Inferring linear invariants with parallelotopes[☆]

Gianluca Amato[a], Marco Rubino[a], Francesca Scozzari[a]

[a] *Università di Chieti-Pescara, Italy*

Abstract

We propose a new numerical abstract domain for inferring linear invariants based on parallelotopes. The domain may encode any linear constraint, as the polyhedra abstract domain, while maintaining the efficiency of weakly relational abstract domains, such as intervals and octagons. We provide the full set of abstract operators, define a reduced product with intervals and present an experimental comparison with polyhedra and octagons. According to these experiments, the reduced product we propose is much more precise than both polyhedra and octagons in inferring interval constraints.

Keywords: Static analysis, abstract interpretation, numerical abstract domain, linear invariant, parallelotopes.

1. Introduction

In the field of static analysis by abstract interpretation, much effort has been devoted in designing domains for inferring numerical properties such as "the value of the variable x in program point p is between 0 and 10". In this paper we are mainly interested in linear invariants, which are typically expressed as linear inequalities such as $2x+3y \leq 42$. The polyhedra domain by Cousot and Halbwachs [1] is able to express any such invariant, but its computational complexity makes it difficult to use it in practice. Many other numerical domains have been proposed to gain efficiency. One of the simplest one is the Cousot and Cousot's interval domain [2], a non-relational abstract domain which can represent only constraints involving a single variable such as $0 \leq x \leq 10$. Analyses using the interval domain are not very precise, since it cannot represent relationships between variables. Many weakly relational domains, more expressive than intervals but less expressive than polyhedra, have been proposed in the last years, such as octagons [3], octahedra [4], logahedra [5] and TVPI [6]. Weakly relational domains have proved to be quite efficient, but the invariants that can be inferred using these domains are seriously limited by syntactic restrictions,

Email addresses: `gamato@unich.it` (Gianluca Amato), `marco.rubino@unich.it` (Marco Rubino), `fscozzari@unich.it` (Francesca Scozzari)

since expressivity is bartered for efficiency. For instance, the octagons abstract domain can only deal with inequalities involving two variables whose coefficients are $\{-1, 0, 1\}$, such as $x - y \leq 5$, while TVPI can only express inequalities with two variables, such as $5x + 2y \leq 8$.

In order to handle more expressive constraints, Sankaranarayanan et al. have proposed a different approach called template polyhedra [7]. For each program, they fix a constraint matrix A and consider all the polyhedra of the form $\{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{b}\}$, where $\boldsymbol{x}$ is the vector of program variables. The *a priori* choice of the matrix differentiates template polyhedra from other domains, where the matrix is fixed for all the programs (such as intervals or octagons) or varies freely (such as polyhedra). Template polyhedra generalize most weakly relational domains, with the difference that its abstract operators are defined by means of linear programming. Along the same direction there are the proposals of generalized template polyhedra [8], which combine template polyhedra and bilinear forms, and template parallelotopes [9, 10], which are a special case of template polyhedra. A parallelotope is a polyhedron defined by at most n linearly independent constraints, where n is the number of variables. Any parallelotope can be described as the set of points $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{b}_1 \leq A\boldsymbol{x} \leq \boldsymbol{b}_2\}$, where $\boldsymbol{x}$ is the vector of program variables and the constraint matrix A is square and invertible. In the special case of template parallelotopes, it is possible to derive efficient abstract operators, without resorting to linear programming tools. In general, when using templates, the result of the analyses greatly varies depending on the choice of the matrix A, and finding such matrices is a hard and still open problem. We are aware of only two proposals for generating templates, one in Sankaranarayanan et al. original paper [7] based on a syntactic inspection of the program, and another one based on the statistical analysis of partial execution traces [9, 10].

Here we propose a different approach which does not use templates and does not restrict the syntactic shape of constraints, still maintaining efficiency. We propose a domain whose abstract objects are parallelotopes, namely we only limit the number of constraints to n (the number of variables) and we require the constraints to be linearly independent.

While our abstract objects are parallelotopes as in [9, 10], there is the fundamental difference that the constraint matrix is not fixed *a priori* but may freely change during the analysis, as for the polyhedra domain. The domain of parallelotopes so defined can encode any linear constraints, does not require templates and can be equipped with very efficient abstract operations, whose complexity is comparable to that of octagons. The key to scalability is the restriction to n linearly independent constraints. This allows the use of algorithms adapted from linear equation solving which are simpler and more efficient than those used in the theory of polyhedra. For instance, minimizing a linear form on a parallelotope is cubic, since it essentially amounts to solving a linear equation system, while the corresponding operation on polyhedra and even on template polyhedra requires the use of the simplex algorithm.

Since many concrete operations are not closed with respect to parallelotopes, in most cases there is not a unique best parallelotope which approximates the

result of the concrete operation. The careful design of suitable abstract operators is then a key point in the parallelotope domain. In some cases, as for the abstract union operator, we propose different operators, and in general we resort to appropriate heuristics in order to ensure good precision in the overall analysis.

We also propose a combination of the domain of parallelotopes and intervals, and we experimentally show that it compares well with respect to polyhedra and octagons. Finally, we show that, if we are interested in inferring precise interval constraints, then the combination of parallelotopes and intervals is far more precise than both polyhedra and octagons.

Preliminary results on the domain of parallelotopes appeared in [11]. We provide here a new comprehensive presentation of the domain, containing several novelties such as new abstract operators on parallelotopes, the reduced product of parallelotopes and intervals, experimental evaluations, and proofs (which are in the Appendix).

2. Notation

2.1. Linear Algebra

We denote by $\bar{\mathbb{R}}$ the set of real numbers extended with $+\infty$ and $-\infty$. Addition and multiplication are partially extended, when possible, to $\bar{\mathbb{R}}$ in the obvious way. We avoid the use of indeterminate forms, with the exception of 0 times $\pm\infty$ which we define to be 0. We use boldface for elements $\boldsymbol{v}$ of $\bar{\mathbb{R}}^n$. Any vector $\boldsymbol{v} \in \bar{\mathbb{R}}^n$ is intended as a column vector, and $\boldsymbol{v}^T$ is the corresponding row vector. We denote by $\boldsymbol{u} \cdot \boldsymbol{v}$ the dot product of $\boldsymbol{u}$ and $\boldsymbol{v}$ and we use it indifferently for both row and column vectors. Given $\boldsymbol{u}, \boldsymbol{v} \in \bar{\mathbb{R}}^n$, and a relation $\bowtie \in \{<, >, \leq, \geq, =\}$, we write $\boldsymbol{u} \bowtie \boldsymbol{v}$ if and only if $u_i \bowtie v_i$ for each $i \in \{1, \ldots, n\}$. Given $\boldsymbol{u} \in \mathbb{R}^n$ and $i \in \{1, \ldots, n\}$, we write $\boldsymbol{u}[i \mapsto x]$ to denote a vector $\boldsymbol{v}$ such that $v_i = x$ and $v_j = u_j$ for $j \neq i$.

If $A = (a_{ij})$ is a matrix, we denote by A^T its *transpose*. If A is invertible, A^{-1} denotes its inverse, and $\mathrm{GL}(n)$ is the group of $n \times n$ invertible matrices. The identity matrix in $\mathrm{GL}(n)$ is denoted by I_n and the standard basis of $\mathbb{R}^n$ is denoted by $\{\boldsymbol{e}^1, \ldots, \boldsymbol{e}^n\}$. Clearly, any $1 \times n$-matrix can be viewed as a vector: in particular, we denote by $\boldsymbol{a}_{i*}$ the row vector given by the i-th row of any matrix A, and by $\boldsymbol{a}_{*i}$ the column vector given by the i-th column of A.

More generally, if J is a set of indexes, then A_{J*} is the submatrix of A composed of the rows in J and A_{*J} the submatrix composed of the columns in J. We use A_{-J*} for the submatrix of A composed of the rows of A whose indexes are not in J.

In the rest of the paper, when talking about the computational complexity of matrix and vector operations, we always assume to use a dense representation, which is the one adopted in our implementation.

2.2. Convex sets

A set $C \subseteq \mathbb{R}^n$ is *convex* when, given $\boldsymbol{x}, \boldsymbol{y} \in C$, for each $\lambda \in [0, 1]$ we have that $\lambda\boldsymbol{x} + (1 - \lambda)\boldsymbol{y} \in C$. A *ray* in C is a non-null vector $\boldsymbol{v} \in \mathbb{R}^n$ such that for

each $\boldsymbol{x} \in C$ and $\alpha \geq 0$, $\boldsymbol{x} + \alpha\boldsymbol{v} \in C$. A convex set C is *bounded* iff it has no rays. A *line* in C is a vector $\boldsymbol{v} \in \mathbb{R}^n$ such that both $\boldsymbol{v}$ and $-\boldsymbol{v}$ are rays. The lines in C form a vector space, called the *linearity space* of C, denote by $\mathrm{lin}(C)$. Given a vector space V, we denote by $V^\perp$ its orthogonal complement.

A set of vectors $S = \{v_1, \ldots, v_m\}$ is *affinely dependent* if there are $\lambda_1, \ldots, \lambda_m \in \mathbb{R}$ such that they are not all zero, $\lambda_1 + \cdots + \lambda_m = 0$ and $\lambda_1\boldsymbol{v}_1 + \cdots + \lambda_m\boldsymbol{v}_m = 0$. The *affine hull* of S is the set $\mathrm{aff.\,hull}(S) = \{\lambda_1\boldsymbol{v}_1 + \cdots + \lambda_m\boldsymbol{v}_m \mid \lambda_1 + \cdots + \lambda_m = 1\}$.

A *flat* F is a set of points of the form $\boldsymbol{a} + H$ where H is a vector space. Given a flat, H is uniquely determined. Therefore, we call the dimension of F the dimension of the corresponding vector space H. By convention, the dimension of the empty set is -1. The *dimension* of a set S is the dimension of the smallest flat containing S, and it is denoted by $\dim(S)$.

2.3. Polyhedra and constraints

An (affine) *constraint* over a vector of n variables is a syntactic object $\boldsymbol{a}\cdot\boldsymbol{x} \leq b$, where $\boldsymbol{a} \in \mathbb{R}^n$ and $b \in \mathbb{R}$. We use $l \leq \boldsymbol{a}\cdot\boldsymbol{x} \leq u$ as a short form for the pair of constraints $\boldsymbol{a}\cdot\boldsymbol{x} \leq u$ and $-\boldsymbol{a}\cdot\boldsymbol{x} \leq -l$. If $C = \{\boldsymbol{a}_1\cdot\boldsymbol{x} \leq b_1, \ldots, \boldsymbol{a}_m\cdot\boldsymbol{x} \leq b_m\}$ is a finite set of constraints, its *solution set* is $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{a}_1\cdot\boldsymbol{x} \leq b_1, \ldots, \boldsymbol{a}_m\cdot\boldsymbol{x} \leq b_m\}$. Given a constraint $c \in C$, we say that c is *redundant* for C when the solution sets of C and $C \setminus \{c\}$ coincide. Equivalently, a constraint $c \equiv \boldsymbol{a}\cdot\boldsymbol{x} \leq b$ is redundant iff there is a family of constraints $\{c_i\}_{i\in I} \subseteq C \setminus \{c\}$ with $c_i \equiv \boldsymbol{a}_i\cdot\boldsymbol{x} \leq b_i$ and a family of positive numbers $\{\lambda_i\}_{i\in I}$ such that $\boldsymbol{a} = \sum_{i\in I}\lambda_i\boldsymbol{a}_i$ and $b \geq \sum_{i\in I}\lambda_i b_i$. Informally, this means that c is implied by a positive linear combinations of the c_i's. This characterization of redundant constraints is known as Farkas' Lemma.

A set $P \subseteq \mathbb{R}^n$ is a *polyhedron* when there exists an $m \times n$ matrix A and a vector $\boldsymbol{u} \in \mathbb{R}^n$ such that

$$P = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{u}\} \ .$$

For each $i \in \{1, \ldots, m\}$, $\boldsymbol{a}_{i*}\cdot\boldsymbol{x} \leq u_i$ is a constraint, therefore P may be viewed as the solution set of a finite set of constraints.

The Fourier-Motzkin elimination [12, Chapter 12.2] is a method to eliminate a variable from a set of affine constraints. Given two constraints $c_1 \equiv \boldsymbol{a}\cdot\boldsymbol{x} \leq b$ and $c_2 \equiv \boldsymbol{d}\cdot\boldsymbol{x} \leq e$ such that $a_1 < 0$ and $d_1 > 0$, we may combine c_1 and c_2 into a new constraint $c_3 \equiv (d_1\boldsymbol{a} - a_1\boldsymbol{d})\cdot\boldsymbol{x} \leq d_1 b - a_1 e$ where the coefficient of the variable x_1 is zero. Given a set of constraints C, we partition C into C^+, C^- and C^0 according to the sign of the coefficient of x_1. Then, we denote by C^{+-} the set of all the constraints obtained combining a constraint in C^- and one in C^+ as shown above. The set $C^{+-} \cup C^0$ is a set of constraints over the variables $x_2, \ldots, x_n$, and has the property that a point $(x_2, \ldots, x_n) \in \mathbb{R}^{n-1}$ is in the solution set of $C^{+-} \cup C^0$ if and only if there exists $x_1 \in \mathbb{R}$ such that $(x_1, \ldots, x_n)$ is in the solution set of C.

2.4. Abstract interpretation

In this paper we adopt a framework for abstract interpretation which is weaker than the common one based on Galois connections (see [13, Section 7]).

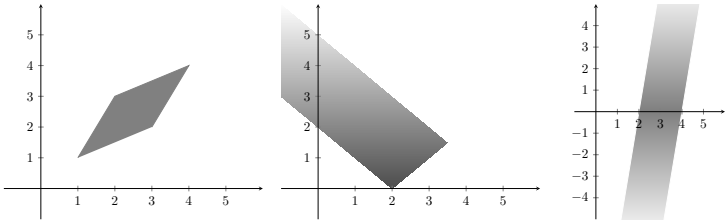


Figure 1: Parallelotopes.

Given a partially-ordered set $(C, \leq_C)$ — the *concrete domain* — and a pre-ordered set $(A, \leq_A)$ — the *abstract domain* — we establish an abstraction–concretization relationship between them with the use of a *concretization map*, which is a monotone function $\gamma : A \to C$. We say that $a \in A$ is a *correct approximation* of $c \in C$ when $c \leq_C \gamma(a)$. Given $c \in C$, there are many correct abstractions. The most interesting ones are those which are minimal w.r.t. the preorder $\leq_A$.

A function $f^\alpha : A \to A$ is a *correct abstraction* of $f : C \to C$ when it preserves correctness of approximation, i.e. when $c \leq_C \gamma(a)$ implies $f(c) \leq_C \gamma(f^\alpha(a))$. It is γ-*complete* when $\gamma \circ f^\alpha = f \circ \gamma$. It is *minimal* when, for any $a, a' \in A$, if $f(\gamma(a)) \leq_C \gamma(a') \leq_C \gamma(f^\alpha(a))$, then $\gamma(a') = \gamma(f^\alpha(a))$, i.e., when $f^\alpha(a)$ is a minimal correct approximation of $f(\gamma(a))$.

When $\leq_A$ is a partial order and γ has a left adjoint, i.e., there exists a monotone function $\alpha : C \to A$ such that, $\forall a \in A \, \forall c \in C \, (\alpha(c) \leq a \iff c \leq \gamma(a))$, we say that α and γ form a *Galois connection*. The function α maps each $c \in C$ to its *best correct approximation* $\alpha(c)$, i.e., the least correct abstraction according to $\leq_A$. For each $f : C \to C$ it is possible to define its *best correct abstraction* as $f^\alpha = \alpha \circ f \circ \gamma$. When γ is injective, we say that α and γ form a *Galois insertion*.

3. Parallelotopes and their representation

A set $P \subseteq \mathbb{R}^n$ is a *parallelotope* when there is an invertible matrix $A \in \mathrm{GL}(n)$ and vectors $\boldsymbol{l}, \boldsymbol{u} \in \bar{\mathbb{R}}^n$ such that

$$P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}\} \ . \tag{1}$$

A parallelotope is a closed convex set. The matrix A is called the *constraint matrix*, while $\boldsymbol{l}$ and $\boldsymbol{u}$ are the *lower* and *upper bounds* respectively. While in the mathematical literature a parallelotope is generally considered to be bounded, we are also considering unbounded ones. We call *box* a parallelotope whose constraint matrix is the identity matrix. Examples of parallelotopes are given in Figure 1.

In this section we define the abstract domain of parallelotopes and its relationship with the concrete domain of subsets of $\mathbb{R}^n$.

3.1. Representation of parallelotopes

First of all, we need a finite representation for parallelotopes. Non-empty parallelotopes are represented with a constraint matrix and corresponding lower and upper bounds, while a special symbol ϵ is used for empty parallelotopes.

Definition 1 (Representations of parallelotopes). A *representation* $\mathcal{P}$ *of a parallelotope* is either the symbol ϵ or a tuple $\langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ with $A \in \mathrm{GL}(n)$, $\boldsymbol{l}, \boldsymbol{u} \in \bar{\mathbb{R}}^n$ subject to the following additional conditions:

- $\boldsymbol{l} \leq \boldsymbol{u}$;
- for all $i \in \{1, \ldots, n\}$ $l_i \neq +\infty$ and $u_i \neq -\infty$.

We denote by Par_n the set of all the representations for parallelotopes in $\mathbb{R}^n$. We simply use Par when n is not relevant.

The last two conditions in Definition 1, together with the fact that A is invertible, ensure that $\langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ represents a non-empty parallelotope. We prefer to keep a different representation for the empty case, since this simplifies the definition of the abstract operators. The relationship between parallelotopes and their representations is formalized by the following concretization map.

Definition 2 (Concretization map). We define a concretization map $\gamma : \mathsf{Par}_n \to \mathscr{P}(\mathbb{R}^n)$ from representations of parallelotopes to paralletopes as follows:

$$\gamma(\epsilon) = \emptyset,$$
$$\gamma(\langle A, \boldsymbol{l}, \boldsymbol{u} \rangle) = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}\}.$$

Theorem 1. *For every parallelotope* P *there is a representation* $\mathcal{P}$ *such that* $\gamma(\mathcal{P}) = P$.

In the following, when this does not cause any ambiguities, we use a representation $\mathcal{P}$ in place of the parallelotope $\gamma(\mathcal{P})$. Moreover, we refer to representations of parallelotopes simply as parallelotopes.

3.2. Some properties of parallelotopes

We want to study the connection between the properties of a parallelotope as a convex set (rays, boundedness, etc. . .) and its representation. Most of these results may be found in standard textbooks with slightly different notations. We provide the proofs in the appendix for clarity of presentation.

Definition 3 (Constrained, unconstrained and equality rows). A row $\boldsymbol{a}_{i*}$ of a representation $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ is said to be: (1) *constrained* when $l_i \in \mathbb{R}$ or $u_i \in \mathbb{R}$; (2) *unconstrained* when $l_i = -\infty$ and $u_i = +\infty$; (3) *equality* when $l_i = u_i$. We denote by $U(\mathcal{P})$ (resp. $E(\mathcal{P})$) the set of indexes of the unconstrained rows (resp. equality rows) in $\mathcal{P}$.

Theorem 2 (Parallelotopes and Representations)**.** *If $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ is a representation of a parallelotope in $\mathbb{R}^n$, the following holds:*

1. $\dim(\mathcal{P})$ *is equal to* $n - |E(\mathcal{P})|$*;* [1]
2. $\boldsymbol{v} \neq 0$ *is a ray in* $\mathcal{P}$ *iff for each* $i \in \{1, \ldots, n\}$, $l_i \in \mathbb{R}$ *implies* $\boldsymbol{a}_{i*} \cdot \boldsymbol{v} \geq 0$ *and* $u_i \in \mathbb{R}$ *implies* $\boldsymbol{a}_{i*} \cdot \boldsymbol{v} \leq 0$*;*
3. $\mathcal{P}$ *is bounded iff all the bounds are finite;*
4. $\boldsymbol{v} \neq 0$ *is a line in* $\mathcal{P}$ *iff it is orthogonal to all the constrained rows.*

Corollary 3. *The linearity space of a parallelotope $\mathcal{P}$ is $V^\perp$ where V is the linear space generated by all the constrained rows of $\mathcal{P}$. It is generated by the orthogonal projections of the unconstrained rows onto $V^\perp$.*

3.3. Minimization and maximization over a parallelotope

One of the basic operations on abstract domains is computing the minimum and maximum values of a linear form over the points in an abstract object. If $\mathcal{P}$ is a parallelotope and $\boldsymbol{c} \in \mathbb{R}^n$, this means computing $\inf_{\boldsymbol{x}\in\mathcal{P}} \boldsymbol{c}\cdot\boldsymbol{x}$ and $\sup_{\boldsymbol{x}\in\mathcal{P}} \boldsymbol{c}\cdot\boldsymbol{x}$.

