The Two Variable Per Inequality Abstract Domain[†]

Axel Simon (axel.simon@ens.fr)
École Normale Supérieure, 45 rue d'Ulm, 75230 Paris Cedex 05, France[‡]

Andy King (a.m.king@kent.ac.uk)
University of Kent, Canterbury, CT2 7NF, United Kingdom

Jacob M. Howe (jacob@soi.city.ac.uk)
City University London, EC1V 0HB, United Kingdom

Abstract. This article presents the Two Variable Per Inequality abstract domain (TVPI domain for short). This so-called weakly-relational domain is able to express systems of linear inequalities where each inequality has at most two variables. The domain represents a sweet-point in the performance-cost tradeoff between the faster Octagon domain and the more expressive domain of general convex polyhedra. In particular, we detail techniques to closely approximate integral TVPI systems, thereby finessing the problem of excessively growing coefficients, yielding – to our knowledge – the only relational domain that combines linear relations with arbitrary coefficients and strongly polynomial performance.

Keywords: polyhedral analysis, integer programming, abstract interpretation

Static analysis methods have evolved from inferring prerequisite invariants for compiler optimisations to tools in their own right that are able to prove the absence of run-time errors of software. The abstract interpretation framework [21] provides a way of constructing and justifying program analyses. The key idea is to simulate each operation in a program with an abstract analog that operates on a description of the program state, rather than the state itself. The descriptions are chosen to trace a property of interest, for instance, an interval that encloses all possible values of a variable would be of value when reasoning about array bounds. These descriptions constitute what is known as an abstract domain. An abstract domain is usually presented as a lattice $\langle D, \sqsubseteq, \sqcup, \sqcap \rangle$ and the analysis itself is formulated as a set of recursive equations over D. The recursive equations are solved iteratively, until a fixpoint is reached which is detected using the ordering predicate $\sqsubseteq$. Each equation expresses how a program statement transforms the state at one program point to the state at another. Equations may be recursive because of loops in the program. The meet $\sqcap$ and join

[†] This paper is a revised extract of the first author's PhD thesis [65], which, in turn, extends [72].

[‡] New address: Technische Universität München, Lehrstuhl für Informatik 2, Boltzmannstraße 3, 85748 Garching, Germany (axel.simon@in.tum.de).

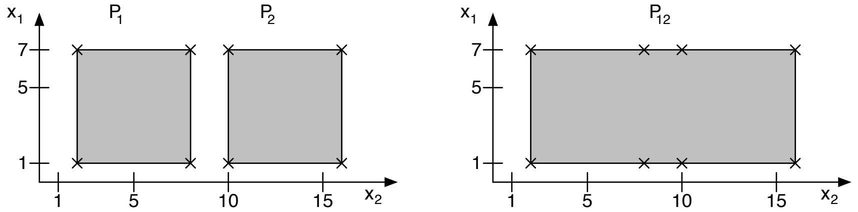


Figure 1. Calculating the convex hull $P_{12} = P_1 \sqcup_P P_2$ of planar polyhedra using the generator representation

$\sqcup$ operators arise when modelling control flow. For example, the state at the beginning of the then-branch of an if-then-else construct can be expressed as the meet of the condition with the state immediately before the conditional. Dually, the join operation $\sqcup$ summarises the two states from the then- and else-branch into a single state. Some lattices contain infinite ascending chains of the form $d_1 \sqsubset d_2 \sqsubset \ldots$ where $d_i \in D$ which can be generated during a fixpoint computation. In this case widening is required to induce termination [24].

Research into abstract domains revolves around the trade-off between the expressiveness of the domain and the cost of its operations. This is no more so than for numeric abstract domains. For instance, the domain of convex polyhedra [25] provides the ability to infer linear relationships between any number of variables. However, common implementations of convex polyhedra [6, 10, 32, 44] suffer from scalability problems that relate to the calculation of the join operation which corresponds to the convex hull in the context of polyhedra. The classic approach to calculating the convex hull of two polyhedra is to convert the half-space representation using inequalities into the generator representation consisting of vertices, rays and lines. Vertices are points in the polyhedron that cannot be represented by a convex combination of other points. Rays and lines are vectors that represent unidirectional and bidirectional trajectories, respectively, towards which the polyhedron extends to infinity. The convex hull of two input polyhedra can be calculated by converting both polyhedra into their generator representation, joining their sets of vertices, rays and lines and converting these three sets back into the half-space representation. In order to illustrate the problems using this approach, consider Fig. 1. Here, the shown polyhedra $P_1 = [\![\{1 \leq x_1 \leq 7, 2 \leq x_2 \leq 8\}]\!]$ and $P_2 = [\![\{1 \leq x_1 \leq 7, 10 \leq x_2 \leq 16\}]\!]$ contain neither rays nor lines as they are both bounded. The set of vertices are shown as crosses. These vertices are included in the resulting convex hull $P_{12} = P_1 \sqcup_P P_2$ which is shown in the right graph of the figure. A similar example in three dimen-

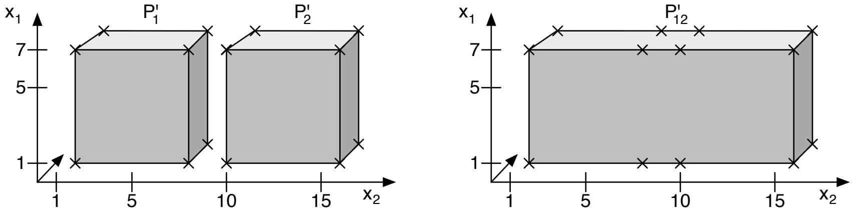


Figure 2. Calculating the convex hull $P'_{12} = P'_1 \sqcup_P P'_2$ of three dimensional polyhedra using the generator representation

sions is shown in Fig. 2 which depicts the convex hull $P'_{12} = P'_1 \sqcup_P P'_2$ of the polyhedra $P'_1 = [\![\{1 \leq x_1 \leq 7, 2 \leq x_2 \leq 8, 0 \leq x_3 \leq 3\}]\!]$ and $P'_2 = [\![\{1 \leq x_1 \leq 7, 10 \leq x_2 \leq 16, 0 \leq x_3 \leq 3\}]\!]$. While in the two-dimensional case each input polyhedron can be described by four inequalities or, equivalently, four vertices, each input in the three-dimensional case is described by six inequalities or, equivalently, eight vertices. In general, calculating the convex hull of two d-dimensional hypercubes requires $2d$ inequalities to represent each input polyhedron or, equivalently, 2^d vertices. Thus, even though input and output polyhedra can be described by a small number of inequalities, the intermediate representation using generators can be exponential.

While circumventing this exponential blow-up is possible by approximating the convex hull operation [69], the most compelling example of large-scale program analysis that uses a relational domain [11] is based on a sub-class of polyhedra, namely the Octagon domain [47]. The Octagon domain [47] only allows linear inequalities of the form $\pm x \pm y \leq c$ where $c \in \mathbb{Q}$ and thereby scales to hundreds of thousand lines of code. However, for certain applications like string buffer analysis [67, 70], the expressiveness of these inequalities is insufficient. Furthermore, analysers such as Astrée use auxiliary domains to recover from the precision loss that occurs when using only Octagons. For example, the byte offset b when accessing an integer array at index i is $b = 4i$ which needs to be expressed as an adjunct to the Octagon domain. Not only does this approach require a second domain, it also requires an accompanying substitution strategy that, for instance, replaces each occurrence of $4i$ with b [48].

This article presents the theory and the practical design of the Two Variable Per Inequality (TVPI) domain in which inequalities take on the form $ax + by \leq c$ where $a, b, c \in \mathbb{Z}$. This domain can thus express arbitrary linear relationships between any two program variables. The TVPI domain thus addresses the need for two communicating domains, for instance, to argue about byte offsets into arrays, leading to an overall simpler analysis design. The central idea of the TVPI domain is to replace operations on n-dimensional polyhedra by operations on planar polyhedra. Specifically, given a system of TVPI inequalities I, two inequalities $ax+by \leq c$, $dx+ez \leq f \in I$ can be combined to obtain $ae\,z - bd\,y \leq af - cd$ if $a > 0$ and $d < 0$. Adding this so-called resultant to the original system, removing redundant inequalities and repeating this process leads to a so-called closed system after a finite number of steps. The key observation is that the join of two closed TVPI systems can be broken down to calculating the planar convex hull for each pair of variables. Thus, instead of running the general convex hull algorithm for n-dimensional polyhedra, an $O(n \log(n))$ planar convex hull algorithm [68] is run for $n(n-1)/2$ pairs of variables. On the contrary, implementing the meet operation, that is, adding new inequalities to the system, must close the TVPI system again by calculating resultants.

Another challenge in using a polyhedral domain is the inference of a fixpoint in finite time. While widening ensures that a loop fixpoint is approximated by a finite number of inequalities [25], it has been observed that coefficients of inequalities can grow excessively which compromises the practicality of an analysis [69]. Assuming that variables in the program are integral, the polyhedron describing the program state can be shrunk around the contained integral points without compromising correctness, resulting in a so-called $\mathbb{Z}$-polyhedron in which every extreme point is integral. In practice, the coefficients of these extreme points are limited by the maximal range of the program variables, thereby also limiting the coefficients of inequalities describing the polyhedron. However, since calculating a $\mathbb{Z}$-polyhedron from a given TVPI polyhedron is NP-complete [42], we present an approximation that is precise in practice but retains the strongly polynomial performance.

In summary, this article describes the design and implementation of the TVPI domain, thereby presenting the following novelties:

- a complete set of domain operations for planar polyhedra consisting of an $O(n \log(n))$ convex hull algorithm, a redundancy removal algorithm for the meet operation, a linear entailment check and a linear programming function to query variable bounds;

- a practical implementation of the TVPI domain including an incremental closure algorithm and an implementation as a reduced product between intervals and TVPI inequalities;

- a refinement of the TVPI domain that shrinks each planar polyhedron around the contained integral points. The ensuing limit on the size of coefficients in the inequalities guarantees a strongly polynomial performance of all domain operations.

After presenting basic notation used in this article, Sections 2–4 describe each of the three aspects above in turn. Section 5 demonstrates the expressiveness of the TVPI domain and Section 6 concludes.

1. The Domain of Convex Polyhedra

1.1. The Lattice of Convex Spaces

A polyhedral analysis expresses numeric constraints over the set of abstract variables $\mathcal{X}$. For the sake of this section, let $\vec{x}$ denote the vector of all variables in $\mathcal{X}$, thereby imposing an order on $\mathcal{X}$. Let $\mathit{Lin}^{\mathbb{R}}$ denote the set of linear expressions of the form $\vec{a} \cdot \vec{x}$ where $\vec{a} \in \mathbb{R}^{|\mathcal{X}|}$ and let $\mathit{Ineq}^{\mathbb{R}}$ denote the set of linear inequalities $\vec{a} \cdot \vec{x} \leq c$ where $c \in \mathbb{R}$. For simplicity, let e.g. $6x_3 \leq x_1 + 5$ abbreviate $\langle -1, 0, 6, 0, \ldots 0\rangle \cdot \vec{x} \leq 5$ and let e.g. $x_2 = 7$ abbreviate the two opposing inequalities $x_2 \leq 7$ and $x_2 \geq 7$, the latter being an abbreviation of $-x_2 \leq -7$. As the analysis only infers integral properties, the notation $e_1 < e_2$ is used to abbreviate $e_1 \leq e_2 - 1$. Each inequality $\vec{a} \cdot \vec{x} \leq c \in \mathit{Ineq}^{\mathbb{R}}$ induces a half-space $[\![\vec{a} \cdot \vec{x} \leq c]\!] = \{\vec{x} \in \mathbb{R}^{|\mathcal{X}|} \mid \vec{a} \cdot \vec{x} \leq c\}$. A set of inequalities $I \subseteq \mathit{Ineq}^{\mathbb{R}}$ induces a closed, convex space $[\![I]\!] = \bigcap_{\iota \in I} [\![\iota]\!]$. Let $\mathcal{S} = \{[\![I]\!] \mid I \subseteq \mathit{Ineq}^{\mathbb{R}}\}$ denote the set of all convex spaces and $S = S_1 \curlyvee S_2$ denote the topological closure of the convex hull of $S_1, S_2 \in \mathcal{S}$, that is, the smallest closed, convex space S such that $S_1 \subseteq S$ and $S_2 \subseteq S$. Together with inclusion $\subseteq$ and intersection $\cap$, $\mathcal{S}$ forms a complete lattice $\langle \mathcal{S}, \subseteq, \curlyvee, \cap \rangle$.

1.2. The Lattice of Convex Polyhedra

Given this lattice, the solution of a set of semantic equations that describe the behaviour of a program can be expressed as a fixpoint calculation. However, the lattice contains infinite ascending chains that converge on infinite sets of inequalities or on inequalities with irrational coefficients. Let Lin denote the set of linear expressions of the form $\vec{a} \cdot \vec{x}$ where $\vec{a} \in \mathbb{Z}^{|\mathcal{X}|}$ and let Ineq denote inequalities $e \leq c$ where $e \in \mathit{Lin}$ and $c \in \mathbb{Z}$. Each inequality $\vec{a} \cdot \vec{x} \leq c \in \mathit{Ineq}$ induces a half-space $[\![\vec{a} \cdot \vec{x} \leq c]\!] = \{\vec{x} \in \mathbb{Q}^{|\mathcal{X}|} \mid \vec{a} \cdot \vec{x} \leq c\}$. Define $\mathit{Poly} = \{[\![I]\!] \mid I \in \mathit{Ineq} \wedge |I| \in \mathbb{N}\}$ to be the set of convex polyhedra. In order to ensure that the fixpoint computation only converges on elements in Poly rather than on states in $\mathcal{S} \supset \mathit{Poly}$, a widening operator $\nabla : \mathit{Poly} \times \mathit{Poly} \to \mathit{Poly}$ with the following properties must be inserted into every loop of the semantic equations where Q denotes the previous and P the current state that includes Q [24, 25]:

1. $\forall P, Q \in \mathit{Poly} \,.\, P \subseteq P \nabla Q$

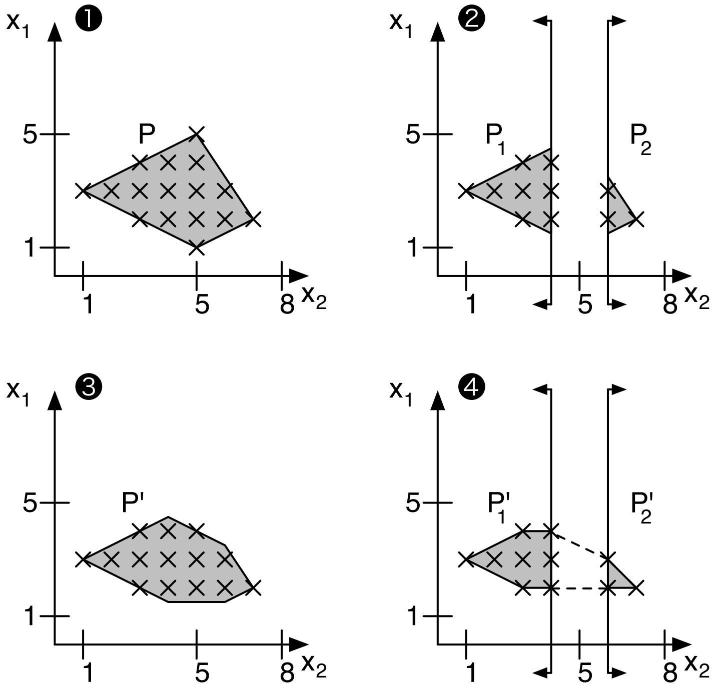


Figure 3. Example to show that $\mathbb{Z}$-polyhedra are not closed under intersection. Integral points are shown as crosses and the half-spaces $[\![x \leq 4]\!]$ and $[\![x \geq 6]\!]$ are indicated by vertical lines with arrows pointing towards the feasible space

2. $\forall P, Q \in \mathit{Poly} \,.\, Q \subseteq P \nabla Q$

3. for all increasing chains $P_0 \subseteq P_1 \subseteq \ldots$, the increasing chain defined by $R_0 = P_0$ and $R_{i+1} = R_i \nabla P_{i+1}$ is ultimately stable.

In order to reflect the restriction of *Poly* to finitely representable state spaces in the operations on *Poly*, we use $\sqcup_P$ for $\overline{\curlyvee}$, furthermore $\sqcap_P$ for $\cap$ and $\sqsubseteq_P$ for $\subseteq$, yielding the lattice of convex polyhedra $\langle \mathit{Poly}, \sqsubseteq_P, \sqcup_P, \sqcap_P \rangle$. This lattice is incomplete as neither the join nor meet of an arbitrary number of polyhedra is necessarily a polyhedron.

1.3. The Lattice of $\mathbb{Z}$-Polyhedra

Widening not only restricts the fixpoint calculation to elements of *Poly* but in fact ensures that the number of inequalities remains computationally tractable. In contrast, the size of coefficients may still grow beyond manageable sizes as a result of applying the $\sqcup_P$ operation. One possible approach for dealing with inequalities with very large coefficients and constants is to merely discard them during the analysis [69], thereby incurring a precision loss that is difficult to understand and anticipate when interpreting the results of an analysis. A more semantic approach can be taken when all variables of interest are integral, which makes it possible to restrict the domain of *Poly* further to the set of convex spaces over $\mathbb{Z}$. These so-called $\mathbb{Z}$-polyhedra can be characterised by the fact that all their vertices, that is, all extreme points of the convex space, have integral coordinates. As a consequence, the inequalities that define $\mathbb{Z}$-polyhedra have coefficients whose size is bounded by the coordinates of the vertices they connect. In particular, by equating polyhedra that contain the same set of integral points, it is possible to define a lattice of $\mathbb{Z}$-polyhedra $\langle Poly_{\equiv\mathbb{Z}}, \sqsubseteq_P^{\mathbb{Z}}, \sqcup_P^{\mathbb{Z}}, \sqcap_P^{\mathbb{Z}}\rangle$. If each equivalence class is represented by its smallest polyhedron it is possible to put $\sqsubseteq_P^{\mathbb{Z}} = \sqsubseteq_P$ and $\sqcup_P^{\mathbb{Z}} = \sqcup_P$. However, the meet operation $\sqcap_P$ is not closed for $\mathbb{Z}$-polyhedra. In order to illustrate this, consider Fig. 1.3. The state space $P \in Poly_{\equiv\mathbb{Z}}$ over x_1, x_2 in the first graph is transformed by evaluating the conditional $x_2 \neq 5$ which is implemented by calculating $P' = (P \sqcap_P [\![x \leq 4]\!]) \sqcup_P (P \sqcap_P [\![x \geq 6]\!])$. Observe that the input P as well as the two half-spaces $[\![x \leq 4]\!]$ and $[\![x \geq 6]\!]$ are $\mathbb{Z}$-polyhedra. The second graph shows the two intermediate results $P_1 = P \sqcap_P [\![x \leq 4]\!]$ and $P_2 = P \sqcap_P [\![x \geq 6]\!]$ which both have two non-integral vertices. As a consequence, the join of P_1 and P_2, shown as third graph, has non-integral vertices as well and is therefore not a $\mathbb{Z}$-polyhedron. However, if the intermediate results were shrunk around the contained integral point sets, as illustrated in the fourth graph, all vertices of the intermediate results would be integral and the join would be a $\mathbb{Z}$-polyhedron, too. However, for general, n-dimensional polyhedra, the number of inequalities necessary to represent a $\mathbb{Z}$-polyhedron can grow exponentially with respect to a polyhedron over $\mathbb{Q}$ that contains the same integral points [58, Chap. 23]. Thus, no efficient algorithm exists to implement the $\sqcap_P$-operation on $\mathbb{Z}$-polyhedra. However, in order to limit the growth of coefficients, Sect. 4 presents efficient techniques to approximate the $\sqcap_P^{\mathbb{Z}}$-operation. In anticipation of this section, we define all operations that query the value of a polyhedron to return integral bounds.

