A Note on the Inversion Join for Polyhedral Analysis

Axel Simon[1]

Lehrstuhl 2 für Informatik, Technical University Munich, 85478 Garching, Germany

Abstract

Linear invariants are essential in many optimization and verification tasks. The domain of convex polyhedra (sets of linear inequalities) has the potential to infer all linear relationships. Yet, it is rarely applied to larger problems due to the join operation whose most precise result is given by the convex hull of two polyhedra which, in turn, may be of exponential size. Recently, Sankaranarayanan et al. proposed an operation called inversion join to efficiently approximate the convex hull. While their proposal has an ad-hoc flavour, we show that it is quite principled and, indeed, complete for planar polyhedra and, for general polyhedra, complete on over 70% of our benchmarks.

Keywords: abstract interpretation, polyhedra analysis, convex hull

1 Introduction

More than three decades ago, Cousot and Halbwachs proposed the lattice of convex polyhedra to infer linear relationships between program variables [4]. While approximating assignments and tests can straightforwardly be implemented using simple manipulations of inequality sets, the join of two abstract states cannot. Indeed, the most precise join operation is the convex hull of the two input polyhedra which might result in an output polyhedron whose inequality set is exponentially larger than the two inputs. In the past, many so-called *weakly-relational* domains have been suggested that restrict inequalities to a certain form for which more efficient join operations exist. Examples include the octagon domain [9], the two-variable-per-inequality (TVPI) domain [16] or, more recently, the logahedra domain [5]. However, many practical tasks require that weakly-relational domains are combined with other

[1] This work was supported by the Emmy Noether Programme SI 1579/1 and the INRIA project "Abstraction" of jointly funded by CNRS and ENS.

domains to achieve the required precision. For instance, the octagon domain was augmented with a domain tracking symbolic expressions [10] to achieve more precision. In contrast, general polyhedra subsume affine constraints of the form $\boldsymbol{a} \cdot \boldsymbol{x} = c$ where $\boldsymbol{a} \in \mathbb{R}^n$, $c \in \mathbb{R}$, enabling them to symbolically track any linear expression assigned to a variable. Furthermore, when the variables $\boldsymbol{x}$ are known to be integral, congruences can be recovered by observing equalities such as $4x = y$; a constraint that is not expressible in the octagon domain. When furthermore using simple integer tightening methods, disjunctive information can be stored using binary variables [14] which can otherwise only be expressed by tracking several states per program location [8]. Since polyhedra can subsume many of these simpler domains, they are attractive as a one-stop solution.

The original join $\sqcup$ for polyhedra proposed in [4] calculates (the topological closure of) the convex hull $P_1 \overline{\curlyvee} P_2$ of two polyhedra P_1, P_2 which is equivalent to the smallest polyhedron that contains P_1 and P_2. When considering the join operation as just another transfer function that the static analyzer evaluates, setting $\sqcup \equiv \overline{\curlyvee}$ means that the join operation $\sqcup$ is a complete transfer function [3] in that it always returns the most precise polyhedron. Approximating the join operation has already been proposed in [15] who re-formulate the convex hull problem as a projection problem which can be approximated when the output inequality set becomes too large. In [12], Sankaranarayanan et al. propose a so-called inversion join that approximates the join by linear combinations of k inequalities taken from conjoined inequalities of the two input systems. The authors only give an implementation for $k = 2$, resulting in a cubic number of output constraints from which redundant inequalities have to be removed. While the algorithm seems to be ad-hoc, it performs surprisingly well in terms of precision. We show that it is complete for planar polyhedra. Furthermore, we present an empirical evaluation which shows that the output of their algorithm coincides with the convex hull in 73% of all cases. In the remaining cases, the output is slightly larger than the convex hull. Thus the algorithm avoids exponentially-sized outputs at the cost of some precision.

In summary, this paper presents the following, previously unappreciated properties of the inversion join, namely:

- we show that the inversion join corresponds to the convex hull for planar polyhedra;
- we demonstrate that, in the application of program analysis, it is complete in over 70% of all cases;
- we show that many of the produced redundancies can be avoided in practice.

Section 2 introduces required notation before Sect. 3 presents the inversion join and the completeness result for planar polyhedra. Section 4 presents measurements before Sect. 5 concludes.