The minimization and maximization operator for parallelotopes may be obtained by viewing a parallelotope as a box over a non-canonical coordinate system. Given a box $\mathcal{B} = \langle I_n, \boldsymbol{l}, \boldsymbol{u}\rangle$ and a vector $\boldsymbol{c} \in \mathbb{R}^n$, it is well known that

$$\inf_{\boldsymbol{x}\in\mathcal{B}} \boldsymbol{c}\cdot\boldsymbol{x} = \inf_{\boldsymbol{l}\leq\boldsymbol{x}\leq\boldsymbol{u}} \boldsymbol{c}\cdot\boldsymbol{x} = \boldsymbol{c}\cdot\boldsymbol{v} \quad \text{where} \quad v_i = \begin{cases} l_i & \text{if } c_i \geq 0 \\ u_i & \text{otherwise.} \end{cases}$$

The computational complexity of this operation is $O(n)$.

Theorem 4. *Given the representation $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ and a vector $\boldsymbol{c} \in \mathbb{R}^n$, we have that*

$$\inf_{\boldsymbol{x}\in\mathcal{P}} \boldsymbol{c}\cdot\boldsymbol{x} = \inf_{\boldsymbol{l}\leq\boldsymbol{y}\leq\boldsymbol{u}} \boldsymbol{c}^T A^{-1}\boldsymbol{y}.$$

The computational complexity of the minimization operation is $O(n^3)$.

The same properties hold for the maximization operator.

3.4. Parallelotopes over a fixed constraint matrix

We say that the parallelotope P *is definable over* $A \in \mathrm{GL}(n)$ if it is empty or there is a representation for P with A as the constraint matrix. We denote by Par_n^A the subset of Par_n comprised of ϵ and the non-empty representations with A as the constraint matrix. Hence, P is definable over A if there is a representation for it in Par_n^A.

[1] As an example of the problems which arise in considering representations with $\boldsymbol{l} \not\leq \boldsymbol{u}$, this property is false in that case, since $\mathcal{P}$ is empty and $\dim(\mathcal{P}) = -1$ regardless of the number of equality rows.

The abstract domain of parallelotopes with a fixed constraint matrix has been thoroughly studied in [10]. Here we recall some results which are relevant to our paper, adapting the notations of [10] to the ones we are using here.

First of all, it is possible to define on Par_n^A a partial order as follows:

$$\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \leq \langle A, \boldsymbol{l}', \boldsymbol{u}'\rangle \text{ iff } \boldsymbol{l}' \leq \boldsymbol{l} \text{ and } \boldsymbol{u}' \geq \boldsymbol{u},$$

extended with the proviso that $\epsilon \leq \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ for each $\boldsymbol{l}, \boldsymbol{u} \in \bar{\mathbb{R}}^n$. With respect to this ordering, Par_n^A is a complete lattice and γ is a complete join-morphism, hence it has left adjoint $\alpha_A : \mathscr{P}(\mathbb{R}^n) \to \mathsf{Par}_n^A$ such that $\alpha_A(\emptyset) = \epsilon$ and, for $C \neq \emptyset$, $\alpha_A(C) = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ with $l_i = \inf_{\boldsymbol{x}\in C} \boldsymbol{a}_{i*}\boldsymbol{x}$ and $u_i = \sup_{\boldsymbol{x}\in C} \boldsymbol{a}_{i*}\boldsymbol{x}$. Maps α_A and γ form a Galois connection. Since γ is injective, it is actually a Galois insertion.

In particular, this means that for each $C \subseteq \mathscr{P}(\mathbb{R}^n)$, $\alpha_A(C)$ gives the least parallelotope which contains C and is definable over A. As a particular case, it is interesting, given a parallelotope P in $\mathbb{R}^n$ and a matrix $A' \in \mathrm{GL}(n)$, to find out the least parallelotope containing P and definable over A'.

Definition 4. Given a parallelotope $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ and a matrix $A' \in \mathrm{GL}(n)$, the operator $\mathsf{rot}_{A'}$ is defined as

$$\mathsf{rot}_{A'}(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle) = \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$$

where

$$B = A'A^{-1} \ , \quad l'_i = \inf_{\boldsymbol{l}\leq\boldsymbol{x}\leq\boldsymbol{u}} \boldsymbol{b}_{i*}\boldsymbol{x} \ , \quad u'_i = \sup_{\boldsymbol{l}\leq\boldsymbol{x}\leq\boldsymbol{u}} \boldsymbol{b}_{i*}\boldsymbol{x} \ .$$

This is extended to the empty parallelotope as $\mathsf{rot}_{A'}(\epsilon) = \epsilon$.

Example 1. Given the parallelotope

$$P = \{\boldsymbol{x} \in \mathbb{R}^n \mid 0 \leq -x_1 + 3x_2 \leq 2, 3 \leq x_1 + 2x_2 \leq 8\}$$

depicted in gray in Figure 2, and given the matrix $A = \left(\begin{smallmatrix} 1 & 1 \\ -1 & 1 \end{smallmatrix}\right)$, $\mathsf{rot}_A(P)$ is the dotted parallelotope.

Theorem 5. *Given a parallelotope $\mathcal{P}$ and $A' \in \mathrm{GL}(n)$, $\mathsf{rot}_{A'}(\mathcal{P})$ is the least parallelotope definable over A' which contains $\mathcal{P}$, i.e., $\mathsf{rot}_{A'}(\mathcal{P}) = \alpha_{A'}(\gamma(\mathcal{P}))$. The computational complexity of $\mathsf{rot}_{A'}$ is $O(n^3)$.*

3.5. Parallelotopes over arbitrary constraint matrices

Given a set $C \subseteq \mathbb{R}^n$, we are interested in approximating C with a parallelotope $P \supseteq C$. In general, there is not a least parallelotope P which contains C, but there are several (possibly infinitely many) minimal parallelotopes with such a property. In particular, for each $A \in \mathrm{GL}(n)$, $\gamma(\alpha_A(C))$ is a minimal parallelotope containing C.

It is possible to define a pre-order over parallelotopes in such a way that $\mathcal{P} \leq \mathcal{P}'$ iff $\gamma(\mathcal{P}) \subseteq \gamma(\mathcal{P}')$. The idea is that if $\gamma(\mathcal{P}) \subseteq \gamma(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle)$, then $\gamma(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle)$ contains the least parallelotope definable over A which contains $\gamma(\mathcal{P})$.

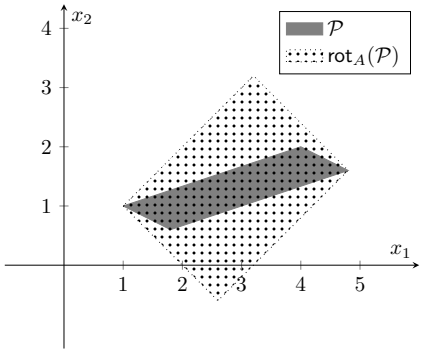


Figure 2: The rot operator (see Example 1).

Definition 5. Given two representations of parallelotopes $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ and $\mathcal{P}' = \langle A', \boldsymbol{l}', \boldsymbol{u}' \rangle$, we say $\mathcal{P} \leq \mathcal{P}'$ iff $\boldsymbol{l}' \leq \boldsymbol{l}''$ and $\boldsymbol{u}'' \leq \boldsymbol{u}'$, where $\langle A', \boldsymbol{l}'', \boldsymbol{u}'' \rangle = \mathsf{rot}_{A'}(\mathcal{P})$. Moreover, $\epsilon \leq \mathcal{P}$ for each representation $\mathcal{P}$.

Note that there is no ambiguity in using the same symbol $\leq$ as for the ordering over Par_A^n. If $A = A'$, then $\mathsf{rot}_{A'}(\langle A, \boldsymbol{l}, \boldsymbol{u} \rangle) = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ and the two definitions coincide.

Theorem 6. *Given two representations $\mathcal{P}$ and $\mathcal{P}'$, we have that $\mathcal{P} \leq \mathcal{P}'$ iff $\gamma(\mathcal{P}) \subseteq \gamma(\mathcal{P}')$. Moreover, $\leq$ is a pre-order. The computational complexity of deciding $\leq$ is $O(n^3)$.*

Theorem 6 allows us to state the precise abstract framework we use in this paper. The concrete domain is the powerset of $\mathbb{R}^n$, ordered by set inclusion, while the abstract domain is the set Par_n of all the representations of parallelotopes, according to Definition 1, ordered by $\leq$ from Definition 5. The two domains are related by the monotonic concretization map γ of Definition 2.

3.6. On inverse matrices and factorizations

Most of the operations for parallelotopes require either computing the inverse of the constraint matrix A or solving a system of linear equations. In our implementation we use Gaussian elimination in both cases,

Solving systems of linear equations may be done more efficiently if we know a factorization (such as LU or QR) of the constraint matrix. Therefore, memoizing this factorization may be useful to improve efficiency. Moreover, it could be investigated whether it is possible to redefine the abstract operators in order to incrementally and automatically keep track of a factorization for the constraint matrix. This is left for future work.

4. Operators on parallelotopes

In this section we consider the operations on $\mathscr{P}(\mathbb{R}^n)$ commonly used when defining the collecting semantics of imperative programming languages, and

for each of them we introduce a correct approximation on parallelotopes. The main difficulty in defining the abstract operations on parallelotopes is that, in the general case, a subset of $\mathbb{R}^n$ has not a best correct abstraction in Par_n. Different constraint matrices give rise to minimal but incomparable abstractions. Therefore, defining the abstract operations on parallelotopes essentially amounts to:

1. choosing a resulting constraint matrix A;
2. computing bounds $\boldsymbol{l}$ and $\boldsymbol{u}$ to get a correct approximation of the result.

Once A is fixed, computing the tightest bounds is quite easy. In practice, the two steps of determining A and $\boldsymbol{l}, \boldsymbol{u}$ will be carried out at the same time, for the sake of efficiency. However, there are generally several possible alternatives for A, which lead to results which are set-theoretically incomparable. The choice between different matrices may only be done under the basis of heuristic considerations, and validation requires extensive tests.

From a theoretical perspective, in evaluating the precision of an abstract operator we look for the following properties, in order of preference:

1. γ-*completeness*, if possible, i.e. when the result of the concrete operator is a parallelotope;
2. *minimality*, i.e. we compute one of the minimal parallelotopes which approximate the concrete result;
3. *relative optimality*, i.e. we fix a matrix A and compute the least parallelotope definable over A which approximates the concrete result.

It is easy to check that γ-completeness implies minimality which, in turn, implies relative optimality.

In general, our abstract operators will not be monotone. Given two inputs $\mathcal{P} \leq \mathcal{P}'$, different choices of the constraint matrix of the corresponding results may lead to non-monotonicity. However, abstract operators are monotone when they are γ-complete.

4.1. Invertible linear assignment

Linear assignment are used to analyze the behavior of the statement $x_i = c_1x_1 + \ldots + c_nx_n + b$. The concrete linear assignment operation $\mathsf{assign}(i, \boldsymbol{c}, b) : \mathscr{P}(\mathbb{R}^n) \to \mathscr{P}(\mathbb{R}^n)$ is defined as

$$\mathsf{assign}(i, \boldsymbol{c}, b)(X) = \{\boldsymbol{x}[i \mapsto \boldsymbol{c} \cdot \boldsymbol{x} + b] \mid \boldsymbol{x} \in X\}.$$

If $c_i \neq 0$, then $\mathsf{assign}(i, \boldsymbol{c}, b)$ is invertible and, most importantly, maps a parallelotope to a parallelotope. In this special case, it is possible to implement the abstract operator along the line of [1]. Intuitively, the operation $\mathsf{assign}(i, \boldsymbol{c}, b)$ corresponds to the assignment $x_i' = \boldsymbol{c} \cdot \boldsymbol{x} + b$, where x_i' is the value of x_i after the assignment. From this equation, it is possible to recover x_i as a function of x_i' and the other elements of $\boldsymbol{x}$, namely $x_i = (x_i' - \sum_{j \neq i} c_j x_j - b)/c_i$. Replacing the variable x_i in $\boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}$ with the latter definition, we obtain a representation for the result of the assignment.

Definition 6. Given $\boldsymbol{c} \in \mathbb{R}^n$ such that $c_i \neq 0$ and $b \in \mathbb{R}$, we define the abstract linear assignment $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b) : \mathsf{Par}_n \to \mathsf{Par}_n$ as

$$\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)(\epsilon) = \epsilon$$
$$\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle) = \langle A - \frac{1}{c_i}\boldsymbol{a}_{*i}(\boldsymbol{c} - \boldsymbol{e}^i)^T, \boldsymbol{l} + \frac{b}{c_i}\boldsymbol{a}_{*i}, \boldsymbol{u} + \frac{b}{c_i}\boldsymbol{a}_{*i}\rangle \ .$$

Theorem 7. *The operation* $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)$ *is correct and* γ*-complete. The computational complexity is* $O(mn)$ *where* m *is the number of non-zero components in* $\boldsymbol{c}$*.*

The case when $c_i = 0$ will be treated after the non-deterministic assignment.

4.2. Non-deterministic assignment

Consider the non-deterministic assignment operation $\mathsf{forget}(i) : \mathscr{P}(\mathbb{R}^n) \to \mathscr{P}(\mathbb{R}^n)$, corresponding to the statement $x_i = ?$ and defined as

$$\mathsf{forget}(i)(X) = \{\boldsymbol{x}[i \mapsto v] \mid \boldsymbol{x} \in X, v \in \mathbb{R}\} \ .$$

Note that, even if P is a parallelotope, $\mathsf{forget}(i)(P)$ may not be parallelotope.

Example 2. Consider the parallelotope P given by the inequalities $-1 \leq x_1 + x_2 + x_3 \leq 1$, $-1 \leq x_1 + x_2 - x_3 \leq 1$ and $-1 \leq x_1 - x_2 + x_3 \leq 1$. The set $\mathsf{forget}(1)(P)$ is a polyhedron whose constraints may be obtained by Fourier-Motzkin elimination:

$$\mathsf{forget}(1)(P) = \{\boldsymbol{x} \in \mathbb{R}^3 \mid -1 \leq x_2 \leq 1, -1 \leq x_3 \leq 1, -1 \leq x_2 - x_3 \leq 1\}.$$

Although P is a parallelotope, $\mathsf{forget}(1)(P)$ is not.

Given a matrix A, we can characterize in a syntactic way the parallelotopes definable over A which approximate the set $\mathsf{forget}(i)(P)$. A parallelotope approximates $\mathsf{forget}(i)(P)$ iff it approximates P and any constraint containing the variable x_i is unbounded.

Proposition 8. *Given two parallelotopes* $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \leq \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$*, we have that*

$$\mathsf{forget}(i)(\gamma(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle)) \subseteq \gamma(\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle) \text{ iff } \forall j(a'_{ji} \neq 0 \to (l'_j = -\infty \wedge u'_j = +\infty)).$$

The above proposition suggests that, in order to compute $\mathsf{forget}^\alpha(i)(\mathcal{P})$, we need to find a parallelotope $\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle \geq \mathcal{P}$ such that each row of A' which contains a non-zero entry in the i-th position must be unconstrained. A naive definition of $\mathsf{forget}^\alpha(i)(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle)$ would replace the bounds of the rows j such that $a_{ji} \neq 0$ with $-\infty$ and $+\infty$. However, this generally yields a gross over approximation.

Example 3. Consider the parallelotope given by the inequalities $0 \leq x_1 + x_2 \leq 0$ and $0 \leq x_1 - x_2 \leq 0$, which consists of a single point $\{(0,0)\}$. After a non-deterministic assignment to x_2, if we apply the naive procedure described above, we get $-\infty \leq x_1 + x_2 \leq +\infty$ and $-\infty \leq x_1 - x_2 \leq +\infty$ which is the entire space. However, by adding the two original inequalities, we get the new constraint $0 \leq 2x_1 \leq 0$, which does not contain x_2 and thus is preserved by the non-deterministic assignment.

Therefore, we need to make explicit the constraints hidden in $\mathcal{P}$ which do not contain the variable x_i we want to forget. The problem is that, in general, there are more entailed constraints than we can represent with a parallelotope. We need a way to choose between competing constraints.

Example 4. Consider the parallelotope P in Example 2. When computing $\mathsf{forget}(1)(P)$ we get the implicit constraints $-1 \leq x_2 \leq 1$, $-1 \leq x_3 \leq 1$ and $-1 \leq x_2 - x_3 \leq 1$. The problem is that the linear forms x_2, x_3 and $x_2 - x_3$ are not linearly independent, hence we cannot keep all of them in the result. Note that although $x_2 - x_3$ is a linear combination of x_2 and x_3, the constraint $-1 \leq x_2 - x_3 \leq 1$ is not implied by the other two: we lose precision when we omit one of them.

In order to deal with the above problems, we propose the following approach, which is detailed in Algorithm 1. First note that we may ignore the rows in $\mathcal{P}$ which are unconstrained or whose i-th entry is zero: the first ones remain unconstrained, while the second ones are not affected by the assignment. Thus, in the following, we focus on the remaining rows only, whose indexes are in $J = \{j \mid a_{ji} \neq 0, l_j \neq -\infty \vee u_j \neq +\infty\}$. The idea is to transform the rows in J in such a way that they remain linearly independent and there is exactly one row whose i-th entry is not zero. Thus, we choose a row $r \in J$ and consider the linear combinations $R = \{a_{ji}\boldsymbol{a}_{r*} - a_{ri}\boldsymbol{a}_{j*} \mid j \in J \setminus \{r\}\} \cup \{\boldsymbol{a}_{r*}\}$. Vectors in R are linearly independent: any of its linear combination has the form

$$\sum_{j \in J \setminus \{r\}} \lambda_j (a_{ji}\boldsymbol{a}_{r*} - a_{ri}\boldsymbol{a}_{j*}) + \lambda_r \boldsymbol{a}_{r*} \quad = \\ \sum_{j \in J \setminus \{r\}} -a_{ri}\lambda_j \boldsymbol{a}_{j*} + \left(\lambda_r + \sum_{j \in J \setminus \{r\}} \lambda_j a_{ji}\right) \boldsymbol{a}_{r*} \ .$$

Since $a_{ri} \neq 0$, this linear combination is zero only if all λ_j with $j \in J$ are zero. Thus, R is a set of $|J|$ linearly independent vectors. Moreover, all the vectors in R are independent from the rows of A not in J. Combining them together, we get the resulting matrix A'. The last step is to ensure that $\mathcal{P}' \geq \mathcal{P}$, by computing the new bounds (with the exception of $\boldsymbol{a}_{r*}$ which must become an unconstrained row).

The main question is how to choose the index r in J. Our intuition is that it is better to choose an index r such that both l_r and u_r are finite, possibly equal,

Algorithm 1 The $\mathsf{forget}^{\alpha}(i)$ abstract operator

Require: $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \in \mathsf{Par}_n$, $i \in \{1, \ldots, n\}$
1: $J = \{j \mid a_{ji} \neq 0, l_j \neq -\infty \vee u_j \neq +\infty\}$
2: **if** $J = \emptyset$ **then**
3: **return** $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$
4: **end if**
5: $\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle \leftarrow \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$
6: $J_0 \leftarrow \{j \in J \mid l_j = u_j \in \mathbb{R}\}$
7: $J_1 \leftarrow \{j \in J \mid l_j, u_j \in \mathbb{R}\}$
8: **if** $J_0 \neq \emptyset$ **then**
9: $r \leftarrow$ an element in J_0
10: **else if** $J_1 \neq \emptyset$ **then**
11: $r \leftarrow$ an element in J_1
12: **else**
13: $r \leftarrow$ an element in J
14: **end if**
15: **for all** $j \in J \setminus \{r\}$ **do**
16: $\boldsymbol{a}'_{j*} \leftarrow a_{ji}\boldsymbol{a}_{r*} - a_{ri}\boldsymbol{a}_{j*}$
17: $(m_r, M_r) \leftarrow$ **if** $a_{ji} < 0$ **then** (u_r, l_r) **else** (l_r, u_r)
18: $(m_j, M_j) \leftarrow$ **if** $-a_{ri} < 0$ **then** (u_j, l_j) **else** (l_j, u_j)
19: $l'_j \leftarrow a_{ji}m_r - a_{ri}m_j$
20: $u'_j \leftarrow a_{ji}M_r - a_{ri}M_j$
21: **end for**
22: **return** $\langle A', \boldsymbol{l}'[r \mapsto -\infty], \boldsymbol{u}'[r \mapsto +\infty]\rangle$

since we get better bounds in $\mathcal{P}'$. In fact, we prove later that when we choose a row r whose lower and upper bounds coincide, then $\mathsf{forget}^{\alpha}(i)$ is γ-complete.

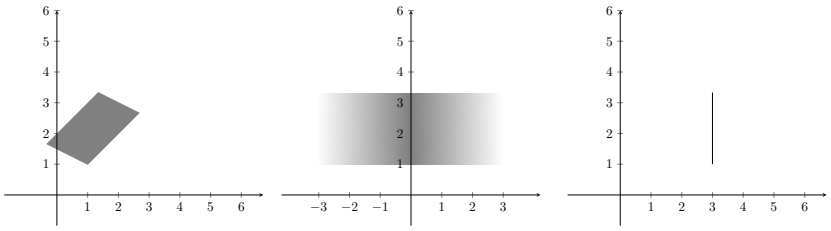
Theorem 9. *The operator* $\mathsf{forget}^{\alpha}(i)$ *described in Algorithm 1 is correct and relatively optimal. The computational complexity is* $O(n^2)$.