Given the basic operations on polyhedra, the next section proposes implementations for them in the context of planar polyhedra.

2. Planar Polyhedra

This section presents operations on planar, that is, two-dimensional polyhedra which are later lifted to sets of inequalities that contain at

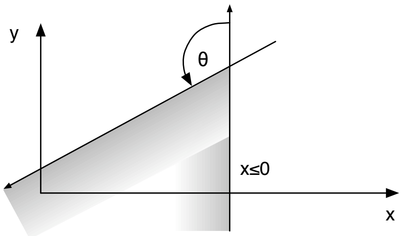


Figure 4. The angle of a planar inequality is measured relative to $x \leq 0$

most two variables but with arbitrary coefficients, thereby extending the abstract domain of Octagons. We show that all domain operations on planar polyhedra can be implemented efficiently, thereby providing a basis for an efficient lifting to arbitrary dimensions. The key observation for implementing efficient algorithms is that inequalities in two dimensions can be sorted by angle. The section therefore commences by presenting basic properties of inequalities in planar space. The notation introduced here is then used to define the various domain operations on planar polyhedra.

2.1. Operations on Inequalities

In this section we shall present some basic concepts from the literature about planar inequalities [59]. For the sake of this section, let $\mathcal{X} = \{x, y\}$ denote the set of polyhedral variables that correspond to the axes of the two-dimensional Euclidian space. Observe that the vector $\langle a, b\rangle$ is orthogonal to the line $ax + by = c$ and points away from the induced half-space $[\![ax + by \leq c]\!]$. This vector induces an ordering on half-spaces via the orientation mapping θ. This map $\theta : \mathit{Ineq} \to [0, 2\pi)$ is defined such that $\theta(ax + by \leq c) = \psi$ where $\cos(\psi) = a/\sqrt{a^2 + b^2}$ and $\sin(\psi) = b/\sqrt{a^2 + b^2}$. The mapping θ corresponds to the counter-clockwise angle which the half-space of $x \leq 0$ has to be turned through to coincide with that of $ax + by \leq c$ as illustrated in Fig. 4. In the context of this work, θ is mainly used to compare the orientation of two half-spaces which is key to sorting a set of inequalities. For the sake of efficiency and numeric stability, it is desirable to implement this comparator without recourse to trigonometric functions [59]. To this end, define the function $\mathit{class} : \mathit{Ineq} \to \{1, 2, \ldots, 8\}$ which classifies an inequality according to its angle as shown in Fig. 5. It is defined by:

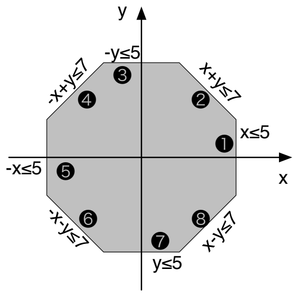


Figure 5. Classifying planar inequalities by their angle

$$class(ax + by \leq c) = \begin{cases} 7 - sign(b) & : \ a < 0 \\ 1 & : \ a = 0 \wedge b \leq 0 \\ 5 & : \ a = 0 \wedge b > 0 \\ 3 + sign(b) & : \ a > 0 \end{cases}$$

Here $sign : \mathbb{Z} \to \{-1, 0, 1\}$ is the function that returns -1 if the given number is negative, 1 if it is positive and zero otherwise. A comparison between the angles of $\iota_1 \equiv a_1 x + b_1 x \leq c_1$ and $\iota_2 \equiv a_2 x + b_2 x \leq c_2$ can now be implemented as follows:

$$\begin{aligned} \theta(\iota_1) \leq \theta(\iota_2) \iff & class(\iota_1) \leq class(\iota_2) \vee \\ & class(\iota_1) = class(\iota_2) \wedge a_1 b_2 \leq a_2 b_1 \end{aligned}$$

Furthermore, define the angular difference $\iota_1 \measuredangle \iota_2$ between two inequalities ι_1 and ι_2 as the counter-clockwise angle between $\theta(\iota_1)$ and $\theta(\iota_2)$. More precisely $\iota_1 \measuredangle \iota_2 = (\theta(\iota_2) - \theta(\iota_1)) \ mod \ 2\pi$. This function is used to test if two inequalities are less than π apart. As above, this test can be implemented without recourse to trigonometric functions.

2.1.1. *Entailment between Single Inequalities*

A recurring function is the test if two inequalities define a sub-space of another inequality. In fact, this test is a building block of the upcoming domain operation which applies this test to compare consecutive elements of a sorted sequence of inequalities and thereby infer information on an inequality with respect to the whole sequence.

We give a definition of the test in the form of a case distinction on the coefficients of the involved inequalities. Let $\iota_i \equiv a_i x + b_i y \leq c_i$ for $i = 1, 2$ and $\iota \equiv ax + by \leq c$. Assume $\iota_1 \measuredangle \iota_2 \leq \pi$, otherwise exchange ι_1 and ι_2. We define the following predicates which are explained below:

$$\{\iota_1\} \sqsubseteq \iota \iff \begin{cases} \mathit{false} & \text{if } a_1 b - a b_1 \neq 0 \\ \mathit{false} & \text{else if } a_1 a < 0 \\ \mathit{false} & \text{else if } b_1 b < 0 \\ \frac{a}{a_1} c_1 \leq c & \text{else if } a_1 \neq 0 \\ \frac{b}{b_1} c_1 \leq c & \text{else if } b_1 \neq 0 \\ (c < 0 \wedge a = 0 \wedge b = 0) \Rightarrow c_1 < 0 & \text{otherwise} \end{cases}$$

$$\{\iota_1, \iota_2\} \sqsubseteq \iota \iff \begin{cases} \{\iota_1\} \sqsubseteq \iota \vee \{\iota_2\} \sqsubseteq \iota & \text{if } d = a_1 b_2 - a_2 b_1 = 0 \\ \mathit{false} & \text{else if } \lambda_1 = (a b_2 - a_2 b)/d < 0 \\ \mathit{false} & \text{else if } \lambda_2 = (a_1 b - a b_1)/d < 0 \\ \lambda_1 c_1 + \lambda_2 c_2 \leq c & \text{otherwise} \end{cases}$$

Intuitively, the predicate $\{\iota_1\} \sqsubseteq \iota$ holds iff $[\![\iota_1]\!] \subseteq [\![\iota]\!]$ and, analogously, $\{\iota_1, \iota_2\} \sqsubseteq \iota$ holds iff $[\![\{\iota_1, \iota_2\}]\!] \subseteq [\![\iota]\!]$. The reasoning behind the above definitions is as follows.

Inclusion between two single inequalities never holds if they are not parallel, that is, if the determinant of their coefficients $a_1 b - a b_1$ is non-zero. Furthermore, the inclusion cannot hold if ι_1 and ι are anti-parallel, which is the case if the coefficients for x have different signs. Similarly for y. Otherwise, the intersection points with the y-axis of ι_1 and ι are calculated and compared. In particular, the subset relation $\{x \mid a_1 x \leq c_1\} \subseteq \{x \mid ax \leq c\}$ implies that $x \leq \frac{c}{a}$ if $x \leq \frac{c_1}{a_1}$, assuming that $a_1 > 0$ and $a > 0$. The latter is equivalent to $\frac{c_1}{a_1} \leq \frac{c}{a}$ and since a is positive, $\frac{a}{a_1} c_1 \leq c$ follows. Now assume $a_1 < 0$ and $a < 0$. From $x \geq \frac{c}{a}$ if $x \geq \frac{c_1}{a_1}$ follows $\frac{c_1}{a_1} \geq \frac{c}{a}$. Multiplying by $a < 0$ yields $\frac{a}{a_1} c_1 \leq c$. In case $a_1 = 0$ but $b_1 \neq 0$, the intersection points with the x-axis can be calculated and compared. If $b_1 = 0$, too, then ι_1 is tautologous or unsatisfiable which is handled by the implication.

The second test $\{\iota_1, \iota_2\} \sqsubseteq \iota$ reduces to the first test whenever ι_1 and ι_2 are parallel, that is, if the determinant $a_1 b_2 - a_2 b_1$ is zero. Otherwise, a linear combination of ι_1 and ι_2 is calculated, yielding an inequality that is parallel to ι. Specifically, λ_1 and λ_2 are calculated such that $\lambda_1 a_1 + \lambda_2 a_2 = a$ and $\lambda_1 b_1 + \lambda_2 b_2 = b$. If either λ_1 or λ_2 is negative, the resulting half-space of the parallel inequality faces the opposite direction and the inequality ι is not entailed. If both λ_1 and λ_2 are positive, entailment can be determined by comparing the constant of the parallel inequality, namely $\lambda_1 c_1 + \lambda_2 c_2$, with the constant of ι.

Due to the ability to sort inequalities by angle, this constant-time entailment check between three inequalities can be lifted to a linear-time entailment check between two planar polyhedra, as presented in the next section.

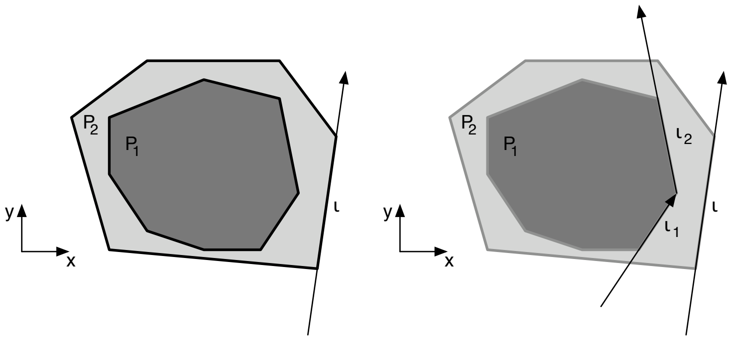


Figure 6. The entailment check $P_1 \sqsubseteq_P P_2$ reduces to entailment checks $\{\iota_1, \iota_2\} \sqsubseteq \iota$ by testing each inequality ι of P_2 with respect to some ι_1, ι_2 of P_1

2.2. Operations on Sets of Inequalities

In the sequel we present operations on planar polyhedra that exploit the fact that inequalities can be sorted by angle. Given this order on inequalities, entailment, redundancy removal, the convex hull and a linear programming algorithm can all be implemented efficiently.

2.2.1. *Entailment Check*

By traversing the inequalities so that their angles are increasing, a linear-time entailment check between two planar polyhedra can be implemented. Consider the task of checking whether $P_1 \sqsubseteq_P P_2$, that is, if $P_1 \subseteq P_2$ holds. It is sufficient to show that each inequality ι that defines a facet of P_2 contains the polyhedron P_1 as shown on the left of Fig. 6. Specifically, it is sufficient to find the two adjacent inequalities ι_1, ι_2 of P_1 that are angle-wise no larger and strictly larger than ι and ensure that $\{\iota_1, \iota_2\} \sqsubseteq \iota$. All inequalities in P_1 which are not tested in this way will only make the inner polyhedron P_1 smaller and, hence, cannot affect the outcome of the entailment check. Algorithm 1 formalises this idea. Here, the trivial cases of $P_1 = \emptyset$ and $P_2 = \emptyset$ are checked before the inequalities that constitute the two polyhedra are extracted with indices that increase with the angle. For each inequality ι_i in the facet list of P_2, the inner loop in lines 12–15 finds two adjacent inequalities $\iota_{l \bmod n}, \iota_{u \bmod n}$ in P_1 that enclose the inequality ι_i angle-wise. If the entailment $\{\iota_{l \bmod n}, \iota_{u \bmod n}\} \sqsubseteq \iota_i$ holds for all ι_i and corresponding $\iota_{l \bmod n}, \iota_{u \bmod n}$, then P_2 is entailed by P_1. Note that the total number of times the inner loop iterates is $|I_2|$ where $[\![I_2]\!] = P_2$. The whole algorithm runs in $O(|I_1| + |I_2|)$ where $[\![I_1]\!] = P_1$ and is thereby linear in the size of its input.

Algorithm 1 Checking entailment between planar polyhedra

```
procedure $entails(P_1, P_2)$ where $P_1, P_2 \in Poly$
if $P_1 = \emptyset$ then
  return $true$
end if
if $P_2 = \emptyset$ then
  return $false$
end if
$[\![\{\iota_0, \ldots \iota_{n-1}\}]\!] \leftarrow P_1$  /*such that $\theta(\iota_0) \leq \theta(\iota_1) \leq \ldots \leq \theta(\iota_{n-1})$*/
$[\![\{\iota'_0, \ldots \iota'_{m-1}\}]\!] \leftarrow P_2$  /*such that $\theta(\iota'_0) \leq \theta(\iota'_1) \leq \ldots \leq \theta(\iota'_{m-1})$*/
$u \leftarrow 0$
$l \leftarrow n - 1$
for $i \in [0, m-1]$ do
  while $u < n \wedge \theta(\iota_u) < \theta(\iota'_i)$ do
    $l \leftarrow u$
    $u \leftarrow u + 1$
  end while
  if $\{\iota_{l \ mod \ n}, \iota_{u \ mod \ n}\} \not\sqsubseteq \iota'_i$ then
    return $false$
  end if
end for
return $true$
```

A slightly more complicated iteration strategy is necessary to remove redundant inequalities. This task is the topic of the next section.

2.2.2. *Removing Redundancies*

The meet operation $\sqcap_P$ on planar polyhedra amounts to conjoining two sets of inequalities. This section presents an algorithm to remove redundant inequalities from the resulting set of inequalities, thereby maintaining a compact representation. Furthermore, redundancy removal also detects when an inequality system has become unsatisfiability as the result of the meet operation.

2.2.2.1. *Principle.* We define the function $nonRedundant(\{\iota_1, \ldots, \iota_m\})$ which takes a set of inequalities $\{\iota_1, \ldots, \iota_m\}$ that are sorted by angle. We use the notation $\langle \iota_1, \ldots, \iota_m \rangle$ to describe a sorted sequence of elements which, in practice, may be implemented as a doubly linked list. In this representation, assigning a sequence $\langle \iota_m, \iota_1, \ldots \iota_{m-1} \rangle$ to I is a constant-time operation that rotates the original sequence $I = \langle \iota_1, \ldots \iota_m \rangle$ one position to the right.

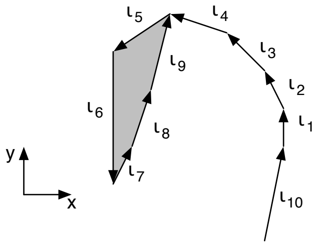


Figure 7. A chain of inequalities $\iota_1, \ldots \iota_4$ that are non-redundant with respect to their neighbours but which are redundant with respect to ι_9 and ι_5.

The key to removing redundant inequalities lies in the observation that the inequality ι_i is redundant if $\{\iota_{(i-1)\ mod\ m}, \iota_{(i+1)\ mod\ m}\} \sqsubseteq \iota_i$. Checking each inequality once against its neighbours is not sufficient to determine that no such pair of inequalities exists since each time an inequality is removed, the previously separate neighbours become adjacent which might make one of them redundant, too. This is illustrated in Fig. 7. Here, the feasible space of the polyhedron is shown in grey and, hence, the inequalities $\iota_{10}, \iota_1, \ldots \iota_4$ are redundant. Consider a redundancy removal function that starts checking each inequality from ι_1 onwards. Since $\{\iota_{10}, \iota_2\} \not\sqsubseteq \iota_1$, the inequality ι_1 is non-redundant with respect to its two neighbours and iteration proceeds to infer $\{\iota_1, \iota_3\} \not\sqsubseteq \iota_2$, $\{\iota_2, \iota_4\} \not\sqsubseteq \iota_3$ and so forth. It is not until iteration ten when $\{\iota_9, \iota_1\} \sqsubseteq \iota_{10}$ and ι_{10} is discarded. If iteration stops here, the redundant inequalities $\iota_1, \ldots \iota_4$ are not detected as such and an incorrect result is returned. In a correct implementation, iteration has to proceed until a fixpoint is reached, that is, until each inequality was found to be non-redundant.

2.2.2.2. *Implementation.* This strategy is implemented as Alg. 2. The input inequalities are stored in I in increasing angular order. The variable *todo* tracks the number of inequalities that still need to be examined and is initially set to the size of the sequence I. Lines 4–5 stop the loop short when the size of the system is so small that there is no neighbour to test against, which, in turn, would lead to the incorrect removal of the single remaining inequality. Otherwise, $|I| \geq 2$ and the conditional in line 8 tests if the first inequality in I is redundant with respect to its two neighbours. In case ι_1 is redundant, lines 9–10 remove this inequality from I and the *todo* counter is reset to enforce that every inequality in I is checked once more. Lines 12–13 deal with the case that ι_1 is non-redundant in which case the sequence I is rotated in order to check the next inequality. In this case the *todo* counter is merely decremented with the effect that the loop eventually

Algorithm 2 Removal of redundant inequalities

```
procedure nonRedundant($\{\iota_1, \dots \iota_n\}$) where $\iota_i \in Ineq$
I ← $\langle \iota_1, \dots \iota_n \rangle$  /*$\theta(\iota_1) \leq \theta(\iota_2) \leq \dots \leq \theta(\iota_n)$*/
todo ← $|I|$
while todo $> 0$ do
    if $|I| \leq 1$ then
        return I
    end if
    $\langle \iota_1, \dots \iota_m \rangle$ ← I
    if $\{\iota_m, \iota_2\} \sqsubseteq \iota_1$ then
        I ← $\langle \iota_m, \iota_2, \iota_3, \dots \iota_{m-1} \rangle$
        todo ← $|I|$
    else
        I ← $\langle \iota_2, \iota_3, \dots \iota_{m-1}, \iota_m, \iota_1 \rangle$
        todo ← todo $- 1$
    end if
end while
return I
```

stops when all remaining inequalities are non-redundant. With respect to the running-time of the algorithm, observe that Fig. 6 constitutes the worst-case scenario in which the loop iterates over a maximum chain of inequalities until the last inequality ι_{10} is found to be redundant. At this point, *todo* is repeatedly reset to $|I|$ until ι_4 is removed and the loop iterates further until ι_9, only to find that all inequalities are non-redundant.

2.2.2.3. *Complexity.* The algorithm runs for at most two complete iterations such that it is in $O(n)$ where n is the number of input inequalities. Note that the first inequality in the returned set I does not necessarily have the smallest angle. Rather than sorting the resulting set, it can be rotated until the smallest inequality is at the beginning of the sequence.

2.2.2.4. *Unsatisfiability.* A special case arises when the input to the *nonRedundant* function is an unsatisfiable set of inequalities. The algorithms will terminate with either $I = \{\iota_0, \iota_1\}$ where $\iota_0 \measuredangle \iota_1 = \pi$ or with $I = \{\iota_0, \iota_1, \iota_2\}$ where $\iota_i \measuredangle \iota_{(i+1)\ mod\ 3} < \pi$. In the former case the coefficients of the inequalities need to be compared in order to detect that their intersection is empty. In the latter case, the boundaries of the half-spaces $[\![\iota_0]\!]$ and $[\![\iota_1]\!]$ intersect in a point $\langle v_x, v_y\rangle$. The system is unsatisfiable if $\langle v_x, v_y\rangle \notin [\![\iota_2]\!]$, i.e. if $av_x+bv_y > c$ where $\iota_2 \equiv ax+by \leq c$.

This completes the description of the redundancy removal algorithm which forms the basis for the meet operation. We now proceed to define the join operation, which turns out to be the most intricate.