2 Preliminaries

Let the analyzer express numeric constraints over a set of variables $\mathcal{X}$ and let $\boldsymbol{x}$ denote the vector of all variables in $\mathcal{X}$. Let Lin_n denote the set of linear expressions of the form $\boldsymbol{a} \cdot \boldsymbol{x}$ where $\boldsymbol{a} \in \mathbb{Z}^n$ and $n = |\mathcal{X}|$. Let $Ineq_n$ denote the set of linear inequalities $\boldsymbol{a} \cdot \boldsymbol{x} \leq c$ where $c \in \mathbb{Q}$. For simplicity, let e.g. $3 \leq x_2 \leq 4$ abbreviate the two inequalities $x_2 \leq 4$ and $x_2 \geq 3$, the latter being an abbreviation of $-x_2 \leq -3$. Each inequality $\boldsymbol{a} \cdot \boldsymbol{x} \leq c \in Ineq_n$ induces a half-space $[\![\boldsymbol{a} \cdot \boldsymbol{x} \leq c]\!] = \{\boldsymbol{x} \in \mathbb{Q}^n \mid \boldsymbol{a} \cdot \boldsymbol{x} \leq c\}$. Let $[\![I]\!] = \bigcap_{\iota \in I} [\![\iota]\!]$ and $Poly_n = \{[\![I]\!] \mid I \subseteq Ineq_n \wedge |I| \in \mathbb{N}\}$ the set of (finitely generated) convex polyhedra. Polyhedra form a lattice $\langle Poly_n, \subseteq, \curlyvee, \cap \rangle$ where $P_1 \curlyvee P_2$ is the (topological closure of) the convex hull of P_1 and P_2 [4] which can be defined as $P_1 \curlyvee P_2 = cl(\{\boldsymbol{x} \in \mathbb{Q}^n \mid \boldsymbol{x} = (1-\lambda)\boldsymbol{x}_1 + \lambda \boldsymbol{x}_2 \wedge 0 \leq \lambda \leq 1 \wedge \boldsymbol{x}_1 \in P_1 \wedge \boldsymbol{x}_2 \in P_2\})$ where $cl(\cdot)$ denotes the closure. An actual analyzer uses the lattice of sets of inequalities $\langle \mathcal{P}(Ineq_n), \sqsubseteq, \sqcup, \sqcap \rangle$. Here $I_1 \sqsubseteq I_2$ iff $[\![I_1]\!] \subseteq [\![\iota]\!]$ for all $\iota \in I_2$ which can be tested as follows: let $c = maxExp(\boldsymbol{a} \cdot \boldsymbol{x}, I)$ be the result of a linear program where $c \in \mathbb{Q}$ is the maximum that the expression $\boldsymbol{a} \cdot \boldsymbol{x} \in Lin_n$ can take on in the polyhedron $[\![I]\!]$ and set $c = \infty$ if no such maximum exists. Then $[\![I_1]\!] \subseteq [\![\boldsymbol{a} \cdot \boldsymbol{x} \leq c]\!]$ iff $c' = maxExp(\boldsymbol{a} \cdot \boldsymbol{x}, I_1) \neq \infty$ and $c' \leq c$. The meet is defined as $I_1 \sqcap I_2 = \textsc{Remove-Redundant}(I_1 \cup I_2)$ where $\textsc{Remove-Redundant}(I)$ removes redundant inequalities, that is, inequalities $\iota \in I$ for which $I \setminus \{\iota\} \sqsubseteq I$ holds. This reduces to testing if $I \setminus \{\iota\} \sqsubseteq \{\iota\}$ using *maxExp*.

3 The Inversion Join

The inversion join originated in the template method [13] that was proposed to infer a constant vector $\boldsymbol{c}$ to a matrix A such that $A\boldsymbol{x}^{tr} \leq \boldsymbol{c}^{tr}$ where $\boldsymbol{x}^{tr}$ denotes a column vector corresponding to the row vector $\boldsymbol{x}$. The disadvantage of the template method is that A is fixed and has to be given by the user who employs the analysis. The inversion join was meant to calculate new rows that could constitute useful invariants. However, since it infers new inequalities from two (possibly different) template systems, it is suitable to calculate a join of two arbitrary inequality systems. The result is, however, only an approximation to the convex hull of the state represented by the two inequality systems since not all inequalities will be found.

The original presentation of the inversion join is cast in terms of conjoining the inequalities of the two input systems A and B into one set $T = A \cup B$ and to calculate the minimal constant c' of each inequality $\iota \equiv \boldsymbol{a} \cdot \boldsymbol{x} \leq c \in T$ such that $A \sqsubseteq \{\iota'\}$ and $B \sqsubseteq \{\iota'\}$ where $\iota' \equiv \boldsymbol{a} \cdot \boldsymbol{x} \leq c'$. Note that the constant c' can be inferred by calculating $c' = \max(maxExp(\boldsymbol{a} \cdot \boldsymbol{x}, [\![A]\!]), maxExp(\boldsymbol{a} \cdot \boldsymbol{x}, [\![B]\!]))$ iff both maximums exist. The inversion join is then defined as a special case of a so-called restricted join that calculates the convex hull on subsets $\psi_1, \psi_2 \subseteq T$

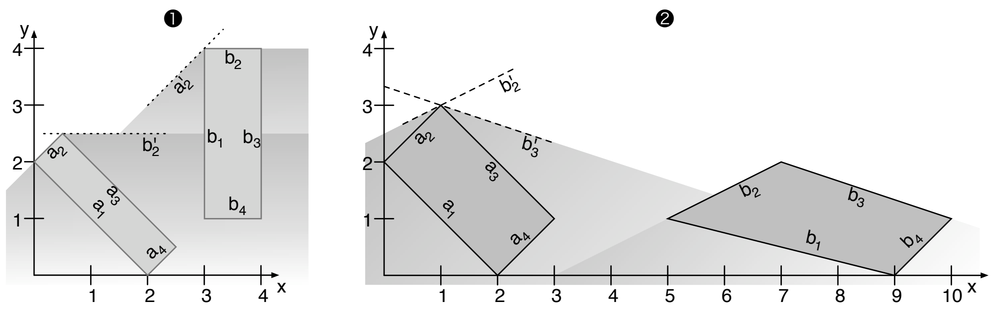

Fig. 1. Choosing two inequalities from $T = \{a_1, \ldots a_4, b_1, \ldots b_4\}$.