Given a parallelotope $P = \gamma(\mathcal{P})$, note that either $\mathsf{forget}(i)(P)$ is equal to P or its linearity space is the sum of the linearity space of P plus $\boldsymbol{e}^i$. In both cases, it is easy to check that the linearity space of $\gamma(\mathsf{forget}^{\alpha}(i)(\mathcal{P}))$ is the same as the linearity space of $\mathsf{forget}(i)(P)$.

Although in the general case the result of $\mathsf{forget}(i)(P)$ is not a parallelotope, in the particular case that $J_0 \neq \emptyset$ in Algorithm 1, $\mathsf{forget}^{\alpha}(i)(\mathcal{P})$ is γ-complete. The same holds if $\boldsymbol{e}^i$ is a line in P, i.e., $J = \emptyset$ in Algorithm 1.

Theorem 10. *If* $\boldsymbol{e}^i$ *is a line in* $\mathcal{P}$ *or* $J_0 \neq \emptyset$*, then it holds that* $\gamma(\mathsf{forget}^{\alpha}(i)(\mathcal{P})) = \mathsf{forget}(i)(\gamma(\mathcal{P}))$.

In the general case, we can prove that the $\mathsf{forget}^{\alpha}(i)$ operator is minimal using the Fourier-Motzkin elimination method [12, Chapter 12.2].



(a) Parallelotope $\mathcal{P}$ (b) $\mathsf{forget}^{\alpha}(x_1)(\mathcal{P})$ (c) Assigning $x_1 = 3$ to $\mathcal{P}$

Figure 3: Non-invertible assignment

Proposition 11. *Given a parallelotope $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ and a column index i, assume that $l_j < u_j$ for each constrained row j such that $a_{ji} \neq 0$. Consider the polyhedron obtained by Fourier-Motzkin elimination of the i-th variable. None of its constraints with finite bounds is redundant.*

Theorem 12. *The operator $\mathsf{forget}^{\alpha}(i)$ is minimal.*

It is possible to think of other heuristics for selecting the implied constraints which appear in the result. For example, sparse constraints might be preferred to dense ones. This would require further investigations.

4.3. Non-invertible assignment

We consider the assignment operator $\mathsf{assign}(i, \boldsymbol{c}, b)$ when $c_i = 0$. In this case, all the constraints involving the variable x_i need to be removed after the assignment, possibly replaced with other implied constraints. Note that if $c_i = 0$ we have $\mathsf{assign}(i, \boldsymbol{c}, b) = \mathsf{assign}(i, \boldsymbol{c}, b) \circ \mathsf{forget}(i)$. This suggests to use the abstract forget operation to make implied constraints explicit.

Example 5. Given the parallelotope

$$\mathcal{P} = \{\boldsymbol{x} \in \mathbb{R}^n \mid 0 \leq -x_1 + x_2 \leq 2, 3 \leq x_1 + 2x_2 \leq 8\}$$

depicted in Figure 3(a), the parallelotope $\mathsf{forget}^{\alpha}(x_1)(\mathcal{P})$ is depicted in Figure 3(b). The result of the non-invertible assignment $x_1 = 3$ to the parallelotope $\mathcal{P}$ is in Figure 3(c).

The procedure is shown in Algorithm 2. Given a parallelotope $\mathcal{P}$, we first compute $\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle = \mathsf{forget}^{\alpha}(i)(\mathcal{P})$ and choose a row j in A' with $a'_{ji} \neq 0$, which is certainly unconstrained. Lines 3–5 ensure that $\boldsymbol{a}'_{j*}$ is the unique line with a non-zero i-th element. They do not change the parallelotope, since they operate on unconstrained rows. Then, we may replace $\boldsymbol{a}'_{j*}$ with $\boldsymbol{e}^i - \boldsymbol{c}$, and both l_j, u_j with b. Since the j-th row of A' is unconstrained, we do not lose precision when we replace it. Since the steps 3–5 ensure that all the other rows in A' have a zero in the i-th column, then $\boldsymbol{e}^i - c$ is linearly independent from them and the final matrix A' is invertible.

Algorithm 2 The non-invertible $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)$ abstract operator

Require: $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \in \mathsf{Par}_n$, $i \in \{1, \ldots, n\}$, $\boldsymbol{c} \in \mathbb{R}^n$, $b \in \mathbb{R}$, $c_i = 0$
1: $\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle \leftarrow \mathsf{forget}^\alpha(i)(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle)$
2: choose an index j with $a'_{ji} \neq 0$
3: **for all** s such that $a'_{si} \neq 0$ and $s \neq j$ **do**
4: $\quad \boldsymbol{a}'_{s*} \leftarrow \boldsymbol{a}'_{s*} - a'_{si}/a'_{ji}\boldsymbol{a}'_{j*}$
5: **end for**
6: $\boldsymbol{a}'_{j*} \leftarrow \boldsymbol{e}^i - \boldsymbol{c}$
7: $\boldsymbol{l}'_j \leftarrow b$
8: $\boldsymbol{u}'_j \leftarrow b$
9: **return** $\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$

Theorem 13. *The operator* $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)$ *described in Algorithm 2 is correct. The computational complexity is* $O(n^2)$.

Note that non-invertible assignment is γ-complete exactly in the same hypothesis under which non-deterministic assignment is γ-complete.

Theorem 14. *Consider* $\boldsymbol{c} \in \mathbb{R}^n$, $b \in \mathbb{R}$ *and* $c_i = 0$. *If* $\mathcal{P}$ *is a parallelotope in* $\mathbb{R}^n$ *and* $\boldsymbol{e}^i$ *is a line in* $\mathcal{P}$ *or* $J_0 \neq \emptyset$ *in Algorithm 1 then* $\gamma(\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)(\mathcal{P})) = \mathsf{assign}(i, \boldsymbol{c}, b)(\gamma(\mathcal{P}))$.

Theorem 15. *The operator* $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)$ *with* $c_i = 0$ *is minimal.*

Even for non-invertible assignments, heuristics can be used to improve the result of the analysis. For example, it is possible that, in order to gain more benefits from the result of $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)$, it is better to tweak $\mathsf{forget}^\alpha(i)$ in order to get the best possible bounds for the row $\boldsymbol{c}$. However, we have not investigated this idea in details.

4.4. Refinement by linear inequality

Given $\boldsymbol{c} \in \mathbb{R}^n$ and $b \in \mathbb{R}$, consider the operation over $\mathscr{P}(\mathbb{R}^n)$ defined by

$$\mathsf{refine}(\boldsymbol{c}, b)(X) = \{\boldsymbol{x} \mid \boldsymbol{x} \in X \wedge \boldsymbol{c} \cdot \boldsymbol{x} \leq b\} \ ,$$

which we call *linear refinement*, corresponding to the conditional statement "if $\boldsymbol{c} \cdot \boldsymbol{x} \leq b$ then". In general, the linear refinement of a parallelotope is not a parallelotope.

Consider Algorithm 3. Given a parallelotope $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$, we first check if there exists an unconstrained row of A such that, if we replace that row with $\boldsymbol{c}$, the matrix is still invertible. This amounts to computing a vector $\boldsymbol{y} \in \mathbb{R}^n$ such that $A^T\boldsymbol{y} = \boldsymbol{c}$ and look for an index $j \in U(\mathcal{P})$ such that $y_j \neq 0$. When it does not exist, we simply compute the least parallelotope $\mathcal{P}'$ containing $\mathsf{refine}(\boldsymbol{c}, b)(\mathcal{P})$ and definable over A: we recall from [10] that $\mathcal{P}' = \langle A, \boldsymbol{l}', \boldsymbol{u}'\rangle$, where $\langle I_n, \boldsymbol{l}', \boldsymbol{u}'\rangle = \mathsf{refine}(\boldsymbol{y}, b)\langle I_n, \boldsymbol{l}, \boldsymbol{u}\rangle$. Hence, we may use the known refine operator over boxes to define a refine operator over parallelotopes.

Algorithm 3 The $\mathsf{refine}^{\alpha}(\boldsymbol{c}, b)$ abstract operator

Require: $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \in \mathsf{Par}_n$, $\boldsymbol{c} \in \mathbb{R}^n$, $b \in \mathbb{R}$
1: $\boldsymbol{y} \leftarrow$ solution of $A^T\boldsymbol{y} = \boldsymbol{c}$
2: **if** $\exists j.\ y_j \neq 0 \wedge l_j = -\infty \wedge u_j = +\infty$ **then**
3: $\quad \boldsymbol{a}_{j*} \leftarrow \boldsymbol{c}^T$
4: $\quad u_j \leftarrow b$
5: $\quad$ **return** $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$
6: **else**
7: $\quad \langle I_n, \boldsymbol{l}', \boldsymbol{u}'\rangle \leftarrow \mathsf{refine}^{\alpha}(\boldsymbol{y}, b)\langle I_n, \boldsymbol{l}, \boldsymbol{u}\rangle$ {using operator on boxes}
8: $\quad$ **return** $\langle A, \boldsymbol{l}', \boldsymbol{u}'\rangle$
9: **end if**

Theorem 16. *The operator* $\mathsf{refine}^{\alpha}(\boldsymbol{c}, b)$ *described in Algorithm 3 is correct and relatively optimal. If* $\boldsymbol{c} \notin \mathrm{lin}(\gamma(\mathcal{P}))^{\perp}$, *then* $\mathsf{refine}^{\alpha}(\boldsymbol{c}, b)(\mathcal{P})$ *is* γ*-complete. The computational complexity is* $O(n^3)$.

We could change the algorithm above in such a way that, when there is no $j \in U(\mathcal{P})$ with $y_j \neq 0$, we choose any j such that $y_j \neq 0$ and either l_j or u_j is infinite. Then, we proceed with lines 3–5 as if j were in $U(\mathcal{P})$. In this case, we replace a bounded constraint with a different bounded constraint. The result is generally incomparable with the original parallelotope $\mathcal{P}$. However, the rationale here is that the new constraint could be more useful in the remaining of the analysis than the old one which has been removed. We have not investigated in practice the impact of such a different approach to linear refinement.

4.5. Union

Let us come to the abstract union of parallelotopes. The result is obvious if one of the parallelotopes is empty. In the rest of this section and in the following ones we assume to be in the non-empty case. The same If we fix a matrix M, the least parallelotope definable over M which contains the parallelotopes $\mathcal{P}_A = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ and $\mathcal{P}_B = \langle B, \boldsymbol{j}, \boldsymbol{k}\rangle$ can be easily obtained by $\mathsf{rot}_M(\mathcal{P}_A)$ and $\mathsf{rot}_M(\mathcal{P}_B)$ simply selecting, for each row in M, the minimum of the two lower bounds and the maximum of the two upper bounds. By choosing M to be either A or B, we can use this method for a simple and fast implementation of abstract union. The biggest drawback of this choice is that it does not generate new constraints. The ability to generate new constraints precisely allows us to fully exploit the power of a non-template abstract domain, by changing the shape of the generated parallelotopes.

We now propose a more complex variant of abstract union, inspired by the *inversion join* operator [14]. The main idea of the algorithm is to generate a bunch of candidate linear forms. The corresponding constraints are obtained from the candidate linear forms by computing the lower and upper bounds on $\mathcal{P}_A$ and $\mathcal{P}_B$. We then assign to each constraint a priority. In general, the candidate linear forms are not linearly independent. At the end, we select exactly n linearly independent constraints, according to their priorities.

Assigning the priorities to the candidate constraints is the key of the abstract union algorithm. As a first consideration, assigning the priorities must take into account whether the domain is used in combination with another abstract domain, thus allowing better optimizations. In particular, in the case of a reduced product of parallelotopes and intervals, it is immediate to see that constraints with a single non-null coefficient (interval constraints) are not useful, and thus we should avoid them. On the contrary, when parallelotopes are used alone, it is very convenient to choose such constraints. Thus, we have developed three slightly different algorithms for assigning priorities, according to whether we prefer to retain interval constraints or not. Choosing the algorithm which best fits depends on the analysis context.

We use a flag *favorAxes* to differentiates the behavior of the three algorithms: when the flag is set to 1, we assign to the interval constraints the highest priority, so that they are always preferred in the abstract union operation. When the flag is set to -1, we assign to the interval constraints the lowest priority, so that the heuristic algorithm tries to avoid them, as much as reasonable. When the flag is set to 0, we assign priorities to intervals using the same algorithm as for non-interval constraints.

The priority of non-interval constraints is chosen according to the values of the bounds: equality constraints come first, followed by constraints which are saturated both in $\mathcal{P}_A$ and $\mathcal{P}_B$. This order is mostly dictated by heuristic considerations. Algorithm 4 computes the bounds and assigns the priorities for a given linear form $\boldsymbol{v}$. The lines "{other tests}" in Algorithm 4 refers to a set of tests where we consider all the possible combinations of finite and infinite lower and upper bounds, in order to better assign priorities.

We now describe how to generate the candidate linear forms for the abstract union. Obvious candidates are the rows of the matrices A and B. Moreover, we also generate new linear forms using (a part of) the inversion join algorithm. Given two constraints in $\mathcal{P}_A$ and/or $\mathcal{P}_B$, the inversion join computes a new linear form obtained as a linear combination of the two constraints, under the condition that they form an inversion.

Definition 7. The constraints $l_h \leq \boldsymbol{a}_{h*} \cdot \boldsymbol{x}$ taken from the parallelotope $\langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ and $j_i \leq \boldsymbol{b}_{i*} \cdot \boldsymbol{x}$ taken from $\langle B, \boldsymbol{j}, \boldsymbol{k} \rangle$ form an *inversion* when, provided $\langle A, \boldsymbol{j}', \boldsymbol{k}' \rangle = \mathsf{rot}_A(\mathcal{P}_B)$, the following conditions hold:

1. all the bounds l_h, j'_h, l_i, j'_i are finite;
2. $\boldsymbol{a}_{h*}$ and $\boldsymbol{b}_{i*}$ are linearly independent;
3. $l_h < j'_h$ and $l_i > j'_i$ (or vice-versa, that is $l_h > j'_h$ and $l_i < j'_i$).

Whenever we find an inversion, we may generate the new constraint $\lambda_{hi} j'_h \leq (\boldsymbol{a}_{h*} + \lambda_{hi} \boldsymbol{b}_{i*}) \cdot \boldsymbol{x}$, where $\lambda_{hi} = \frac{l_h - j'_h}{j'_i - l_i}$. The same procedure can also be applied to upper bounds, just considering $\boldsymbol{a}_{h*} \cdot \boldsymbol{x} \leq u_h$ as equivalent to $-u_h \leq -a_{h*} \cdot \boldsymbol{x}$. The complete procedure is illustrated in Algorithm 5.

Algorithm 4 Bounds and priorities for the linear form $\boldsymbol{v}$ (Sketch)

Require: $\mathcal{P}_A, \mathcal{P}_B \in \mathsf{Par}_n$, $\boldsymbol{v} \in \mathbb{R}^n$, $favorAxes \in \{-1, 0, 1\}$

$l_v \leftarrow \inf\{\boldsymbol{v} \cdot \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}_A\}$
$u_v \leftarrow \sup\{\boldsymbol{v} \cdot \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}_A\}$
$j_v \leftarrow \inf\{\boldsymbol{v} \cdot \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}_B\}$
$k_v \leftarrow \sup\{\boldsymbol{v} \cdot \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}_B\}$
if $\boldsymbol{v}$ is an interval constraint and $favorAxes \neq 0$ **then**
  if $favorAxes = 1$ **then**
    $p \leftarrow -\infty$ // this is the highest priority
  else if $favorAxes = -1$ **then**
    $p \leftarrow \infty$ // this is the lowest priority
  end if
else
  if $l_v = u_v = j_v = k_v$ **then**
    $p \leftarrow 0$
  else if $l_v = j_v \in \mathbb{R}$ and $u_v = k_v \in \mathbb{R}$ **then**
    $p \leftarrow 1$
  else if ... **then**
    $\vdots$ {other tests}
  else
    $p \leftarrow +\infty$
  end if
end if
return $\langle \min(l_v, j_v), \max(u_v, k_v), p \rangle$

Example 6. Given the parallelotopes

$$\mathcal{P}_1 = \{\boldsymbol{x} \in \mathbb{R}^n \mid 1 \leq x_1 \leq 2, 2 \leq x_2 \leq 4\}$$
$$\mathcal{P}_2 = \{\boldsymbol{x} \in \mathbb{R}^n \mid 2 \leq x_1 \leq 3, 1 \leq x_2 \leq 3\}$$

depicted in Figure 4 in grey and light-grey respectively, $\mathcal{P}_1 \cup^{\alpha} \mathcal{P}_2$ is the dotted parallelotope.

Theorem 17. *The abstract union operator described in Algorithm 5 is correct and relatively optimal. The computational complexity is* $O(n^4)$.

4.6. Widening

We use the standard definition of widening which appears in [15] since, although developed for monotonic abstract operators, it also works in the non-monotonic case [16].

A first widening for parallelotopes, denoted by $\triangledown$, is essentially inspired by the standard widening on intervals. In order to compute $\mathcal{P}_A \triangledown \mathcal{P}_B$, we first compute $\mathcal{P}' = \mathsf{rot}_A(\mathcal{P}_B)$. Each bound of $\mathcal{P}_A$ which has been enlarged in $\mathcal{P}'$ is changed into $+\infty$ or $-\infty$.

Algorithm 5 The abstract union operator

Require: $\mathcal{P}_A = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$, $\mathcal{P}_B = \langle B, \boldsymbol{j}, \boldsymbol{k} \rangle \in \mathsf{Par}_n$

1: $Q \leftarrow \emptyset$ {a priority queue}
2: **for all** $i \in \{1, \ldots, n\}$ **do**
3: $\langle c, d, p \rangle \leftarrow$ result of Algorithm 4 applied to $\boldsymbol{a}_{i*}$
4: add $\langle \boldsymbol{a}_{i*}, c, d \rangle$ to Q with priority p
5: **end for**
6: same procedure of lines 2–5 applied to rows in B
7: **for all** $\boldsymbol{v}_1, \boldsymbol{v}_2$ rows of A and B **do**
8: {here we check if $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ form an inversion}
9: $h_1 \leftarrow \inf\{\boldsymbol{v}_1 \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}_A\}$
10: $h_2 \leftarrow \inf\{\boldsymbol{v}_2 \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}_A\}$
11: $i_1 \leftarrow \inf\{\boldsymbol{v}_1 \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}_B\}$
12: $i_2 \leftarrow \inf\{\boldsymbol{v}_2 \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}_B\}$
13: **if** $h_1, i_1, h_2, i_2 \in \mathbb{R}$ and $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ are linearly independent and $((h_1 < i_1 \wedge h_2 > i_2) \vee (h_1 > i_1 \wedge h_2 < i_2))$ **then**
14: {we know that $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ form an inversion}
15: $\boldsymbol{w} \leftarrow \boldsymbol{v}_1 + \frac{h_1 - i_1}{i_2 - h_2} \boldsymbol{v}_2$ {$\boldsymbol{w}$ is the linear form obtained by inversion join}
16: $\langle c, d, p \rangle \leftarrow$ result of Algorithm 4 applied to $\boldsymbol{w}$
17: add $\langle \boldsymbol{w}, c, d \rangle$ to Q with priority p
18: **end if**
19: **end for**
20: same procedure of lines 7–18 applied to upper bounds
21: same proc. of lines 7–18 applied to lower bounds for $\boldsymbol{v}_1$ and upper bounds for $\boldsymbol{v}_2$
22: same proc. of lines 7–18 applied to upper bounds for $\boldsymbol{v}_1$ and lower bounds for $\boldsymbol{v}_2$
23: $\langle R, \boldsymbol{l}', \boldsymbol{u}' \rangle \leftarrow$ empty set of constraints
24: **while** $|R| < n$ **do**
25: extract $\langle \boldsymbol{w}, c, d \rangle$ from Q with maximal priority
26: **if** $\boldsymbol{w}$ is linearly independent from R **then**
27: add $\langle \boldsymbol{w}, c, d \rangle$ to $\langle R, \boldsymbol{l}', \boldsymbol{u}' \rangle$
28: **end if**
29: **end while**
30: **return** $\langle R, \boldsymbol{l}', \boldsymbol{u}' \rangle$

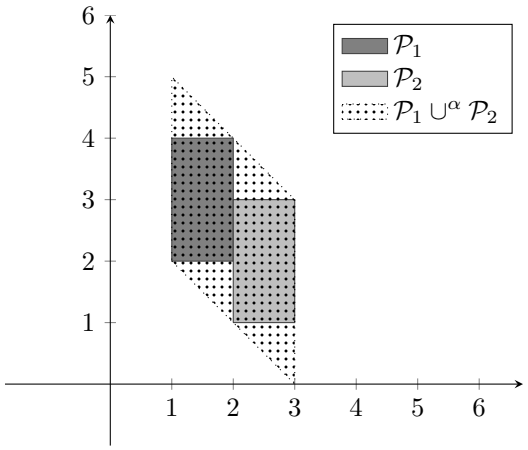


Figure 4: Union of parallelotopes $\mathcal{P}_1$ and $\mathcal{P}_2$.