2.2.3. *Convex Hull*

In 1972 Graham published the first efficient algorithm to compute the convex hull of a set of points in planar space [29]. Since then numerous improvements [1, 2, 3, 15, 41] and extensions to polytopes [53, 74] have been proposed. An overview can be found in [54, 60]. Interestingly, some of these improvements turned out to be incorrect [30, 74, 75] which suggests that geometric algorithms are difficult to construct correctly. While the convex hull of polytopes (bounded polyhedra) can be calculated straightforwardly by taking the convex hull of their extreme points, calculating the convex hull of unbounded polyhedra, that is, convex spaces containing rays and lines [54, 60], turns out to be more subtle due to a large number of geometric configurations. Even for planar polyhedra, the introduction of rays makes it necessary to handle polyhedra such as a single half-space, a single ray, a single line, two facing (not coinciding) half-spaces, etc., all of which require special cases in a point-based algorithm. The problem is exacerbated by the number of ways these special polyhedra can be combined. In order to simplify the correctness argument, we present a direct reduction of the convex hull problem of planar polyhedra to the classic convex hull problem for a set of points. The idea of the presented algorithm is to confine vertices of the input polyhedra to a box and to use the rays to translate these points outside the box. A linear pass around the convex hull of all these points is then sufficient to determine the resulting polyhedron. This approach inherits the time complexity of the underlying convex hull algorithm, which is in $O(n \log n)$ [29]. Our algorithm follows the standard tactic for calculating the convex hull of polyhedra that are represented as sets of inequalities, namely to convert the input into an intermediate ray and vertex representation. Two approaches to the general (n-dimensional) conversion problem are the double description method [49] (also known as the Chernikova algorithm [16, 43]) and the vertex enumeration algorithm of Avis and Fukuda [4]. While our approach is linear, the Chernikova method leads to a cubic time solution for calculating the convex hull of planar polyhedra [43] whereas the method of Avis and Fukuda is quadratic.

Before we detail the algorithm itself, we define a few auxiliary functions. We then give an explanation in the context of an example in which the join of a bounded polyhedron and a polyhedron with two

Algorithm 3 Calculating an inequality from two points

```
procedure $connect(\langle x_1, y_1\rangle, \langle x_2, y_2\rangle)$ where $\langle x_i, y_i\rangle \in \mathbb{Q}^2$
  return $(y_2 - y_1)x + (x_1 - x_2)y \leq (y_2 - y_1)x_1 + (x_1 - x_2)y_1$
```

Algorithm 4 Calculating the generators of a planar polyhedron

```
procedure $extreme(\{\iota_0, \ldots \iota_{n-1}\})$ where $\theta(\iota_0) \leq \theta(\iota_1) \leq \ldots \theta(\iota_{n-1})$
$\langle V, R\rangle \leftarrow \langle\emptyset, \emptyset\rangle$
if n=1 then
  $ax + by \leq c \leftarrow \iota_0$
  $R \leftarrow \langle -a/\sqrt{a^2+b^2}, -b/\sqrt{a^2+b^2}\rangle$
end if
for $i \in [0, n-1]$ do
  $ax + by \leq c \leftarrow \iota_i$
  $d_{pre} \leftarrow \iota_{(i-1)\ mod\ n} \measuredangle \iota_i \geq \pi \vee n = 1$
  $d_{post} \leftarrow \iota_i \measuredangle \iota_{(i+1)\ mod\ n} \geq \pi \vee n = 1$
  if $d_{pre}$ then
    $R \leftarrow R \cup \{\langle b/\sqrt{a^2+b^2}, -a/\sqrt{a^2+b^2}\rangle\}$
  end if
  if $d_{post}$ then
    $R \leftarrow R \cup \{\langle -b/\sqrt{a^2+b^2}, a/\sqrt{a^2+b^2}\rangle\}$
  else
    $V \leftarrow V \cup intersect(\iota_i, \iota_{(i+1)\ mod\ n})$
  end if
  if $d_{pre} \wedge d_{post}$ then
    $V \leftarrow V \cup \{v\}$ where $v \in \{\langle x, y\rangle \mid ax + by = c\}$
  end if
end for
return $\langle V, R\rangle$
```

rays is calculated. A note on degenerate input polyhedra and their treatment completes the description of the algorithm.

2.2.3.1. *Auxiliary Functions* The auxiliary function $intersect(a_1x + b_1y \leq c_1,\ a_2x + b_2y \leq c_2)$ calculates the set of intersection points of the two lines $a_1x + b_1y = c_1$ and $a_2x + b_2y = c_2$. In practice, an implementation of this function only needs to be partial since it is only applied when the resulting set contains a single point. Algorithm 3 presents *connect* which generates an inequality from two points subject to the following constraints: the half-space induced by $connect(p_1, p_2)$ has p_1 and p_2 on its boundary and, if p_1, p_2, p_3 are sorted counter-

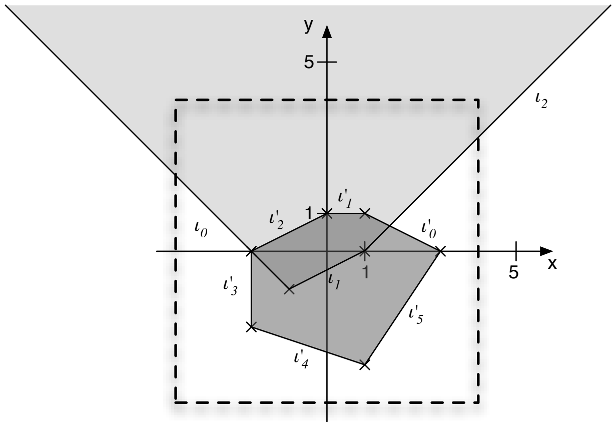


Figure 8. Calculate a square around the origin which includes all vertices

clockwise, then p_3 is in the feasible space. The notation $\overline{p_1, p_2}$ is used to abbreviate $\mathit{connect}(p_1, p_2)$. Furthermore, the predicate $\mathit{saturates}(p, \iota)$ holds whenever the point p is on the boundary of the half-space defined by the inequality ι, that is, $\mathit{saturates}(\langle x_1, y_1 \rangle, ax{+}by \leq c)$ iff $ax_1{+}by_1 = c$. Moreover, the predicate $\mathit{inBox}(s, p)$ holds whenever the point p is strictly contained within a square of width $2s$ that is centred on the origin. Specifically, $\mathit{inBox}(s, \langle x, y \rangle)$ iff $|x| < s \wedge |y| < s$. Finally, let $\mathrm{scan}(Q)$ denote the application of Graham's algorithm to calculate the convex hull of a sequence of points Q that is already ordered by angle.

2.2.3.2. *A Typical Run of the Algorithm* The algorithm divides into a decomposition and a reconstruction phase. The *hull* function, presented as Alg. 5, decomposes the input polyhedra into their corresponding ray and vertex representations by calling the function *extreme* in lines 4 and 5 which is defined as Alg. 4. The remainder of the *hull* function reconstructs a set of inequalities whose half-spaces enclose both sets of rays and points. The algorithm requires the input polyhedra to be non-redundant and sorted; its output is also non-redundant and sorted.

In order to illustrate the algorithm consider Fig. 8. The polyhedron $I = \{\iota_0, \iota_1, \iota_2\}$ and the polytope $I' = \{\iota'_0, \ldots, \iota'_5\}$ constitute the input to the *hull* function. They are passed to the function *extreme* at line 4 and 5. Note that we assume that the set of inequalities is sorted by angle such that their indices increase with the angle. The

Algorithm 5 Calculating the convex hull of planar polyhedra

```
procedure $hull(I_1, I_2)$ where $I_i \subseteq Ineq$ satisfiable, non-redundant
if $I_1 = \emptyset \vee I_2 = \emptyset$ then
  return $\emptyset$
end if
$\langle P_1, R_1 \rangle \leftarrow extreme(I_1)$
$\langle P_2, R_2 \rangle \leftarrow extreme(I_2)$
$P \leftarrow P_1 \cup P_2$
$R \leftarrow R_1 \cup R_2$ /*Note: $|R| \leq 8$*/
$s \leftarrow 1 + \max\{|x_0|, |y_0| \mid \langle x_0, y_0 \rangle \in P\}$
$Q \leftarrow P$
for $\langle \langle x_0, y_0 \rangle, \langle a, b \rangle \rangle \in P \times R$ do
  $Q \leftarrow Q \cup \{\langle x_0 + 2\sqrt{2}sa, y_0 + 2\sqrt{2}sb \rangle\}$
end for
if $Q = \{\langle x_1, y_1 \rangle\}$ then /*result is zero dimensional (a point)*/
  return $\{x \leq x_1, y \leq y_1, -x \leq -x_1, -y \leq -y_1\}$
end if
$q_p \leftarrow \langle \sum_{\langle x_0, y_0 \rangle \in Q} x_0 / |Q|, \sum_{\langle x_0, y_0 \rangle \in Q} y_0 / |Q| \rangle$ /*$q_p$ is interior point*/
$\langle q_0, \ldots, q_{n-1} \rangle \leftarrow \mathrm{sort}(q_p, Q)$ /*sort points by angle with $q_p$*/
$\langle q_{k_0}, \ldots, q_{k_{m-1}} \rangle \leftarrow \mathrm{scan}(\langle q_0, \ldots, q_{n-1} \rangle)$
$I_{\mathrm{res}} \leftarrow \emptyset$
for $i \in [0, m-1]$ do
  $\langle x_1, y_1 \rangle \leftarrow q_{k_i}$
  $\langle x_2, y_2 \rangle \leftarrow q_{k_{(i+1)\ mod\ m}}$
  $\iota \leftarrow connect(\langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle)$ /*add $\iota$ if $q_{k_i}$ or $q_{k_{i+1}}$ is in box*/
  $add \leftarrow inBox(s, \langle x_1, y_1 \rangle) \vee inBox(s, \langle x_2, y_2 \rangle) \vee m = 2$
  $j \leftarrow (k_i + 1)\ mod\ n$
  while $\neg add \wedge j \neq k_{i+1}$ do /* ...or any point on $\iota$ is in the box*/
    $add \leftarrow saturates(q_j, \iota) \wedge inBox(s, q_j)$
    $j \leftarrow (j + 1)\ mod\ n$
  end while
  if $m = 2 \wedge inBox(s, \langle x_1, y_1 \rangle)$ then
    if $y_1 = y_2$ then
      $I_{\mathrm{res}} \leftarrow I_{\mathrm{res}} \cup \{sgn(x_1 - x_2)x \leq sgn(x_1 - x_2)x_1\}$
    else
      $I_{\mathrm{res}} \leftarrow I_{\mathrm{res}} \cup \{sgn(y_1 - y_2)y \leq sgn(y_1 - y_2)y_1\}$
    end if
  end if
  if $add$ then
    $I_{\mathrm{res}} \leftarrow I_{\mathrm{res}} \cup \{\iota\}$
  end if
end for
return $I_{\mathrm{res}}$
```

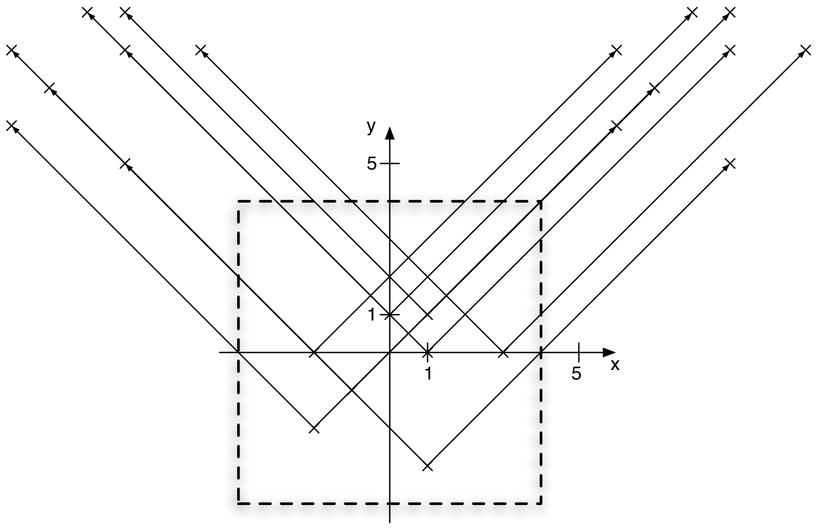


Figure 9. Translate all vertices along the rays such that they lie outside the square

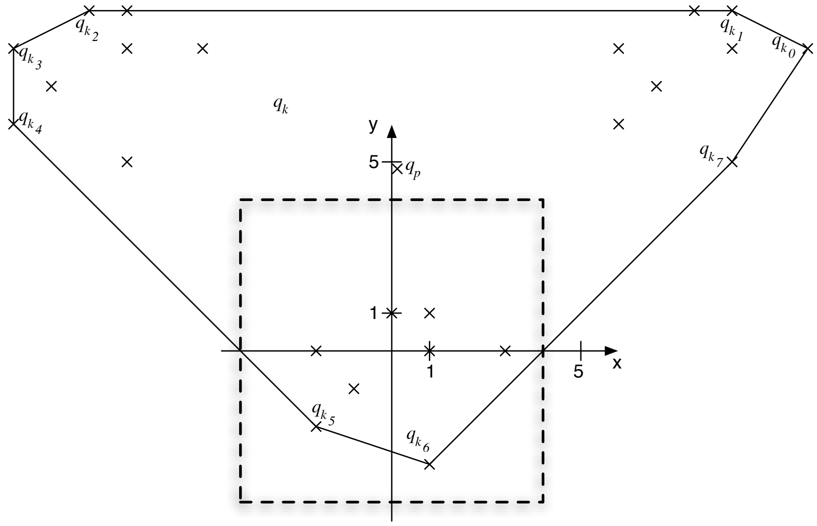


Figure 10. Calculate the convex hull of all points

loop at lines 6–22 of Alg. 4 examines the relationship of each inequality with its two angular neighbours. If d_{post} is false, the intersection point $intersect(\iota_i, \iota_{(i+1)\ mod\ n})$ is a vertex which is added at line 16. In the example, two vertices are created for I, namely v_1 and v_2 where $\{v_1\} = intersect(\iota_0, \iota_1)$ and $\{v_2\} = intersect(\iota_1, \iota_2)$. Six vertices are created for I'. Conversely, if d_{post} is true, the intersection point is

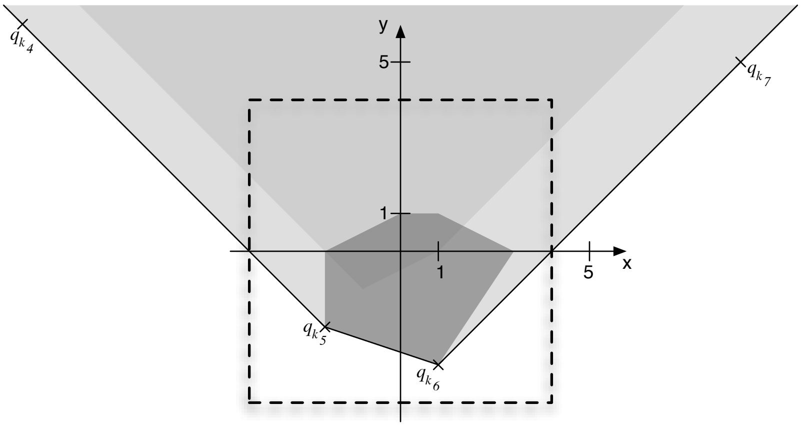


Figure 11. The three inequalities $\overline{q_{k_4}, q_{k_5}}$, $\overline{q_{k_5}, q_{k_6}}$ and $\overline{q_{k_6}, q_{k_7}}$ define a polyhedron that includes the two polyhedra $I = \{\iota_0, \iota_1, \iota_2\}$ and $I' = \{\iota'_0, \ldots \iota'_5\}$ from Fig. 8

degenerate, that is, either I contains a single inequality or the angular difference between the current inequality and its successor is greater than or equal to π. For instance, $\mathit{intersect}(\iota_2, \iota_0)$ is degenerated and thus it is not added to V. In case of degenerated intersection points, d_{pre} or d_{post} is true and rays are created at line 11 or 14, respectively. The two rays along the boundaries of ι_i and $\iota_{(i+1)\ mod\ n}$ are generated in loop iteration i when d_{post} is true and iteration $(i + 1)\ mod\ n$ when d_{pre} is true. In our example d_{post} is true for ι_2, generating a ray along the boundary of ι_2 which recedes in the direction of the first quadrant, whereas d_{pre} is true for ι_0 yielding a ray along ι_0 which recedes towards the second quadrant. No rays are created for the polytope I' because d_{post} and d_{pre} are false for all inequalities $\iota'_0, \ldots \iota'_5$.

In general, both flags might true, e.g. for anti-parallel half-spaces. In this case the inequality ι_i cannot define a vertex and an arbitrary point on the boundary of the half-space of ι_i is created at line 19 to fix its representing rays in space. Another case not encountered in this example arises when the polyhedron consists of a single half-space ($|I| = 1$). In this case, line 4 creates a third ray to indicate on which side the feasible space lies. Note that R never has more than four elements, a case that arises when describing two facing half-spaces.

The remainder of the *hull* function is dedicated to the reconstruction phase. The point and ray sets, returned by *extreme*, are merged at line 6 and 7. At line 8 the size of a square is calculated which includes all points in P. The square has $\langle s, s\rangle$, $\langle -s, s\rangle$, $\langle s, -s\rangle$, $\langle -s, -s\rangle$ as its corners. The square in the running example is depicted at all stages with a dashed line. Fig. 9 shows how each point $p \in P$ is then translated

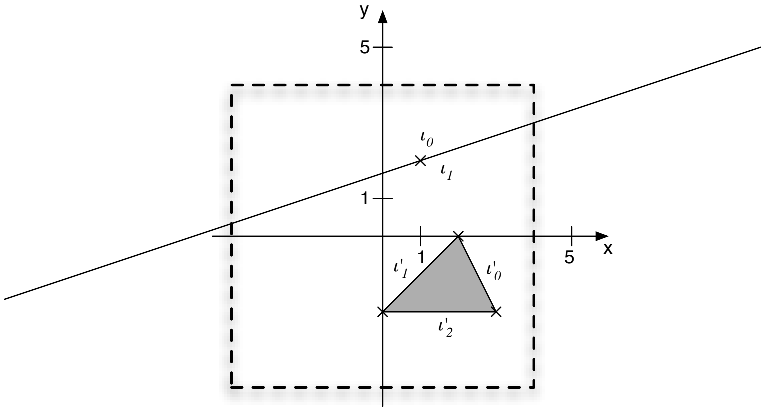


Figure 12. Creating a point in the box for each line

by each ray $r \in R$ yielding the point set Q. The translated points are always outside the square since rays are normalised and then scaled by $2\sqrt{2}s$, which corresponds to the largest extent of the square, namely its diagonal. Lines 13–14 are not relevant to this example as they trap the case when the output polyhedron consists of a single point. Line 16 calculates a feasible point q_p of the convex hull of Q which is not a vertex. This point serves as the pivot point in the classic Graham scan. Firstly, the point set Q is sorted counter-clockwise with respect to q_p. Secondly, all interior points are removed, yielding the indices of all vertices, in the case of the example $k_0, \ldots k_7$ as shown in Fig. 10. What follows is a round-trip around the hull which translates pairs of adjacent vertices into inequalities by calling *connect* at line 23. Whether this inequality actually appears in the result depends on the state of the *add* flag. In our particular example the *add* flag is only set at line 24. Whenever it is set, it is because one of the two vertices lies within the square. The resulting polyhedron is shown in Fig. 11 and consists of the inequalities $\overline{q_{k_4}, q_{k_5}}$, $\overline{q_{k_5}, q_{k_6}}$ and $\overline{q_{k_6}, q_{k_7}}$ which is a correct solution for this example.