of at most size k. Specifically, the inversion join picks exactly two inequalities $\boldsymbol{a}_i \cdot \boldsymbol{x} \leq c_i, \boldsymbol{a}_j \cdot \boldsymbol{x} \leq c_j \in T$ such that ψ_1, ψ_2 take on the following form:

$$\psi_1 = \{\boldsymbol{a}_i \cdot \boldsymbol{x} \leq c_i^A, \boldsymbol{a}_j \cdot \boldsymbol{x} \leq c_j^A\}$$
$$\psi_2 = \{\boldsymbol{a}_i \cdot \boldsymbol{x} \leq c_i^B, \boldsymbol{a}_j \cdot \boldsymbol{x} \leq c_j^B\}$$

Here, the constants $c_i^A, c_j^A, c_i^B, c_j^B$ are chosen such that $A \sqsubseteq \psi_1$ and $B \sqsubseteq \psi_2$, that is, $c_i^A = \mathit{maxExp}(\boldsymbol{a}_i \cdot \boldsymbol{x}, A)$ and analogously for c_j^A, c_i^B, c_j^B. An inversion exists if the coefficients $\boldsymbol{a}_i$ and $\boldsymbol{a}_j$ are linearly independent and $c_i^A < c_j^A \wedge c_i^B > c_j^B$ or $c_i^A > c_j^A \wedge c_i^B < c_j^B$. Two of the possible configurations that satisfy these conditions are depicted in Fig. 1 which shows two polyhedra defined by $A = \{a_1, \ldots a_4\}$ and $B = \{b_1, \ldots b_4\}$. In Diagram ❶, the inequalities $a_2, b_2 \in A \cup B = T$ are chosen. The relaxed inequality $a_2' \equiv \boldsymbol{a}_2 \cdot \boldsymbol{x} \leq c'$ is inferred from $a_2 \equiv \boldsymbol{a}_2 \cdot \boldsymbol{x} \leq c$ by calculating $c' = \mathit{maxExp}(\boldsymbol{a}_2 \cdot \boldsymbol{x}, [\![B]\!])$ and analogously for b_2'. We combine one inequality from each input polyhedron, thus $\psi_1 = \{a_2, b_2'\}$ and $\psi_2 = \{a_2', b_2\}$. The spaces $[\![\psi_1]\!]$ and $[\![\psi_2]\!]$ are depicted as two areas that extend towards infinity. The idea of the inversion join is to calculate a new inequality that connects the two tips (vertices) of these areas. Diagram ❷ shows two similar areas, except that these are defined by $\psi_1 = \{b_2', b_3'\}$ and $\psi_2 = \{b_2, b_3\}$, that is, both relaxed inequalities reside in ψ_1 while the original inequalities reside in ψ_2. In particular, the combined inequalities are both taken from one input set, here B. A symmetric example can be constructed in which both inequalities are taken from A. Thus, the inversion join combines inequalities in two principle modes: the *bilateral* mode, in which an inequality form A is combined with an inequality from B; and the *unilateral* mode, in which the inequalities that are combined both stem from either A or B. For each of these modes, we now discuss how a new inequality is calculated that connects the vertex of the area $[\![\psi_1]\!]$ with the vertex of $[\![\psi_2]\!]$.

3.0.1 Calculating a New Inequality

We commence by presenting a problem that only requires bilateral combinations, that is, the combination of an inequality in A with one in B. The input

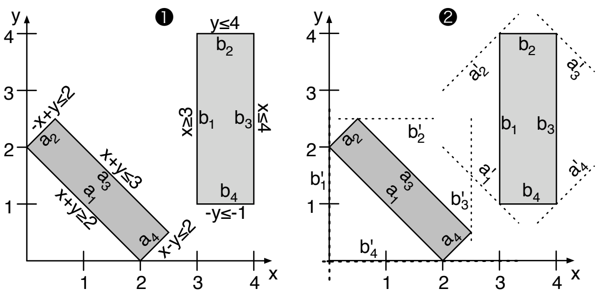

Fig. 2. Maximizing a_i in $[\![B]\!]$ and vice-versa.