Definition 8. Given two parallelotopes $\mathcal{P}_A = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ and $\mathcal{P}_B = \langle B, \boldsymbol{j}, \boldsymbol{k} \rangle$, let $\langle A, \boldsymbol{j}', \boldsymbol{k}' \rangle = \mathsf{rot}_A(\mathcal{P}_B)$. We define $\mathcal{P}_A \triangledown \mathcal{P}_B = \langle A, \boldsymbol{l}', \boldsymbol{u}' \rangle$ where for each $i \in \{1, \ldots, n\}$, we have that:

$$l'_i = \begin{cases} -\infty & \text{if } j'_i < l_i \\ l_i & \text{otherwise} \end{cases} \qquad u'_i = \begin{cases} \infty & \text{if } k'_i > u_i \\ u_i & \text{otherwise.} \end{cases}$$

The problem with this widening is that it cannot change the constraint matrix. For this reason, it is better to use it with delayed widening. Even a small delay allows the constraint matrix in loops to stabilize to a good value. Then, when widening kicks in, constraints cannot change anymore.

We introduce an alternative widening operator which tries to change the constraint matrix while avoiding infinite ascending chains and that will be used in the rest of the paper.

Definition 9. Given two parallelotopes $\mathcal{P}_A = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ and $\mathcal{P}_B = \langle B, \boldsymbol{j}, \boldsymbol{k} \rangle$, we define

$$\mathcal{P}_A \mathbin{\overline{\triangledown}} \mathcal{P}_B = \begin{cases} \mathsf{rot}_B(\mathcal{P}_A) \triangledown \mathcal{P}_B & \text{if } \mathsf{rot}_B(\mathcal{P}_A) < \mathcal{P}_B \\ \mathcal{P}_A \triangledown \mathcal{P}_B & \text{otherwise} \end{cases}$$

The above definition works because applying the rot operator does not decrease the number of infinite bounds in the parallelotope.

Theorem 18. *The operator $\overline{\triangledown}$ is a widening.*

4.7. Narrowing

First of all, we give a definition of narrowing which, as we will discuss later, slightly differs from the standard one.

Definition 10 (Narrowing). Given a lattice $(C, \subseteq)$ (the concrete domain) and a partial order $(A, \leq)$ (the abstract domain) with monotone concretization map $\gamma : A \to C$, a *narrowing* is a binary operator $\triangle : A \times A \to A$ such that:

- $a_1 \triangle a_2 \leq a_1$;
- $\gamma(a_1) \cap \gamma(a_2) \subseteq \gamma(a_1 \triangle a_2)$;
- for each sequence $a_0, \ldots, a_i, \ldots$ of elements of A, the sequence defined as $b_0 = a_0$ and $b_i = b_{i-1} \triangle a_i$ for $i > 0$ is definitively stationary.

This generalizes the standard narrowing in [15] to the case when the second argument is not bigger than the first one. The generalization is needed since abstract operators on parallelotopes are not monotone, hence the arguments to narrowing may not be in the correct relation as expected in [15]. Another generalization of narrowing, designed to work with non-monotonic abstract operators, recently appeared in [16]. However, we find the latter not completely satisfactory, since correctness of the descending iterations does not depend on the definition of narrowing alone, but on the interaction between narrowing and the abstract operators.

The first condition in Definition 10 ensures that an iteration sequence built with narrowing is decreasing, the second condition means that $\triangle$ is a correct approximation of the concrete meet, while the last condition enforces termination. When $a_2 \leq a_1$, then $\gamma(a_2) \cap \gamma(a_1) = \gamma(a_2)$, and the condition $\gamma(a_1) \cap \gamma(a_2) \subseteq \gamma(a_1 \triangle a_2)$ is enforced by the stronger condition $a_2 \leq a_1 \triangle a_2$ of the standard narrowing.

A narrowing satisfying the definition above may be used to improve the result of the ascending phase of the analysis:

Theorem 19. *In the hypothesis of Definition 10, let $\bar{F}$ be a (possibly non-monotone) abstract operator $\bar{F} : A \to A$ which is a correct abstraction of the monotone operator $F : C \to C$. Given $a \in A$, let $c \in C$ be a fixpoint of F such that $c \subseteq \gamma(a)$. We define the iteration sequence $y_0 = a$, $y_{i+1} = y_i \triangle \bar{F}(y_i)$. Then*

- *the iteration sequence $\{y_i\}_{i \in \mathbb{N}}$ is decreasing and definitively stationary;*
- *for each $i \in \mathbb{N}$, y_i is a correct approximation of c.*

Our definition of narrowing ensures that the descending phase terminates on a correct abstraction, which might not be a post-fixpoint of the set of semantic equations.

For parallelotopes $\mathcal{P}_A$ and $\mathcal{P}_B$, the narrowing is defined by first computing $\mathcal{P}' = \mathsf{rot}_A(\mathcal{P}_B)$ and then refining with $\mathcal{P}'$ the unbounded constraints in $\mathcal{P}_A$.

Definition 11. Given two parallelotopes $\mathcal{P}_A = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ and $\mathcal{P}_B = \langle B, \boldsymbol{j}, \boldsymbol{k} \rangle$, let $\langle A, \boldsymbol{j}', \boldsymbol{k}' \rangle = \mathsf{rot}_A(\mathcal{P}_B)$. We define $\mathcal{P}_A \triangle \mathcal{P}_B = \langle A, \boldsymbol{l}', \boldsymbol{u}' \rangle$ where for each $i \in \{1, \ldots, n\}$, we have that:

$$l'_i = \begin{cases} j'_i & \text{if } l_i = -\infty \\ l_i & \text{otherwise} \end{cases} \qquad u'_i = \begin{cases} k'_i & \text{if } u_i = +\infty \\ u_i & \text{otherwise} \end{cases}$$

Theorem 20. *The operator $\triangle$ for parallelotopes is a narrowing according to Definition 10.*

5. Parametric parallelotopes

In Sections 4.5 we have designed a parametric algorithm for the abstract union depending on the flag *favorAxes*. Changing the flag we modify the priorities assigned to interval constraints, thus obtaining different parallelotopes during the analysis. We now show that the resulting three abstract union operations produce quite different results.

We have implemented a prototype of the abstract domain of parallelotopes in the static analyzer Jandom [17] and did some preliminary test on the ALICe[2] benchmarks [18] (plus some additional test programs which can be found in [17]). The test-suite comprises a total of 105 models with 326 program points. The models have at most 11 different program points and 10 variables.

The domain of parallelotopes Par has been implemented with the standard union, namely $favorAxes = 0$. The domain $\mathsf{Par}_{+\mathsf{axes}}$ is tuned with $favorAxes = 1$, thus assigning to the interval constraints the highest priority, so that they are always preferred. On the contrary, in the domain $\mathsf{Par}_{-\mathsf{axes}}$ we use $favorAxes = -1$, so producing parallelotopes which hardly use interval constraints.

Figure 5 shows the results of the experimental comparison between the standard parallelotope Par, $\mathsf{Par}_{+\mathsf{axes}}$ and $\mathsf{Par}_{-\mathsf{axes}}$. The experimental results show that in about one third of the experiments we get incomparable results. This is due to the fact that we deal with a non-template numerical domain, and thus the shape of the result may considerably vary when changing the abstract operations. The experiments show that the domain with the best performance in precision is clearly $\mathsf{Par}_{+\mathsf{axes}}$, which improves over the results of both Par and $\mathsf{Par}_{-\mathsf{axes}}$. This result was largely expected, since variable initialization is an important source of information which should be carried on during the analysis. Thus, when the abstract domain of parallelotopes is used alone, the preferred version should certainly be $\mathsf{Par}_{+\mathsf{axes}}$.

On the other hand, we show in the next section that things change when considering a reduced product of $\mathsf{Par}_{-\mathsf{axes}}$ with intervals which dramatically improves the analysis precision.

6. Reduced product

Although the abstract domain of parallelotopes is very expressive and can potentially represent any linear invariant, it is limited by the reduced number of constraints allowed in the constraint matrix. In order to overcome this limit, we strongly believe that parallelotopes should be used in combination with another simple, even non-relational, numerical abstract domain. In particular, the combination of a non-template, fully relational domain with a rich expressiveness,

[2] http://alice.cri.ensmp.fr/

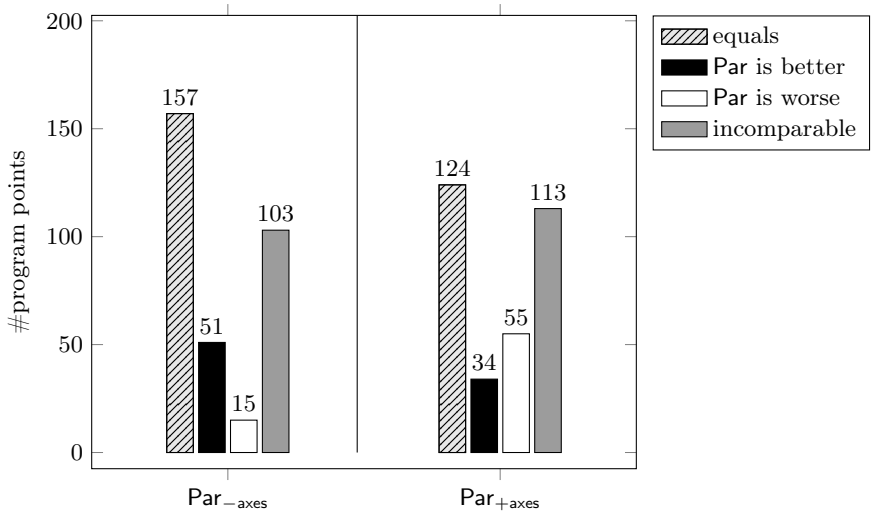


Figure 5: Comparing parallelotopes with different flags *favorAxes*. On the left we compare Par with $\mathsf{Par}_{-\mathsf{axes}}$, on the right Par with $\mathsf{Par}_{+\mathsf{axes}}$.

such as parallelotopes, and a template, weakly relational domain, able to carry on the basic information at least on interval constraints, allows parallelotopes to exploit its full potential.

In this section we define a simple reduced product between parallelotopes and intervals. We denote by Box the domain of intervals with the standard operations [2]. Since intervals are parallelotopes (whose constraint matrix is the identity matrix), we abuse notation and use the same denotations for the concretization map and the abstract operations.

6.1. Representation of reduced product

The (reduced) product of the domains Par and Box is defined as

$$\mathsf{Par} \sqcap \mathsf{Box} = \{\langle P, B\rangle \mid P \in \mathsf{Par}, B \in \mathsf{Box}\}$$

and the concretization map is simply given by

$$\gamma(\langle P, B\rangle) = \gamma(P) \cap \gamma(B) \ .$$

6.2. Abstract operators on the reduced product

The abstract operations on $\mathsf{Par} \sqcap \mathsf{Box}$ are obtained by combining the operations of the original domains, with the help of a *reduction* operator responsible for transferring information between the two domains.

We first need to introduce a new abstract operator of *weak intersection* on parallelotopes which is defined asymmetrically by first performing a rotation of the second argument on the constraint matrix of the first one.

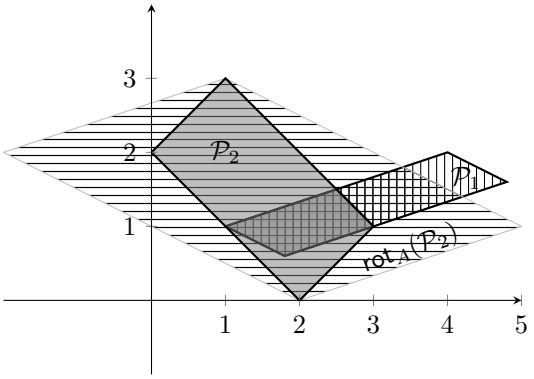


Figure 6: Weak intersection of parallelotopes P_1 and P_2.

Definition 12 (Weak intersection). Given two parallelotopes $\mathcal{P}_1 = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$, $\mathcal{P}_2 \in \mathsf{Par}_n$, let $\langle A, \boldsymbol{l}', \boldsymbol{u}' \rangle = \mathsf{rot}_A(\mathcal{P}_2)$. We define the weak intersection of $\mathcal{P}_1$ and $\mathcal{P}_2$ as:

$$\mathcal{P}_1 \cap^\alpha \mathcal{P}_2 = \langle A, \boldsymbol{l}'', \boldsymbol{u}'' \rangle$$

where $l''_i = \max(l_i, l'_i)$ and $u''_i = \min(u_i, u'_i)$.

The idea of the weak intersection is to preserve the constraint matrix of the first parallelotope. Algorithm 6 shows the weak intersection operator.

Example 7. Given the parallelotopes

$$\mathcal{P}_1 = \{\boldsymbol{x} \in \mathbb{R}^n \mid 0 \leq -x_1 + 3x_2 \leq 2, 3 \leq x_1 + 2x_2 \leq 8\}$$

$$\mathcal{P}_2 = \{\boldsymbol{x} \in \mathbb{R}^n \mid 2 \leq x_1 + x_2 \leq 4, -2 \leq -x_1 + x_2 \leq 2\}$$

depicted in Figure 6, $\mathcal{P}_1 \cap^\alpha \mathcal{P}_2$ is the parallelotope filled with both vertical and horizontal lines.

Proposition 21. *The operator $\cap^\alpha$ is a correct approximation of the concrete intersection. It is γ-complete when the two arguments are defined over the same constraint matrix.*

We can now define the reduction operator in a standard way, by using the weak intersection operator.

Definition 13 (Reduction). Given a parallelotope $\mathcal{P} \in \mathsf{Par}_n$ and a box $B \in \mathsf{Box}$, we define the reduction operator $\mathsf{red} : \mathsf{Par} \sqcap \mathsf{Box} \to \mathsf{Par} \sqcap \mathsf{Box}$ as:

$$\mathsf{red}(\langle P, B \rangle) = \langle P \cap^\alpha B, B \cap^\alpha P \rangle$$

Proposition 22. *The reduction operator is correct, i.e., given any $P \in \mathsf{Par}$ and $B \in \mathsf{Box}$, we have that $\gamma(\mathsf{red}(\langle P, B \rangle)) = \gamma(\langle P, B \rangle)$.*

Algorithm 6 The weak intersection operator

Require: $P_A = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle, P_B = \langle B, \boldsymbol{j}, \boldsymbol{k}\rangle \in \mathsf{Par}_n$
1: $\langle A, \boldsymbol{l'}, \boldsymbol{u'}\rangle \leftarrow \mathsf{rot}_A(P_B)$
2: **for all** $i \in \{1, \ldots, n\}$ **do**
3: $\quad l'_i = \max(l'_i, l_i)$
4: $\quad h'_i = \min(h'_i, h_i)$
5: $\quad$ **if** $(l'_i > h'_i)$ **then**
6: $\quad\quad$ **return** ϵ
7: $\quad$ **end if**
8: **end for**
9: **return** $\langle A, \boldsymbol{l'}, \boldsymbol{u'}\rangle$

Since weak intersection is not relatively optimal, it happens that multiple iterations of the reduction operator may improve the precision of the result [19], at the cost of an increase in the computational cost of the operation.

All the operators $\odot^{P,B}$ on $\mathsf{Par} \sqcap \mathsf{Box}$ except widening and narrowing are simply defined componentwise, by first performing the operation on the original domain and then applying the reduction operator:

$$\langle P_1, B_1\rangle \odot^{P,B} \langle P_2, B_2\rangle = \mathsf{red}(\langle P_1 \odot^P P_2, B_1 \odot^B B_2\rangle) \tag{2}$$

Corollary 23. *All the operations on* $\mathsf{Par} \sqcap \mathsf{Box}$ *defined according to Equation 2 are correct, namely:*

$$\gamma(\langle P_1, B_1\rangle \odot^{P,B} \langle P_2, B_2\rangle) \supseteq \gamma(\langle P_1, B_1\rangle) \odot \gamma(\langle P_2, B_2\rangle)$$

The widening and narrowing operators cannot be defined using the reduction, since it may prevent termination. Thus we use a standard definition for reduced product omitting the reduction step.

Definition 14 (Widening and narrowing). Given $P_1, P_2 \in \mathsf{Par}$ and $B_1, B_2 \in \mathsf{Box}$, the abstract widening and narrowing on $\mathsf{Par} \sqcap \mathsf{Box}$ are defined as

$$\langle P_1, B_1\rangle \triangledown^{P,B} \langle P_2, B_2\rangle = \langle (P_1 \mathbin{\overline{\triangledown}}^P P_2), (B_1 \triangledown^B B_2)\rangle$$

$$\langle P_1, B_1\rangle \triangle^{P,B} \langle P_2, B_2\rangle = \langle (P_1 \triangle^P P_2), (B_1 \triangle^B B_2)\rangle$$

7. Evaluating the reduced product

In the previous section, we have defined the reduced product $\mathsf{Par} \sqcap \mathsf{Box}$. According to Section 5, we can easily design a parametric reduced product along the lines of $\mathsf{Par} \sqcap \mathsf{Box}$, by simply tuning the flag *favorAxes*, so obtaining the reduced products $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ and $\mathsf{Par}_{+\mathsf{axes}} \sqcap \mathsf{Box}$. We have implemented a prototype of the reduced products in the static analyzer Jandom and performed the test on the ALICe benchmarks. We have performed two different kinds of

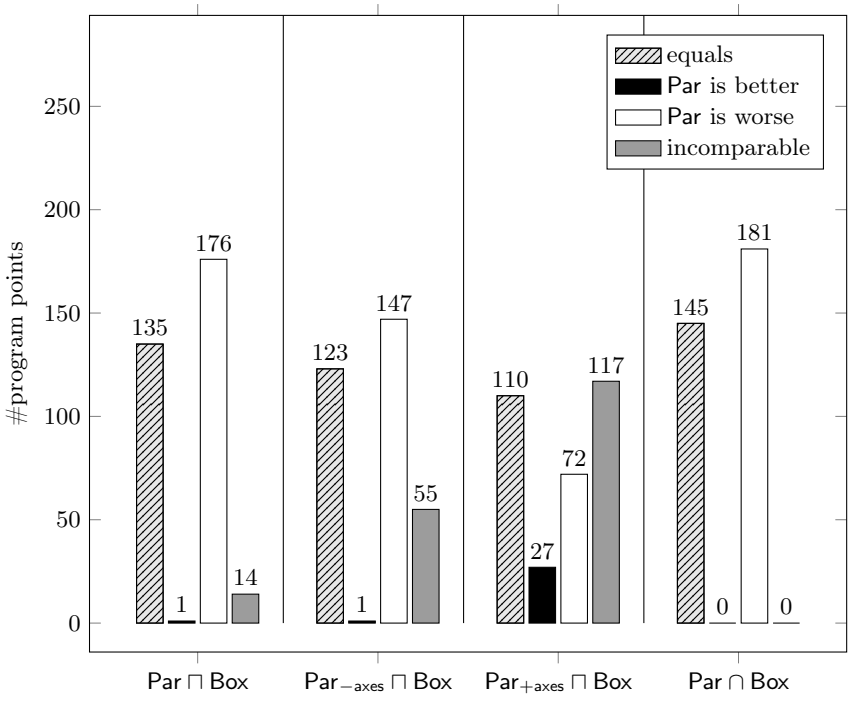


Figure 7: Comparing Par with different versions of reduced product

evaluations. Firstly, we have compared the parametric versions of the reduced product with the domain Par, in order to validate our intuition that the domain $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ is generally more precise than the other versions. Secondly, we have tested the domain $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ versus the domains Par, $\mathsf{Par}_{+\mathsf{axes}}$ and $\mathsf{Par}_{-\mathsf{axes}}$, in order to show the increment in precision that we gain when using the reduced product.

7.1. Parametric reduced products

We have compared the domain of parallelotope Par to the reduce products $\mathsf{Par} \sqcap \mathsf{Box}$, $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ and $\mathsf{Par}_{+\mathsf{axes}} \sqcap \mathsf{Box}$. In addition, for completeness, we have also implemented a naive analysis, denoted by $\mathsf{Par} \cap \mathsf{Box}$, obtained by separately performing the analysis with parallelotopes and intervals, and then taking the intersection of the results only once at the end of the two analyses.

The results of the experiments are given in Figure 7. From the experiments it emerges that the domain $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ is more precise and more expressive, since it is able to compute more and different invariants with respect to the other domains, according to the two last columns of the bar chart which show the program points with a better behavior (147) and those where we can produce (at least a) new invariants (55). We strongly believe that the ability to found new invariants is specially important.

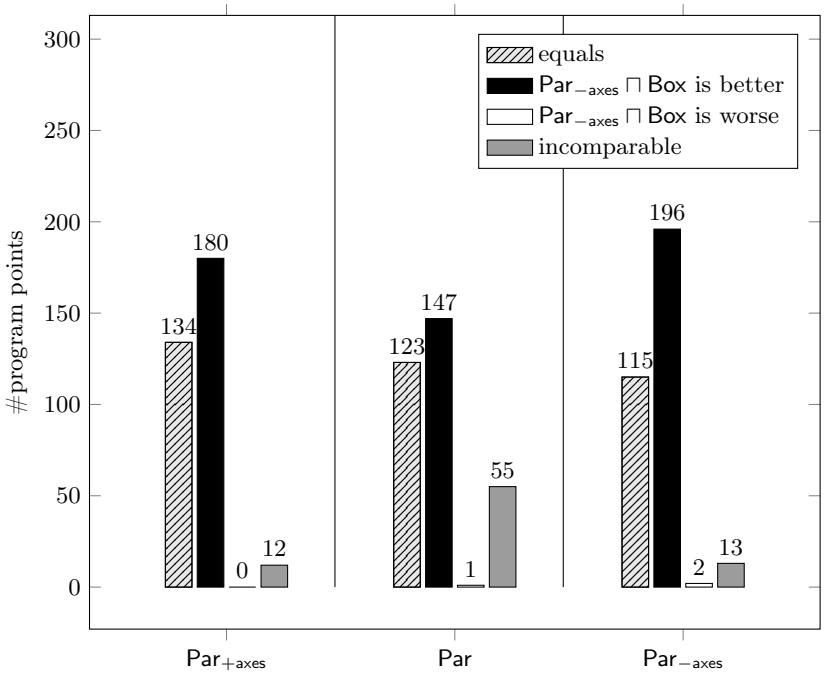


Figure 8: Comparing the reduced product $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ with $\mathsf{Par}_{+\mathsf{axes}}$, Par and $\mathsf{Par}_{-\mathsf{axes}}$.