2.2.3.3. *Pathological Configurations* The reconstruction phase has to consider certain anomalies that mainly arise in outputs of lower dimensionality. One subtlety in the two dimensional case is the handling of polyhedra that contain lines. This is illustrated in Fig. 12 where the two inequalities ι_0, ι_1 are equivalent to one equation which defines a space that is a line or, equivalently, a set of two opposing rays. The result of translating the vertices by the two rays and their convex hull is shown in Fig. 13. Observe that no point in the square is a

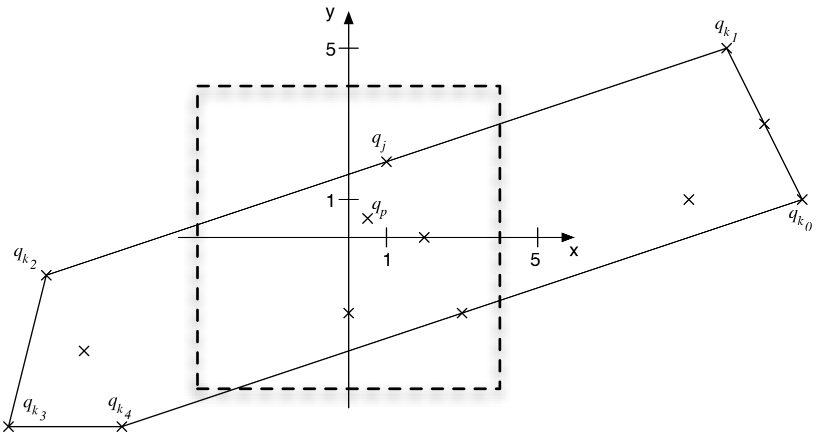


Figure 13. Creating inequalities if one of the points lies in the box

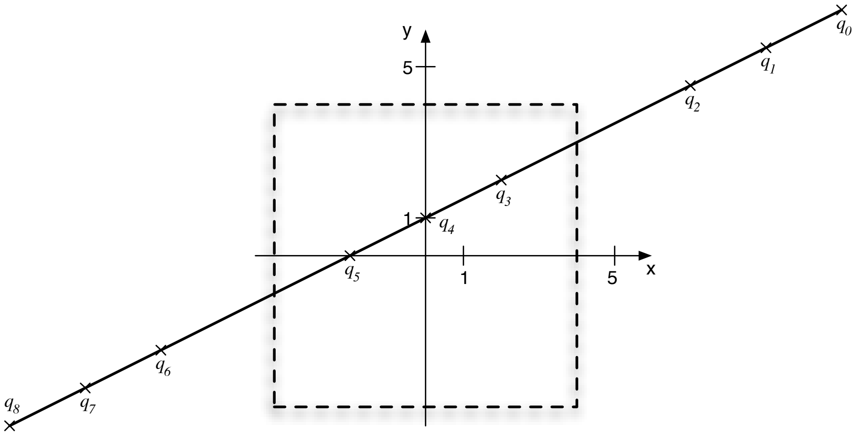


Figure 14. Handling the one-dimensional case

vertex in the hull of Q. Therefore the predicate *inBox* does not hold for the two vertices q_{k_1} and q_{k_2} and the desired inequality $\overline{q_{k_1}, q_{k_2}}$ is not emitted. The same holds for q_{k_4} and q_{k_0}. However, in such cases there always exists a point $q_j \in Q$ with $\overline{q_p, q_{k_i}} \measuredangle \overline{q_p, q_j} < \overline{q_p, q_{k_i}} \measuredangle \overline{q_p, q_{k_{(i+1)\ mod\ m}}}$ which lies in the square. Hence, it is sufficient to search for an index $j \in [k_i + 1, k_{(i+1)\ mod\ m} - 1]$ such that q_j is both in the square and on the line connecting the vertices q_{k_i} and $q_{k_{(i+1)\ mod\ m}}$. The inner loop at lines 26–29 tests if Q contains such a point and sets *add* appropriately. In the example $\overline{q_{k_1}, q_{k_2}}$ and $\overline{q_{k_4}, q_{k_5}}$ are each saturated by a point in the square and are, in fact, the only inequalities in the output.

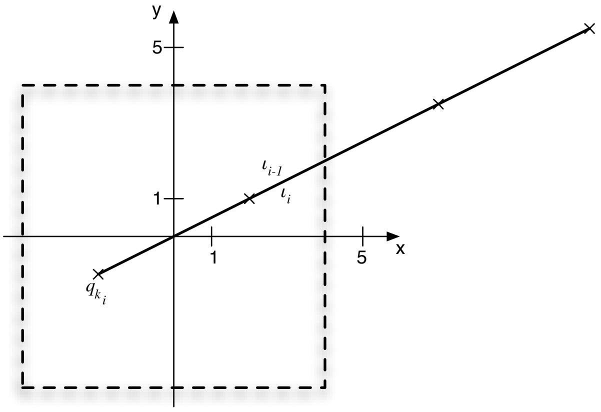


Figure 15. Joining a vertex that lies on a ray with the ray itself

The one-dimensional case is handled by the $m = 2$ tests at line 24 and 30. Fig. 14 illustrates the necessity of the first test. Suppose I_1 and I_2 are given such that $\mathit{extreme}(I_1) = \langle\{q_4, q_5\}, \emptyset\rangle$ and $\mathit{extreme}(I_2) = \langle\{q_3\}, \{r, -r\}\rangle$ where r is any ray parallel to $\overline{q_4, q_5}$. Observe that all points are collinear, thus the pivot point is on the line and a stable sort could return the ordering depicted in the figure. The correct inequalities for this example are $I_{\text{res}} = \{\overline{q_0, q_8}, \overline{q_8, q_0}\}$. The Graham scan will identify $q_{k_0} = q_0$ and $q_{k_1} = q_8$ as vertices. Since there exists $j \in [k_0 + 1, k_1 - 1]$ such that $\mathit{inBox}(s, q_j)$ holds, $\overline{q_0, q_8} \in I_{\text{res}}$. In contrast, although there are boundary points between q_8 and q_0, the loop is not aware of them since sorting the points removed all points between q_8 and q_0. In this case the $m = 2$ test sets *add* and thereby forces $\overline{q_8, q_0} \in I_{\text{res}}$.

Another complication arises when generating line segments as shown in Fig. 15. Observe that the output polyhedron must include q_{k_i} as a vertex whenever $\mathit{inBox}(s, q_{k_i})$ holds. If $\mathit{inBox}(s, q_{k_i})$ holds, the algorithm generates the inequalities $\iota_{i-1} = \overline{q_{k_{(i-1)\ mod\ m}}, q_{k_i}}$ and $\iota_i = \overline{q_{k_i}, q_{k_{(i+1)\ mod\ m}}}$. If $\iota_{i-1} \measuredangle \iota_i < \pi$, then $\{q_{k_i}\} = \mathit{intersect}(\iota_{i-1}, \iota_i)$ and the vertex q_{k_i} is realised. However, if $m = 2$ then $\iota_{i-1} \measuredangle \iota_i = \pi$ which requires an additional inequality to define the vertex q_{k_i}. This is the rôle of the inequalities generated on line 32 and 34. This new inequality ι obeys $\iota_{i-1} \measuredangle \iota < \pi$ and $\iota \measuredangle \iota_i < \pi$ and thus suffices to define q_{k_i}.

The last special case to be considered is a result that is zero dimensional. This case can only occur when both input polyhedra consist of the same single point v. Line 13 traps this case and returns a set of inequalities describing $\{v\}$.

Even though some subtle problems arise in dealing with the occurring pathological cases, note that the zero and one dimensional case

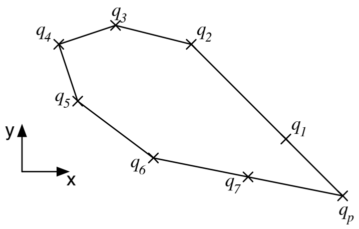


Figure 16. Sorting points that are on a line with the pivot point q_p

only require minute changes to the general two dimensional case. We show in [68] that these modifications are indeed sufficient to ensure that the algorithm is correct on all possible inputs.

2.2.3.4. *Implementation Issues* Special care has to be taken when implementing the actual convex hull algorithm. The pivot point calculated in line 16 of Alg. 5 is likely to have rational coordinates with large numerators and denominators, thereby slowing down the algorithm. The original algorithm of Graham [29] creates an interior point as the pivot point by choosing two arbitrary points q_1, q_2 and searching the point set for a point q_i which does not saturate the line $\overline{q_1, q_2}$. The center of the triangle q_1, q_2, q_i is also guaranteed to be an interior point of Q. While the pivot point found using this method may have a smaller representation, generating inequalities from the resulting sequence of vertices does not guarantee that the first inequality generated will have the smallest angle of all inequalities. The disadvantage is that the resulting sequence of inequalities needs to be rotated until the sequence is in increasing order. Another pitfall is that two or more points may lie on a line with the pivot point and thereby compare as equal even though the points have different coordinates. Graham suggests to retain only the point that lies furthest away from the pivot point, since only the outermost point can ever become a vertex of the convex hull. However, the removal of points is also at odds with our algorithm, as the input point set may not be modified. A common way to address this problem is to perturb the input point set slightly, thereby guaranteeing that no three points lie on a line [27]. For a sound analysis, the perturbation would have to be removed after the convex hull algorithm finishes which complicates the algorithm further.

Our approach to circumvent these problems is to pick one point from the input set that is a definite vertex and to use this point as pivot point (as proposed in [2]). Specifically, we choose the vertex with the largest x-coordinate and the smallest y-coordinate (in that order). Creating inequalities starting with this vertex is guaranteed to generate a sequence with strictly increasing angles. As before, complications arise when ordering points that lie on a line with the pivot point. For instance, $q_1, \ldots q_7$ in Fig. 16 must be returned in increasing sequence. However, the pairs q_1, q_2 and q_6, q_7 compare as equal since they have the same angle to the pivot point q_p. By enhancing the comparison function to lexicographically sort by angle, then by larger x, then by smaller y-coordinate, the points q_6 and q_7 are sorted in correct order, thereby leaving the other pair in the incorrect order q_2, q_1. To ensure that the points q_1 and q_2 appear in increasing sequence, all points at the beginning of the sequence that lie on a line with q_p are reversed.

Finally observe that the loop at lines 26–29 can often be skipped: if the line between q_{k_i} and $q_{k_{(i+1)\ mod\ m}}$ does not intersect with the square, $inBox(s, q)$ cannot hold for any $q \in Q$. In this case, add cannot be set in line 27 and the loop has no effect. Hence, if all corners of the box lie in the feasible region of the potential inequality $\overline{q_{k_i}, q_{k_{(i+1)\ mod\ m}}}$, the loop can safely be skipped.

2.2.4. *Linear Programming and Planar Polyhedra*

The semantics of certain constructs in a program are dependant on the range of a variable. For instance, the evaluation of an array access requires the minimum and maximum value that the index may take on at the given program point. In general, possible values of an expression $\vec{a} \cdot \vec{x}$ with respect to a given polyhedron $P \in Poly$ can be inferred by running a linear program twice; once for an upper and once for a lower bound on the expression. In fact, finding the tightest bounds $[l, u]$ such that $l \leq \vec{a} \cdot \vec{x} \leq u$ holds in P can be implemented more simply in planar space: finding the upper bound u amounts to finding a minimal u such that $P \sqcap_P [\![ax + by \leq u]\!] = P$. To this end, Fig. 17 depicts the line $ax + by = c$ and its coefficient vector $\langle a, b\rangle$ as the vector which points towards the direction of larger values of c. The minimal value of u can be found by increasing u until the $P \sqcap_P [\![ax + by \leq u]\!] = P$ holds. In practice, an inequality ι_i of P can be found in $O(\log n)$ time such that $\theta(\iota_i) \leq \theta(ax + by \leq u) < \theta(\iota_{(i+1)\ mod\ n})$ by performing a binary search on the sorted set of inequalities with $\theta(ax + by \leq u)$ as the key. If $\theta(\iota_i) = \theta(ax + by \leq u)$ then $u = c_i$ where $\iota_i \equiv a_i x + b_i y \leq c_i$. Otherwise if $\iota_i \measuredangle \iota_{(i+1)\ mod\ n} < \pi$ then the intersection point $\langle x', y'\rangle$ of the two inequalities yields the minimal value of u, namely $u = ax' + by'$. Otherwise, no upper bound exists and $u = \infty$. The lower bound l can be inferred in a similar way using the angle $\theta(-ax - by \leq -l)$ as key.

Inferring the bound of a single variable x is a special case that reduces to finding the vertex with the smallest and largest x-coordinate. Incidentally, our implementation stores the upper and lower bounds of

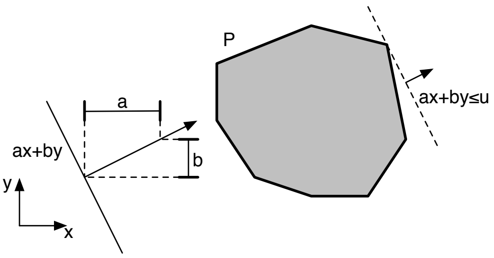


Figure 17. Calculating the maximum of a linear expression in a planar polyhedron

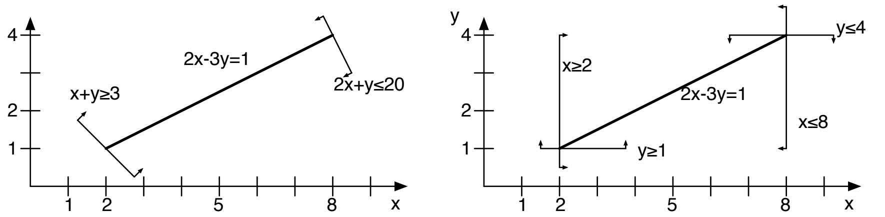


Figure 18. Planar polyhedra may have several representations if they extend in fewer than two dimensions (degrees of freedom)

each variable explicitly, so that the these bounds can simply be read off. The explicit representation of bounds also impacts on the way widening is implemented which is the next and final planar operation.

2.2.5. *Widening Planar Polyhedra*

Calculating the fixpoint of a loop using polyhedra may result in an infinite chain of iterates. By removing inequalities that describe changing facets of a polyhedron, the fixpoint calculation is accelerated and, in fact, forced to converge, a process known as widening [24, 25]. Suppose that two consecutive loop iterates $I_1 = \{x+y \geq 3, 2x+y \leq 20, 2x-3y = 1\}$ and $I_2 = \{x \geq 2, y \geq 1, x \leq 8, y \leq 4, 2x - 3y = 1\}$ are given as shown in Fig. 18. Both systems describe the same set of points $\llbracket I_1 \rrbracket = \llbracket I_2 \rrbracket$ and hence $\llbracket I_1 \rrbracket \nabla \llbracket I_2 \rrbracket = \llbracket I_1 \rrbracket$ since the iterates are stable. Thus, since $I_1 \neq I_2$, widening cannot generally be implemented as a syntactic operation and has to be defined semantically, that is, in terms of entailment [25]. In particular, the original widening is defined such that $\llbracket I_1 \rrbracket \nabla \llbracket I_2 \rrbracket = \llbracket I' \rrbracket$ where $\llbracket I' \rrbracket = \emptyset$ if $\llbracket I_1 \rrbracket = \emptyset$ and otherwise $\iota \in I'$ if $\iota \in I_1$ and $\llbracket I_2 \rrbracket \sqsubseteq_P \llbracket \iota \rrbracket$. While this operation can be implemented similarly to the entailment check on planar polyhedra, an even simpler implementation is possible in the context of the TVPI domain described in the next section, where each planar polyhedron

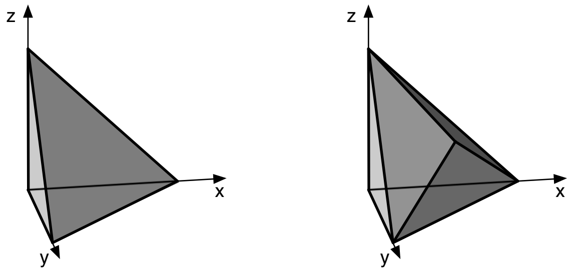


Figure 19. Approximating a 3-dimensional polyhedron with TVPI inequalities

has a unique representation. Given a unique representation of a planar polyhedron (which follows from the reduced product representation), widening can be implemented in purely syntactic way as it reduces to a simple set-difference operation.

The next section elaborates on how to use the presented algorithms on planar polyhedra to implement polyhedra of arbitrary dimension where each inequality has at most two non-zero coefficients.

3. The TVPI Abstract Domain

This section presents the abstract domain of polyhedra where each facet can be described by an inequality that has at most two non-zero variables. These so-called TVPI polyhedra form a proper subset of general convex polyhedra. For instance, consider the inequality set $\{x \geq 0, y \geq 0, z \geq 0, x + y + z \leq 1\}$. The resulting state space is depicted on the left of Fig. 19. This system can be approximated with TVPI inequalities by replacing the inequality $x + y + z \leq 1$ with three inequalities of the form $x + y \leq c_{xy}, x + z \leq c_{xz}$ and $y + z \leq c_{yz}$. The constant c_{xy} can be determined by inserting the bounds for z into $x + y + z \leq 1$, yielding $x + y + [0, 1] \leq 1$. Moving the interval to the right yields $x + y \leq 1 - [-1, 0]$, that is, $x + y \leq 1 + [0, 1]$. Thus, the tightest bounds that can be inferred for $x + y$ is $c_{xy} = 1$, and similarly $c_{xz} = c_{yz} = 1$. The resulting space is depicted on the right of the figure. Note that the best TVPI approximation to the polyhedron $[\![x + y + z \leq 1]\!]$ is an empty TVPI system. Thus, TVPI polyhedra are a strict subset of general polyhedra. However, every polyhedron has a best approximation in the TVPI domain, so that the domain of polyhedra and the TVPI domain form a Galois insertion.

An interesting property of TVPI inequalities is that they are closed under projection. Consider projecting $I = \{2x + 3y \leq 4, -2y + 2z \leq 2\}$

onto the x, z-plane by applying Fourier-Motzkin variable elimination [37] on y. This is carried out by scaling the first inequality by 2, the second by 3 and adding them to yield $2x + 3z \leq 7$ which describes all possible x, z-values of the original polyhedron $[\![I]\!]$. The observation is that projecting out variables of TVPI inequalities removes a common variable and thereby yields an inequality with at most two variables.

Interestingly, when the coefficients of inequalities are normalised to their lowest common denominator, the number of inequalities that can be added through projection is polynomial in the size of the input system [51]. The process of calculating all projections is called closure. In a closed system, the set of inequalities containing the variables $x_i, x_j \in \mathcal{X}$ expresses all information that is available with respect to these variables. In fact, the key idea of the TVPI domain is to apply the planar operations on each x_i, x_j-projection of a closed TVPI system which is equivalent to applying the domain operations of general polyhedra to the whole TVPI system, albeit more efficient.

The principle of applying planar operations on a closed TVPI system is formalised in Sect. 3.1. In particular, Sect. 3.1.1 formalises closure before we show how planar algorithms for checking entailment, calculating the convex hull, and projecting out variables naturally lift to TVPI systems. The liftings for these three operations are presented in Sects. 3.1.2, 3.1.3, and 3.1.4, respectively. Other domain operations are more dependant on how TVPI systems are represented. Section 3.2 therefore presents our choice of representation which separates inequalities over a single variable from those over two. This construction can viewed as a reduced product [22]. We then introduce those operations that benefit from the reduced product representation. These are the removal of redundant inequalities in Sect. 3.2.4, the intersection with new TVPI inequalities in Sect. 3.2.5 and the intersection with inequalities involving more than two variables in Sect. 3.2.6. Dealing with more than two variables is also a concern for linear programming which is therefore considered in Sect. 3.2.7. The reduced product allows for a straightforward implementation of widening, presented in Sect. 3.2.8. We conclude Section 3 with a discussion of related work.