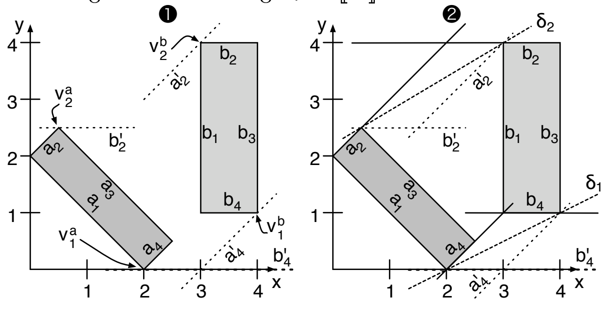

Fig. 3. Calculating weighted combinations.

polyhedra $A = \{a_1, \dots a_4\}$ and $B = \{b_1, \dots b_4\}$ where $a_i \equiv \boldsymbol{a}_i \cdot \boldsymbol{x} \leq c^A_i$ and $b_i \equiv \boldsymbol{b}_i \cdot \boldsymbol{x} \leq c^B_i$ are shown in ❶ of Fig. 2. Using linear programming we infer $c'^A_i = \mathit{maxExp}(\boldsymbol{a}_i \cdot \boldsymbol{x}, B)$ and $c'^B_i = \mathit{maxExp}(\boldsymbol{b}_i \cdot \boldsymbol{x}, A)$. Diagram ❷ of Fig 2 shows the resulting inequalities $a'_1, \dots a'_4$ and $b'_1, \dots b'_4$ as dotted lines.

In order to check if the inequality sets $\psi_1 = \{a_i, b'_j\}$ and $\psi_2 = \{a'_i, b_j\}$ form an inversion, we calculate the amount an inequality a_i and b_i was shifted, by calculating $\delta^a_i := c^A_i - c'^A_i$ and $\delta^b_i := c^B_i - c'^B_i$ for $i = 1, \dots 4$. Figure 3 shows how this information can be used to calculate inequalities that describe the convex hull of $[\![A]\!]$ and $[\![B]\!]$: Consider the relaxed inequalities a'_4 and b'_4 that are depicted in diagram ❶. The task is to find an inequality that connects the two shown vertices v^a_1 and v^b_1. Diagram ❷ shows that these two vertices lie on opposite corners of the parallelogram spanned by a_4, a'_4 and b_4, b'_4. The diagonal is the sum of its two sides (that is, inequalities) whose length is given by δ^a_2 and δ^b_2. In order to obtain an inequality that pivots in the upper right corner of this parallelogram, we calculate a linear combination of a'_4 and b_4. Given that both $\delta^a_4 > 0$ and $\delta^b_2 > 0$, the following weighted combination of a_4 and b'_4 includes both $[\![A]\!]$ and $[\![B]\!]$ and is saturated by v^a_1 and v^b_1:

$$(\delta^b_j \boldsymbol{a}_i + \delta^a_i \boldsymbol{b}_j) \cdot \boldsymbol{x} \leq (\delta^b_j c^A_i + \delta^a_i c'^B_j). \tag{1}$$

A proof of this claim is straightforward and can be found in [12]. In general, δ^a_i and δ^b_j may be negative as, for example, in the case of a_2 and b_2. In this case, the following generalized inequality holds:

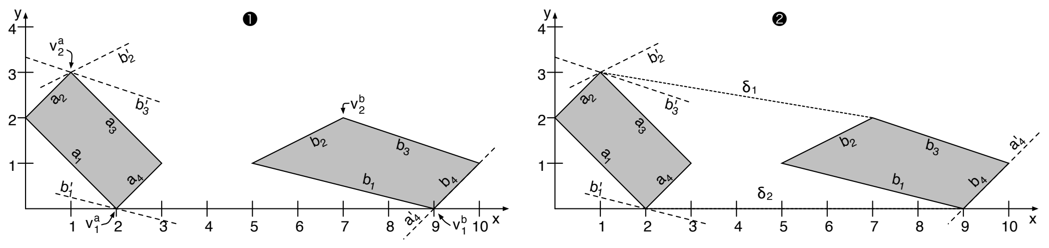

Fig. 4. Combining inequalities from just one polyhedron.

$$(|\delta_j^b|\boldsymbol{a}_i + |\delta_i^a|\boldsymbol{b}_j) \cdot \boldsymbol{x} \leq (|\delta_j^b|c_i^A + |\delta_i^a|c'^B_j) \tag{2}$$

Note that this inequality subsumes the previous one. The resulting inequalities δ_1 and δ_2 are shown in Diagram ❷ of Fig. 3.

An example for a unilateral combination is shown in Fig. 4. Here, diagram ❶ depicts two systems in which $b_2, b_3 \in B$ chosen from $T = A \cup B$. Thus, $\psi_1 = \{b_2', b_3'\}$ and $\psi_2 = \{b_2, b_3\}$. Again, we use an inequality similar to (1) to calculate a weighted, linear combination, this time from $\boldsymbol{b}_2'$ and $\boldsymbol{b}_3$:

$$(|\delta_j^b|\boldsymbol{b}_i + |\delta_i^b|\boldsymbol{b}_j) \cdot \boldsymbol{x} \leq (|\delta_j^b|c_i^B + |\delta_i^b|c'^B_j) \tag{3}$$

The resulting new inequality is dubbed δ_1 in Diagram ❷. We now specify how to calculate the inversion join in the general case which leads us to a comment on how to reduce the number of inequality pairs that are considered.

