Available online at www.sciencedirect.com

SciVerse ScienceDirect

Electronic Notes in Theoretical Computer Science 287 (2012) 17–28

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

The Abstract Domain of Parallelotopes

Gianluca Amato Francesca Scozzari

Università "G. d'Annunzio" di Chieti-Pescara – Italy

Abstract

We propose a numerical abstract domain based on parallelotopes. A parallelotope is a polyhedron whose constraint matrix is squared and invertible. The domain of parallelotopes is a fully relational abstraction of the Cousot and Halbwachs' polyhedra abstract domain, and does not use templates. We equip the domain of parallelotopes with all the necessary operations for the analysis of imperative programs, and show optimality results for the abstract operators.

Keywords: Abstract interpretation, numerical domains, polyhedra, parallelotopes.

1 Introduction

In recent years, we have seen many different proposals of numerical domains derived from Cousot and Halbwachs' [7] polyhedra abstract domain. Weakly relational domains, such as octagons [10] and logahedra [8], have proved to be quite efficient, but the assertions that can be discovered using these domains are limited by many syntactic restrictions. In order to handle more expressive constraints, Sankaranarayanan et al. have proposed a different approach called template polyhedra [12], which is a generalization of most weakly relational domains. For each program, the authors fix *a priori* a constraint matrix A and consider all the polyhedra of the type $A\boldsymbol{x} \leq \boldsymbol{b}$. The choice of the matrix is what 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). Abstract operators on template polyhedra have been defined by means of linear programming and can be computed in polynomial time.

Along the same direction there are the proposals of generalized template polyhedra [5], which combine template polyhedra and bilinear forms, and template parallelotopes [1,3], which are a special case of template polyhedra. A parallelotope is a polyhedron defined by at most n linearly independent constraints, where

[1] Email: amato@sci.unich.it, scozzari@sci.unich.it

doi:10.1016/j.entcs.2012.09.003

n is the number of variables. This amounts to say that the constraint matrix of a parallelotope is squared and invertible. In the special case of template parallelotopes, it is possible to derive efficient abstract operators, without resorting to linear programming tools. While template polyhedra seem to be a valid alternative to polyhedra from the point of view of efficiency, and to weakly relational domains from the point of view of expressivity, a general method to find "good" templates still does not exist. We are aware of only two proposals for generating templates, one in Sankaranarayanan et al. original paper [12] based on a syntactic inspection of the program, and another one based on the statistical analysis of partial execution traces [1,3].

In this paper we try a different approach to balance expressivity and efficiency. Instead of restricting the syntactic form of constraints, we limit the number of constraints to n linearly independent ones. It turns out that our abstract objects are parallelotopes, as in [1,3], but with the fundamental difference that the constraint matrix is not fixed *a priori* but may change freely, as for the polyhedra domain. We provide the basic abstract operators, and show that the domain of parallelotopes can be equipped with very efficient abstract operations, whose complexity is comparable to that of octagons.

When defining the abstract operators, the main problem we need to face is that many concrete operations are not closed with respect to parallelotopes. Moreover, in many cases there is no unique best parallelotope which approximates the result of the concrete operation. Therefore, we need to carefully choose, even resorting to appropriate heuristics, an approximating parallelotope which brings a good precision in the overall analysis.

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 extended to $\bar{\mathbb{R}}$ in the obvious way, with the exception that 0 times $\pm\infty$ is 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. 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.

2.2 Abstract interpretation

In this paper we adopt a framework for abstract interpretation which is weaker than the common one based on Galois connections (see [6, Section 7]). Given two pre-ordered sets $(C, \leq_C)$ — the *concrete domain* — and $(A, \leq_A)$ — the *abstract domain* — we establish an abstraction–concretization relationship between them with the use of a *concretization map*, which is a monotone function $\gamma : C \to A$. 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 possible abstractions. The most interesting ones are those which are minimal w.r.t. the ordering $\leq_A$.