3.1. Principles of the TVPI Domain

For the sake of this section, let $\mathit{var} : \mathit{Ineq} \to \mathcal{P}(\mathcal{X})$ extract the variables that occur in an inequality, that is $\mathit{var}(\vec{a} \cdot \vec{x} \leq c) = \{x_i \mid a_i \neq 0\}$. Let $\mathit{Ineq}^2 \subset \mathit{Ineq}$ denote all TVPI inequalities, that is, $\mathit{Ineq}^2 = \{\iota \in \mathit{Ineq} \mid |\mathit{var}(\iota)| \leq 2\}$. The set of all finite TVPI systems is therefore defined as $\mathit{Two} = \{I \subseteq \mathit{Ineq}^2 \mid |I| \in \mathbb{N}\}$. In contrast to $\mathit{Poly} \subseteq \mathbb{Q}^n$, elements of Two are inequalities rather than sets of convex spaces. This syntactic form is required to distinguish between closed and non-closed TVPI systems. We define these concepts in the next section.

3.1.1. *Closure*
Define a family of syntactic projection operators $\pi_X(I) = \{\iota \in I \mid \mathit{var}(\iota) \subseteq X\}$ for all $X \subseteq \mathcal{X}$. The set of closed TVPI systems can now be defined as $\mathit{Two}^{cl} = \{I \subseteq \mathit{Two} \mid \forall \iota \in \mathit{Ineq}^2 \,.\, [\![I]\!] \sqsubseteq_P [\![\iota]\!] \Rightarrow [\![\pi_{\mathit{var}(\iota)}(I)]\!] \sqsubseteq_P [\![\iota]\!]\}$, that is, a TVPI system is closed if any TVPI inequality ι that is valid in the whole system I is also valid when considering only those inequalities of I that contain variables of ι. Intuitively, this definition implies that all information about a pair of variables $x_i, x_j \in \mathcal{X}$ is expressed as inequalities over these two variables. In particular, combining inequalities such as $ax_i - x_k \leq c_1$ and $x_k + bx_j \leq c_2$ to $ax_i + bx_j \leq c_1 + c_2$ does not add any new information about x_i and x_j. For instance, the system $I = \{x \leq y, y \leq z\}$ is not closed, since $[\![\pi_{\{x,z\}}(I)]\!] = [\![\emptyset]\!] = \mathbb{Q}^2$ although the inequality $x \leq y$ fulfills $[\![I]\!] \sqsubseteq_P [\![x \leq z]\!]$ and $[\![\emptyset]\!] \not\sqsubseteq_P [\![x \leq y]\!]$.

A closed form does always exist within *Two*, as stated by the following proposition (the proof of all propositions can be found in [71]):

Proposition 1. *For any $I \in \mathit{Two}$ there exists $I' \in \mathit{Two}^{cl}$ such that $I \subseteq I'$ and $[\![I]\!] = [\![I']\!]$.*

In fact, any given TVPI system $I \in \mathit{Two}$ can be closed to obtain $I' \in \mathit{Two}^{cl}$ by calculating so-called resultants of I using $\mathit{result} : \mathit{Two} \to \mathit{Two}$, defined as follows:

$$\mathit{result}(I) = \left\{ ae\,z - bd\,y \leq af - cd \;\middle|\; \begin{array}{l} \iota_1, \iota_2 \in I \qquad\quad \wedge \\ \iota_1 \equiv ax + by \leq c \wedge \\ \iota_2 \equiv dx + ez \leq f \wedge \\ a > 0 \wedge d < 0 \end{array} \right\}$$

The purpose of the *result* function is to combine inequalities over the variables x, y and y, z to new inequalities over x, z, thereby making information on x, z explicit that was only implicitly available before. In particular, the above combination resembles Fourier-Motzkin variable elimination where all information on x, z is made explicit in order to remove the variable y from the system. Nelson observed that merely adding inequalities (rather than removing those containing y as in Fourier-Motzkin variable elimination) eventually leads to a closed system [50]. For instance, consider applying the *result* function to the following system of inequalities:

$$I_0 = \{x_0 \leq x_1, x_1 \leq x_2, x_2 \leq x_3, x_3 \leq x_4\}$$

We calculate $I_1 = \mathit{result}(I_0)$ and $I_2 = \mathit{result}(I_0 \cup I_1)$ resulting in the following sets:

$$\begin{aligned} \mathit{result}(I_0) &= \{x_0 \leq x_2, x_1 \leq x_3, x_2 \leq x_4\} \\ \mathit{result}(I_0 \cup I_1) &= I_1 \cup \{x_0 \leq x_3, x_0 \leq x_4, x_1 \leq x_4\} \end{aligned}$$

Here, $I_3 = \mathit{result}(\bigcup_{i=0}^{2} I_i)$ is a fixpoint in that $\mathit{result}(I_3) \subseteq I_3$. An important property of $I \cup \mathit{result}(I)$ is the way it halves the number of variables required to entail a given inequality $\iota \in \mathit{Two}$: suppose $[\![I]\!] \sqsubseteq_P [\![\iota]\!]$ then there exists $I' \subseteq I \cup \mathit{result}(I)$ such that $[\![I']\!] \sqsubseteq_P [\![\iota]\!]$ and I' contains no more than half the variables of I. Lemma 1 formalises this and is a reformulation of Lemma 1b of [50].

Lemma 1. *Let $I \in \mathit{Two}$ and $\iota \in \mathit{Ineq}^2$ such that $[\![I]\!] \sqsubseteq_P [\![\iota]\!]$. Then there exists $Y \subseteq \mathcal{X}$ s.t. $|Y| \leq \lfloor |\mathit{var}(I)|/2 \rfloor + 1$ and $[\![\pi_Y(I \cup \mathit{result}(I))]\!] \sqsubseteq_P \iota$.*

Lemma 1 suggests a way to obtain a closed system by applying the *result* function approximately $\log_2(|\mathit{var}(I)|)$ times to any system of inequalities $I \in \mathit{Two}$. More precisely, a TVPI system can be closed in $O(k^2 d^3 \log(d)(\log(k) + \log(d)))$ steps where d is the number of variables and k the maximum number of inequalities for any pair of variables [72]. This bound can be refined by sorting individual projections (rather than the whole system of inequalities) to $O(k^2 d^3 \log(d) \log(k))$. Empirical evidence suggests that k is bounded by a small constant in practice, such that the bound collapses to $O(d^3 \log(d))$ [72]. In fact, on average, the number of inequalities k needed to describe a set of n random points is merely $O(\sqrt[3]{n})$ [9].

Rather than presenting such an algorithm, Sect. 3.2.5 details how an initially empty inequality system can be incrementally closed each time a new inequality is added. An incremental closure is more amenable to abstract interpretation where the meet operation is mostly used to add a few inequalities to a system before the inequality system is further passed on to entailment checks, join and projection operations.

Interestingly, applying the planar algorithms from the last section to each syntactic projection $\pi_{\{x_i, x_j\}}(I)$ of a TVPI system $I \in \mathit{Two}^{cl}$ in most cases results in a closed system. Besides this practical property, the next sections also attest the correctness of lifting the planar entailment check, join and projection algorithms to the TVPI domain.

3.1.2. *Entailment Check*

We show that checking entailment between two closed TVPI systems can be reduced to checking entailment on each two-dimensional projection.

Proposition 2. *Let $I' \in \mathit{Two}^{cl}$ and $I \in \mathit{Two}$. Then $[\![I']\!] \sqsubseteq_P [\![I]\!]$ iff $[\![\pi_Y(I')]\!] \sqsubseteq_P [\![\pi_Y(I)]\!]$ for all $Y = \{x, y\} \subseteq \mathcal{X}$.*

Note that the proposition does not require both inequality systems to be closed. This observation is interesting when applying operations to individual projections that can lead to a non-closed system.

As a consequence of Proposition 2, it suffices to check that entailment holds for all planar projections which can be checked with the *entails* test presented as Alg. 1 in Sect. 2.2.1.

3.1.3. *Convex Hull*

In order to show that calculating the join of two TVPI polyhedra can be reduced to calculating the convex hull of each planar projection, we define the operation $\curlyvee^S$ as follows:

Definition 1. *The piece-wise convex hull* $\curlyvee^S : \mathit{Two} \times \mathit{Two} \to \mathit{Two}$ *is defined* $I_1 \curlyvee^S I_2 = \cup\{I_{x,y} \mid x, y \in \mathcal{X}\}$ *where* $[\![I_{x,y}]\!] = [\![\pi_{x,y}(I_1)]\!] \curlyvee [\![\pi_{x,y}(I_2)]\!]$.

The following proposition states that calculating the convex hull on each planar projection results in a TVPI system that is closed if the two input TVPI systems were closed. This value of this observation lies in the fact that it is not necessary to calculate the $O(d^3(\log(d)^2))$ complete closure after each convex hull.

Proposition 3. $I_1' \curlyvee^S I_2' \in \mathit{Two}^{cl}$ *if* $I_1', I_2' \in \mathit{Two}^{cl}$.

The following proposition relates the correctness and precision of the convex hull on TVPI systems to planar convex hull operations on each projection $\pi_{\{x_i,x_j\}}(I)$.

Proposition 4. *Let* $I_1', I_2' \in \mathit{Two}^{cl}$. *Then the following holds:*

- $[\![I_1']\!] \curlyvee [\![I_2']\!] \sqsubseteq_P [\![I_1' \curlyvee^S I_2']\!]$
- *If* $I \in \mathit{Two}^{cl}$ *and* $[\![I_1']\!] \curlyvee [\![I_2']\!] \sqsubseteq_P [\![I]\!]$ *then* $[\![I_1' \curlyvee^S I_2']\!] \sqsubseteq_P [\![I]\!]$.

Thus, an efficient way to calculate the convex hull of two closed TVPI systems I_1 and I_2 is to apply the planar convex hull algorithm in Sect. 2.2.3 to each pair $x_i, x_j \in \mathcal{X}$ of the syntactic projections $\pi_{\{x_i,x_j\}}(I_1)$ and $\pi_{\{x_i,x_j\}}(I_2)$. (Note that the corresponding proposition in [72] was stated incorrectly.)

3.1.4. *Projection*

Projection returns the most precise system without a given variable. Semantically, the family of operators $\exists_{x_i} : \mathit{Poly} \to \mathit{Poly}$ is defined as $\exists_{x_i}(P) = \{\langle x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots x_n\rangle \mid \langle x_1, \ldots x_n\rangle \in P, x \in \mathbb{R}\}$. This operator is sometime called the "forget" operator [47] as it removes all information from a polyhedron pertaining to x.

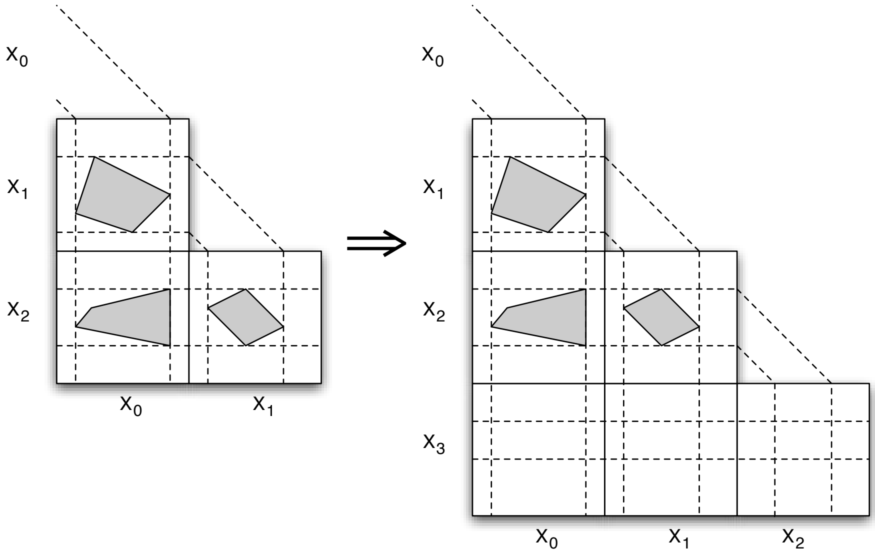


Figure 20. Adding a variable to a TVPI system adds a row to the triangular matrix

An algorithmic definition of projection is easily possible for closed systems. Proposition 5 states that projection coincides with the definition of syntactic projection π, that is, projection can be implemented by removing all those inequalities that contain variables that are to be eliminated. Furthermore, we prove that this operation preserves closure. We commence by defining syntactic projection.

Definition 2. *The syntactic projection operator* $\exists_x^S : Two^{cl} \to Two^{cl}$ *is defined* $\exists_x^S(I) = \cup\{\pi_Y(I) \mid Y \subseteq X \setminus \{x\} \wedge |Y| = 2\}$.

Note that the projection function $\exists_x^S$ above operates on the inequality representation of a polyhedron, rather than a set of points. The following proposition states that the syntactic projection defined above and the projection operator on polyhedra, which operates on sets of points, coincide.

Proposition 5. $\exists_x(\llbracket I' \rrbracket) = \llbracket \exists_x^S(I') \rrbracket$ *and* $\exists_x^S(I') \in Two^{cl}$ *for all* $I' \in Two^{cl}$.

As an example, consider again the system consisting of $x \leq y$ and $y \leq z$. Closure will introduce $x \leq z$. Projecting out y will only preserve $x \leq z$ which coincides with projection in that $\exists_y(\llbracket\{x \leq y, y \leq z\}\rrbracket) = \llbracket x \leq z \rrbracket$.

3.2. Reduced Product Between Bounds and Inequalities

Given that most operations on a closed TVPI system are executed as operations on planar polyhedra over one specific pair of variables, it is prudent to use a data structure for storing TVPI systems that groups the inequalities by the pair of variables occurring in them. Given a domain of $n = |\mathcal{X}|$ variables, $n(n-1)/2$ unique combinations of variables exist which can be stored in a triangular matrix as shown in Fig. 20 for $n = 3$ and $n = 4$. The rows of the triangular matrix are mapped to a one-dimensional array that dynamically resizes. Each projection $\langle x_i, x_j \rangle$ is stored at the index $j(j-1)/2+i$ if $i < j$, see [40]. While indexing into this array is more complicated, it enables the implementation to add and remove variables without copying the matrix. This is important since the size of the matrix grows quadratically with n. Since not all variables $\mathcal{X}$ are present in the polyhedron at all times, it is possible to remove rows that correspond to variables that are projected out and add a new row whenever a variable is mentioned that is not currently represented in the matrix.

3.2.1. *Adding Variables to a TVPI System*

If the underlying array is large enough, a new row with the index $n + 1$ can be added by merely appending n new planar polyhedra to the end of the array, otherwise the matrix has to be copied to a larger array. In the likely case that some of the variables with indices $0 \ldots n$ are bounded from above or from below, these bounds have to be inserted into the newly added polyhedra. Rather than replicating upper and lower bounds of each variable in each planar projection, we chose to implement the TVPI domain as a reduced product [18] between intervals and planar polyhedra that contain TVPI inequalities of the form $ax + by \leq c$ where both coefficients, a and b, are non-zero. Indeed, avoiding the replication of bounds is the main benefit of the reduced product representation as it avoids the need to keep bounds consistent across different projections. The principle is shown in the left schematic drawing of Fig. 20. Here, the dashed lines depict the upper and lower bounds of each variable which are stored separately from the TVPI inequalities. We refer to the projection in row i and column j with $\langle x_i, x_j \rangle$ such that x_i and x_j correspond to the y- and x-axis, respectively. The grey polyhedra over $\langle x_1, x_0 \rangle$, $\langle x_2, x_0 \rangle$ and $\langle x_2, x_1 \rangle$ are defined by these bounds and the sets of TVPI inequalities that are specific to the given projection. In this representation, a fourth variable is introduced by merely adding a new range and three projections

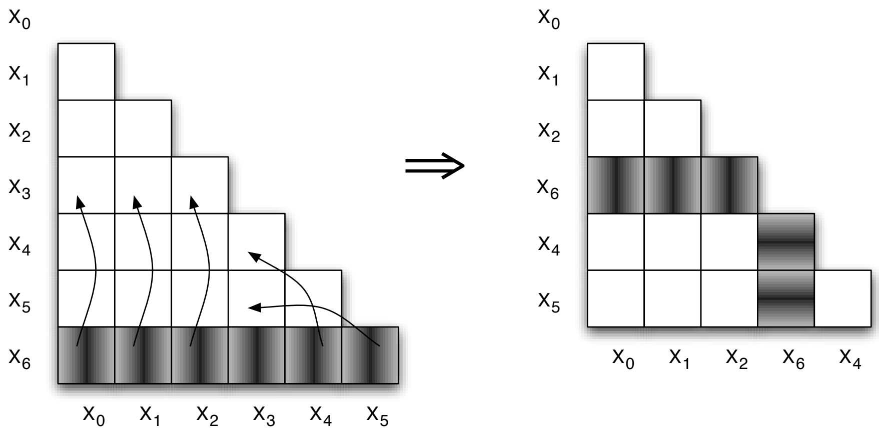


Figure 21. Removing a variable from a TVPI domain that is stored as matrix

without any inequalities, one for each variable pair $\langle x_3, x_0\rangle$, $\langle x_3, x_1\rangle$ and $\langle x_3, x_2\rangle$, as shown in Fig. 20.

3.2.2. *Removing Variables from a TVPI System*

While the triangular matrix can always be extended with a new variable by adding a new row to the matrix, the removal of a variable might require the removal of a row that resides within the triangular matrix. For example, consider the removal of x_3 in the system depicted in Fig. 21. In order to avoid holes in the matrix, the projections of x_3 are replaced with those of the last row in the system, namely x_6. Specifically, the first three projections in the last row, namely $\langle x_6, x_i\rangle$ for $i = 1, \ldots 3$, simply replace the planar polyhedra $\langle x_3, x_i\rangle$ for $i = 1, \ldots 3$ of the row that is to be removed. While the projection $\langle x_6, x_3\rangle$ is merely deleted, the remaining two projections $\langle x_6, x_4\rangle$ and $\langle x_6, x_5\rangle$ cannot overwrite the projections $\langle x_4, x_3\rangle$ and $\langle x_5, x_3\rangle$. This is because the y- and x-axis of $\langle x_6, x_4\rangle$ are x_6 and x_4 but the new projection at the $\langle x_4, x_3\rangle$ entry in the matrix must have x_4 and x_6 as its y- and x-axes. A similar problem arises when moving the $\langle x_6, x_5\rangle$ entry. Thus, prior to replacing these target projections, the variables of the two planar polyhedra have to be swapped which geometrically amounts to a mirroring along the $x = y$ line which is a linear-time operation. Suppose a polyhedron is stored as an array of inequalities that are sorted by angle. Mirroring along the $x = y$ line can then be realised by reversing the order of inequalities with angles in $[0, \pi/2)$ and then separately reversing inequalities with angles in $[\pi/2, 2\pi)$. After these reversals, the x- and y-coefficients in each inequality are swapped.

3.2.3. *Assignments*

Adding and removing rows provides an efficient framework to implement *in situ* updates of variables. Consider an assignment of the form `x=`e where e is an expression and x is the polyhedral variable representing `x`. In the case that `x` does not appear in e, no information on x is needed to calculate the new value of x. In this case, the row storing projections over x can be overwritten with new projections that define the new value of x. Then closure is applied. For instance, if $e \equiv y + 1$, then the projection $\langle x, y\rangle$ contains $\{x = y+1\}$ and all other projections $\langle x, z\rangle$ where $z \neq y$ are empty. Closure will then instantiate these empty projections. Suppose now that e contains x. In the special case where e is of the form $ax + b$, the projections over x can be updated by performing an affine transformation [25]. However, we treat this case no different from the case when e is a more complex (e.g., non-linear or multi-variable) expression that must be approximated. In this general case, a new row is added to hold a temporary variable t. The value of e is then described by inserting inequalities into the projections $\langle t, y\rangle$. In order to assign t to x and to project out t, it is sufficient to merely project out x: since t is stored in the last row of the triangular matrix, the projections containing t overwrite those of x as illustrated in Fig. 21. Thus, updating a variable can be implemented by adding a new row, calculating the result within this new row, and replacing the target variable with the last row.