3.1 Notes on the Implementation

The algorithm for the inversion join can be specified as follows:

(i) For each $\iota \in A \cup B$, calculate ι' and δ_ι using Simplex.

(ii) Define the set of stable constraints as $S = \{\iota \in A \cup B \mid \delta_\iota \leq 0\}$.

(iii) For each $a_i \equiv \boldsymbol{a}_i \cdot \boldsymbol{x} \leq c_i \in A \cup B$ and for each $a_j \equiv \boldsymbol{a}_j \cdot \boldsymbol{x} \leq c_j \in A \cup B$ check that $\boldsymbol{a}_i$ is linearly independent of $\boldsymbol{a}_j$. Add the weighted combination $(|\delta_j|\boldsymbol{a}_i + |\delta_i|\boldsymbol{a}_j) \cdot \boldsymbol{x} \leq (|\delta_j|c_i + |\delta_i|c_j')$ to the result set R if
- $\delta_i\delta_j < 0$ and $\{a_i, a_j\} \subseteq A$ or $\{a_i, a_j\} \subseteq B$ or
- $\delta_i\delta_j > 0$ and $a_i \in A \wedge a_j \in B$.

(iv) Calculate the *weak* join $W = \{\iota' \mid \iota \in A \cup B \wedge \delta_\iota > 0\}$, see [12].

(v) Return Remove-Redundant$(S \cup R \cup W)$.

In step (iii), the inversion join considers $O(|A||B|)$ weighted combinations. Many of these are redundant and have to be removed using Remove-Redundant. Many combinations produce duplicate inequalities that can be identified as redundant by storing inequalities appropriately. During the analysis of loops, many inequalities remain unchanged which can be exploited to reduce the number of Simplex runs. Let $I_C = A \cap B$ denote the common inequalities of A and B. Obviously, for each inequality $\boldsymbol{a} \cdot \boldsymbol{x} \leq c \in A \cup B$ it holds that $\mathit{maxExp}(\boldsymbol{a} \cdot \boldsymbol{x}, A) = \mathit{maxExp}(\boldsymbol{a} \cdot \boldsymbol{x}, B) = 0$ and hence the displacement δ is zero. No sensible combination can be calculated using this inequality since the result is always the inequality itself. Indeed, a facet that is present in both input polyhedra is also present in the output. Thus, it is prudent to identify I_C and avoid calculating *maxExp* for these. The set I_C can be maximised by putting each polyhedron into a normal form which can be obtained by factoring out all equalities from the inequality systems [6].

3.2 Completeness of the Inversion Join for Planar Polyhedra

We now show that the inversion join that combines two inequalities from the input system $A \cup B$ is complete in the two dimensional case, that is, it delivers a result that is as precise as possible or, equivalently, that corresponds to the convex hull of the two input polyhedra. We commence by assuming that both input systems specify bounded polyhedra, that is, each polyhedron can be represented by a set of vertices without rays or lines.

Suppose $A, B \subseteq \mathit{Ineq}_n$ be non-redundant inputs to the inversion join and set $I = S \cup R$ as defined in step (ii) and (iii) of the algorithm. For the sake of a contradiction, assume there exists an inequality ι such that $P = [\![A]\!] \curlyvee [\![B]\!] \subseteq [\![\iota]\!]$ but $A \sqcup B \not\sqsubseteq \{\iota\}$. Since P is bounded and planar, ι connects two vertices (extreme points) v_1, v_2 of P. These vertices must have been vertices in $[\![A]\!]$ or $[\![B]\!]$. Suppose that both, v_1 and v_2, are vertices in $[\![A]\!]$. Since A is non-redundant, there exists an inequality $a \in A$ that connects these vertices. In this case $B \sqsubseteq \{a\}$ as otherwise the convex hull P would not include $[\![B]\!]$. Thus, $[\![a]\!] = [\![\iota]\!]$ and $\delta_a = 0$. Therefore $\iota \in S$.

We can thus suppose that, without loss of generality, v_1 is a vertex in $[\![A]\!]$ and v_2 is a vertex in $[\![B]\!]$. For each vertex v in a planar polyhedron, there exist two inequalities that define it. Let $a_1, a_2 \in A$ define v_1 and let $b_1, b_2 \in B$ define v_2. We assume that a_1 (b_1) can be rotated clockwise around v_1 (v_2) until it coincides with a_2 (b_2). Then there exists a clockwise ordering of $\{a_1, b_1\}$, followed by ι, followed by $\{a_2, b_2\}$. We consider the different cases:

- (i) $a_1, b_1, \iota, a_2, b_2$: Since b_1 lies angle-wise between a_1 and ι, v_1 saturates b_1'. Since a_2 lies angle-wise between ι and b_2, v_2 saturates a_2'. Thus, the weighted bilateral combination of b_1 and a_2 would connect v_1 with v_2 and coincide with ι which is a contradiction.
- (ii) $a_1, b_1, \iota, b_2, a_2$: As above, except that a unilateral combination is created using b_1 and b_2' (or b_1' and b_2).
- (iii) $b_1, a_1, \iota, b_2, a_2$: Analogous to case (i).