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 $c \leq \gamma(a)$ 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 $a \in A$, there exists no $b \in A$ such that $f(\gamma(a)) \leq_A b <_A f^\alpha(a)$, i.e. when $f^\alpha(a)$ is a minimal correct approximation of $f(\gamma(a))$.

3 Parallelotopes

A set $\mathcal{P} \subseteq \mathbb{R}^n$ is a *parallelotope* [2] when there is a matrix $A \in \mathrm{GL}(n)$ and vectors $\boldsymbol{l}, \boldsymbol{u} \in \bar{\mathbb{R}}^n$ such that

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

A parallelotope is a closed convex set.

Definition 3.1 (Representation of parallelotopes) *A* representation of parallelotopes *is a tuple* $P = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ *such that* $A \in \mathrm{GL}(n)$, $\boldsymbol{l}, \boldsymbol{u} \in \bar{\mathbb{R}}^n$ *and* $\boldsymbol{l} \leq \boldsymbol{u}$. *We denote by* $\mathsf{ParTope}_n$ *the set of all the representations of parallelotopes in* $\mathbb{R}^n$.

The matrix A *is called the* constraint matrix, *while* $\boldsymbol{l}$ *and* $\boldsymbol{u}$ *are the* lower *and* upper *bounds respectively. We denote by* $\gamma(P)$ *the corresponding parallelotope which, according to Eq. 1, is* $\gamma(P) = \{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{l} \leq A\boldsymbol{x} \leq \boldsymbol{u}\}$. *We say that a parallelotope* $\mathcal{P}$ is definable over $A \in \mathrm{GL}(n)$ *if there is a representation for* $\mathcal{P}$ *whose constraint matrix is* A.

For every non-empty parallelotope $\mathcal{P}$ there is a representation P such that $\gamma(P) = \mathcal{P}$. We have ruled out the case $\boldsymbol{l} \not\leq \boldsymbol{u}$ because this would considerably complicate the formalism. When we need a representation for the empty parallelotope, we will use the symbol ϵ, with the proviso that $\gamma(\epsilon) = \emptyset$. However, in most cases we will only consider non-empty parallelotopes.

It is worth noting that there are many different representations which correspond to the same parallelotope. In the following, when this does not cause ambiguities, we will use a representation P in place of the parallelotope $\gamma(P)$, and we will refer to representations of parallelotopes simply as parallelotopes.

[2] In the mathematical literature, a parallelotope is generally considered to be bound, while we are also considering unbounded ones.

3.1 Minimization and maximization over a parallelotope

Given a box[3] $\mathcal{B} = \langle I_n, \boldsymbol{l}, \boldsymbol{u}\rangle$ and a vector $\boldsymbol{c} \in \mathbb{R}^n$, we have that

$$\inf_{\boldsymbol{x}\in\mathcal{B}} \boldsymbol{c}^T\boldsymbol{x} = \inf_{\boldsymbol{l}\leq\boldsymbol{x}\leq\boldsymbol{u}} \boldsymbol{c}^T\boldsymbol{x} = \boldsymbol{c}^T\boldsymbol{v} \quad \text{where} \quad v_i = \begin{cases} l_i & \text{if } c_i \geq 0 \\ u_i & \text{otherwise.} \end{cases} \tag{2}$$

The computational complexity of this operation is $O(n)$. The minimization operator for parallelotopes may be obtained by viewing a parallelotope as a box over a non-canonical coordinate system.

Proposition 3.2 *Given a parallelotope $P = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ and a vector $\boldsymbol{c} \in \mathbb{R}^n$, we have that*

$$\inf_{\boldsymbol{x}\in P} \boldsymbol{c}^T\boldsymbol{x} = \inf_{\boldsymbol{l}\leq\boldsymbol{y}\leq\boldsymbol{u}} \boldsymbol{c}^T A^{-1}\boldsymbol{y}. \tag{3}$$

The computational complexity is $O(n^3)$.

Symmetric properties hold for the maximization operators.

