LNCS*, pages 189–204. Springer Verlag, 2008.
3. R. Bagnara, P. M. Hill, and E. Zaffanella. Widening operators for powerset domains. In *VMCAI'04*, volume 2477 of *LNCS*, pages 135–148. Springer Verlag, 2004.
4. 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.
5. J.W. Chineck and K. Ramadan. Linear programming with interval coefficients. *Journal of the Operational Research Society*, 51(2):209–220, 2000.
6. 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.
7. 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, New York, 1977.
8. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *ACM POPL'79*, pages 269–282. ACM Press, New York, 1979.
9. P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages). In *ICCL'94*, pages 95–112. IEEE Computer Society Press, 1994.
10. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *ACM POPL'78*, pages 84–96. ACM Press, New York, 1978.
11. R. Giacobazzi and F. Ranzato. Optimal domains for disjunctive abstract interpretation. *Sci. Comput. Program*, 32(1-3):177–210, 1998.
12. P. Granger. Static analysis of arithmetical congruences. *International Journal of Computer Mathematics*, 30:165–199, 1989.
13. 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*, volume 5123 of *LNCS*, pages 370–384. Springer-Verlag, 2008.
14. C. Jansson. Calculation of exact bounds for the solution set of linear interval systems. *Linear Algebra and Its Applications*, 251:321–340, 1997.
15. C. Jansson. Rigorous lower and upper bounds in linear programming. *SIAM Journal on Optimization*, 14(3):914–935, 2004.
16. G. Lalire, M. Argoud, and B. Jeannet. Interproc. http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc/.
17. 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 Verlag, 2009.
18. A. Makhorin. The GNU Linear Programming Kit, 2000. http://www.gnu.org/software/glpk/.
19. A. Miné. Relational abstract domains for the detection of floating-point run-time errors. In *ESOP'04*, volume 2986 of *LNCS*, pages 3–17. Springer, 2004.
20. A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
21. A. Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *VMCAI'06*, volume 3855 of *LNCS*, pages 348–363. Springer, 2006.
22. W. Oettli and W. Prager. Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *Numer. Math.*, 6:405–409, 1964.
23. J. Rohn. A handbook of results on interval linear problems. Technical report, Czech Academy of Sciences, Prague, Czech Republic, April 2005.
24. J. Rohn. Solvability of systems of interval linear equations and inequalities. In *Linear Optimization Problems with Inexact Data*, pages 35–77. Springer, 2006.