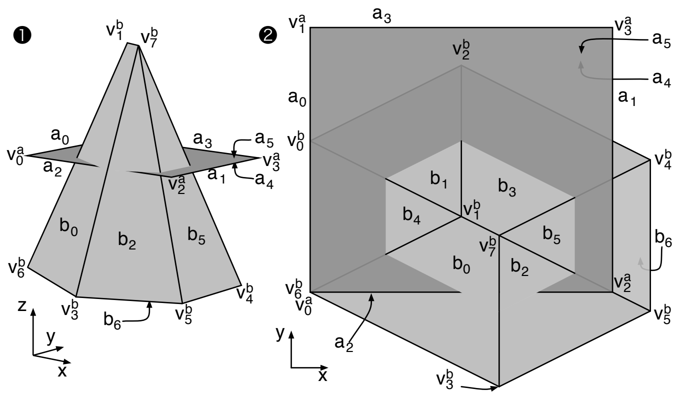

Fig. 5. Polyhedron A, depicted as dark rectangle, is to be joined with pyramid B.

(iv) $b_1, a_1, \iota, a_2, b_2$: Analogous to case (ii).

Thus, for bounded polyhedra, $P = [\![S \cup R]\!]$. For unbounded polyhedra the inequalities ι_i' do occasionally not exist, namely when linear programming returns infinity. In this case, one or both polyhedra contain at least one inequality that defines a ray or a line. In case of a ray, the inequality $a_i \equiv \boldsymbol{a}_i \cdot \boldsymbol{x} \leq c_i \in A$ is saturated by one one vertex in $[\![A]\!]$, in case of a line a is saturated by no vertex. In case of a ray, the correct set of inequalities are generated to connect the vertex with the remaining vertices, as per the argument above. What remains to add to the output is an inequality that defines the ray. If a_i' exists, that is, if $maxExp(\boldsymbol{a}_i \cdot \boldsymbol{x}, B) \neq \infty$, then a_i' itself is such an inequality. Indeed, these inequalities are added through the calculation of W, the so-called *weak* join. The inequality set $W \cup S \cup R$ therefore defines the convex hull of the two planar input polyhedra A and B.

As a consequence for the implementation, if in step (i) of the algorithm each inequality ι has a corresponding ι' then the inclusion of W in the final result is not necessary as they do not restrict the output space any further. Note that this is only true for the two-dimensional case.

3.3 Incompleteness of the Inversion Join for General Polyhedra

While the algorithm above is complete for planar polyhedra, it remains an approximation to the convex hull of polyhedra of higher dimensions. In order to illustrate this, consider the task of joining the two polyhedra in Fig. 5. Figures 6 and 7 show how the inversion join finds two facets of the actual convex hull. In both figures, we use a_i' to denote the inequality a_i in which the constant has to be modified to the maximum in the polyhedron B and vice-versa for b_i' and b_i and polyhedron A.

In order to illustrate how the inversion misses inequalities that are required to define the exact convex hull of the input polyhedra, consider Fig. 8.

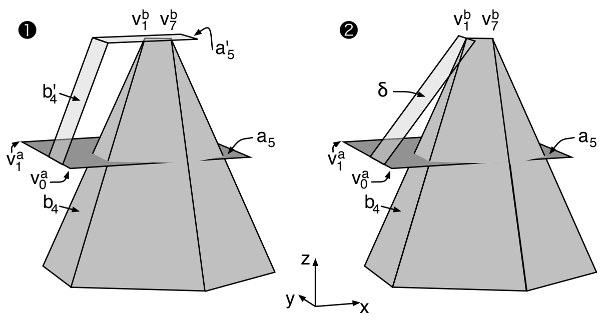

Fig. 6. The relaxed constraints a'_5 and b'_4 are combined to the constraint δ.

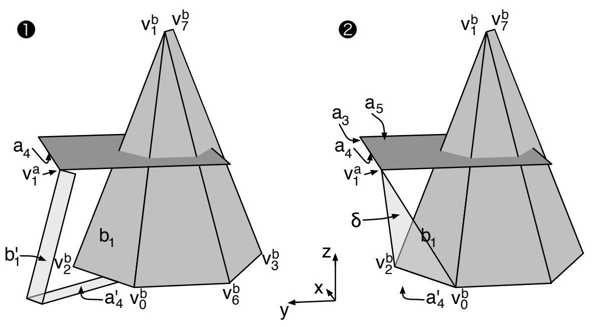

Fig. 7. A weighted combination of inequalities a_4 and b_1.