3.2 Approximation by parallelotopes

Given a set $\mathcal{C} \in \mathbb{R}^n$, we are interested in approximating $\mathcal{C}$ with a parallelotope $\mathcal{P} \supseteq \mathcal{C}$. In general, there is not a least parallelotope $\mathcal{P}$ which contains $\mathcal{C}$, but there are several (possibly infinite) minimal parallelotopes with such a property. Among them, the choice may be done according to some heuristic.

Things change if we fix the matrix A and consider only the parallelotopes definable over A, as in [3]. In this case the least parallelotope exists. As a particular case, given a parallelotope $\mathcal{P}$ in $\mathbb{R}^n$ and a matrix $A \in \mathrm{GL}(n)$, it is interesting to seek the least parallelotope containing $\mathcal{P}$ and definable over A.

Definition 3.3 (Approximation operator) *The approximation operator on parallelotopes $\alpha_{\mathrm{A}} : \mathsf{ParTope}_n \to \mathsf{ParTope}_n$ is defined as*

$$\alpha_{\mathrm{A}}(P) \stackrel{\text{def}}{=} \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \qquad \textit{where} \qquad l_i = \inf_{\boldsymbol{x}\in P} \boldsymbol{a}_{i*}\boldsymbol{x} \ , \quad u_i = \sup_{\boldsymbol{x}\in P} \boldsymbol{a}_{i*}\boldsymbol{x} \ . \tag{4}$$

Theorem 3.4 *Given a parallelotope P and $A \in \mathrm{GL}(n)$, $\alpha_{\mathrm{A}}(P)$ is the least parallelotope definable over A which contains P. The computational complexity is $O(n^3)$.*

3.3 Ordering of parallelotopes

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

[3] We represent a box as a parallelotope with the identity constraint matrix.

Definition 3.5 *Given parallelotopes* $P = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ *and* $P' = \langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$*, we define* $P \leq P'$ *iff* $\boldsymbol{l}' \leq \boldsymbol{l}''$ *and* $\boldsymbol{u}'' \leq \boldsymbol{u}'$*, where* $\langle A', \boldsymbol{l}'', \boldsymbol{u}''\rangle = \alpha_{\mathrm{A}'}(P)$.

Theorem 3.6 *Given parallelotopes* P *and* P'*, we have that* $P \leq P'$ *iff* $\gamma(P) \subseteq \gamma(P')$*. Moverover,* $\leq$ *is a pre-order. The computational complexity is* $O(n^3)$.

This theorem 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. The abstract domain is the set $\mathsf{ParTope}_n \cup \{\epsilon\}$, where the ordering over parallelotopes is extended to ϵ in such a way that $\epsilon \leq P$ for each $P \in \mathsf{ParTope}_n$. Finally, we take γ as concretization map.

4 Abstract operators on parallelotopes

We now consider the operations on $\wp(\mathbb{R}^n)$ commonly used when defining the collecting semantics of imperative programming languages, and for each of them we introduce a correct approximation on parallelotopes. Computing operations on parallelotopes essentially amounts to:

(i) choosing a resulting constraint matrix A;

(ii) computing bounds $\boldsymbol{l}$ and $\boldsymbol{u}$ to get a correct approximation of the result.

Once A is fixed, computing the best bounds is quite easy. However, there are generally several possible alternatives for A, which lead to results which are set-theoretically incomparable. From a theoretical perspective, in evaluating the precision of an abstract operator, we will look for the following properties, in order of preference:

(i) γ-*completeness*, if possible, i.e. when the result of the concrete operator is a parallelotope;

(ii) *minimality*, i.e. we compute one of the minimal parallelotopes which approximate the concrete result;

(iii) *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. The choice between competing minimal parallelotopes may only be done under the basis of heuristic considerations, and validation requires extensive tests.

4.1 Invertible linear assignment

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

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

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 [7]. Intuitively, the operation $\mathsf{assign}(i, \boldsymbol{c}, b)$ corresponds to the assignment $x_i' = \boldsymbol{c}^T\boldsymbol{x} + b$, 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 with the above definition we get the solution.