Next to updates using the meet operation, the performance of the domain hinges on the join and entailment operations. In our implementation, a TVPI system is an array in which each projection is a pointer to a planar polyhedron which can be shared amongst several TVPI systems. Thus, creating a new copy of a system merely requires to copy the pointers in the array. Furthermore, performing minor changes to the copy and calculating the join with the original is cheap: if only few projections are changed, most projections still refer to the same shared polyhedron. In this case, calculating the convex hull on these projections can be avoided since the result is the same shared polyhedron. Similarly, an entailment check holds automatically if the pointers of two projections are the same.

3.2.4. *Redundancy Removal in the Reduced Product*

Closure may add many redundant inequalities to a projection which have to be removed in order to keep the size of the domain in check. Redundancy removal in the TVPI domain requires some substantial changes to the planar redundancy removal algorithm since the domain is represented as a reduced product of interval bounds and TVPI in-

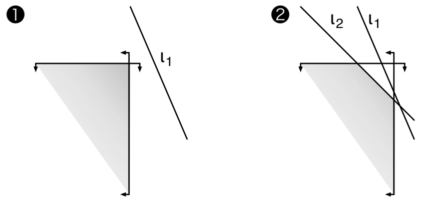


Figure 22. Variations of entailment check between an inequality and interval bounds

equalities. While the basic round-trip algorithm of Sect. 2.2.2 remains intact, special care has to be taken with respect to interval bounds. In particular, they cannot be converted to inequalities since they are often redundant with respect to other TVPI inequalities and would thus be removed. Thus, the redundancy removal algorithm only tests TVPI inequalities for redundancy but has to consult the inequalities and the interval bounds for entailment checks. In particular, special entailment tests are needed whenever inequalities lie in different quadrants, that is, if a bound lies angle-wise between two adjacent inequalities. Fig. 22 shows the two necessary entailment tests between inequalities and interval bounds: The first schematic drawing depicts an inequality ι_1 with no adjacent inequalities in its quadrant. In this case it is necessary to test the inequality against the two nearest interval bounds. The second drawing shows a redundant inequality ι_1 that is the first in that quadrant. This inequality needs to be tested for redundancy with respect to ι_2 and one bound, and ι_2 and the other bound. Similarly, the last inequality in each quadrant needs to be tested against the previous inequality and two nearest interval bounds. In the case where an inequality has a neighbour on either side and within the same quadrant, the normal entailment check can be applied.

In contrast to inequalities, redundant interval bounds cannot be removed but have to be tightened. Interval bounds are tightened in four principal ways as shown in Fig. 23. The first case applies if the angle between the last inequality of a quadrant, ι_1, and the first inequality in the next quadrant, ι_2, is less than π, in which case the inequalities intersect in a point that might imply a tighter interval bound. The second case applies if two inequalities ι_1, ι_2 obey $\iota_1 \measuredangle \iota_2 < \pi$ but this time have an empty quadrant (i.e. two bounds) between them. In this case their intersection point might tighten two bounds at once. If case two has not updated both bounds, one or both bounds might be tighter than what the intersection point suggests, leading to the third case. In this case, one bound is tightened with respect to the intersection point of one of the inequalities and the other bound. The last graph in

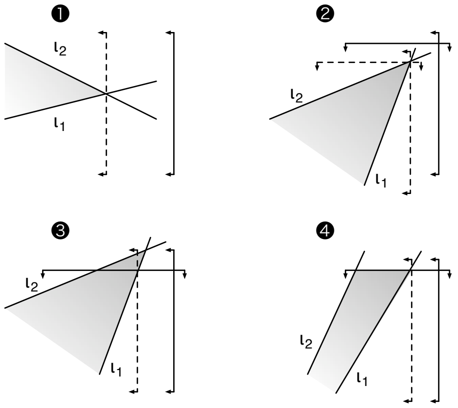


Figure 23. Tightening interval bounds given two adjacent inequalities ι_1, ι_2. Bounds shown with solid lines are tightened to the bounds shown with dashed lines. Case one applies if $\iota_1 \measuredangle \iota_2 < \pi$ and one bound lies between them, case two applies if two bounds lie between ι_1 and ι_2, case three applies to those bounds that case two could not tighten, and case four applies if $\iota_1 \measuredangle \iota_2 > \pi$

Fig. 23 depicts the fourth case where the angle between two inequalities is greater or equal to π. Here the single inequality ι_1 might tighten the adjacent bound with the bound adjacent to ι_2. This completes the suite of entailment checks and tightenings for interval bounds.

The above entailment checks and tightenings have to be adapted to all four cardinal directions. Since a full presentation of the algorithm is repetitive, for brevity, we do not replicate the above cases for the other quadrants. More insightful is the implementation of the incremental closure, which implicitly uses the redundancy removal algorithm when adding inequalities to a projection.

3.2.5. *Incremental Closure*

Calculating the closure of a TVPI system as discussed in Sect. 3.1 can be implemented by a variant of the Floyd-Warshall algorithm [20] which infers the shortest paths between any pair of nodes in a graph. Specifically, the original cubic Floyd-Warshall algorithm on n variables creates n different $n \times n$ matrices where an element at the matrix entry $\langle x_i, x_j \rangle$ describes the cost of traversing the graph from node x_i to x_j. Note that the cost of travelling from x_i to x_j might be different to the cost of travelling from x_j to x_i. In the context of the triangular matrix of a TVPI domain, the planar polyhedron $\langle x_i, x_j \rangle$ represents both directions of travel due to the fact that inequalities of that polyhedron may have negative as well as positive coefficients. Apart from this difference, the Floyd-Warshall algorithm can be adapted to close a TVPI system

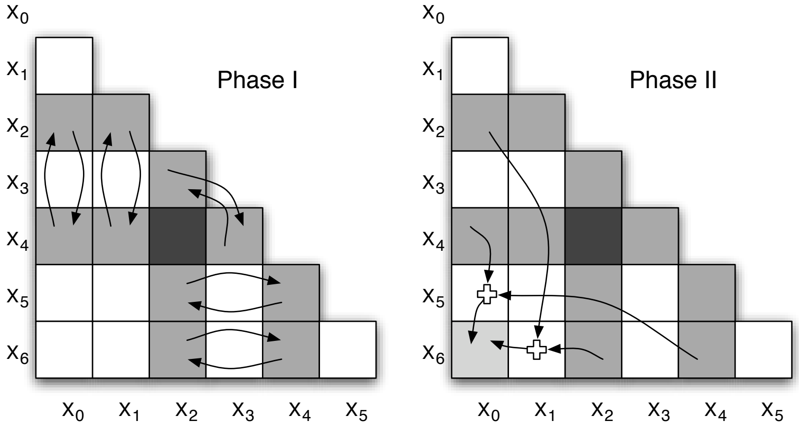


Figure 24. Incremental closure after changing the projection x_2, x_4. New inequalities in x_3, x_5 serve to calculate distance-one projections (Phase I). For each set of new inequalities in these, distance-two projections are calculated (Phase II)

when the operation of adding the cost of two edges between $\langle x_i, x_j \rangle$ and $\langle x_j, x_k \rangle$ is replaced by calculating the resultants of the inequalities of the planar polyhedra over $\langle x_i, x_j \rangle$ and $\langle x_j, x_k \rangle$. Updating the edge $\langle x_i, x_k \rangle$ with the smaller of current cost and the cost via x_j corresponds to inserting these resultants into the target projection $\langle x_i, x_k \rangle$.

Unfortunately, calculating a complete closure of a TVPI system is at odds with the needs of program analysis. Here, inequalities are usually added one by one through conditionals or assignments. Furthermore, adding inequalities to the domain is interleaved with variable removal and the calculation of joins. Thus, an incremental closure is required that takes a closed system and a set of inequalities that are to be intersected with a given projection $\langle x_i, x_j \rangle$ and which returns a new, closed system in which the inequalities are incorporated. The incremental closure uses operations similar to those used in the Floyd-Warshall algorithm, albeit with a different strategy. Specifically, the Floyd-Warshall algorithm calculates a sequence of n matrices $m_1, \ldots m_n$ such that the distance between nodes in matrix m_i that are no more than i edges apart is minimal. In contrast, the incremental closure operates on a system in which all distances are minimal, except those that involve a particular projection $\langle x_i, x_j \rangle$. Thus, the task is to update all other projections with the new information available on $\langle x_i, x_j \rangle$. This task, illustrated in Fig. 24, is performed as follows.

In the shown example, the polyhedron for $\langle x_4, x_2 \rangle$ is intersected with new inequalities. In order to close the system, all inequalities that are new and which were non-redundant have to be propagated to all other projections. This propagation is performed in two phases. The first phase propagates information to all projections that share one variable with the inequalities over $\langle x_4, x_2 \rangle$. For instance, all inequalities in the projection $\langle x_2, x_0 \rangle$ are combined with the new inequalities, yielding inequalities over $\langle x_4, x_0 \rangle$. Before these inequalities are inserted into the projection $\langle x_4, x_0 \rangle$, the current inequalities over $\langle x_4, x_0 \rangle$ are combined with those over $\langle x_4, x_2 \rangle$, yielding new inequalities for the symmetric case, namely $\langle x_2, x_0 \rangle$. Analogously, the new inequalities over $\langle x_4, x_2 \rangle$ are combined with $\langle x_2, x_i \rangle$ and $\langle x_4, x_i \rangle$ for $i = 1, 3, 5, 6$. At this point, all resultants that share one variable with the projection $\langle x_4, x_2 \rangle$ are up-to-date. In terms of a graph, all nodes (variables) that are immediate neighbours of the nodes x_2 and x_4 are up-to-date; the polyhedra on these edges are called distance-one results. The second phase uses the distance-one results to update the remaining projections that have no variables in common with the $\langle x_4, x_2 \rangle$ projection. Specifically, each projection $\langle x_i, x_j \rangle$ where $i, j \in \{0, 1, 3, 5, 6\}, i \neq j$ is updated by calculating the resultants of the projections $\langle x_i, x_2 \rangle$ and $\langle x_2, x_j \rangle$ in addition to the resultants of the projections $\langle x_i, x_4 \rangle$ and $\langle x_4, x_j \rangle$. For instance, as shown on the right of Fig. 24, the projection $\langle x_6, x_0 \rangle$ is updated by calculating the resultants of $\langle x_6, x_2 \rangle$ and $\langle x_2, x_0 \rangle$ followed by $\langle x_6, x_4 \rangle$ and $\langle x_4, x_0 \rangle$. The projections $\langle x_i, x_j \rangle$ are called distance-two results because in the context of the graph interpretation, these edges are two nodes away from the nodes x_2 and x_4 for which closure is run. By adapting an argument for the Octagon domain [47], it can be shown that after calculating all distance-two projections, the TVPI system is closed.

Algorithm 6 sketches the structure of the incremental closure algorithm. The shown algorithm assumes that the domain is not implemented as a reduced product between TVPI inequalities and intervals, thereby simplifying the presentation significantly. Furthermore, the algorithm does not assume that the projections are stored in a triangular matrix. In particular, we define the inequality sets $I_{\{i,j\}}$ for every variable set $\{x_i, x_j\}$. As the index is a set, $I_{\{i,j\}} = I_{\{j,i\}}$ follows. Furthermore, the sets are not a partitioning of I since inequalities ι with $var(\iota) = \{x_i\}$ appear in all sets $I_{\{x_i,x_j\}}$ with $j \neq i$. The correctness of this simplified algorithm is stated below.

Proposition 6. *Given $I_{new} \in Two$ with $|var(I_{new})| = 2$ and let $I \in Two^{cl}$. Moreover, let $I' = intersect(I, I_{new})$. Then $[\![I \cup I_{new}]\!] = [\![I']\!]$ and $I' \in Two^{cl}$.*

Note that the above algorithm only handles inequalities with exactly two variables. Inserting inequalities over a single variable can be im-

Algorithm 6 Intersection with inequalities over x_j, x_k and closure

```
procedure $intersect(I, I_{new})$ where $I, I_{new} \subseteq Ineq$
$\{x_j, x_k\} \leftarrow var(I_{new})$
$I_{\{i,j\}} \leftarrow \{\iota \in I \mid var(\iota) \subseteq \{x_i, x_j\}\}$
$I'_{\{j,k\}} \leftarrow nonRedundant(I_{\{j,k\}} \cup I_{new})$
if $[\![I'_{\{j,k\}}]\!] = \emptyset$ then
  return $\{0 \leq -1\}$
end if
$n \leftarrow |var(I)|$
for $i \in [0, n-1] \setminus \{j, k\}$ do
  $I'_{\{j,i\}} \leftarrow nonRedundant(I_{\{j,i\}} \cup result(I'_{\{j,k\}} \cup I_{\{k,i\}}))$
  $I'_{\{k,i\}} \leftarrow nonRedundant(I_{\{k,i\}} \cup result(I'_{\{k,j\}} \cup I_{\{j,i\}}))$
  if $[\![I'_{\{j,i\}}]\!] = \emptyset \vee [\![I_{\{k,i\}'}]\!] = \emptyset$ then
    return $\{0 \leq -1\}$
  end if
end for
for $x \in [0, n-1] \setminus \{j, k\}$ do
  for $y \in [x+1, n-1] \setminus \{j, k\}$ do
    $I'_{\{x,y\}} \leftarrow nonRedundant(I_{\{x,y\}} \cup result(I'_{\{x,j\}} \cup I'_{\{j,y\}}) \cup$
                                          $result(I'_{\{x,k\}} \cup I'_{\{k,y\}}))$
    if $[\![I'_{\{x,y\}}]\!] = \emptyset$ then
      return $\{0 \leq -1\}$
    end if
  end for
end for
return $\{I'_{x,y} \mid 0 \leq x < y < n\}$
```

plemented by a simpler algorithm that merely updates the upper and lower bounds of a single row/column. Adding inequalities with more than two variables is considered in the next section.

3.2.6. *Approximating General Inequalities*

Figure 19 at the beginning of this section depicts the problem of adding the inequality $x + y + z \leq 1$ over three variables to the TVPI domain. The resulting domain is necessarily an approximation as only inequalities with at most two variables can be represented. In general, calculating the intersection of $I \in Two$ with an inequality of the form $a_1 x_1 + \ldots a_n x_n \leq c$ can be approximated by inserting the set of inequalities $a_j x_j + a_k x_k \leq c - c_{j,k}$ into I where $1 \leq j < k \leq n$, $c_{j,k} = minExp(\sum_{i \in [1,n] \setminus \{j,k\}} a_i x_i, I)$ and $minExp(e, P)$ calculates the minimum of an expression e in P using linear programming. In case $c_{j,k}$ is unbounded, no approximation is possible. The number of inequalities that are generated this way may be quadratic in the number of non-zero coefficients n. In practice, programs rarely give rise to inequalities with more than three variables, so that the number of TVPI inequalities needed to approximate a single inequality is not a bottleneck.

Surprisingly, the presented approximation of inequalities is not always optimal: Consider the task of adding $x - 2y + z \leq 0$ to the closed TVPI system $I = \{x - y = 0\}$. Since neither x, $-2y$ nor z have an upper bound, the above approximation would fail to deduce any information. However, the new inequality can be rewritten to $x - y - y + z \leq 0$ and substituting the inequality $x - y = 0 \in I$ yields the approximation $-y + z \leq 0$. While a better approximation algorithm that can deduce this relationship is certainly desirable, we observed no precision loss in practice. In fact, many inequalities with more than two variables can be reduced to TVPI inequalities through substitution, by making use of equality relationships between variables.

In order to calculate the above approximation to an n-dimensional inequality, an algorithm for *minExp* is required, which is discussed next.

3.2.7. *Linear Programming in the TVPI Domain*

In the context of a TVPI system a linear expression can be minimised straightforwardly using general linear programming techniques such as Danzig's Simplex method. Interestingly, a linear programming algorithm for arbitrary linear cost functions that exploits the special structure of a TVPI system was proposed by Wayne [77]. This algorithm runs in $O(m^3 n^2 \log A)$ where m is the number of inequalities, n the number of variables and A the upper bound on the absolute value of the constants. Observe that for a closed TVPI system, a TVPI objective function can be minimised using the $O(\log k)$ algorithm presented in Sect. 2.2.4 on the appropriate projection. For objective functions with more than two variables, Prop. 5 states that linear programming may be run on just those TVPI inequalities whose variables appear in the expression to be minimised. This reduces the number of inequalities m that need to be considered which is important since closure introduces many redundant inequalities. Nevertheless, implementing a general or specialised linear programming algorithm is not attractive due to the large number of inequalities that can arise in a closed system. Hence, we approximate the minimum of a linear expression over several variables using efficient linear programs over two variables. For instance, the minimum of $c_a x_a + c_b x_b + c_c x_c \in Lin$ in the TVPI domain I is approximated by the minimum of the following expressions:

$$\begin{gathered} \mathit{minExp}(c_a x_a + c_b x_b, I) + \mathit{minExp}(c_c x_c, I) \\ \mathit{minExp}(c_a x_a + c_c x_c, I) + \mathit{minExp}(c_b x_b, I) \\ \mathit{minExp}(c_b x_b + c_c x_c, I) + \mathit{minExp}(c_a x_a, I) \end{gathered}$$

Similarly, the minimum value of an expression $c_a x_a + c_b x_b + c_c x_c + c_d x_d \in \mathit{Lin}$ is the minimum of the following expressions:

$$\begin{gathered} \mathit{minExp}(c_a x_a + c_b x_b, I) + \mathit{minExp}(c_c x_c + c_d x_d, I) \\ \mathit{minExp}(c_a x_a + c_c x_c, I) + \mathit{minExp}(c_b x_b + c_d x_d, I) \\ \mathit{minExp}(c_a x_a + c_d x_d, I) + \mathit{minExp}(c_b x_b + c_c x_c, I) \end{gathered}$$

The last operation that needs to be lifted from planar polyhedra to TVPI polyhedra is widening which is the topic of the next section.

3.2.8. *Widening of TVPI Polyhedra*

As mentioned in Sect. 2.2.5, widening the individual planar projections of the TVPI domain is simpler if the representation of each projection is unique. This is the case for the reduced product representation where each projection is defined by interval bounds and TVPI inequalities. For instance, Figure 18 on page 26 shows two inequality sets that describe the same polyhedron. While the left graph shows a non-redundant set of TVPI inequalities, two of the interval bounds in the right graph are redundant. However, the representation using bounds and TVPI inequalities is unique in that no other set of bounds and inequalities defines the same polyhedron. As a consequence, widening, which removes facets of a polyhedron that are unstable, reduces to a simple set-difference operation and we therefore omit the pseudo-code.

Note that widening each planar projection results in a TVPI system that is not closed in general. For instance, let $I_1 = \{x \leq y + 1, y \leq z + 1, x \leq z + 1\}$ and $I_2 = \{x \leq y + 1, y \leq z + 1, x \leq z + 2\}$ represent two consecutive loop iterates. The result of $I = I_1 \nabla I_2$ discards the inequality $x \leq z+1$ as it has changed to $x \leq z+2$. However, $\mathit{result}(I) = \{x \leq z+2\} \notin I$, thus the widened TVPI system I is not closed. In fact, $\mathit{result}(I) \cup I = I_2$ which hints at the fact that closure might interfere with widening in that it re-introduces inequalities that were widened away. This phenomenon has been observed by Miné in the context of the Octagon domain [47]. The solution to this termination problem is merely to avoid closing the state that is stored at a widening point.

While widening is a prerequisite to ensure that a fixpoint calculation using the TVPI domain will terminate, it may not be sufficient. Infinite chains manifest themselves in a continuously growing number of inequalities and infinitely increasing coefficients. While widening tackles both aspects, it is not always sufficient to ensure that coefficients within inequalities remain tractable. Section 4 therefore presents techniques to reduce the size of coefficients by tightening the inequalities around the set of integral points that they enclose.