7.2. Reduced product vs parallelotopes

We have compared the best domain resulting from the previous section, $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$, with the different versions of parallelotopes Par, $\mathsf{Par}_{+\mathsf{axes}}$ and $\mathsf{Par}_{-\mathsf{axes}}$, in order to show the increment in precision obtained by considering the combination with intervals. In all the tests, we have used a delayed widening and narrowing, fixing the delay to 5, i.e. we avoid applying the widening and narrowing operators during the first 5 iterations of any loop.

The comparison in Figure 8 shows that the reduced product highly improves over the results of the analysis when using the parallelotopes alone.

8. Experimental comparison with other domains

We have compared the domain $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ with the polyhedra [1], octagons [3] and intervals [2] abstract domains. The tests have been conducted in the Jandom static analyzer. The comparison is mostly performed in precision, since parallelotopes and intervals are implemented natively in Jandom, while for polyhedra and octagons we use the PPL (Parma Polyhedra Library) [20] and comparing the execution times would be unfair.

We have conducted two different evaluations. First, we have compared the domain $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ with the polyhedra, octagons and intervals domains. For each model in the benchmarks and for each program point, we have evaluated

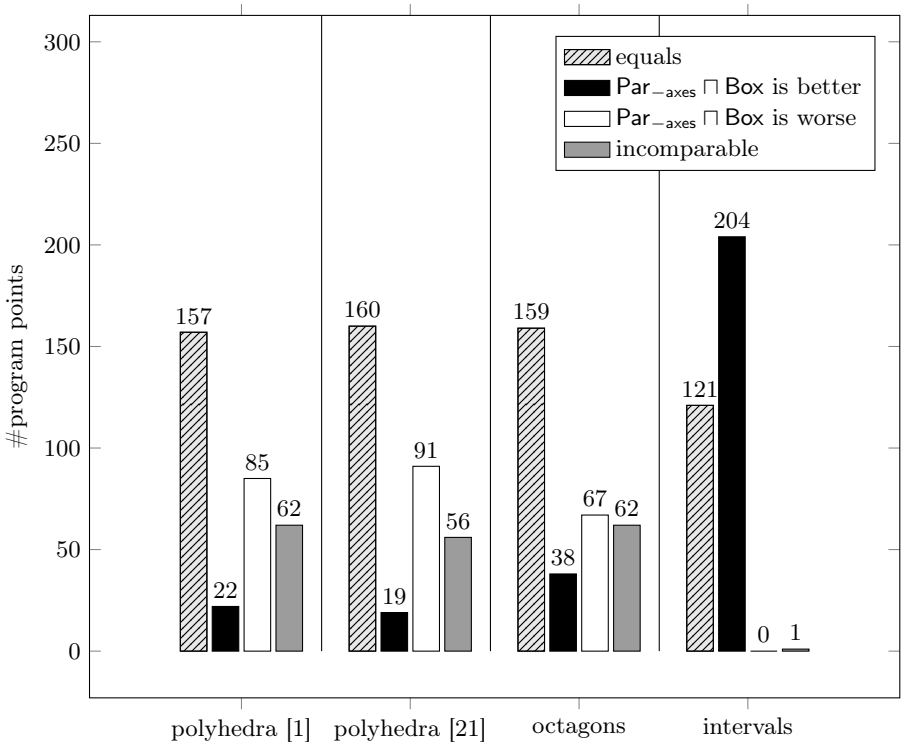


Figure 9: Comparing $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ with polyhedra, octagons and intervals. The first polyhedra domain is equipped with the standard polyhedra widening [1], the second one with the widening in [21].

both the shapes and the bounds of the invariants found by the analyses. In the experiments we use two different polyhedra domains, the first equipped with the standard polyhedra widening [1], the second one with the widening in [21].

The results in Figure 9 show that in about one half of the program points we obtain the same results as polyhedra and octagons. In the comparison with octagons, we obtain a better result in 38 program points (out of 326), we get worse results in 67 program points, and incomparable results in 62 program points. Thus, in about one third of the cases (100 program points), we get a better or different result than octagons. A similar conclusion holds for the comparison with polyhedra.

Even if octagons seem to be better than our domain in many program points, this is mainly due to the fact that octagons can represent a greater number of constraints. In fact, it is worth noting that this comparison measures not only the ability of the domain to discover better bounds or new invariants, but also the total number of constraints found by each domain. Of course, this is unfair with respect to the domain $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$, which can represent at most $2n$ constraints, while octagons can represent n^2 constraints, and the polyhedra

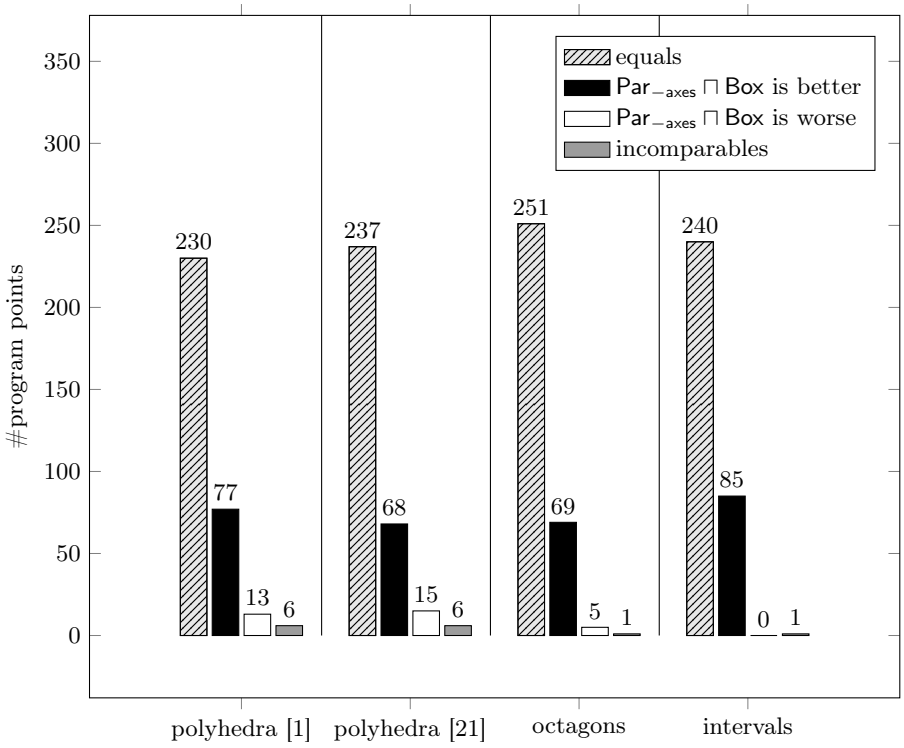


Figure 10: Comparing $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ with polyhedra, octagons and intervals on the interval constraints.

potentially any number. Things change if we are interested in (a limited number of) specific constraints, for instance interval constraints, which can be expressed in all the domains. A precise interval analysis is very helpful in many contexts, therefore it is interesting to compare the relative precision of different domains with respect to interval properties. Thus, we have compared the analysis results projecting them on the domain of intervals. The results are in Figure 10.

In this case, the domain $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ dramatically outperforms the other ones. This evaluation shows that, if we are interested in interval bounds only, then the domain $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ is strictly more precise than octagons in 69 program points (21.1%), octagons is better in 5 cases (1.5%), the results are incomparable in only 1 program point (0.3%), and we get the same result on the other program points.

Surprisingly, we get a similar result also when comparing $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ with polyhedra. In fact, our domain is strictly more precise than polyhedra [1] in 77 program points (23.6%), polyhedra is better in 13 cases (3.9%) and the results are incomparable in 6 program points (1.8%). Similar results hold when we use the widening in [21]. We believe that this improvement is mainly due to the widening on polyhedra, which is responsible for losing precision with respect to

Delay	3	4	5	6	7
equals	245	234	230	231	225
$\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ is better	58	73	77	79	85
polyhedra [1] is better	16	12	13	12	13
incomparable	7	7	6	4	3

Table 1: Comparing $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ with polyhedra [1] using various widening and narrowing delays.

Domain	Time
$\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$	3.7
polyhedra [21]	2.6
polyhedra [1]	2.4
octagons	1.8
intervals	0.2

Table 2: Comparing analysis time. Results are in seconds.

more abstract domains.

Our experiments also show that the widening and narrowing delays may greatly change the results of the comparison. In Table 1 we compare $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ with polyhedra [1] on the interval constraints using delays from 3 to 7. For instance, when choosing the delay 5, we obtain the same result for 230 program points, $\mathsf{Par}_{-\mathsf{axes}} \sqcap \mathsf{Box}$ is better in 77 program points, polyhedra is better in 13 program points, and in 6 we get incomparable results.

Finally, we have briefly compared the efficiency of our domain with respect to polyhedra, octagons and intervals. We believe that this is not completely fair because we use the PPL [20] for polyhedra and octagons, which is carefully engineered and optimized, so comparing their efficiency against our implementation is difficult. Moreover, the PPL is written in `C++`, while our analyzer Jandom [17], where we have implemented parallelotopes, box and their reduced product, is written in Scala, so that we also pay the overhead of the Java virtual machine. Table 2 shows some preliminary results. Analysis time is computed on the full set of benchmarks and it is in seconds. Because of the above consideration, and since there is still space for improvements on the abstract operators, we believe that this result is actually very promising.

9. Conclusions

We have proposed a new, non-template abstract domain based on parallelotopes. The domain of parallelotopes can represent any linear constraint, thus retaining the expressivity of the polyhedra abstract domain, with reasonable computational costs. We have provided the full set of abstract operators and have defined a reduced product with intervals. Operations are mostly performed exploiting the inversion of the constraint matrix and, unlike template polyhedra, do not resort to linear programming.

We have experimentally validated the precision of parallelotopes versus the domains of polyhedra and octagons. In many cases we compute more invariants or invariants with tighter bounds. In particular, if we are interested in inferring interval constraints, the reduced product of parallelotopes and intervals greatly improves over polyhedra and octagons, still having a reasonable computational cost similar to octagons.

We believe that the ability to potentially represent any constraint is the key of the parallelotope domain, and makes it a valid alternative to template polyhedra.

We think that the domain of parallelotopes can be further improved by exploring alternative heuristics used in some abstract operators, which need an extensive experimental evaluation. Also the implementation of all the abstract operators could be enhanced, avoiding to recompute the inverse matrix (or solving a full linear system) when only few rows of the constraint matrix vary.

Appendix (Proofs)

Theorem 1. *For every parallelotope P there is a representation $\mathcal{P}$ such that $\gamma(\mathcal{P}) = P$.*

Proof. If P is a parallelotope, there is $A \in \mathrm{GL}(n)$ and $\boldsymbol{l}, \boldsymbol{u} \in \bar{\mathbb{R}}^n$ such that (1) holds. If P is empty, its representation is ϵ and $\gamma(\epsilon) = \emptyset$ by definition. If P is non-empty, then $\boldsymbol{l} \leq \boldsymbol{u}$ and for all $i \in \{1, \ldots, n\}$, $l_i \neq +\infty$ and $u_i \neq -\infty$. Hence $\langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ is a representation of P. □

Theorem 2 (Parallelotopes and Representations)**.** *If $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ is a representation of a parallelotope in $\mathbb{R}^n$, the following holds:*

1. $\dim(\mathcal{P})$ *is equal to* $n - |E(\mathcal{P})|$; [3]
2. $\boldsymbol{v} \neq 0$ *is a ray in* $\mathcal{P}$ *iff for each* $i \in \{1, \ldots, n\}$, $l_i \in \mathbb{R}$ *implies* $\boldsymbol{a}_{i*} \cdot \boldsymbol{v} \geq 0$ *and* $u_i \in \mathbb{R}$ *implies* $\boldsymbol{a}_{i*} \cdot \boldsymbol{v} \leq 0$*;*
3. $\mathcal{P}$ *is bounded iff all the bounds are finite;*
4. $\boldsymbol{v} \neq 0$ *is a line in* $\mathcal{P}$ *iff it is orthogonal to all the constrained rows.*

Proof. We start proving the first point. Assume without loss of generality that $E(\mathcal{P}) = \{1, \ldots, m\}$. Since the rows of A are linearly independent, we know from linear algebra that the solution set of the first m constraints in $\mathcal{P}$ is a flat of dimension $n - m$. Therefore $\dim(\mathcal{P}) \leq n - m$.

Now we prove the opposite inequality. Let us define vectors $\bar{\boldsymbol{l}}, \bar{\boldsymbol{u}} \in \mathbb{R}^n$ such that

$$\bar{l}_i = \begin{cases} l_i & \text{if } l_i \in \mathbb{R} \\ u_i - 1 & \text{if } l_i \notin \mathbb{R} \text{ and } u_i \in \mathbb{R} \\ -1 & \text{otherwise} \end{cases} \qquad \bar{u}_i = \begin{cases} u_i & \text{if } u_i \in \mathbb{R} \\ l_i + 1 & \text{if } u_i \notin \mathbb{R} \text{ and } l_i \in \mathbb{R} \\ +1 & \text{otherwise} \end{cases}$$

[3] As an example of the problems which arise in considering representations with $\boldsymbol{l} \not\leq \boldsymbol{u}$, this property is false in that case, since $\mathcal{P}$ is empty and $\dim(\mathcal{P}) = -1$ regardless of the number of equality rows.

Note that $\boldsymbol{l} \leq \bar{\boldsymbol{l}} \leq \bar{\boldsymbol{u}} \leq \boldsymbol{u}$ and $\bar{l}_i < \bar{u}_i$ for each $i > m$. For each $i > m$ we define $\boldsymbol{u}^i = \bar{\boldsymbol{l}}[i \mapsto \bar{u}_i]$, and note that $\bar{\boldsymbol{l}} \leq \bar{\boldsymbol{u}}^i \leq \boldsymbol{u}$. To ease notation, we also define $\boldsymbol{u}^{n+1} = \bar{\boldsymbol{l}}$. We prove that the set $\{\boldsymbol{u}^i\}_{i=m+1}^{n+1}$ is affinely independent. Assume otherwise, i.e. $\sum_{i=m+1}^{n+1} \lambda_i \boldsymbol{u}^i = \boldsymbol{0}$ with $\sum_{i=m+1}^{n+1} \lambda_i = 0$ and $\lambda_j \neq 0$. We have $\sum_{i=m+1}^{n+1} \lambda_i \boldsymbol{u}^i_j = (\sum_{k \neq j} \lambda_k)\bar{l}_j + \lambda_j \bar{u}_j = -\lambda_j \bar{l}_j + \lambda_j \bar{u}_j = 0$. However, since $\bar{u}_j \neq \bar{l}_j$, this is only possible if $\lambda_j = 0$, which contradicts our hypothesis. Then $\{\boldsymbol{u}^i\}_{i=m+1}^{n+1}$ is affinely independent.

Since A is invertible, it is possible to find points $\{\boldsymbol{v}^i\}_{i=m+1}^{n+1}$ such that $A\boldsymbol{v}^i = \boldsymbol{u}^i$. By definition of $\boldsymbol{u}^i$ we have that $\boldsymbol{v}^i$ is a point in $\mathcal{P}$. Since $\{\boldsymbol{u}^i\}_{i=m+1}^{n+1}$ is a set of affinely independent vectors, the same is true for $\{\boldsymbol{v}^i\}_{i=m+1}^{n+1}$. Therefore $\dim(\mathcal{P}) \geq n - m$ and this proves the result.

In order to prove the second point, assume $\boldsymbol{v} \neq \boldsymbol{0}$ is a ray in $\mathcal{P}$. If $l_i \in \mathbb{R}$ and $\boldsymbol{a}_{i*} \cdot \boldsymbol{v} = z < 0$, consider a generic $\boldsymbol{x} \in P$. Then it is possible to find $k \geq 0$ such that $\boldsymbol{a}_{i*} \cdot (\boldsymbol{x} + k\boldsymbol{v}) = \boldsymbol{a}_{i*} \cdot \boldsymbol{x} + kz < l_i$, which implies $\boldsymbol{x} + k\boldsymbol{v} \notin P$. This is absurd, hence $\boldsymbol{a}_{i*} \cdot v \geq 0$. The case when $u_i \in \mathbb{R}$ is similar. For the converse implication, assume $v \neq \boldsymbol{0}$ such that for each $i \in \{1, \ldots, n\}$, $l_i \in \mathbb{R}$ implies $\boldsymbol{a}_{i*} \cdot \boldsymbol{v} \geq 0$ and $u_i \in \mathbb{R}$ implies $\boldsymbol{a}_{i*} \cdot \boldsymbol{v} \leq 0$. We need to prove that $\boldsymbol{v}$ is a ray. Given $\boldsymbol{x} \in P$ and $\alpha \geq 0$, consider the point $\boldsymbol{w} = \boldsymbol{x} + \alpha \boldsymbol{v}$. Then, $A\boldsymbol{w} = A\boldsymbol{x} + \alpha A\boldsymbol{v}$, where $\boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}$ by definition. Given $i \in \{1, \ldots, n\}$, if $l_i \in \mathbb{R}$ then $\boldsymbol{a}_{i*} \cdot \boldsymbol{v} = z \geq 0$, hence $\boldsymbol{a}_{i*} \cdot \boldsymbol{w} = \boldsymbol{a}_{i*} \cdot \boldsymbol{x} + \alpha z \geq \boldsymbol{a}_{i*} \cdot \boldsymbol{x} \geq l_i$. The same holds for the case $u_i \in \mathbb{R}$, hence $\boldsymbol{w} \in P$.

We come to the third point. It is clear that $\mathcal{P}$ is bounded iff it has no rays. Assume there is a row $\boldsymbol{a}_{i*}$ with $u_i = +\infty$, and we show that $\mathcal{P}$ has a ray. We write $\boldsymbol{a}_{i*}$ as $\boldsymbol{v} + \boldsymbol{w}$ where $\boldsymbol{v}$ is in the space V of the rows of A different from i, and $\boldsymbol{w}$ in $V^\perp$. Note that $\boldsymbol{w} \neq 0$, otherwise A is not invertible, and $\boldsymbol{a}_{i*} \cdot \boldsymbol{w} = \boldsymbol{w} \cdot \boldsymbol{w} > 0$. Given a point $\boldsymbol{x} \in P$, we have that $\boldsymbol{a}_{j*} \cdot (\boldsymbol{x} + \lambda \boldsymbol{w}) = \boldsymbol{a}_{j*} \cdot \boldsymbol{x}$ for $j \neq i$, while $\boldsymbol{a}_{i*} \cdot (\boldsymbol{x} + \lambda \boldsymbol{w}) \geq \boldsymbol{a}_{i*} \cdot \boldsymbol{x}$. Hence $\boldsymbol{w}$ is a ray. The same happens if $l_i = -\infty$. On the converse, assume $\mathcal{P}$ has a ray $\boldsymbol{v}$. If all bounds are finite, by the point 2 of this Lemma, we have that $A\boldsymbol{v} = 0$, i.e., $\boldsymbol{v} = 0$, which is not possible since a ray, by definition, cannot be zero.

Finally, if $\boldsymbol{v} \neq \boldsymbol{0}$ is orthogonal to all the constrained rows, then, according to the second point, both $\boldsymbol{v}$ and $-\boldsymbol{v}$ are rays, hence $\boldsymbol{v}$ is a line. On the contrary, assume $\boldsymbol{v} \neq \boldsymbol{0}$ is a line and $\boldsymbol{a}_{i*}$ has a real bound, let us say $l_i \in \mathbb{R}$. Since $\boldsymbol{v}$ is a ray, we have $\boldsymbol{a}_{i*} \cdot \boldsymbol{v} \leq 0$, and since $-\boldsymbol{v}$ is a ray, we have $a_{i*} \cdot \boldsymbol{v} \geq 0$, hence $\boldsymbol{v}$ is orthogonal to $\boldsymbol{a}_{i*}$. □

Corollary 3. *The linearity space of a parallelotope $\mathcal{P}$ is $V^\perp$ where V is the linear space generated by all the constrained rows of $\mathcal{P}$. It is generated by the orthogonal projections of the unconstrained rows onto $V^\perp$.*

Proof. The fact that the linearity space of a parallelotope is $V^\perp$ is an immediate consequence of the last point of the previous theorem.