Definition 4.1 *Given $\boldsymbol{c} \in \mathbb{R}^n$ such that $c_i \neq 0$ and $b \in \mathbb{R}$, we define the abstract linear assignment* $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)$ *as*

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

Theorem 4.2 *The operation* $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)$ *is correct and γ-complete. The computational complexity is $O(n^2)$.*

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 $\mathsf{forget}(i) : \wp(\mathbb{R}^n) \to \wp(\mathbb{R}^n)$ defined as

$$\mathsf{forget}(i)(\mathcal{C}) = \{\boldsymbol{x} + \alpha\boldsymbol{e}^i \mid \boldsymbol{x} \in \mathcal{C}, \alpha \in \mathbb{R}\} \ .$$

First note that, even if $\mathcal{P}$ is a parallelotope, $\mathsf{forget}(i)(\mathcal{P})$ may fail to be a parallelotope.

A naive definition of $\mathsf{forget}^\alpha(i)(\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle)$ would replace the bounds of the lines j such that $a_{ji} \neq 0$ with $-\infty$ and $+\infty$. However, this generally yields a gross approximation.

Example 4.3 *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 inequalities, we get the new constraint $0 \leq 2x_1 \leq 0$, which does not contain x_2 and thus is preserved by non-deterministic assignments.*

Therefore, we need to make explicit the constraints hidden in 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 4.4 *Consider the parallelotope 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 $-1 \leq x_1 - x_2 + x_3 \leq 1$. By considering all the pairs of inequalities and simplifying, 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 overcome the above problems, we propose the following operator. First note that we may ignore the rows in P which are unbounded (i.e. with infinite lower and upper bounds) or whose i-th entry is zero: the first remain unbounded, while the second are not affected by the assignment. Thus, in the following, we will 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 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 $R = \{a_{ji}\boldsymbol{a}_{r*} - a_{ri}\boldsymbol{a}_{j*} \mid j \in J \setminus \{r\}\} \cup \{\boldsymbol{a}_{r*}\}$. Since the rows in J are linearly independent, it follows that their linear combinations are still linearly independent, and thus R is a set of $|J|$ linearly independent rows. Moreover, all the rows 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 $P' \geq P$, by computing the new bounds (with the exception of $\boldsymbol{a}_{r*}$ which must become an unbounded 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, since we will get better bounds in P'. In fact, when we choose a row r whose lower and upper bounds coincide, then the $\mathsf{forget}^{\alpha}(i)$ abstract operator is γ-complete. The detailed procedure is shown in Algorithm 1.

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

Require: $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \in \mathsf{ParTope}_n$, $i \in \{1, \ldots, n\}$
$J = \{j \mid a_{ji} \neq 0, l_j \neq -\infty \vee u_j \neq \infty\}$
if $J = \emptyset$ **then**
  return $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$
end if
$\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle \leftarrow \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$
$J_0 \leftarrow \{j \in J \mid l_j = u_j \in \mathbb{R}\}$
$J_1 \leftarrow \{j \in J \mid l_j, u_j \in \mathbb{R}\}$
if $J_0 \neq \emptyset$ **then**
  $r \leftarrow$ an element in J_0
else if $J_1 \neq \emptyset$ **then**
  $r \leftarrow$ an element in J_1
else
  $r \leftarrow$ an element in J
end if
for all $j \in J \setminus \{r\}$ **do**
  $\boldsymbol{a}'_{j*} \leftarrow a_{ji}\boldsymbol{a}_{r*} - a_{ri}\boldsymbol{a}_{j*}$
  $(m_r, M_r) \leftarrow$ **if** $a_{ji} < 0$ **then** (u_r, l_r) **else** (l_r, u_r)
  $(m_j, M_j) \leftarrow$ **if** $-a_{ri} < 0$ **then** (u_j, l_j) **else** (l_j, u_j)
  $l'_j \leftarrow a_{ji}m_r - a_{ri}m_j$
  $u'_j \leftarrow a_{ji}M_r - a_{ri}M_j$
end for
$l'_r \leftarrow -\infty$
$u'_r \leftarrow +\infty$
return $\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$

Theorem 4.5 *The operator $\mathsf{forget}^{\alpha}(i)$ described in Algorithm 1 is correct and minimal. It is γ-complete when $J_0 \neq \emptyset$. The computational complexity is $O(n^2)$.*

4.3 Non-invertible assignment

We consider again 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 discharged after the assignment, possibly replaced by 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.