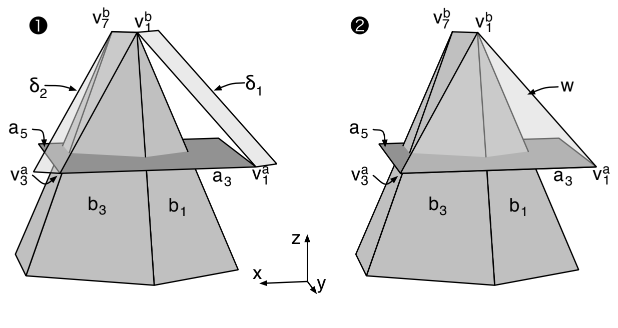

Fig. 8. The weighted combinations of a_5, b_1 and a_5, b_3 do not form a facet of the convex hull. The facet w would be the most precise constraint but cannot be found by combining two inequalities.

Formally, the task of calculating the convex hull of two polyhedra amounts to finding facets that connect a single vertex $\boldsymbol{v}$ of one polyhedron with a so-called horizon ridge of the other polyhedron. A ridge is the intersection of two adjacent facets, which, in turn, can be seen as the intersection of the boundaries of two inequalities ($[\![\boldsymbol{a} \cdot \boldsymbol{x} = c_a]\!] \cap [\![\boldsymbol{b} \cdot \boldsymbol{x} = c_b]\!]$ for inequalities $\boldsymbol{a} \cdot \boldsymbol{x} \leq c_a$ and $\boldsymbol{b} \cdot \boldsymbol{x} \leq c_b$). A horizon ridge is formed by two inequalities of which one is "visible" from the vertex $\boldsymbol{v}$ while the other one is obscured, that is, $\boldsymbol{a} \cdot \boldsymbol{v} \not\leq c_a$ indicates that the first facet is visible while $\boldsymbol{b} \cdot \boldsymbol{v} \leq c_b$ indicates that the second is obscured. The challenge in calculating the convex hull lies in finding

dims.	total # of	incomplete join		off by > 1 ineq.		off by > 2 ineq.	
(vars)	test cases	# of cases	in %	# of cases	in %	# of cases	in %
2	848	0	0	0	0	0	0
3	574	62	11	9	2	4	1
4	340	60	18	25	7	7	2
5–6	586	167	28	104	18	64	11
7–8	344	134	39	87	25	77	22
9–11	296	189	64	161	54	143	48
12–17	225	134	60	123	55	112	50
18–32	238	175	74	168	71	159	67
≥ 33	41	17	41	17	41	15	37
≥ 2	3492	939	27	694	20	581	17

Table 1
Performance of the join algorithm on a 2.4GHz Core 2 Linux computer.

ridges of a polyhedron. The inversion join combines two inequalities and optimistically assumes that the result touches a ridge in one of the polyhedra. In the case of b_1 in ❶ of Fig 8, this assumption is wrong: In the diagram, the inequalities a_5 and b_1 are combined. While b_1 forms a ridge with b_3, b_4 and b_6; and a_5 forms a ridge with $a_0, \ldots a_3$, the resulting inequality δ_1 touches none of these ridges. Indeed, it connects a vertex to a vertex and is therefore redundant in the actual convex hull. On the contrary, the inversion join will miss opportunities to connect certain ridges in a polyhedron to a vertex in the other polyhedron. This is illustrated in Diagram ❷ of Fig. 8. Here, the ridge formed by a_3, a_5 could be connected to the vertex v^b_1. However, since a'_3 touches the vertex v^b_2 rather than vertex v^b_1, this opportunity is missed.

We now present our prototype implementation and its evaluation.

4 Evaluation

The major challenge in implementing the inversion join in a sound way is the use of the Simplex solver. Since off-the-shelf solvers use floating-point arithmetic, the result is not always correct. Thus, given $c = \mathit{maxExp}(\boldsymbol{a} \cdot \boldsymbol{x}, I)$, we discard the floating point optimum c and query which inequalities $I_B \subseteq I$ the Simplex solver used as a basis when observing the maximum. We then find a vector $\boldsymbol{\lambda}$ of multipliers such that $\boldsymbol{\lambda} A = \boldsymbol{a}$ where $I_B \equiv A\boldsymbol{x} \leq \boldsymbol{c}$. If

$c \neq \infty$ and no $\boldsymbol{\lambda}$ exists then the floating-point solver is wrong and we re-run the linear program using exact arithmetic which is about 70 times slower.

We applied our implementation of the inversion join to a benchmark suite that was gathered in the context of the PIPS project [1] which pursued advanced optimizations and parallelization of Fortran code. The benchmarks [11] contain every 100th input to the convex hull algorithm of the analyser while analyzing the Perfect Club and Spec 95 Fortran benchmarks. The data in the benchmark consist of pairs of polyhedra that were joined during the analysis. Thus, the benchmarks only contain few examples that are exponential as the analyzer would not have progressed after encountering a hard problem. Hence, there are no instance where our algorithm delivers a result while the exact convex hull algorithm times out. We generated a C++ program that calculates the convex hull of each test case using the Parma Polyhedra Library [2] and a Haskell program that implements the inversion join, using the GNU Linear Programming Toolkit as solver [7]. The test programs record the results to disk which we used to count the number of inequalities that the inversion join lacks from the exact solution of the convex hull. The test results in Table 1 are partitioned by the number of dimensions (variables) in the output polyhedron except for the last row which shows the results for running all 3492 tests. Next to the dimension we show the number of test cases and then three double columns that state how many of these test cases do not match the convex hull; those that lack at most one inequality and those that lack at most two. In each double column the number of incomplete cases is given together with the percentage of the total. From the last row, it can be seen that the algorithm is exact in 73% of all cases and misses less than two inequalities in 83% of all cases. Note that the number of incomplete joins rises with the dimension which is to be expected as the convex hull problem is exponential in the dimension.