For each unconstrained row $\boldsymbol{a}_{i*}$ in $\mathcal{P}$, consider its projection $\boldsymbol{w}_i$ over $V^\perp$. Since it is a vector in $V^\perp$, it is a line. Moreover, all the $\boldsymbol{w}_i$'s are linearly independent. Actually, we have that $\boldsymbol{a}_{i*} = \boldsymbol{w}_i + \boldsymbol{v}_i$ with $\boldsymbol{v}_i \in V$ and $\boldsymbol{w}_i \in V^\perp$. Since $\boldsymbol{v}_i \in V$, we have $\boldsymbol{v}_i = \sum_j \beta_{ij} \boldsymbol{a}_{j*}$ where j ranges over the constrained

rows. If $\sum_i \alpha_i \boldsymbol{w}_i = 0$, we have that $\sum_i \alpha_i \boldsymbol{a}_{i*} - \alpha_i \sum_i \sum_j \beta_{ij} \boldsymbol{a}_{j*} = 0$, which would mean that A is singular. □

Theorem 4. *Given the representation $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$ and a vector $\boldsymbol{c} \in \mathbb{R}^n$, we have that*

$$\inf_{\boldsymbol{x} \in \mathcal{P}} \boldsymbol{c} \cdot \boldsymbol{x} = \inf_{\boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}} \boldsymbol{c}^T A^{-1} \boldsymbol{y}.$$

The computational complexity of the minimization operation is $O(n^3)$.

Proof. By replacing $A\boldsymbol{x}$ with $\boldsymbol{y}$ we may transform the linear programming problem $\inf_{\boldsymbol{x} \in \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle} \boldsymbol{c} \cdot \boldsymbol{x} = \inf_{\boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}} \boldsymbol{c}^T \boldsymbol{x}$ into $\inf_{\boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}} \boldsymbol{c}^T A^{-1} \boldsymbol{y}$. The computational complexity is bounded by the cost for computing $\boldsymbol{c}^T A^{-1}$, which may be found out by solving for $\boldsymbol{z}$ in the system of linear equations $\boldsymbol{z} A = \boldsymbol{c}^T$. □

Theorem 5. *Given a parallelotope $\mathcal{P}$ and $A' \in \mathrm{GL}(n)$, $\mathsf{rot}_{A'}(\mathcal{P})$ is the least parallelotope definable over A' which contains $\mathcal{P}$, i.e., $\mathsf{rot}_{A'}(\mathcal{P}) = \alpha_{A'}(\gamma(\mathcal{P}))$. The computational complexity of $\mathsf{rot}_{A'}$ is $O(n^3)$.*

Proof. If $\mathcal{P} = \epsilon$ the result is obvious. Otherwise, let $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$. If we want to compute $\alpha_{A'}(\gamma(\mathcal{P}))$, it is enough to find out, for each row $\boldsymbol{a}'_{i*}$ in A', the minimum and maximum value of $\boldsymbol{a}'_{i*} \boldsymbol{x}$ for $\boldsymbol{x} \in \gamma(\mathcal{P})$. These will be the values for l'_i and u'_i respectively. Note that $\inf_{\boldsymbol{x} \in \mathcal{P}} \boldsymbol{a}'_{i*} \boldsymbol{x} = \inf_{\boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}} \boldsymbol{a}'_{i*} A^{-1} \boldsymbol{y}$ by Theorem 4. Moreover $\boldsymbol{a}'_{i*} A^{-1} = \boldsymbol{b}_{i*}$, which proves that $\alpha_{A'}(\gamma(\mathcal{P})) = \mathsf{rot}_{A'}(\mathcal{P})$. The computational cost is bounded by the cost for computing B, which may be found out as the solution on X of the equation $XA = A'$. □

Theorem 6. *Given two representations $\mathcal{P}$ and $\mathcal{P}'$, we have that $\mathcal{P} \leq \mathcal{P}'$ iff $\gamma(\mathcal{P}) \subseteq \gamma(\mathcal{P}')$. Moreover, $\leq$ is a pre-order. The computational complexity of deciding $\leq$ is $O(n^3)$.*

Proof. Given the Galois connection between $\alpha_{A'}$ and γ, we have $\gamma(\mathcal{P}) \subseteq \gamma(\mathcal{P}')$ iff $\alpha_{A'}(\gamma(\mathcal{P})) = \mathsf{rot}_{A'}(\mathcal{P}) \leq \mathcal{P}'$ which is equivalent to $\mathcal{P} \leq \mathcal{P}'$. The fact that $\leq$ is a pre-order immediately follows from its characterization in terms of γ and $\subseteq$. The computational complexity of deciding $\leq$ is clearly bounded by the complexity of rot which is $O(n^3)$. □

Theorem 7. *The operation $\mathsf{assign}^{\alpha}(i, \boldsymbol{c}, b)$ is correct and γ-complete. The computational complexity is $O(mn)$ where m is the number of non-zero components in $\boldsymbol{c}$.*

Proof. It trivially holds that $\mathsf{assign}(i, \boldsymbol{c}, b)(\gamma(\epsilon)) = \mathsf{assign}(i, \boldsymbol{c}, b)(\emptyset) = \emptyset = \gamma(\epsilon)$. Given $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle \in \mathsf{Par}_n$, we have

$$\begin{aligned} &\mathsf{assign}(i, \boldsymbol{c}, b)(\gamma(\mathcal{P})) \\ &\quad = \{\boldsymbol{x}[i \mapsto \boldsymbol{c} \cdot \boldsymbol{x} + b] \mid \boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}\} \\ &\quad = \{K\boldsymbol{x} + b\boldsymbol{e}^{\boldsymbol{i}} \mid \boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}\} \end{aligned}$$

where $K = I_n + \boldsymbol{e}^i(\boldsymbol{c} - \boldsymbol{e}^i)^T$. Note that K is invertible and $K^{-1} = I_n - \frac{1}{c_i}\boldsymbol{e}^i(\boldsymbol{c} - \boldsymbol{e}^i)^T$. Therefore, by the change of variable $\boldsymbol{y} = K\boldsymbol{x} + b\boldsymbol{e}^i$, we have

$$\begin{aligned}
&\mathsf{assign}(i, \boldsymbol{c}, b)(\gamma(\mathcal{P})) \\
&\quad = \{\boldsymbol{y} \mid \boldsymbol{l} \leq AK^{-1}(\boldsymbol{y} - b\boldsymbol{e}^i) \leq \boldsymbol{u}\} \\
&\quad = \{\boldsymbol{y} \mid \boldsymbol{l} + bAK^{-1}\boldsymbol{e}^i \leq AK^{-1}\boldsymbol{y} \leq \boldsymbol{u} + bAK^{-1}\boldsymbol{e}^i\}.
\end{aligned}$$

Since $AK^{-1} = A - \frac{1}{c_i}\boldsymbol{a}_{*i}(\boldsymbol{c} - \boldsymbol{e}^i)^T$ we have $AK^{-1}\boldsymbol{e}^i = \boldsymbol{a}_{*i} - \frac{1}{c_i}\boldsymbol{a}_{*i}(c_i - 1) = \frac{1}{c_i}\boldsymbol{a}_{*i}$. Hence

$$\begin{aligned}
&\mathsf{assign}(i, \boldsymbol{c}, b)(\gamma(\mathcal{P})) \\
&\quad = \left\{\boldsymbol{y} \mid \boldsymbol{l} + \frac{b}{c_i}\boldsymbol{a}_{*i} \leq \left(A - \frac{1}{c_i}\boldsymbol{a}_{*i}(\boldsymbol{c} - \boldsymbol{e}^i)^T\right)\boldsymbol{y} \leq \boldsymbol{u} + \frac{b}{c_i}\boldsymbol{a}_{*i}\right\} \\
&\quad = \gamma(\mathsf{assign}^\alpha(i, \boldsymbol{c}, b))(\mathcal{P}).
\end{aligned}$$

This proves that $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)$ is γ-complete. The computational complexity of this operation is dominated by the cost of computing all non-zero products in $\boldsymbol{a}_{*i}(\boldsymbol{c} - \boldsymbol{e}^i)^T$, which is $O(mn)$. This is true even if we use a dense representation for vectors and matrices, since $\frac{1}{c_i}\boldsymbol{a}_{*i}(\boldsymbol{c} - \boldsymbol{e}^i)^T$ may be subtracted from A without materializing it. □

Proposition 8. *Given two parallelotopes $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \leq \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$, we have that*

$$\mathsf{forget}(i)(\gamma(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle)) \subseteq \gamma(\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle) \text{ iff } \forall j(a'_{ji} \neq 0 \to (l'_j = -\infty \wedge u'_j = +\infty)).$$

Proof. If $a'_{ji} \neq 0 \to (l'_j = -\infty \wedge u'_j = +\infty)$ for all j, it follows immediately that for each $v \in \mathbb{R}$ and $\boldsymbol{x} \in \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$, $\boldsymbol{x}[i \mapsto v] \in \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$, since the variable x_i is not used in $\boldsymbol{l}' \leq A'\boldsymbol{x} \leq \boldsymbol{u}'$. Since $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \leq \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$, it follows that $\mathsf{forget}(i)(\gamma(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle)) \subseteq \gamma(\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle)$.

Now, let $\mathsf{forget}(i)(\gamma(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle)) \subseteq \gamma(\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle)$. By contradiction, assume there exists an index j such that $a'_{ji} \neq 0$ and $l'_j \in \mathbb{R}$. Let $\boldsymbol{x} \in \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ and let $v \in \mathbb{R}$ such that $va'_{ji} < l'_j - \boldsymbol{a}'_{j*}\boldsymbol{x}$. It follows that $\boldsymbol{a}'_{j*} \cdot (\boldsymbol{x} + v\boldsymbol{e}^i) = \boldsymbol{a}'_{j*} \cdot \boldsymbol{x} + va'_{ji} < l'_j$, and thus $\boldsymbol{x} + v\boldsymbol{e}^i \notin \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$. However, $\boldsymbol{x} + v\boldsymbol{e}^i \in \mathsf{forget}(i)(\gamma(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle))$, which is a contradiction. The case that there exists an index j such that $a'_{ji} \neq 0$ and $u'_j \in \mathbb{R}$ is analogous. □

Theorem 9. *The operator $\mathsf{forget}^\alpha(i)$ described in Algorithm 1 is correct and relatively optimal. The computational complexity is $O(n^2)$.*

Proof. The algorithm begins at line 1 by computing a set J of indexes. Each element $j \in J$ is the index of a constrained row $\boldsymbol{a}_{j*}$ of the constraint matrix A, where the i-th component a_{ji} is non-zero. If J is empty, by Proposition 8, $\gamma(\mathcal{P}) = \mathsf{forget}(i)(\gamma(\mathcal{P}))$ and therefore we return $\mathcal{P}$. The result is γ-complete, and thus relatively optimal.

Otherwise, lines from 6 to 14 of the algorithm choose an index $r \in R$, according to a given heuristic. The loop at lines 15–21 finds a new constraint matrix

A' and builds at the same time the parallelotope $\langle A', \boldsymbol{l}', \boldsymbol{u}' \rangle = \mathsf{rot}_{A'}(P)$. In particular, the r-th constraint of the parallelotope ($l_r \leq \boldsymbol{a}_{*r} \leq u_r$) is combined with the j-th constraint ($l_j \leq \boldsymbol{a}_{j*} \leq u_j$) to obtain a new constraint $l'_j \leq \boldsymbol{a}'_{j*} \leq u'_j$ which is implied by the first two. The set of rows $\{\boldsymbol{a}'_{j*} \mid j \in J\}$ generates the same vector space as $\{\boldsymbol{a}_{j*} \mid j \in J\}$, therefore A' is invertible. Moreover, after the loop $a'_{ji} = 0$ for each $j \in J \setminus \{r\}$.

From Proposition 8, we know that the smallest parallelotope whose constraint matrix is A' and which contains $\mathsf{forget}(i)(\gamma(\mathcal{P}))$ is given by turning the constrained rows which contain a non-zero entry in the i-th position into unconstrained rows. There is only one such a row in A', which is row r. The bounds for this row are changed in the final return statement.

The complexity is bounded by the cost of determining all the linear combinations in the loop at lines 15–21. □

Theorem 10. *If $\boldsymbol{e}^i$ is a line in $\mathcal{P}$ or $J_0 \neq \emptyset$, then it holds that $\gamma(\mathsf{forget}^\alpha(i)(\mathcal{P})) = \mathsf{forget}(i)(\gamma(\mathcal{P}))$.*

Proof. If $\boldsymbol{e}^i$ is a line in $\gamma(\mathcal{P})$, then $\mathsf{forget}(i)(\gamma(\mathcal{P})) = \gamma(\mathcal{P})$. Moreover, since $\boldsymbol{e}^i$ is orthogonal to the constrained rows in $\mathcal{P}$ (Theorem 2), it means that J in line 1 of the algorithm is empty, hence $\mathsf{forget}^\alpha(i)(\mathcal{P}) = \mathcal{P}$.

Otherwise, assume the set J_0 in line 6 of the algorithm is not empty and the row $\boldsymbol{a}_{r*}$ chosen in line 9 is an equality constraint, with bounds $l_r = u_r = \alpha$.

We now prove that the steps 15–21 in the algorithm do not change the parallelotope. Consider the constraint $c_1 \equiv l'_j \leq \boldsymbol{a}'_{j*} \cdot \boldsymbol{x} \leq u'_j$ computed during the for loop. We prove that this constraint together with the r-th constraint of $\mathcal{P}$, i.e. $c \equiv \boldsymbol{a}_{r*} = \alpha$, is equivalent to the combination of the r-th and j-th constraint of $\mathcal{P}$, i.e. $c_2 \equiv l_j \leq \boldsymbol{a}_{j*} \cdot \boldsymbol{x} \leq u_j$. Given what we have said in the proof of correctness, we only need to check that the j-th constraint of $\mathcal{P}$ is a logical consequence of the new constraint and the r-th constraint in $\mathcal{P}$. Assume without loss of generality that $a_{ri} > 0$ and $a_{ji} > 0$. Then

$$c_1 \equiv a_{ji}\alpha - a_{ri}u_j \leq (a_{ji}\boldsymbol{a}_{r*} - a_{ri}\boldsymbol{a}_{j*}) \cdot \boldsymbol{x} \leq a_{ji}\alpha - a_{ri}l_j \ .$$

It is easy to check that $(a_{ji}c - c_1)/a_{ri} = c_2$. Since this holds for all $j \in J \setminus \{r\}$, we have that, at the end of the loop, $\langle A', \boldsymbol{l}', \boldsymbol{u}' \rangle$ and $\mathcal{P}$ are representations for the same parallelotope.

The last step is removing the bounds in the row r, which is the only constrained row with a non-null i-th coefficient.

Now, given a generic point $\boldsymbol{x} \in \gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}))$, we need to prove that $\boldsymbol{x} \in \mathsf{forget}(i)(\gamma(\mathcal{P}))$. It is obvious that $\boldsymbol{y} = \boldsymbol{x} + \lambda \boldsymbol{e}^i \in \gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}))$ for each $\lambda \in \mathbb{R}$. Note that, for each constrained row $\boldsymbol{a}'_{j*}$ with $j \neq r$, we have that $\boldsymbol{a}'_{j*} \cdot \boldsymbol{y} = \boldsymbol{a}'_{j*} \cdot \boldsymbol{x}$, while $\boldsymbol{a}_{r*} \cdot \boldsymbol{y} = \boldsymbol{a}_{r*} \cdot \boldsymbol{x} + \lambda a_{ri}$. By choosing $\lambda = \alpha - (\boldsymbol{a}_{r*} \cdot \boldsymbol{x})/a_{ri}$, we have $\boldsymbol{y} \in \gamma(\langle A', \boldsymbol{l}', \boldsymbol{u}' \rangle) = \gamma(\mathcal{P})$. Since $\boldsymbol{x} = \boldsymbol{y} - \lambda \boldsymbol{e}^i$ and $\boldsymbol{y} \in \gamma(\mathcal{P})$, then $\boldsymbol{x} \in \mathsf{forget}(i)(\gamma(\mathcal{P}))$. □

Lemma 24. *Consider the polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}\}$ such that $A \in \mathbb{R}(m, n)$ has maximal row rank, $\boldsymbol{a}_{*r} = \boldsymbol{1}$ (the column vector with 1 in all*

its components) and $\boldsymbol{l}, \boldsymbol{u} \in \bar{\mathbb{R}}^n$ *with* $\boldsymbol{l} < \boldsymbol{u}$*. The set of constraints generated by Fourier-Motzkin elimination is* $\{(\boldsymbol{a}_{i*} - \boldsymbol{a}_{j*}) \cdot \boldsymbol{x} \leq u_i - l_j \mid i, j \in \{1, \ldots, m\}\}$*. None of these constraints is redundant.*

Proof. First of all, considering $\boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}$ as a short form of $A\boldsymbol{x} \leq \boldsymbol{u}$ and $-A\boldsymbol{x} \leq -\boldsymbol{l}$ and applying Fourier-Motzkin elimination, we obtain the set

$$\{(\boldsymbol{a}_{i*} - \boldsymbol{a}_{j*}) \cdot \boldsymbol{x} \leq u_i - l_j \mid i, j \in \{1, \ldots, n\}\} \ .$$

Without loss of generality, consider the inequality $(\boldsymbol{a}_{1*} - \boldsymbol{a}_{2*}) \cdot \boldsymbol{x} \leq u_1 - l_2$ and assume $u_1 - l_2$ is finite. We want to prove that this constraint is not redundant.

By Farkas Lemma, this inequality is redundant iff there are $\lambda_{ij} \geq 0$ such that $\lambda_{12} = 0$ and

$$\boldsymbol{a}_{1*} - \boldsymbol{a}_{2*} = \sum_{i,j} \lambda_{ij}(\boldsymbol{a}_{i*} - \boldsymbol{a}_{j*}) \tag{3}$$

and

$$\sum_{i,j} \lambda_{ij}(u_i - l_j) \leq u_1 - l_2 \ .$$

We may rewrite $\sum_{i,j} \lambda_{ij}(\boldsymbol{a}_{i*} - \boldsymbol{a}_{j*})$ as $\sum_{i,j} \lambda_{ij}\boldsymbol{a}_{i*} - \sum_{i,j} \lambda_{ij}\boldsymbol{a}_{j*} = \sum_{i,j} \lambda_{ij}\boldsymbol{a}_{i*} - \sum_{i,j} \lambda_{ji}\boldsymbol{a}_{i*}$ Since the $\boldsymbol{a}_{i*}$ are linearly independent, we have that (3) holds iff

$$\sum_j \lambda_{ij} - \sum_j \lambda_{ji} = \begin{cases} 1 & \text{if } i = 1 \\ -1 & \text{if } i = 2 \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Let $\alpha_i = \sum_j \lambda_{ji}$ and $\beta_i = \sum_j \lambda_{ij}$. Equation 4 means that $\beta_1 = 1 + \alpha_1$, $\beta_2 = -1 + \alpha_2$ and $\alpha_i = \beta_i$ for each $i > 2$. Using these relationships, we may rewrite $\sum \lambda_{ij}(u_i - l_j) \leq u_1 - l_2$ as follows:

$$\begin{aligned} \sum_{i,j} \lambda_{ij}(u_i - l_j) &= \sum_i \Big(\sum_j \lambda_{ij}\Big) u_i - \sum_i \Big(\sum_j \lambda_{ji}\Big) l_i \\ &= \sum_i (\beta_i u_i - \alpha_i l_i) \\ &= (\alpha_1 + 1)u_1 - \alpha_1 l_1 + \sum_{i>2} (\alpha_i u_i - \alpha_i l_i) + \beta_2 u_2 - (1 + \beta_2) l_2 \\ &= u_1 - l_2 + \sum_{i \neq 2} \alpha_i (u_i - l_i) + \beta_2 (u_2 - l_2) \ . \end{aligned}$$

Since $u_i > l_i$, $\lambda_{ij} \geq 0$ and $\sum_{i,j} \lambda_{ij}(u_i - l_j) \leq u_1 - l_2$, then $\alpha_i = 0$ for each $i \neq 2$ and $\beta_2 = 0$. This entails that the only possibly positive λ's are the λ_{j2} with $j > 2$. Let $h > 2$ such that $\lambda_{h2} > 0$, then $\beta_h = \alpha_h > 0$, which is a contradiction. Therefore all the λ_{ij} are zero, i.e. $\boldsymbol{a}_{1*} = \boldsymbol{a}_{2*}$ which is not possible since A is of maximal row rank. Therefore $\boldsymbol{a}_{1*} - \boldsymbol{a}_{2*} \leq u_1 - l_2$ is not redundant. □

Corollary 25. *Consider the polyhedron $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}\}$ such that $A \in \mathbb{R}(m,n)$ has maximal row rank, $\boldsymbol{a}_{*r}$ has no zero components and $\boldsymbol{l}, \boldsymbol{u} \in \bar{\mathbb{R}}^n$ with $\boldsymbol{l} < \boldsymbol{u}$. Then, none of the constraints obtained by Fourier-Motzkin elimination of P is redundant.*

Proof. Consider the matrix A' such that $a'_{ij} = a_{ij}/a_{ir}$, and vectors $\boldsymbol{l}'$ and $\boldsymbol{u}'$ such that

$$l'_i = \begin{cases} l_i/a_{ir} & \text{if } a_{ir} > 0; \\ u_i/a_{ir} & \text{otherwise;} \end{cases} \qquad u'_i = \begin{cases} u_i/a_{ir} & \text{if } a_{ir} > 0; \\ l_i/a_{ir} & \text{otherwise.} \end{cases}$$

Then $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l}' \leq A'\boldsymbol{x} \leq \boldsymbol{u}'\}$. We may apply Lemma 24 and note that each constraint $(\boldsymbol{a}'_{i*} - \boldsymbol{a}'_{j*}) \cdot \boldsymbol{x} \leq u'_i - l'_j$ is equivalent to one of the constraints obtained by Fourier-Motkin elimination of P. For example, if $a_{ir} > 0$ and $a_{jr} < 0$, constraints $\boldsymbol{a}_{i*} \cdot \boldsymbol{x} \leq u_i$ and $\boldsymbol{a}_{j*} \cdot \boldsymbol{x} \leq u_j$ are combined into $(-a_{jr}\boldsymbol{a}_{i*} + a_{ir}\boldsymbol{a}_{j*}) \cdot \boldsymbol{x} \leq -a_{jr}u_i + a_{ir}u_j$, which is equivalent to $(\boldsymbol{a}_{i*}/a_{ij} - \boldsymbol{a}_{j*}/a_{jr}) \cdot \boldsymbol{x} \leq u_i/a_{ir} - u_j/a_{jr}$, i.e, $(\boldsymbol{a}'_{i*} - \boldsymbol{a}'_{j*}) \cdot \boldsymbol{x} \leq u'_i - l'_j$. The other cases are similar. □

Proposition 11. *Given a parallelotope $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ and a column index i, assume that $l_j < u_j$ for each constrained row j such that $a_{ji} \neq 0$. Consider the polyhedron obtained by Fourier-Motzkin elimination of the i-th variable. None of its constraints with finite bounds is redundant.*

Proof. Let J be defined as in Algorithm 1, i.e. $J = \{j \mid a_{ji} \neq 0, l_j \neq -\infty \vee u_j \neq +\infty\}$. We have that the result of Fourier-Motzkin elimination is the polyhedron

$$\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l}_{-J} \leq A_{-J*}\boldsymbol{x} \leq \boldsymbol{u}_{-J}\} \cap \{\boldsymbol{x} \in \mathbb{R}^n \mid D\boldsymbol{x} \leq \boldsymbol{d}\}$$

where $D\boldsymbol{x} \leq \boldsymbol{d}$ is the set of constraints determined by Fourier-Motzkin elimination from the polyhedron $\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l}_J \leq A_{J*} \cdot \boldsymbol{x} \leq \boldsymbol{u}_J\}$. We need to prove that all the constraints with finite bounds are not redundant. All finitely bounded constraints of the kind $l_j \leq \boldsymbol{a}_{j*} \cdot \boldsymbol{x}$ or $\boldsymbol{a}_{j*} \cdot \boldsymbol{x} \leq u_j$ for $j \notin J$ are not redundant simply because $\boldsymbol{a}_{j*}$ is linearly independent from all the rows in D and in $A_{-(J\cup\{j\})*}$. Therefore, it is not possible for such a constraint to be a positive linear combination of other constraints.