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

Require: $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \in \mathsf{ParTope}_n$, $i \in \{1, \ldots, n\}$, $\boldsymbol{c} \in \mathbb{R}^n$, $b \in \mathbb{R}$, $c_i = 0$
1: $\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle \leftarrow \mathsf{forget}^\alpha(i)\langle \boldsymbol{A}, \boldsymbol{l}, \boldsymbol{u}\rangle$
2: choose an index j with $\boldsymbol{a}'_{ji} \neq 0$
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: $\boldsymbol{l}'_i \leftarrow b$
8: $\boldsymbol{u}'_i \leftarrow b$
9: **return** $\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle$

The procedure is shown in Algorithm 2. Given a parallelotope P, we first compute $\langle A', \boldsymbol{l}', \boldsymbol{u}'\rangle = \mathsf{forget}^\alpha(i)(P)$ and choose a row j in A' with $a'_{ji} \neq 0$. 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 operate on unbounded rows. Then, we may replace $\boldsymbol{a}'_{j*}$ with $\boldsymbol{e}^i - \boldsymbol{c}$, l_j and u_j with b. Since the j-th row of A is unbounded, we do not lose precision when we replace it. Thanks to the steps 3–5, the final matrix A' is invertible.

Theorem 4.6 *The operator* $\mathsf{assign}^\alpha(i, \boldsymbol{c}, b)$ *described in Algorithm 2 is correct and minimal. The computational complexity is* $O(n^2)$.

4.4 Refinement by linear inequality

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

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

which we will call *linear refinement*. In general, the linear refinement of a parallelotope is not a parallelotope.

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

Require: $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \in \mathsf{ParTope}_n$, $\boldsymbol{c} \in \mathbb{R}^n$, $b \in \mathbb{R}$
1: $\boldsymbol{y} \leftarrow$ solution of $A^T\boldsymbol{y} = c$
2: **if** $\exists j.\ y_j \neq 0 \wedge l_i = -\infty \wedge u_i = +\infty$ **then**
3: $\quad \boldsymbol{a}_{j*} \leftarrow \boldsymbol{c}$
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**

Given a parallelotope $P = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$, we first investigate if there exists an unbounded row of A such that, if we replace that row with $\boldsymbol{c}$, then 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 such that $y_j \neq 0$. When it does not exists, we simply compute the least parallelotope P' containing $\mathsf{refine}(\boldsymbol{c}, b)(P)$ and definable over A. We recall from [3] that, if $\mathsf{refine}(\boldsymbol{y}, b)\langle I_n, \boldsymbol{l}, \boldsymbol{u}\rangle = \langle I_n, \boldsymbol{l}', \boldsymbol{u}'\rangle$, then $P' = \langle A, \boldsymbol{l}', \boldsymbol{u}'\rangle$. Hence, we may use the known refine operator over boxes to define a refine operator over parallelotopes.

Theorem 4.7 *The operator* $\mathsf{refine}^{\alpha}(\boldsymbol{c}, b)$ *described in Algorithm 3 is correct and relatively optimal. The computational complexity is* $O(n^3)$.

4.5 Union

Let us come to the abstract union of parallelotopes. If we fix a matrix M, the least parallelotope definable over M which contains the parallelotopes $P_A = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle$ and $P_B = \langle B, \boldsymbol{j}, \boldsymbol{k}\rangle$ can be easily obtained by $\alpha_{\mathrm{M}}(P_A)$ and $\alpha_{\mathrm{M}}(P_B)$ simply selecting, for each row in M, the *least* lower bound and the *greatest* upper bound. 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.