5 Conclusion and Discussion

We assessed the precision of the inversion join in a qualitative and quantitative way. In particular, we argue that the inversion join with $k = 2$ is complete for planar polyhedra. This begs the question if similar results are obtainable for $k > 2$. The completeness in two dimensions implies that a TVPI system of inequalities can be joined using the inversion join in a way that the result is more precise than that of the TVPI domain. However, the result of applying the inversion join to a TVPI system is not necessarily a TVPI system since inequalities may have more than two variables per inequality. Furthermore, widening, a process necessary to ensure termination, is non-monotonic. Widening could therefore render an analysis using the inversion join less precise than that of the TVPI system. Further empirical studies that compare the precision of the two domains would thus be of interest.

On general polyhedra, our observation is that the inversion join produces very good approximations to the precise convex hull algorithm while avoiding the exponentially sized output cases. The inversion join therefore presents a sweetpoint between precision and efficiency and we would argue for the inclusion of this algorithm into the common off-the-shelf polyhedra libraries.

The author wishes to thank Duong Nguyen Que for making the benchmark suite available and also Liqian Chen for useful discussions. The author would also like to thank Antoine Miné for his diligent work as editor.

References

[1] Ancourt, C., F. Coelho, F. Irigoin and R. Keryell, *A Linear Algebra Framework for Static High Performance Fortran Code Distribution*, Scientific Programming **6** (1997), pp. 3–27.

[2] Bagnara, R., P. M. Hill and E. Zaffanella, *Not Necessarily Closed Convex Polyhedra and the Double Description Method*, Formal Aspects of Computing **17** (2005), pp. 222–257.

[3] Cousot, P. and R. Cousot, *Abstract Interpretation and Application to Logic Programs*, Journal of Logic Programming **13** (1992), pp. 103–179.

[4] Cousot, P. and N. Halbwachs, *Automatic Discovery of Linear Constraints among Variables of a Program*, in: *Principles of Programming Languages* (1978), pp. 84–97.

[5] Howe, J. M. and A. King, *Logahedra: a New Weakly Relational Domain*, in: Z. Lu and A. P. Ravn, editors, *Automated Technology for Verification and Analysis*, LNCS (2009).

[6] Lassez, J.-L. and K. McAloon, *A Canonical Form for Generalized Linear Constraints*, Journal of Symbolic Computation **13** (1993), pp. 1–24.

[7] Makhorin, A., *GLPK (GNU Linear Programming Kit)* (2008), version 4.32.
URL http://www.gnu.org/software/glpk/

[8] Mauborgne, L. and X. Rival, *Trace Partitioning in Abstract Interpretation Based Static Analyzers*, in: M. Sagiv, editor, *European Symposium on Programming*, LNCS **3444** (2005), pp. 5–20.

[9] Miné, A., *The Octagon Abstract Domain*, Higher-Order and Symbolic Computation **19** (2006), pp. 31–100.

[10] Miné, A., *Symbolic Methods to Enhance the Precision of Numerical Abstract Domains*, in: E. A. Emerson and K. S. Namjoshi, editors, *Verification, Model Checking and Abstract Interpretation*, LNCS **3855** (2006), pp. 348–363.

[11] Nguyen-Que, D., *Polybench* (2006).
URL http://www.cri.ensmp.fr/people/duong/polybench

[12] Sankaranarayanan, S., M. Colón, H. B. Sipma and Z. Manna, *Efficient Strongly Relational Polyhedral Analysis.*, in: E. A. Emerson and K. S. Namjoshi, editors, *Verification, Model Checking and Abstract Interpretation*, LNCS **3855** (2006), pp. 111–125.

[13] Sankaranarayanan, S., H. B. Sipma and Z. Manna, *Scalable Analysis of Linear Systems Using Mathematical Programming*, in: R. Cousot, editor, *Verification, Model Checking, and Abstract Interpretation*, LNCS **3385** (2005), pp. 25–41.

[14] Simon, A., *Splitting the Control Flow with Boolean Flags*, in: M. Alpuente and G. Vidal, editors, *Static Analysis Symposium*, LNCS **5079** (2008), pp. 315–331.

[15] Simon, A. and A. King, *Exploiting Sparsity in Polyhedral Analysis*, in: C. Hankin and I. Siveroni, editors, *Static Analysis Symposium*, LNCS **3672** (2005), pp. 336–351.

[16] Simon, A., A. King and J. M. Howe, *Two Variables per Linear Inequality as an Abstract Domain*, in: M. Leuschel, editor, *Logic-Based Program Synthesis and Transformation*, LNCS **2664** (2003), pp. 71–89.