Moreover, since the rows of D are linearly independent from the rows of A_{-J*}, if a constraint generated by D is redundant, then it should be a positive linear combination of other constraints generated by D. This is not possible by Corollary 25. □

Lemma 26. *Let $P \subseteq P' \subseteq P''$ be convex sets, and H an hyper-plane. If $P \cap H$ and $P'' \cap H$ are faces of dimension k, then $P' \cap H$ is a face of dimension k.*

Proof. By definition, if C is a convex set and H is a hyper-plane, $C \cap H$ is a face when it is non-empty and C is entirely on one side of H. Since $P \cap H$ is non-empty and $P' \supseteq P$, then $P \cap H$ is not-empty. Moreover, since P'' is entirely on one side of H and $P' \subseteq H$, then P' is entirely on one side of H. Then $P' \cap H$ is a face. Moreover, since $P \cap H \subseteq P' \cap H \subseteq P'' \cap H$ and both $P \cap H$ and $P'' \cap H$ have dimension k, also $P' \cap H$ has dimension k. □

Theorem 12. *The operator* $\mathsf{forget}^\alpha(i)$ *is minimal.*

Proof. Let $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ be a parallelotope representation and $P = \gamma(\mathcal{P}) \subseteq \mathbb{R}^n$. If we are in the case where $\mathsf{forget}^\alpha(i)$ is γ-complete, then it is also minimal. Otherwise, assume $\mathcal{P}$ has a equality rows and u unconstrained rows. Therefore, $\dim(P) = n - a$ and $\dim(\mathrm{lin}(P)) = u$. If j is an equality row, then $\boldsymbol{a}_{ji} = 0$, otherwise we fall in the case where $\mathsf{forget}^\alpha(i)$ is γ-complete.

Consider now the polyhedron $\mathsf{forget}(i)(P)$. By the properties of P discussed above, we have that $\dim(\mathsf{forget}(i)(P)) = \dim(P)$, $\mathrm{aff.\,hull}(\mathsf{forget}(i)(P)) = \mathrm{aff.\,hull}(P)$ and $\dim(\mathrm{lin}(\mathsf{forget}(i)(P))) = u + 1$. Assume $c \equiv \boldsymbol{v} \cdot \boldsymbol{x} \leq u$ is a constraint generated by Fourier-Motzkin elimination and $H = \{\boldsymbol{x} \mid \boldsymbol{v} \cdot \boldsymbol{x} \leq u\}$. Since c is not redundant, by a property of polyhedron we have that $\mathsf{forget}(i)(P) \cap H$ is a facet of $\mathsf{forget}(i)(P)$, i.e., a face of dimension $\dim(P) - 1$.

If $\mathcal{P}' = \mathsf{forget}^\alpha(i)(\mathcal{P}) = \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$ and $P' = \gamma(\mathcal{P}')$ we have that $\mathcal{P}'$ has a equality rows and $u + 1$ unconstrained rows. The constraints corresponding to the equality rows are unchanged from those in $\mathcal{P}$, hence $\dim(P') = \dim(P)$, $\mathrm{aff.\,hull}(P') = \mathrm{aff.\,hull}(P)$ and $\dim(\mathrm{lin}(P')) = u + 1$. The other constraints are a selection of constraints obtained by Fourier-Motzkin elimination. Since these constraints were non-redundant in $\mathsf{forget}(i)(P)$ by Proposition 11, they are also non-redundant in $\mathcal{P}'$. Hence if $\boldsymbol{a}'_{j*} \cdot \boldsymbol{x} \leq u_j$ is a bounded constraint in $\mathcal{P}'$ and $H = \{\boldsymbol{x} \mid \boldsymbol{a}'_{j*} \cdot \boldsymbol{x} \leq u_j\}$, then $P' \cap H$ is a facet of P'. The same holds for constraints of the kind $l_j \leq \boldsymbol{a}'_{j*} \cdot \boldsymbol{x}$.

Now assume $\mathcal{P}''$ is a correct approximation of $\mathsf{forget}(i)(P)$ such that $\mathcal{P}'' \leq \mathcal{P}'$ and $P'' = \gamma(\mathcal{P}'')$. Therefore, $\dim(P'') = \dim(P') = \dim(\mathsf{forget}(i)(P))$, $\mathrm{aff.\,hull}(P'') = \mathrm{aff.\,hull}(P') = \mathrm{aff.\,hull}(\mathsf{forget}(i)(P))$ and $\dim(\mathrm{lin}(P'')) = u + 1$. We want to prove that in $\mathcal{P}''$ should explicitly appear all the constraints which appear in $\mathcal{P}'$. Without loss of generality, consider the constraint $\boldsymbol{a}'_{j*} \cdot \boldsymbol{x} \leq u_j$ in $\mathcal{P}'$ and $H = \{\boldsymbol{x} \mid \boldsymbol{a}'_{j*} \cdot \boldsymbol{x} \leq u_j\}$. We know that both $P' \cap H$ and $\mathsf{forget}(i)(P) \cap H$ are faces of dimension $\dim(P') - 1$. By Lemma 26, $P'' \cap H$ is a face of P'' of the same dimension, and since $\dim(P'') = \dim(P')$ then $P'' \cap H$ is a facet.

For a polyhedron, and therefore also for a parallelotope, facets exactly corresponds to the inequality constraints in its definition. Therefore, $\boldsymbol{a}'_{j*} \cdot \boldsymbol{x} \leq u_j$ (or an equivalent multiple) should appear in $\mathcal{P}''$. This holds for all constrained inequality rows of A' and for constraints determined by either lower or upper bounds. Since a rows in $\mathcal{P}''$ are needed for the equality constraints which force $\mathrm{aff.\,hull}(P'') = \mathrm{aff.\,hull}(P')$ and $u + 1$ row should remain unconstrained, so that $\dim(\mathrm{lin}(P'')) = \dim(\mathrm{lin}(P'))$, we have that all remaining constraints in $\mathcal{P}''$ are multiples of constraints in $\mathcal{P}'$, and therefore $P'' = P'$. □

Theorem 13. *The operator* $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)$ *described in Algorithm 2 is correct. The computational complexity is* $O(n^2)$.

Proof. We begin by proving that lines 2–5 of the algorithm change the representation $\mathcal{P}' = \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$ into another representation for the same parallelotope. First of all, we need to check that the new vectors $\boldsymbol{v}_s = \boldsymbol{a}'_{s*} - (a'_{si}/a'_{ji})\boldsymbol{a}'_{j*}$ correspond to unbounded directions in $\gamma(\mathcal{P}')$, i.e., that the linear form $\boldsymbol{x} \mapsto \boldsymbol{v}_s \cdot \boldsymbol{x}$ is unbounded for $\boldsymbol{x} \in \gamma(\mathcal{P}')$. Note that this does not mean that $\boldsymbol{v}_s$ is a line.

Since A' is invertible, $\boldsymbol{v}_s$ cannot be obtained as a linear combination of the constrained rows of A', hence $\boldsymbol{v}_s = \lambda_1 \boldsymbol{w}_1 + \lambda_2 \boldsymbol{w}_2$ where $\boldsymbol{w}_2$ is a line and $\boldsymbol{w}_1$ is orthogonal to the linearity space of $\mathcal{P}'$. Hence, $\inf\{\boldsymbol{v}_s \cdot \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}'\} = -\infty$ and $\sup\{\boldsymbol{v}_s \cdot \boldsymbol{x} \mid \boldsymbol{x} \in \mathcal{P}'\} = +\infty$. Moreover, the operations at line 4 are *basic row operations*, which are known to preserve the rank of a matrix.

At line 6, $\boldsymbol{a}'_{j*}$ is the only row in A' with a non-zero i-th component. If we replace it with another row which has a non-empty i-th component, such as $\boldsymbol{e}^i - \boldsymbol{c}$, the matrix remains invertible. Moreover, $\boldsymbol{e}^i - \boldsymbol{c}$ is an unbounded direction in $\mathsf{forget}^\alpha(i)(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle)$. Therefore, at the end of line 6, we still have $\gamma(\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle) = \gamma(\mathsf{forget}^\alpha(i)(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle))$.

Lines 7 and 8 in the algorithm amount to building the parallelotope representation $\alpha_{A'}(\mathsf{assign}(i, \boldsymbol{c}, b)(\gamma\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle)) = \alpha_{A'}(\mathsf{assign}(i, \boldsymbol{c}, b)(\gamma(\mathsf{forget}^\alpha(i)\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle)))$. Since $\mathsf{forget}^\alpha(\_)$ is correct, this proves the correctness of the non-invertible assignment.

The computational complexity of the algorithm is dominated by the cost of the loop at lines 3–5, which is $O(n^2)$. □

Theorem 14. *Consider $\boldsymbol{c} \in \mathbb{R}^n$, $b \in \mathbb{R}$ and $c_i = 0$. If $\mathcal{P}$ is a parallelotope in $\mathbb{R}^n$ and $\boldsymbol{e}^i$ is a line in $\mathcal{P}$ or $J_0 \neq \emptyset$ in Algorithm 1 then $\gamma(\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)(\mathcal{P})) = \mathsf{assign}(i, \boldsymbol{c}, b)(\gamma(\mathcal{P}))$.*

Proof. Assume $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ and $\langle A'', \boldsymbol{l}'', \boldsymbol{u}''\rangle = \mathsf{assign}^\alpha(i, \boldsymbol{c}, b)(\mathcal{P})$. Consider a generic $\boldsymbol{x} \in \gamma(\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)(\mathcal{P}))$. If j is the row determined at line 2 of Algorithm 2, then $\boldsymbol{a}''_{j*}\boldsymbol{x} = b$, which means that $\boldsymbol{x}$ satisfies the equation $x_i = \boldsymbol{c} \cdot \boldsymbol{x} + b$. At then end of line 5 of Algorithm 2, we have values of A', $\boldsymbol{l}'$ and $\boldsymbol{u}'$ which only differ from A'', $\boldsymbol{l}''$ and $\boldsymbol{u}''$ for the j-th constraint, which directly comes from the result of $\mathsf{forget}^\alpha(i)$. From the definition of $\mathsf{forget}^\alpha(i)$, it turns out that $l'_j = -\infty$ and $u'_j = +\infty$, hence $\boldsymbol{x} \in \gamma(\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle)$. Since lines 2–5 in the algorithm do not change the parallelotope, we have $\boldsymbol{x} \in \gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}))$. By Theorem 10, $\boldsymbol{x} \in \mathsf{forget}(i)(\gamma(\mathcal{P}))$, and since $\boldsymbol{x}$ satisfies the equation $x_i = \boldsymbol{c} \cdot \boldsymbol{x} + c$, then also $\boldsymbol{x} \in \mathsf{assign}(i, \boldsymbol{c}, b)(\mathsf{forget}(i)(\gamma(\mathcal{P}))) = \mathsf{assign}(i, \boldsymbol{c}, b)(\gamma(\mathcal{P}))$. □

Theorem 15. *The operator $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)$ with $c_i = 0$ is minimal.*

Proof. Let $\mathcal{P}'' = \langle A'', \boldsymbol{l}'', \boldsymbol{u}''\rangle$ be a correct approximation of $\mathsf{assign}(i, \boldsymbol{c}, b)(\mathcal{P})$, $\mathcal{P}' = \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle = \mathsf{assign}^\alpha(i, \boldsymbol{c}, b)(\mathcal{P})$ and assume $\mathcal{P}'' \leq \mathcal{P}'$.

Note that $\gamma(\mathcal{P}')$ lies on the hyper-plane $(\boldsymbol{e}^i - \boldsymbol{c}) \cdot \boldsymbol{x} = b$. Since $\mathcal{P}'' \leq \mathcal{P}'$, the same holds for $\gamma(\mathcal{P}'')$. It is easy to check that this implies

$$\gamma(\mathcal{P}'') = \mathsf{assign}(i, \boldsymbol{c}, b)(\mathsf{forget}(i)(\gamma(\mathcal{P}''))) \ . \tag{5}$$

Since $\mathcal{P}''$ lies in the hyper-plane $(\boldsymbol{e}^i - \boldsymbol{c}) \cdot \boldsymbol{x} = b$, one of its equality rows must have a non-zero i-th component. The same also holds for $\mathcal{P}'$. By Theorem 10 we get:

$$\gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}')) = \mathsf{forget}(i)(\gamma(\mathcal{P}')) \ , \tag{6}$$

$$\gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}'')) = \mathsf{forget}(i)(\gamma(\mathcal{P}'')) \ . \tag{7}$$

Moreover, it is clear that $\mathsf{forget}(i) \circ \mathsf{assign}(i, \boldsymbol{c}, b) = \mathsf{forget}(i)$ and $\gamma \circ \mathsf{forget}^\alpha(i) \circ \mathsf{assign}^\alpha(i, \boldsymbol{c}, b) = \gamma \circ \mathsf{forget}^\alpha(i)$. Therefore, by (7),

$$\gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}'')) = \mathsf{forget}(i)(\gamma(\mathcal{P}'')) \supseteq \\ \supseteq \mathsf{forget}(i)(\mathsf{assign}(i, \boldsymbol{c}, b)(\gamma(\mathcal{P}))) = \mathsf{forget}(i)(\gamma(\mathcal{P}))$$

which means that $\mathsf{forget}^\alpha(i)(\mathcal{P}'')$ is a correct approximation of $\mathsf{forget}(i)(\mathcal{P})$. Moreover, by (6) and (7),

$$\gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}'')) = \mathsf{forget}(i)(\gamma(\mathcal{P}'')) \subseteq \mathsf{forget}(i)(\gamma(\mathcal{P}')) = \\ = \gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}')) = \gamma(\mathsf{forget}^\alpha(i)(\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)(\mathcal{P}))) = \gamma(\mathsf{forget}^\alpha(i)(\mathcal{P})) \ ,$$

hence $\gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}'')) \subseteq \gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}))$. However, by Theorem 12, we have that $\mathsf{forget}^\alpha(i)(\mathcal{P})$ is a minimal correct approximation of $\mathsf{forget}(i)(\mathcal{P})$. Hence, we have $\gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}'')) = \gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}))$, and together with (7) this gives $\mathsf{forget}(i)(\gamma(\mathcal{P}'')) = \gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}))$. Therefore, by (5),

$$\gamma(\mathcal{P}'') = \mathsf{assign}(i, \boldsymbol{c}, b)(\mathsf{forget}(i)(\gamma(\mathcal{P}''))) = \mathsf{assign}(i, \boldsymbol{c}, b)(\gamma(\mathsf{forget}^\alpha(i)(\mathcal{P}))$$

and since $\boldsymbol{e}^i$ is a line in $\mathsf{forget}^\alpha(i)(\mathcal{P})$, by Theorem 14 we have that $\gamma(\mathcal{P}'') = \gamma(\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)(\mathsf{forget}^\alpha(i)(\mathcal{P}))) = \gamma(\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)(\mathcal{P}))$. □

Theorem 16. *The operator* $\mathsf{refine}^\alpha(\boldsymbol{c}, b)$ *described in Algorithm 3 is correct and relatively optimal. If* $\boldsymbol{c} \notin \mathrm{lin}(\gamma(\mathcal{P}))^\perp$*, then* $\mathsf{refine}^\alpha(\boldsymbol{c}, b)(\mathcal{P})$ *is* γ*-complete. The computational complexity is* $O(n^3)$*.*

Proof. Assume $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u} \rangle$, and let $y \in \mathbb{R}^n$ be the solution of the equation $A^T \boldsymbol{y} = \boldsymbol{c}$. First of all, we prove γ-optimality when $\boldsymbol{c} \notin \mathrm{lin}(\gamma(\mathcal{P}))^\perp$. In this case there is an index j that satisfies the condition in line 2 of the algorithm: since $\boldsymbol{c} \notin \mathrm{lin}(\gamma(\mathcal{P}))^\perp$, then $\boldsymbol{c}$ is not a linear combination of the constrained rows in $\mathcal{P}$. Steps 3 and 4 introduce the new constraint and, due to the choice of j, the modified matrix A is invertible. Let A', $\boldsymbol{l}'$, $\boldsymbol{u}'$ be the values of variables A, $\boldsymbol{l}$ and $\boldsymbol{u}$ at line 5, i.e., $\langle A', \boldsymbol{l}', \boldsymbol{u}' \rangle = \mathsf{refine}^\alpha(\boldsymbol{c}, b)(\mathcal{P})$. Then $\boldsymbol{x} \in \gamma(\langle A', \boldsymbol{l}', \boldsymbol{u}' \rangle)$ iff $\boldsymbol{x} \in \gamma(\langle A, \boldsymbol{l}, \boldsymbol{u} \rangle)$ and $\boldsymbol{c} \cdot \boldsymbol{x} \leq b$ iff $\boldsymbol{x} \in \mathsf{refine}(\boldsymbol{c}, b)(\gamma(\mathcal{P}))$. This proves that, when $\boldsymbol{c} \notin \mathrm{lin}(\gamma(\mathcal{P}))^\perp$, $\mathsf{refine}^\alpha(\boldsymbol{c}, b)$ is γ-complete.

We now prove correctness when $\boldsymbol{c} \in \mathrm{lin}(\gamma(\mathcal{P}))^\perp$. Note that, in this case,

$$\begin{aligned} \mathsf{refine}(\boldsymbol{c}, b)(\mathcal{P}) = \{\boldsymbol{x} \mid \boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u} \wedge \boldsymbol{c} \cdot \boldsymbol{x} \leq b\} = \\ \{A^{-1}\boldsymbol{z} \mid \boldsymbol{l} \leq \boldsymbol{z} \leq \boldsymbol{u} \wedge \boldsymbol{c} \cdot (A^{-1}\boldsymbol{z}) \leq b\} = \\ \{A^{-1}\boldsymbol{z} \mid \boldsymbol{l} \leq \boldsymbol{z} \leq \boldsymbol{u} \wedge ((A^T)^{-1}\boldsymbol{c}) \cdot \boldsymbol{z} \leq b\} = \\ \{A^{-1}\boldsymbol{z} \mid \boldsymbol{l} \leq \boldsymbol{z} \leq \boldsymbol{u} \wedge \boldsymbol{y} \cdot \boldsymbol{z} \leq b\} = \\ \{A^{-1}\boldsymbol{z} \mid \boldsymbol{z} \in \mathsf{refine}(\boldsymbol{y}, b)(\gamma(\langle I_n, \boldsymbol{l}, \boldsymbol{u} \rangle))\} = \\ \{\boldsymbol{x} \mid A\boldsymbol{x} \in \mathsf{refine}(\boldsymbol{y}, b)(\gamma(\langle I_n, \boldsymbol{l}, \boldsymbol{u} \rangle))\} \subseteq \\ \{\boldsymbol{x} \mid A\boldsymbol{x} \in \gamma(\mathsf{refine}^\alpha(\boldsymbol{y}, b)(\langle I_n, \boldsymbol{l}, \boldsymbol{u} \rangle))\} \end{aligned}$$

If $\mathsf{refine}^{\alpha}(\boldsymbol{y}, b)(\langle I_n, \boldsymbol{l}, \boldsymbol{u}\rangle) = \langle I_n, \boldsymbol{l}', \boldsymbol{u}'\rangle$, the last set in the previous equation is equal to $\gamma(\langle A, \boldsymbol{l}', \boldsymbol{u}'\rangle)$. Since $\langle A, \boldsymbol{l}', \boldsymbol{u}'\rangle$ is the result computed in lines 7–8 of the algorithm, we have that Algorithm 3 is correct. Moreover, since $\mathsf{refine}^{\alpha}(\boldsymbol{c}, b)$ on boxes is optimal, the algorithm is also relatively optimal.