We now propose a more complex variant of abstract union, inspired by the recently developed *inversion join* operator [11]. 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 lowest and upper bounds on P_A and P_B. We then assign to each constraint a priority.

In general, the candidate linear forms are not linearly independent. At the end, we will select exactly n linearly independent constraints, according to their priorities (where 0 is the highest one).

The priority is chosen according to the values of the bounds. In order of preference, we will select: equality constraints, constraints which are saturated both in P_A and P_B, and so on. This order is mostly dictated by heuristic considerations. Algorithm 4 computes the bounds and assigns the priorities for a given linear form $\boldsymbol{v}$.

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

Require: $P_A, P_B \in \mathsf{ParTope}_n$, $\boldsymbol{v} \in \mathbb{R}^n$
$l_v \leftarrow \inf\{\boldsymbol{x} \in P_A \mid \boldsymbol{v}^T\boldsymbol{x}\}$
$u_v \leftarrow \sup\{\boldsymbol{x} \in P_A \mid \boldsymbol{v}^T\boldsymbol{x}\}$
$j_v \leftarrow \inf\{\boldsymbol{x} \in P_B \mid \boldsymbol{v}^T\boldsymbol{x}\}$
$k_v \leftarrow \sup\{\boldsymbol{x} \in P_B \mid \boldsymbol{v}^T\boldsymbol{x}\}$
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**
  ⋮ {other tests}
else
  $p \leftarrow +\infty$
end if
return $\langle \min(l_v, j_v), \max(u_v, k_v), p\rangle$

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 P_A and/or 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. The complete procedure is illustrated in Algorithm 5.

Algorithm 5 The abstract union operator

Require: $P_A = \langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \in \mathsf{ParTope}_n$, $P_B = \langle B, \boldsymbol{j}, \boldsymbol{k}\rangle \in \mathsf{ParTope}_n$
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 v_1 and v_2 form an inversion}
9: $h_1 \leftarrow \inf\{\boldsymbol{x} \in P_A \mid \boldsymbol{v}_1\boldsymbol{x}\}$
10: $h_2 \leftarrow \inf\{\boldsymbol{x} \in P_A \mid \boldsymbol{v}_2\boldsymbol{x}\}$
11: $i_1 \leftarrow \inf\{\boldsymbol{x} \in P_B \mid \boldsymbol{v}_1\boldsymbol{x}\}$
12: $i_2 \leftarrow \inf\{\boldsymbol{x} \in P_B \mid \boldsymbol{v}_2\boldsymbol{x}\}$
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 v_1 and 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 C 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 v_1 and upper bounds for v_2
22: same proc. of lines 7–18 applied to upper bounds for v_1 and lower bounds for 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 C 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** $\langle R, \boldsymbol{l}', \boldsymbol{u}'\rangle$

Theorem 4.8 *The abstract union operator described in Algorithm 5 is correct and relatively optimal. The computational complexity is $O(n^4)$.*

4.6 Widening

Given two parallelotopes P_A and P_B, we first compute $\alpha_{\mathrm{A}}(P_B) = \langle A, \boldsymbol{j}', \boldsymbol{k}'\rangle$ and then apply, separately for each row in A, a standard widening which extrapolates unstable bounds to infinity. We define $\langle A, \boldsymbol{l}, \boldsymbol{u}\rangle \nabla \langle B, \boldsymbol{j}, \boldsymbol{k}\rangle = \langle A, \boldsymbol{l}', \boldsymbol{u}'\rangle$ where, for each $i \in \{1, \ldots, n\}$, we have that:

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

We will combine this widening operator with delayed widening, to ensure that the union operator is initially applied, and new constraints can be generated.