3.3. Related Work

Using n-dimensional polyhedra as an abstract domain for program analysis is expressive, but expensive [25]. Recent proposals suggest only inferring certain inequalities that are deemed to be important to prove a property [56], only use special geometric shape of polyhedra [13], impose a fixed dependency between variables [57] or to simply approximate the exponential operations when the size of the system becomes too large [69]. In contrast, the TVPI domain limits the precision of the inferred polyhedra up front. TVPI polyhedra form a so-called weakly relational domain and thereby constitute a proper subclass of general polyhedra. Other sub-classes include Difference Bounds Matrices (DBMs for short) [5, 61, 46], the Octagon domain [47] and the Octahedron domain [17]. The abstract domain of DBMs represents inequalities of the form $x_i - x_j \leq c_{ij}$, $x_i, x_j \in \mathcal{X}$ by storing c_{ij} in an $n \times n$ matrix such that the entry at position $\langle i, j \rangle$ is c_{ij}. A special value ∞ is stored at this position if $x_i - x_j$ is not constrained. Closure is computed with an all-pairs Floyd-Warshall shortest-path algorithm that is $O(n^3)$ and echos ideas in the early work of Pratt [52]. The Octagon domain [47] represents inequalities of the form $ax_i + bx_j \leq c$ where $a, b \in \{1, 0, -1\}$ and $x_i, x_j \in \mathcal{X}$. The key idea of [47] is to simultaneously work with a set of positive variables x_i^+ and negative variables x_i^- and consider a DBM over $\{x_1^+, x_1^-, \ldots, x_n^+, x_n^-\}$ where $n = |\mathcal{X}|$. Then $x_i - x_j \leq c$, $x_i + x_j \leq c$ and $x_i \leq c$ can be encoded respectively as $x_i^+ - x_j^+ \leq c$, $x_i^+ - x_j^- \leq c$ and $x_i^+ - x_i^- \leq 2c$. Thus a $2n \times 2n$ square DBM matrix is used to store this domain. The Octagon domain has been successfully applied to verify large-scale embedded software [11, 12]. While the matrix representation makes adding and removing variables cumbersome, matrix elements can be simple integers or floating point variables rather than arbitrary-precision integers as required for the TVPI domain. In fact, the operations of the Octagon abstract domain are so simple that they can be implemented efficiently on high-end graphics hardware [8]. Even then, the cubic closure is too expensive when considering the number of variables that arise in some applications. One solution is to restrict the relational information to so-called packs of variable and use one Octagon domain for each pack [47]. An alternative approach is to use decoupling techniques developed in the context of general polyhedra

Table I. An overview of TVPI operations and their properties.

		closed		
operation	implementation	inputs	result	running-time
join	piece-wise convex hull	yes	yes	$O(d^2k\log(k))$
meet	full closure	no	yes	$O(d^3k^2\log(d)\log(k))$
meet	incremental closure	yes	yes	$O(d^2k^2\log(k))$
subset	piece-wise entailment	first	—	$O(d^2k)$
widen	piece-wise widening	first	no	$O(d^2k)$

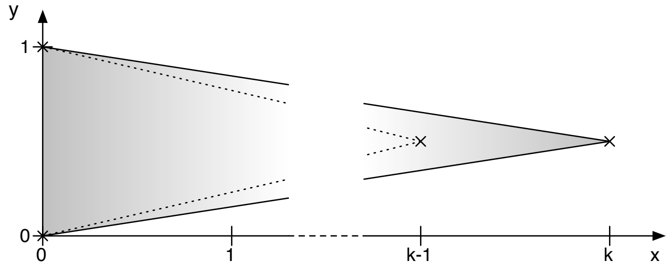


Figure 25. Calculating a new set of Cutting Planes will refine the rational vertex $\langle k, \frac{1}{2}\rangle$ to $\langle k-1, \frac{1}{2}\rangle$. Thus $k-1$ more steps are necessary to obtain a $\mathbb{Z}$-polyhedron

[31] where linear independence is detected between sets of constraints and exploited by applying meet and join on these smaller sets.

The Octagon domain was generalised into the Octahedron domain [17], allowing more than two variables with zero or unary coefficients whilst maintaining a hull operation that is polynomial in the number of variables.

Finally, to aid comparison with other domains, Table 3.3 summarises the different operations on TVPI polyhedra and their properties. Here, d denotes the dimension of the TVPI system and k the maximal number of inequalities in each projection.

4. The Integral TVPI Domain

Most properties that are of interest in static analysis can be expressed with integral numbers. Hence, it is possible to restrict the inferred polyhedra to the contained integral points. In fact, shrinking a polyhedron around the contained integral points is highly desirable for precision as well as for performance, as discussed in Sect. 4.1. However, this task is computationally hard [58]. A first step towards an integral TVPI domain is to tighten each individual inequality $ax + by \leq c$, $a, b, c \in \mathbb{Z}$ by replacing it with the inequality $(a/d)x + (b/d)y \leq \lfloor c/d \rfloor$ where $d = \gcd(a, b)$. Note that every integral point $\langle x', y' \rangle$ with $ax' + by' \leq c$ satisfies the tightened inequality since $ax' + by'$ as well as $(ax' + by')/d$ are integral and thus $(ax' + by')/d \leq \lfloor c/d \rfloor$ [55]. However, tightening individual inequalities is not enough to ensure that the vertices of the polyhedron are integral and additional inequalities need to be added. One way to add these extra inequalities is Gomory's famous Cutting Plane method [58, Chap. 23]. This method systematically infers inequalities $ax + by \leq c$ for a given polyhedron $I \in \mathit{Two}$ such that $[\![ax + by \leq c]\!] \subseteq [\![I]\!]$. The tightened inequality $(a/d)x + (b/d)y \leq \lfloor c/d \rfloor$ where $d = \gcd(a, b)$ is then added to the representation of P, thereby cutting off space of I that contains no integral points. This process is repeated until no more inequalities $ax + by \leq c$ can be inferred in which $\lfloor c/d \rfloor < c/d$ at which point the polyhedron is integral. The method terminates after generating a finite number of cutting planes; however, the number of cutting planes may be exponential in the diameter of the polyhedron. This is illustrated by an example presented in [58, p. 344]: consider the rational polyhedron shown in Fig. 25 that is defined by the vertices $\langle 0, 0 \rangle$, $\langle 0, 1 \rangle$ and $\langle k, \frac{1}{2} \rangle$. One step of Gomory's algorithm infers new inequalities such that $\langle k - 1, \frac{1}{2} \rangle$ is a new vertex. By induction, $k - 1$ further steps are necessary to derive the $\mathbb{Z}$-polyhedron containing only the vertices $\langle 0, 0 \rangle$ and $\langle 0, 1 \rangle$.

While all but the last planes in this example were redundant, even the number of non-redundant inequalities that need to be added to define an integral polyhedron can be exponential in the number of inequalities that describe the rational input polyhedron [58]. Thus, it is not possible to implement an efficient (polynomial) static analysis using $\mathbb{Z}$-polyhedra as the abstract domain. For the special case of planar polyhedra, Harvey proposed an efficient algorithm to shrink a rational polyhedron around the contained integral points [33]. Section 4.2 presents Harvey's algorithm and its implementation in the context of the TVPI domain when it is realised as reduced product between interval bounds and TVPI inequalities. Not surprisingly, shrinking each planar projection is not sufficient to obtain an integral n-dimensional TVPI system. In fact, testing whether a TVPI polyhedron has an integral solution is NP-complete [42]. Hence, Sect. 4.3 discusses how Harvey's algorithm can be combined with the TVPI closure presented in the previous section to approximate an integral TVPI domain.

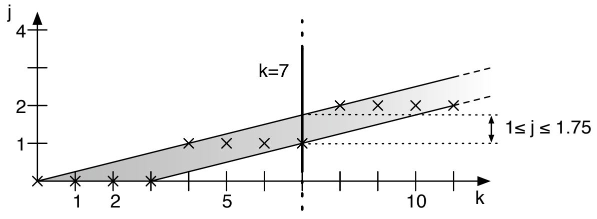


Figure 26. The state space after executing `j=k/4` where `j` and `k` are integers. The crosses mark possible variable valuations. The dashed lines denote the admissible solutions for j after intersecting the polyhedron with $k = 7$

4.1. The Merit of $\mathbb{Z}$-Polyhedra

This section motivates the use of $\mathbb{Z}$-polyhedra in an analysis: On one hand, the precision of an analysis can be improved by removing non-integral state space. On the other hand, shrinking the state space around the contained integral points avoids excessive growth of coefficients in the inequalities that describe the polyhedron and limits the number of inequalities in each projection. The following sections discuss each aspect in turn.

4.1.1. *Improving Precision*

Restricting the solution set of a polyhedron to integral points can improve the precision of an analysis to the extent that certain properties can be verified which are too coarsely approximated when using rational polyhedra. Spurious state space that contains no integral points may be transformed by scaling, that is, evaluating multiplication operations, to a state that contains spurious integral points. For instance, consider Fig. 26 which shows the state space after executing the first assignment in the following C function:

```
void f(unsigned int k) {
  int i,j;
  j = k/4;
  i = j*2;
  if (k==7) { assert(i<3); }
};
```

On nearly all architectures, integer division rounds towards zero and, when assuming that $0 \leq k \leq 2^{32} - 1$, the smallest polyhedron that contains all solutions of the division `j=k/4` contains $4j \leq k$ and $k - 3 \leq 4j$ since the remainder that is discarded is at most 3. The resulting polyhedron is $P = [\![\{0 \leq k \leq 2^{32} - 1, 4j \leq k \leq 4j + 3\}]\!]$ which is shown in grey. The multiplication `i=j*2` adds $i = 2j$ to the description, yielding $2i \leq k \leq 2i + 3$ as the relationship between k and i. The assertion in the branch of the conditional therefore does not appear to hold since with $k = 7$, it only follows that $2i \leq 7 \leq 2i+3$, i.e. $i \in [2, 7/2]$. However, when $i = 3$ then $j = 3/2$ which is not a possible state in the actual program. In fact, the largest value of j for $k = 7$ is 1, and hence the maximal value for i is 2. The necessary precision to verify the assertion can be attained by shrinking the polyhedron P around the containing integral points after the intersection with $k = 7$: While the possible rational values for j are $[1, 7/4]$, the only integral point in this polyhedron $P \sqcap_P [\![\{k = 7\}]\!]$ satisfies $j = 1, k = 7$ which indeed implies $i = 2$ and thus verifies the assertion.

4.1.2. *Limiting the Growth of Coefficients*

While improved precision is important in some circumstances, a more pressing reason to perform tightening around the integral grid is the growth of coefficients that can occur otherwise. Specifically, repeated application of the join operator during a fixpoint calculation can lead to coefficients that are excessively large [69]. Note that arbitrarily large coefficients can arise even in the planar case when expressing floating point computations using rational polyhedra [64]. In principle, widening can be applied to remove inequalities with excessive coefficients. However, this may incur a precision loss that is difficult to understand and anticipate when interpreting the results of an analysis. Even when pursuing this idea, the question of whether an inequality contains an excessively large coefficient, and should therefore be discarded, has no straightforward answer. For instance, wrapping the variable x to an unsigned 32-bit integer in the polyhedron $[\![\{x = y - 1, 0 \leq y \leq 1\}]\!]$ yields $\{x + (2^{32} - 1)y = 2^{32} - 1, 0 \leq y \leq 1\}$ which is the most precise set of inequalities that contains the two integral points and removing any of these inequalities would discard valuable information. In contrast, calculating intersection points between inequalities with coefficients as large as 2^{32} can easily generate numbers that exceed 64-bit integers which requires expensive arbitrary-precision arithmetic. Thus a threshold lower than 2^{32} is desirable which, however, would compromise the polyhedron above. A more principled way to prevent coefficients from growing excessively is, again, to shrink the polyhedron around its integral points. For instance, adding the inequality $x \leq 7$ to the above system $\{x + (2^{32} - 1)y = 2^{32} - 1, 0 \leq y \leq 1\}$ results in a polyhedron that only contains the integral point $\langle x, y\rangle = \langle 0, 1\rangle$. Tightening the rational polyhedron around this integral point results in the inequality set $\{0 \leq x \leq 0, 1 \leq y \leq 1\}$ which contains none of the large coefficients of the rational system. In general, the coefficients of inequalities in a $\mathbb{Z}$-polyhedron are bound by the admissible range of the variables in the inequalities. For instance, the wrapped system above constitutes an inequality set that spans the whole 32-bit range of x and whose coefficients are bound by 2^{32}. In fact, this system represents the worst-case scenario as any other system of inequalities that contains the values 0 and $2^{32}-1$ for x has the same or smaller coefficients for x. Thus, tightening combined with wrapping guarantees upper bounds on the coefficient sizes.

4.1.3. *Limiting the Number of Inequalities*

While limiting the size of coefficients, integer tightening may add additional inequalities as well as make inequalities that do not cut off any integral points redundant. However, the number of inequalities in a planar $\mathbb{Z}$-polyhedron is, in fact, polynomial in the absolute value of any vertex coordinate. To illustrate this, observe that any inequality in a $\mathbb{Z}$-polyhedron connects two integral vertices and that its slope is different from all other inequalities since it would otherwise be redundant. Consider the sequence of points $\langle 1, 1\rangle, \langle 2, 3\rangle, \langle 3, 6\rangle, \ldots, \langle k, \sum_{i=1}^{k} i\rangle$ that forms the vertices of the $\mathbb{Z}$-polyhedron defined by the inequality sequence $x - y \leq 0, 2x - y \leq -1, 3x - y \leq -3, \ldots$. This non-redundant sequence gives the minimal increase in the y-coordinate for each unit increase in the x-coordinate whilst preserving convexity. This minimal sequence covers a range of $\frac{k(k-1)}{2}$ units on the y-axis for k units on the x-axis. Point reflection of this inequality sequence yields $\ldots, x - 3y \leq 1, x - 2y \leq 0, x - y \leq 0$ which spans $\frac{k(k-1)}{2}$ units on the x-axis for k units on the y-axis. Thus, the maximal range spanned by combining the two sequences to one sequence of $2k-1$ inequalities is $k + \frac{k(k-1)}{2} = \frac{k(k+1)}{2}$ on each axis. Thus, we observe that the maximal number of inequalities is $O(\sqrt{m})$ where m is the size of the smallest box that contains all vertices of the $\mathbb{Z}$-polyhedron. The force of this result is that the number of inequalities is polynomial in the size of the bounding box. Since the running time of all domain operations depends on the number of inequalities, this implies that all operations are polynomial. Together with the limit on the size of coefficients in inequalities, this implies that these polynomial operations of the integral TVPI domain are, in fact, strongly polynomial.

The next sections details the tightening process that calculates planar $\mathbb{Z}$-polyhedra and its implications on the closure.

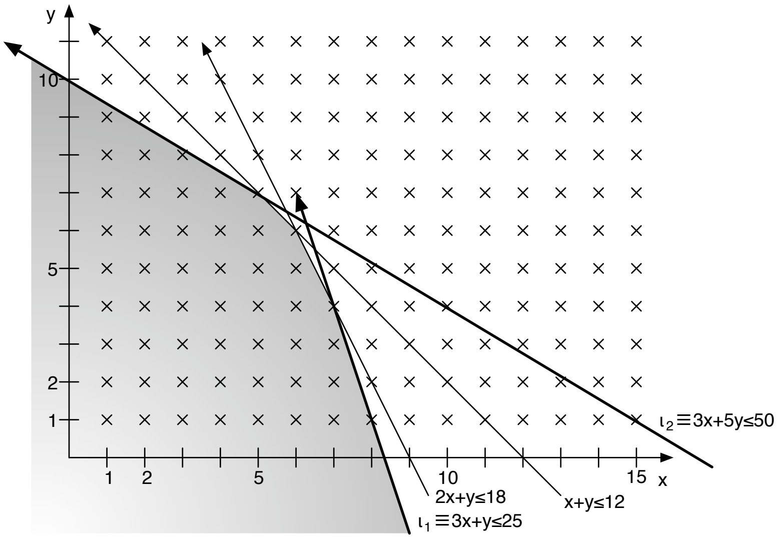


Figure 27. Given two neighbouring inequalities, Harvey's algorithm calculates so-called cuts that tighten these inequalities around the contained integral points

4.2. Harvey's Integral Hull Algorithm

The integral TVPI domain used in our analysis is based on Harvey's integral hull algorithm which tightens a planar polyhedron around the contained integral points. The idea of the integral hull algorithm is to calculate cutting planes between adjacent inequalities. The number of new cutting planes is bounded logarithmically by the size of the (coefficients of) the polyhedron, hence, Harvey's algorithm runs in $O(n \log A)$ where A represents the maximum coefficient in the inequality set. Calculating cutting planes between two adjacent inequalities is presented next. Section 4.2.2 adapts the integral hull algorithm to the reduced product between intervals and TVPI inequalities with two non-zero coefficients. Our algorithm is novel in that it exploits the representation of the TVPI domain as a reduced product between interval bounds and TVPI inequalities. It thereby avoids rather exotic data structures that Harvey relies on to achieve the stated time bound.