Finally, the computational complexity of the algorithm is bounded by the time needed to solve the system of linear equations in line 1, which is $O(n^3)$. □

Theorem 17. *The abstract union operator described in Algorithm 5 is correct and relatively optimal. The computational complexity is $O(n^4)$.*

Proof. We first prove that the result of Algorithm 5 is correct.

Given two parallelotopes $\mathcal{P}_A, \mathcal{P}_B \in \mathsf{Par}_n$ and a vector $\boldsymbol{v} \in \mathbb{R}^n$, Algorithm 4 computes a triple $\langle l, u, p\rangle \in \bar{\mathbb{R}}^3$. By line 22 in Algorithm 4 we know that $l = \min(\inf\{\boldsymbol{v} \cdot \boldsymbol{x} \mid x \in \mathcal{P}_A\}, \inf\{\boldsymbol{v} \cdot \boldsymbol{x} \mid x \in \mathcal{P}_B\})$, and thus $l = \inf\{\boldsymbol{v} \cdot \boldsymbol{x} \mid x \in \mathcal{P}_A \cup \mathcal{P}_B\}$. Symmetrically we have that $u = \sup\{\boldsymbol{v} \cdot \boldsymbol{x} \mid x \in \mathcal{P}_A \cup \mathcal{P}_B\}$.

In Algorithm 5, we first fill a priority queue Q with the candidate constraints. The lines 2-6 ensure that all the rows of the matrices A and B are added to Q. By line 4, we insert in Q the constraints with their lower and upper bounds, as computed by Algorithm 4. With the lines 7-22 we insert in Q the candidate constraints which form an inversion, with their lower and upper bounds computed by Algorithm 4. The while loop in lines 24-29 extracts n linearly independent elements from Q which form the constraint matrix. First note that the queue Q contains at least n such linear forms. In fact, all the n constraints of A are in the queue, and they are linearly independent. Thus, lines 25-28 which extract n linearly independent elements from Q always succeed.

Moreover, for each $\langle \boldsymbol{v}, c, d\rangle$, in the queue, we know by Algorithm 4 that $c = \inf\{\boldsymbol{v} \cdot \boldsymbol{x} \mid x \in \mathcal{P}_A \cup \mathcal{P}_B\}$, and $d = \sup\{\boldsymbol{v} \cdot \boldsymbol{x} \mid x \in \mathcal{P}_A \cup \mathcal{P}_B\}$. Thus, $\gamma(\{\boldsymbol{x} \in \mathbb{R}^n \mid c \leq \boldsymbol{v} \cdot \boldsymbol{x} \leq d\}) \supseteq \gamma(\mathcal{P}_A) \cup \gamma(\mathcal{P}_B)$. By line 27 we know that the resulting matrix R is formed by n linear forms all enjoying this property. Then for the algorithm output $\langle R, \boldsymbol{l}', \boldsymbol{u}'\rangle$ it holds that $\gamma(\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l}' \leq R\boldsymbol{x} \leq \boldsymbol{u}'\}) = \cap_{i \in \{1,\ldots,n\}} \gamma(\{\boldsymbol{x} \in \mathbb{R}^n \mid l'_i \leq \boldsymbol{r}_{i*} \cdot \boldsymbol{x} \leq u'_i\})$ which, by the previous property, $\supseteq \gamma(\mathcal{P}_A) \cup \gamma(\mathcal{P}_B)$. Thus the union operator is correct.

Relative optimality immediately follows from the fact that Algorithm 4 computes the tightest bounds in lines 1-4.

The computational complexity of Algorithm 4 is $O(n^3)$ by Theorem 4, and all the other operators on vectors have a lower complexity. But the complexity is $O(n^3)$ only the first time, since we need to compute the inverse of the two matrices. Successive calls have a complexity of $O(n^2)$.

Since Algorithm 4 is called n times in lines 2-5, n times in line 6, at most n^2 times in the loop in 7-19, and the same for lines 20, 21 and 22, we have that the complexity of all the vector operations is at most $O(n^4)$. The other part comes from the complexity for insert and extract operations in the priority queue. We perform at most $O(n^2)$ insert operations in lines 7-22, and at most $O(n^2)$ extract operations in line 25 (since the number of elements in the queue is at most $O(n^2)$), and thus the complexity is bounded by $O(n^4)$. □

Lemma 27. *Given the parallelotope $\mathcal{P} = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ in $\mathbb{R}^n$ and $A' \in \mathrm{GL}(n)$, let $\mathcal{P}' = \mathsf{rot}_{A'}(\mathcal{P}) = \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$. Then, the sum of the number of infinite components in $\boldsymbol{l}'$ and $\boldsymbol{u}'$ is greater or equal then the sum of the number of infinite components in $\boldsymbol{l}$ and $\boldsymbol{u}$.*

Proof. Each row of A' may be written as a linear combination of the rows in A. Assume we have

$$\boldsymbol{a}'_{j*} = \sum_{i=1}^{n} \lambda_i \boldsymbol{a}_{i*}$$

with $\lambda_k > 0$ and assume $u_k = +\infty$. Then either $\sup_{\boldsymbol{x}\in\mathcal{P}} \boldsymbol{a}'_{j*} \cdot \boldsymbol{x} = +\infty$ or $\inf_{\boldsymbol{x}\in\mathcal{P}} \boldsymbol{a}'_{j*} \cdot \boldsymbol{x} = -\infty$. Consider the set R of all the rows in A with an infinite upper bound. Then, there are at least $|R|$ rows in A' whose linear combinations involve rows in R. Actually, if this is not the case, it means that there are strictly more than $n-|R|$ linearly indipendent rows in A' which may be written as linear combination of $n-|R|$ rows in A, and this is impossible. Therefore, to each infinite upper bound in $\mathcal{P}$ corresponds an infinite lower or upper bound in $\mathcal{P}'$. The same happens for the lower bounds. Therefore, the set of infinite bounds in $\mathcal{P}'$ is no less than the set of infinite bounds in $\mathcal{P}$. □

Theorem 18. *The operator $\overline{\triangledown}$ is a widening.*

Proof. The fact that $\overline{\triangledown}$ is an upper bound immediately derives from the fact that $\triangledown$ is an upper bound. Now consider a sequence $\mathcal{P}_0, \ldots, \mathcal{P}_i, \ldots$ of parallelotopes. We do not require the sequence to be increasing. We need to prove that the sequence $\{\mathcal{P}_i^w\}$ defined as $\mathcal{P}_0^w = \mathcal{P}_0$ and $\mathcal{P}_i^w = \mathcal{P}_{i-1}^w \,\overline{\triangledown}\, \mathcal{P}_i$ for $i > 0$ is definitively stationary. Let $\mathcal{P}_i = \langle A_i, \boldsymbol{l}_i, \boldsymbol{u}_i\rangle$ and $\mathcal{P}_i^w = \langle A_i^w, \boldsymbol{l}_i^w, \boldsymbol{u}_i^w\rangle$.

We show that for each i, either $\mathcal{P}_i^w = \mathcal{P}_{i+1}^w$ or the number of infinite components in $\boldsymbol{l}_i^w$ and $\boldsymbol{u}_i^w$ is strictly smaller than the number of infinite components in $\boldsymbol{l}_{i+1}^w$ and $\boldsymbol{u}_{i+1}^w$. This is obviously true when $\mathcal{P}_{i+1}^w = \mathcal{P}_i^w \triangledown \mathcal{P}_i$. Now we consider the case $\mathsf{rot}_{A_{i+1}}(\mathcal{P}_i^w) < \mathcal{P}_{i+1}$, which implies $\mathcal{P}_{i+1}^w = \mathsf{rot}_{A_{i+1}}(\mathcal{P}_i^w) \triangledown \mathcal{P}_{i+1}$. By Lemma 27 we now that the number of infinite bounds in $\mathsf{rot}_{A_{i+1}}(\mathcal{P}_i^w)$ is greater or equal than the number of infinite bounds in $\mathcal{P}_i^w$. Since $\mathcal{P}_{i+1}$ is strictly greater than $\mathsf{rot}_{A_{i+1}}(\mathcal{P}_i^w)$ and we apply the $\triangledown$ widening, at least one finite bound is turned into an infinite bound. □

Theorem 19. *In the hypothesis of Definition 10, let $\bar{F}$ be a (possibly non-monotone) abstract operator $\bar{F} : A \to A$ which is a correct abstraction of the monotone operator $F : C \to C$. Given $a \in A$, let $c \in C$ be a fixpoint of F such that $c \subseteq \gamma(a)$. We define the iteration sequence $y_0 = a$, $y_{i+1} = y_i \,\triangle\, \bar{F}(y_i)$. Then*

- *the iteration sequence $\{y_i\}_{i\in\mathbb{N}}$ is decreasing and definitively stationary;*
- *for each $i \in \mathbb{N}$, y_i is a correct approximation of c.*

Proof. The fact that $\{y_i\}_{i\in\mathbb{N}}$ is decreasing and definitively stationary immediately follows from the first and third properties in the definition of narrowing. Now we prove that if $c \subseteq \gamma(y_i)$, i.e., y_i is a correct approximation of c, then $c \subseteq \gamma(y_{i+1})$. By the second property of narrowing, $\gamma(y_{i+1}) \supseteq \gamma(y_i) \cap \gamma(\bar{F}(y_i))$.

By correctness of $\bar{F}$, we have $\gamma(\bar{F}(y_i)) \supseteq F(\gamma(y_i)) \supseteq F(c) = c$, and $\gamma(y_{i+1}) \supseteq c$. □

Theorem 20. *The operator $\triangle$ for parallelotopes is a narrowing according to Definition 10.*

Proof. Since $\mathcal{P}_A \triangle \mathcal{P}_B$ is built on the same constraint matrix of $\mathcal{P}_A$ by possibly restricting the bounds, then $\mathcal{P}_A \triangle \mathcal{P}_B \leq \mathcal{P}_A$. Now, let $\mathcal{P}_A$, $\mathcal{P}_B$ and $\mathsf{rot}_A(\mathcal{P}_B)$ as in Definition 11. Assume $\boldsymbol{x} \in \gamma(\mathcal{P}_A) \cap \gamma(\mathcal{P}_B)$. Then, for each row i in A, we have $l_i \leq \boldsymbol{a}_{i*}\boldsymbol{x} \leq u_i$ and $j'_i \leq \boldsymbol{a}_{i*}\boldsymbol{x} \leq k'_i$, hence $\max(l_i, j'_i) \leq \boldsymbol{a}_{i*}\boldsymbol{x} \leq \min(u_i, k'_i)$. Since $l'_i \leq \max(l_i, j'_i)$ and $u'_i \geq \min(u_i, k'_i)$, then $l'_i \leq \boldsymbol{a}_{i*}\boldsymbol{x} \leq u'_i$, hence $\boldsymbol{x} \in \gamma(\mathcal{P}_A \triangle \mathcal{P}_B)$. Termination is guaranteed because bounds may be enlarged at most $2n$ times. □

Proposition 21. *The operator $\cap^\alpha$ is a correct approximation of the concrete intersection. It is γ-complete when the two arguments are defined over the same constraint matrix.*

Proof. We first prove that when $\mathcal{P}_1 = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ and $\mathcal{P}_2 = \langle A, \boldsymbol{l}', \boldsymbol{u}'\rangle$ are defined over the same constraint matrix, then $\cap_\alpha$ is γ-complete. In this case the rot operation is a no-op, hence $\mathsf{rot}_A(\mathcal{P}_2) = \mathcal{P}_2$. Let $\boldsymbol{x} \in \gamma(\mathcal{P}_1) \cap \gamma(\mathcal{P}_2)$. This is equivalent to $\boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}$ and $\boldsymbol{l}' \leq A\boldsymbol{x} \leq \boldsymbol{u}'$. If we use $\boldsymbol{l}''$ and $\boldsymbol{u}''$ as in the definition of weak intersection, we have the equivalent property $\boldsymbol{l}'' \leq A\boldsymbol{x} \leq \boldsymbol{u}''$. If $\boldsymbol{l}'' \leq \boldsymbol{u}''$, this means $\boldsymbol{x} \in \gamma(\langle A, \boldsymbol{l}'', \boldsymbol{u}''\rangle)$, otherwise the intersection is empty. In both cases $\boldsymbol{x} \in \gamma(\mathcal{P}_1) \cap^\alpha \gamma(\mathcal{P}_2)$ iff $\boldsymbol{x} \in \gamma(\mathcal{P}_1 \cap^\alpha \mathcal{P}_2)$.

Now, let us consider the general case. It is immediate to check that if $\mathcal{P}_1 = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$, then $\mathcal{P}_1 \cap^\alpha \mathcal{P}_2 = \mathcal{P}_1 \cap^\alpha \mathsf{rot}_A(\mathcal{P}_2)$. By the γ-completeness proved above, $\gamma(\mathcal{P}_1 \cap^\alpha \mathcal{P}_2) = \gamma(\mathcal{P}_1) \cap \gamma(\mathsf{rot}_A(\mathcal{P}_2)) \supseteq \gamma(\mathcal{P}_1) \cap \gamma(\mathcal{P}_2)$, i.e., $\cap^\alpha$ is a correct abstraction of $\cap$. □

Proposition 22. *The reduction operator is correct, i.e., given any $P \in \mathsf{Par}$ and $B \in \mathsf{Box}$, we have that $\gamma(\mathsf{red}(\langle P, B\rangle)) = \gamma(\langle P, B\rangle)$.*

Proof. We need to prove that, given any $P \in \mathsf{Par}$ and $B \in \mathsf{Box}$, we have that $\gamma(\mathsf{red}(\langle P, B\rangle)) \supseteq \gamma(P) \cap \gamma(B)$.

First note that, given any $P \in \mathsf{Par}$ and matrix $A \in \mathrm{GL}(n)$, we have that $P \leq \mathsf{rot}_A(P)$. Thus, the weak intersection on parallelotopes correctly approximates the concrete intersection, namely

$$\gamma(P_1 \cap^\alpha P_2) \supseteq \gamma(P_1) \cap \gamma(P_2)$$

and therefore

$$\gamma(\mathsf{red}(\langle P, B\rangle)) \supseteq \gamma(P) \cap \gamma(B)$$

which proves the result. □

Corollary 23. *All the operations on $\mathsf{Par} \sqcap \mathsf{Box}$ defined according to Equation 2 are correct, namely:*

$$\gamma(\langle P_1, B_1\rangle \odot^{P,B} \langle P_2, B_2\rangle) \supseteq \gamma(\langle P_1, B_1\rangle) \odot \gamma(\langle P_2, B_2\rangle)$$

Proof. It immediately follows from Proposition 22. □

References

[1] P. Cousot, N. Halbwachs, Automatic discovery of linear restraints among variables of a program, in: POPL '78: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, ACM Press, New York, NY, USA, 84–97, doi:10.1145/512760.512770, 1978.

[2] P. Cousot, R. Cousot, Static determination of dynamic properties of programs, in: Proceedings of the Second International Symposium on Programming, Dunod, Paris, France, 106–130, 1976.

[3] A. Miné, The Octagon Abstract Domain, Higher-Order and Symbolic Computation 19 (1) (2006) 31–100, ISSN 1388-3690, doi:10.1007/s10990-006-8609-1.

[4] R. Clarisó, J. Cortadella, The octahedron abstract domain, Science of Computer Programming 64 (2007) 115–139, ISSN 0167-6423, doi:10.1016/j.scico.2006.03.009.

[5] J. M. Howe, A. Simon, Logahedra: A New Weakly Relational Domain, in: Z. Liu, A. P. Ravn (Eds.), Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings., vol. 5799 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, ISBN 978-3-642-04760-2, 306–320, doi:10.1007/978-3-642-04761-9_23, 2009.

[6] A. Simon, A. King, J. M. Howe, Two Variables per Linear Inequality as an Abstract Domain, in: M. Leuschel (Ed.), Logic Based Program Synthesis and Transformation 12th International Workshop, LOPSTR 2002, Madrid, Spain, September 17–20, 2002. Revised Selected Papers, vol. 2664 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, ISBN 978-3-540-40438-5, 71–89, doi:doi://10.1007/3-540-45013-0_7, 2003.

[7] S. Sankaranarayanan, H. B. Sipma, Z. Manna, Scalable Analysis of Linear Systems using Mathematical Programming, in: R. Cousot (Ed.), Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005. Proceedings, vol. 3385 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, ISBN 978-3-540-24297-0, 25–41, doi:10.1007/b105073, 2005.

[8] M. Colón, S. Sankaranarayanan, Generalizing the template polyhedral domain, in: G. Barthe (Ed.), Programming Languages and Systems: 20th European Symposium on Programming, ESOP 2011, held as part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011. Proceedings, vol. 6602 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, ISBN 978-3-642-19717-8, 176–195, 2011.

[9] G. Amato, M. Parton, F. Scozzari, Deriving Numerical Abstract Domains via Principal Component Analysis, in: R. Cousot, M. Martel (Eds.), 17th

International Symposium, SAS 2010, Perpignan, France, September 14-16, 2010, Proceedings, vol. 6337 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, 134–150, doi:10.1007/978-3-642-15769-1, 2010.

[10] G. Amato, M. Parton, F. Scozzari, Discovering invariants via Simple Component Analysis, Journal of Symbolic Computation 47 (12) (2012) 1533–1560, doi:10.1016/j.jsc.2011.12.052.

[11] G. Amato, F. Scozzari, The abstract domain of parallelotopes, in: J. Midtgaard, M. Might (Eds.), Proceedings of the Fourth International Workshop on Numerical and Symbolic Abstract Domains, NSAD 2012, vol. 287 of *Electronic Notes in Theoretical Computer Science*, Elsevier, 17–28, doi:10.1016/j.entcs.2012.09.003, 2012.

[12] A. Schrijver, Theory of Linear and Integer Programming, John Wiley & Sons, 1986.

[13] P. Cousot, R. Cousot, Abstract Interpretation Frameworks, Journal of Logic and Computation 2 (4) (1992) 511–549, doi:10.1093/logcom/2.4.511.

[14] S. Sankaranarayanan, M. Colón, H. B. Sipma, Z. Manna, Efficient Strongly Relational Polyhedral Analysis, in: E. A. Emerson, K. S. Namjoshi (Eds.), VMCAI, vol. 3855 of *Lecture Notes in Computer Science*, Springer, ISBN 3-540-31139-4, 111–125, 2006.

[15] P. Cousot, R. Cousot, Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints, in: POPL '77: Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, ACM Press, New York, NY, USA, 238–252, doi:10.1145/512950.512973, 1977.

[16] P. Cousot, Abstract Induction by Extrapolation and Interpolation, in: D. D'Souza, A. Lal, K. G. Larsen (Eds.), Verification, Model Checking, and Abstract Interpretation. 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015, Proceedings, vol. 8931 of *Lecture Notes in Computer Science*, Springer, 19–42, doi:10.1007/978-3-662-46081-8_2, 2015.

[17] G. Amato, F. Scozzari, Jandom, `https://github.com/jandom-devel/Jandom`, 2016.

[18] V. Maisonneuve, O. Hermant, F. Irigoin, ALICe: A framework to improve affine loop invariant computation, in: 5th Workshop on INvariant Generation, 2014.

[19] P. Granger, Improving the Results of Static Analyses Programs by Local Decreasing Iteration, in: R. Shyamasundar (Ed.), Foundations of Software Technology and Theoretical Computer Science. 12th Conference New Delhi, India, December 18–20, 1992 Proceedings, Lecture Notes in Computer Science, Springer, 68–79, doi:10.1007/3-540-56287-7_95, 1992.

[20] R. Bagnara, P. M. Hill, E. Zaffanella, The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems, Science of Computer Programming 72 (1–2) (2008) 3–21, ISSN 0167-6423, doi: 10.1016/j.scico.2007.08.001.

[21] R. Bagnara, P. M. Hill, E. Ricci, E. Zaffanella, Precise widening operators for convex polyhedra, Science of Computer Programming 58 (1) (2005) 28–56, ISSN 0167-6423, doi:10.1016/j.scico.2005.02.003.