5 Experiments and conclusions

We show some simple examples to give a rough idea of the potentialities and limits of the new domain, using our implementation of parallelotopes in `RANDOM` [4,2]. Consider the program `cousot78` in Figure 1, taken from [7]. The parallelotope

```
cousot78 = function()
{
  i = 2
  j = 0
  while (TRUE) {
    if (i*i==4)
      i = i+4
    else {
      j = j+1
      i = i+2
} } }
```

```
karr76 = function(k)
{
  i = 2
  j = k+5
  while (TRUE) {
        i = i+1
        j = j+3
  }
}
```

```
absval = function (x) {
  assume(x>=-100)
  assume(x <=100)
  y = x
  if (y<=0) y = -y
  if (y<=69) y = y
}
```

Fig. 1. Example programs

domain is able to prove that, in the last line of the `while`, the inequalities $i+2j \geq 6$ and $j \geq 0$ hold. However, since it is not able to represent more than two constraints, it cannot prove that $2j - i \leq -2$. For the program `karr76`, taken from [9], it can prove the invariants $3i - j + k = 1$ and $i \geq 2$. Finally, the program `absval`, taken from [10], is an example where our domain performs poorly, since it cannot prove that, inside the last `if`, x is between -69 and 69.

While the theoretical work is mostly complete, and we also got interesting and non-trivial minimality results, more work should be devoted to evaluate the domain in practice and to study its combination with other numerical domains. Encouraged by the results of our early experiments, we plan to conduct an extensive test of the parallelotope domain, in order to improve the heuristics (especially the choice of priorities in Algorithm 4, which seems of utter importance to achieve good precision), and to better understand its weak points. In some cases, the limit on the number of constraints appears too restrictive. For instance, in the `cousot78` example we were not able to prove that $i \geq 2$, essentially because we are already using two constraints to prove $2i - j \leq -2$ and $j \geq 0$. We strongly believe that the domain of parallelotopes could benefit from being coupled with intervals or with a weakly relational domain, such as octagons, to keep track of additional constraints, while remaining fully relational.

References

[1] Amato, G., M. Parton and F. Scozzari, *Deriving numerical abstract domains via principal component analysis*, *SAS 2010, Proceedings*, LNCS **6337**, Springer, 2010 pp. 134–150.

[2] Amato, G., M. Parton and F. Scozzari, *A tool which mines partial execution traces to improve static analysis*, *RV 2010. Proceedings*, LNCS **6418**, Springer, 2010 pp. 475–479.

[3] Amato, G., M. Parton and F. Scozzari, *Discovering invariants via simple component analysis*, Journal of Symbolic Computation **47** (2012).

[4] Amato, G. and F. Scozzari, *Random: R-based Analyzer for Numerical DOMains*, *LPAR-18, 2012. Proceedings*, LNCS **7180**, Springer, 2012 pp. 375–382.

[5] Colón, M. and S. Sankaranarayanan, *Generalizing the template polyhedral domain*, in *ESOP 2011, Proceedings* , LNCS **6602**, Springer 2011 pp. 176–195.

[6] Cousot, P. and R. Cousot, *Abstract interpretation frameworks*, Journal of Logic and Computation **2** (1992), pp. 511–549.

[7] Cousot, P. and N. Halbwachs, *Automatic discovery of linear restraints among variables of a program*, in: *POPL '78, Proceedings* (1978), pp. 84–97.

[8] Howe, J. M. and A. Simon, *Logahedra: A new weakly relational domain*, in *ATVA 2009, Proceedings.*, LNCS **5799**, Springer, 2009 pp. 306–320.

[9] Karr, M., *Affine relationships among variables of a program*, Acta Informatica **6** (1976), pp. 133–151.

[10] Miné, A., *The octagon abstract domain*, Higher-Order and Symbolic Computation **19** (2006), pp. 31–100.

[11] Sankaranarayanan, S., M. Colón, H. B. Sipma and Z. Manna, *Efficient strongly relational polyhedral analysis*, in *VMCAI, Proceedings.*, LNCS **3855** (2006), pp. 111–125.

[12] Sankaranarayanan, S., H. B. Sipma and Z. Manna, *Scalable analysis of linear systems using mathematical programming*, in *VMCAI, Proceedings.*, LNCS **3385**, Springer, 2005 pp. 25–41.