4.2.1. *Calculating Cuts Between two Inequalities*

The building block of Harvey's algorithm calculates cuts between two adjacent inequalities that have a rational intersection point. These cuts correspond to Gomory's cutting planes, except that they are always non-redundant. Suppose the following inequalities are adjacent in the input polyhedron:

$$\begin{aligned} \iota_1 &\equiv 3x + y \leq 25 \\ \iota_2 &\equiv 3x + 5y \leq 50 \end{aligned}$$

Figure 27 shows that the intersection point of ι_1 and ι_2 is not integral. In order to calculate the shown cuts $2x+y \leq 18$ and $x+y \leq 12$, the initial inequalities are mapped to a different coordinate system in which ι_2 is parallel to the y-axis. This is achieved by applying a transformation $T \in \mathbb{Z}^{2\times 2}$ to their coefficients such that $\det(T) \in \{1, -1\}$ and

$$\begin{pmatrix} 3 & 1 \\ 3 & 5 \end{pmatrix} T = \begin{pmatrix} t & u \\ 1 & 0 \end{pmatrix},$$

where the values of t and u are fixed by the constraints on T. In particular, the inequality $\iota_1 \equiv 3x + y \leq 25$ is mapped to $\iota_1' \equiv tx + uy \leq 25$ and $\iota_2 \equiv 3x + 5y \leq 50$ to $\iota_2' \equiv x \leq 50$. The restriction $\det(T) \in \{1, -1\}$ implies that T is unimodular, that is, the transformation maps every integral point in the original system to an integral point in the new coordinate system [26]. In the example above, a suitable matrix is

$$T = \begin{pmatrix} -3 & -5 \\ 2 & -3 \end{pmatrix}.$$

Applying this transformation matrix to the inequality ι_1 yields $\iota_1' \equiv -7x + 12y \leq 25$ which is shown together with ι_2' in Fig. 28. As before, ι_1' has a non-integral intersection with $\iota_2' \equiv x \leq 50$. However, observe that the first feasible integral point that lies on the boundary of ι_1' is at $\langle 41, 26\rangle$ such that the problem of calculating cuts reduces to finding integral points with $41 \leq x \leq 50$. The idea is to consider the slope of ι_1' as a fraction $12/7$ and to calculate approximations to $12/7$ using fractions made up of smaller numbers. These approximations provide potential slopes for a cut that originates in $\langle 41, 26\rangle$. Note that an inequality touching $\langle 41, 26\rangle$ with approximated slope $a/b \geq 12/7$ touches the next integral vertex at $\langle 41+a, 26+b\rangle$ and thus does not cut off integral points with x-coordinates between 41 and $41+a$. Hence, it suffices to find two consecutive slopes a/b and a'/b' such that $41 + a \leq 50 < 41 + a'$ to ensure that no feasible integral point is cut off. In order to find these slopes, observe that every rational number can be represented as a finite continued fraction that takes on the following form:

$$a_1 + \cfrac{1}{a_2 + \frac{1}{a_3 + \cdots \frac{1}{a_n}}}$$

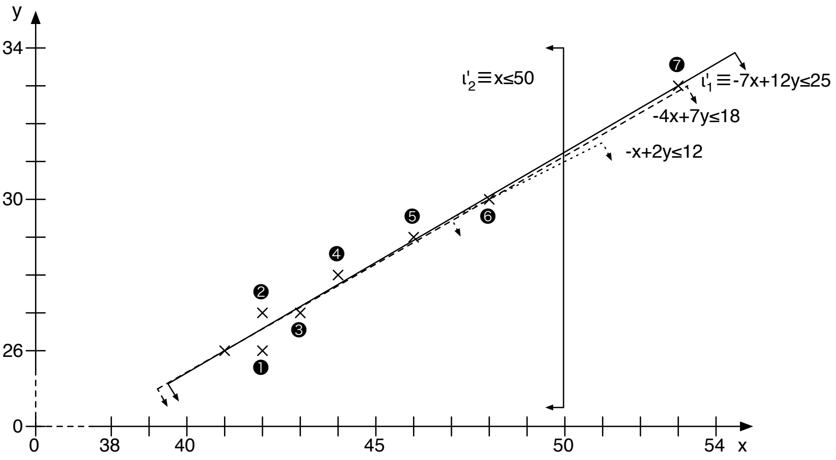


Figure 28. The inequalities in the transformed space. The seven convergents are shown relative to $\langle 41, 26\rangle$, the integral point on ι'_1 that lies on the feasible side of $\iota'_2 = x \leq 50$

The coefficients a_i can be inferred by observing the intermediate results of Euclid's greatest common divisor algorithm when applied to 12 and 7 [26]:

$$\begin{aligned} 12 &= 1 \times 7 + 5 \\ 7 &= 1 \times 5 + 2 \\ 5 &= 2 \times 2 + 1 \\ 2 &= 2 \times 1 + 0 \end{aligned}$$

The coefficients are thus $a_1 = 1, a_2 = 1, a_3 = 2, a_4 = 2$ and $12/7$ can therefore be reexpressed as follows:

$$\frac{12}{7} = 1 + \frac{1}{1 + \frac{1}{2 + \frac{1}{2}}}$$

Approximations to $12/7$ can now be derived by calculating a series of prefixes of the continued fraction, namely

$$\frac{5}{3} = a_1 + \frac{1}{a_2 + \frac{1}{a_3}}; \qquad \frac{2}{1} = a_1 + \frac{1}{a_2}; \qquad \frac{1}{1} = a_1$$

where the empty prefix is $1/0$, which represents the coarsest approximation. Let the fractions A_i/B_i be equal to the prefix of the continued fraction up to and including a_i. Rather than calculating these fractions using rational arithmetic, the following recurrence equations provide a way to calculate these fractions using integral numbers only:

$$\begin{array}{lll} A_0 = 1 & A_1 = a_1 & A_m = a_m A_{m-1} + A_{m-2} \\ B_0 = 0 & B_1 = 1 & B_m = a_m B_{m-1} + B_{m-2} \end{array}$$

The following table shows the values of A_i and B_i:

i	0	1	2	3	4
A_i	1	1	2	5	12
B_i	0	1	1	3	7

For all indices i where $a_i > 1$, namely $i = 3, 4$, there exists further approximations that have to be considered for possible slopes for generating cuts. These slopes are obtained by calculating A_i and B_i where a_i is substituted with the values $1, \ldots a_i - 1$. The following table augments the above with indices $i.j$ where $j = 1, \ldots a_i - 1$ for $a_i > 1$:

#	1	2	3	4	5	6	7
i or $i.j$	0	1	2	3.1	3	4.1	4
A_i	1	1	2	3	5	7	12
B_i	0	1	1	2	3	4	7

This table gives the complete set of slopes A_i/B_i that suffices to generate all possible cuts. For the example, these seven slopes are shown in Fig. 28 as a displacement to the integral point $\langle 41, 26 \rangle$. Note that slopes with odd indices i are not feasible with respect to ι'_1 and can therefore be discarded as an end-point for a cut. In particular, the coefficients for the first cut are taken from the largest even index i (or sub-index $i.j$ with i even) that yields a point that is still satisfied by $\iota'_2 \equiv x \leq 50$. The first cut in the transformed space is therefore $-4x + 7y \leq 18$ using the sixth fraction. The next cut originates in the end-point of the first cut which is $\langle 48, 30 \rangle$. Calculating the next cut is a matter of approximating the slope $7/4$ of the first cut. Since the continued fraction coefficients of $7/4$ form a suffix of those of $12/7$, we can reuse the table above to find a suitable slope. The slope $2/1$ gives a displacement that reaches $\langle 50, 31 \rangle$ which lies on the boundary of ι'_2. Thus the corresponding inequality $-x + 2y \leq 12$ is the final cut with respect to the two input inequalities. The two cuts are translated to the original coordinate system by multiplying the coefficients with

$$T^{-1} = \begin{pmatrix} 3 & 5 \\ 2 & 3 \end{pmatrix},$$

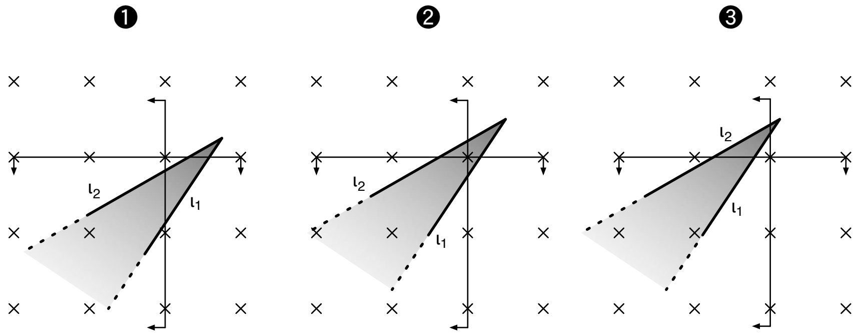


Figure 29. Determining the bounds with which to calculate integral cuts

yielding $2x + y \leq 18$ for the first cut and $x + y \leq 12$ for the second, as shown in Fig. 27. Given that Euclid's algorithm requires $\log(A)$ steps where A bounds the size of the two input coefficients and the fact that the fraction in each step may give rise to at most one cut, no more than $\log(A)$ new inequalities are generated. However, some of these new cuts may be redundant with respect to adjacent inequalities of the rational polyhedron. The next section discusses the challenges of implementing an algorithm that combines tightening of two inequalities with redundancy removal and thereby provides a practical implementation of Harvey's algorithm in $O(n \log A)$ time if the inequalities to be added are sorted by angle. Here n is the number of given inequalities (which may have rational intersection points) plus the number of inequalities to be added to the polyhedron.

4.2.2. *Applying the Integer Hull in the Reduced Product Domain*

Based on the method of calculating cuts between two inequalities, Harvey suggested an incremental algorithm that tightens a rational input polyhedron by adding its inequalities one by one to an initially empty tree of inequalities that constitutes the output. The complexity of $O(n \log A)$ is based on a level-linked finger tree [45] that is implemented in a circular fashion. In this section we present a way to calculate the integral hull that is likely to be faster for small n that occur in program analysis and which is more in tune with the reduced product representation of the TVPI domain where interval bounds are represented separately from the relational information and the latter is stored as an ordered, non-redundant array of inequalities.

Calculating the integral hull of a polyhedron has to be performed whenever new inequalities are added through the meet-operation since this may create non-integral intersection points. On the one hand, any redundant inequality that arises when adding new inequalities needs to be removed as cuts must be calculated on pairs of inequalities that are themselves non-redundant. On the other hand, any new cut may make other inequalities redundant such that the redundant inequalities must be removed while calculating cuts. Satisfying both requirements in one algorithm is difficult as the redundancy removal algorithm in Sect. 2.2.2 on p. 12 reduces the number of inequalities until a fixpoint is reached while calculating cuts creates new inequalities. Thus, we present a strategy that separates these concerns by exploiting the fact that the TVPI domain is implemented as a reduced product between TVPI inequalities and interval bounds. In particular, by observing that interval bounds are tightened explicitly during redundancy removal as shown in Fig. 23 on p. 37, we propose to tighten the interval bounds further, namely to the values that they will take on in the final $\mathbb{Z}$-polyhedron. Given these tightened interval bounds, cuts can be calculated separately within each quadrant without requiring a fixpoint computation to remove inequalities that become redundant with respect to the calculated cuts.

4.2.2.1. *Bounds With Integral Points.* We describe how to tighten interval bounds to the bounds of the final $\mathbb{Z}$-polyhedron. Since a $\mathbb{Z}$-polyhedron is characterised by the fact that all vertices are integral, it follows that a $\mathbb{Z}$-polyhedron has at least one feasible, integral point on each (finite) bound of its bounding box. Thus, in order to find this bounding box, the intervals of the rational polyhedron must be tightened until at least one integral point lies on each bound. Suppose that the bounds are rationally tightened such that the intersection of two adjacent bounds defines a feasible (but possibly non-integral) point, corresponding to graph 2–4 in Fig. 23 on p. 37. Rounding the bounds to the nearest feasible integral values may lead to one of the situations depicted in Fig. 29.

The interval bounds need no tightening if a point lies on them that is integral and feasible, such as in the second graph. Algorithm 7 implements this test for pairs of inequalities ι_1, ι_2 where the normal vector of ι_1 points south-east as is the case in Fig. 29. Only the upper interval bounds, namely x_u and y_u are relevant for this test. The algorithm returns *false* if the feasible section of a bound contains no integral point. In particular, lines 6 and 7 calculate the lower and upper y-values of the intersection point of the inequalities with the x-bound x_u which corresponds to the first graph in Fig. 29. Line 8 tests if these intersection points are feasible with respect to the upper bound on y, namely y_u. If these two bounds lie either side of y_u then an integral point has been found since $y_u \in \mathbb{Z}$ (lines 9–11 and graph two in the

Algorithm 7 Test for a feasible, integral point on the upper bounds.

```
procedure hasZPoint4th($\iota_1, \iota_2, x_u, y_u$) where $x_u, y_u \in \mathbb{Z} \cup \{\infty\}$ and
    $\iota_1, \iota_2 \in Ineq$ with $\frac{3}{2}\pi < \iota_1 < 2\pi \wedge \iota_1 \measuredangle \iota_2 \leq 2\pi \wedge \theta(\iota_2) \neq 0 \wedge \theta(\iota_2) \neq \frac{\pi}{2}$
 1: $a_1x + b_1y \leq c_1 \leftarrow \iota_1$
 2: $a_2x + b_2y \leq c_2 \leftarrow \iota_2$
 3: if $x_u = \infty$ then
 4:   return true
 5: end if
 6: $lower \leftarrow (c_1 - a_1x_u)/b_1$
 7: $upper \leftarrow (c_2 - a_2x_u)/b_2$
 8: if $y_u < \infty \wedge upper > y_u$ then
 9:   if $lower \leq y_u$ then
10:     return true
11:   end if
12:   $upper \leftarrow (c_1 - b_1y_u)/a_1$
13:   $lower \leftarrow (c_2 - b_2y_u)/a_2$
14: end if
15: return $\lceil lower \rceil \leq \lfloor upper \rfloor$
```

figure). Otherwise, the upper and lower x-values of the intersection between the inequalities and y_u is calculated (lines 12–13 and graph three in the figure). If rounding these values towards each other results in a non-empty interval, a feasible point on one of the bounds has been found and line 15 returns *true*. Two special cases may occur during the test. Firstly, $\iota_1 \measuredangle \iota_2 = \pi$, that is, the inequalities define an equality. In this case the polyhedron may have no upper bounds and is thus automatically a valid $\mathbb{Z}$-polyhedron and lines 3–5 return immediately. Secondly, in case the inequalities ι_1 and ι_2 lie in adjacent quadrants (graph one in Fig. 23) y_u may be infinite and line 8 ensures that calculating the intersection with y_u in lines 9–13 is skipped.

4.2.2.2. *Tightening Bounds Using Inequalities.* Since Alg. 7 only tests whether an integral point exists on the upper x-bound, three more variants of this test are necessary for the other bounds. If these tests returns *true*, shrinking the polyhedron around the contained integral points will not affect the corresponding bound. In case *false* is returned, the corresponding bound must be tightened until it contains a feasible integral point. This process is illustrated in Figure 30 where the objective is to find the two circled integral points that define the tightened x- and y-bound. In order to find these integral points we calculate a sequence of cuts $c_0, c_1, \ldots$ from the two inequalities ι_1 and ι_2 as

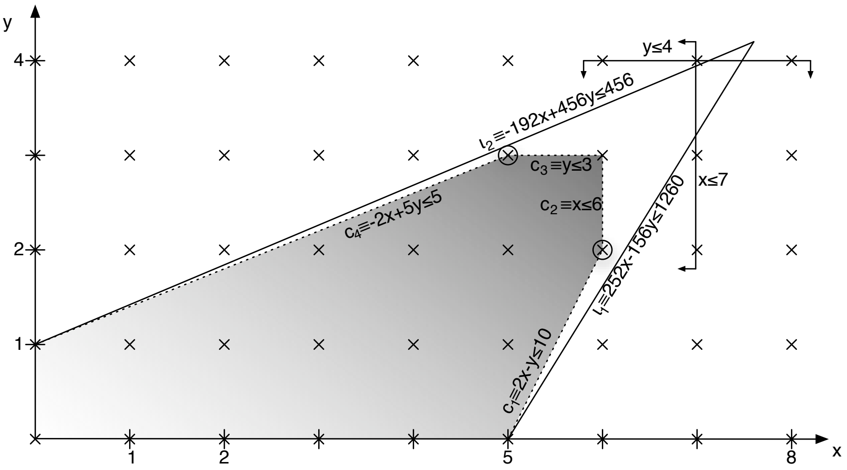


Figure 30. Deriving new bounds by calculating intersection points between cuts

follows: set $c_0 = \iota_1$ unless $\iota_1 \measuredangle \iota_2 = \pi$ in which case c_0 is set to the cut between ι_1 and the next bound. (In Figure 29, this is the upper x-bound, such that the cut is calculated with respect to $1x+0y \leq x_u$.) Let c_i denote the cut between the inequalities c_{i-1} and ι_2. Suppose there are n such cuts such that $c_0, \ldots c_n, c_{n+1}$ denotes a sequence of inequalities with integral intersection points where $c_{n+1} = \iota_2$. Furthermore, let i denote the smallest index such that $class(c_i) \neq class(\iota_1)$, that is, the cut c_i lies in the next quadrant. Then the intersection between c_{i-1}, c_i is an integral vertex that represents the largest extent of the polyhedron towards that direction. Analogously, let j denote the largest index such that $class(c_j) \neq class(\iota_2)$ and use the intersection of c_j and c_{j+1} to refine the next bound. For example, consider Fig. 30 where $\iota_1 = c_0$ and $\iota_2 = c_5$ define the first and the last cut. Here, the class boundaries are $class(\iota_1) = class(c_1) = 8$, $class(c_2) = 1$, $class(c_3) = 2$ and $class(c_4) = class(\iota_2) = 3$. Thus, $i = 2$ and $j = 3$ such that the intersection point between c_1 and c_2 defines the upper x-bound and similarly, c_3 and c_4 define the upper y-bound. After tightening the bounds, all cuts are discarded and the redundancy removal algorithm continues, possibly identifying ι_1 or ι_2 as redundant, in which case the bounds might need tightening again.

4.2.2.3. *Tightening Quadrants.* By applying the above procedure for all quadrants of the planar space, the redundancy removal algorithm will infer a polyhedron in which the bounds coincide with the bounds of the corresponding $\mathbb{Z}$-polyhedron. With cuts being calculated on-the-

Algorithm 8 Calculating all cuts between two non-redundant bounds

```
procedure tightenFirstQuadrant($(I, x_u, y_u)$), $I$ sorted, $x_u, y_u \in \mathbb{Z} \cup \{\infty\}$
 1: if $x_u < \infty$ then
 2:     $I \leftarrow \langle 1x + 0y \leq x_u \rangle \cdot I$
 3: end if
 4: if $y_u < \infty$ then
 5:     $I \leftarrow I \cdot \langle 0x + 1y \leq y_u \rangle$
 6: end if
 7: $O \leftarrow \emptyset$
 8: while $|I| > 0$ do
 9:     if $|O| = 0$ then
10:         $\langle \iota_0, \ldots \iota_n \rangle \leftarrow I$
11:         $I \leftarrow \langle \iota_1, \ldots \iota_n \rangle$
12:         $O \leftarrow \langle \iota_0 \rangle$
13:     else
14:         $\langle o_1, \ldots o_m \rangle \leftarrow O$
15:         $\langle \iota_1, \ldots \iota_n \rangle \leftarrow I$
16:         if $|I| > 1 \wedge \{o_m, \iota_2\} \sqsubseteq \iota_1$ then
17:             $O \leftarrow \langle o_1, \ldots o_{m-1} \rangle$
18:             $I \leftarrow \langle o_m, \iota_2, \ldots \iota_n \rangle$
19:         else if $\mathit{intersect}(o_m, \iota_1) \in \mathbb{Z}^2$ then
20:             $O \leftarrow \langle o_1, \ldots o_m, \iota_1 \rangle$
21:             $I \leftarrow \langle \iota_2, \ldots \iota_n \rangle$
22:         else
23:             $O \leftarrow \langle o_1, \ldots o_{m-1} \rangle$
24:             $I \leftarrow \langle o_m, \mathit{calculateCut}(o_m, \iota_1), \iota_1, \ldots \iota_n \rangle$
25:         end if
26:     end if
27: end while
28: return $O$
```

fly, rather than being inserted into the sequence of inequalities, there is no need to alter the fixpoint calculation of the redundancy removal algorithm. The integral bounds can now serve to tighten each quadrant of the polyhedron separately, as implemented by Alg. 8 for inequalities $I = \{\iota_1, \ldots \iota_n\}$ with $0 < \theta(\iota_1) < \ldots < \theta(\iota_n) < \frac{\pi}{2}$. The first three lines prepend the upper x bound as inequality to I (using a dot to denote concatenation), thereby ensuring that the sequence begins with a non-redundant inequality that is known to be part of the output $\mathbb{Z}$-polyhedron. Analogously, lines 4–6 append the upper y-bound. Note that in case a bound is infinite, the nearest inequality is in both cases a non-redundant ray that is satisfiable by an integral point and must

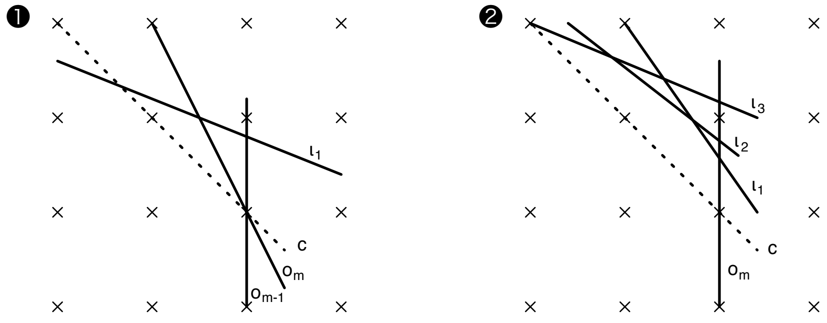


Figure 31. Removing inequalities that were made redundant by a new cut

therefore be part of the output $\mathbb{Z}$-polyhedron. While lines 1–6 are specific to the first quadrant, the code in lines 7–28 is applicable to all quadrants. The shown loop examines the polyhedron described by I and calculates its $\mathbb{Z}$-polyhedron in form of the initially empty sequence O. Specifically, lines 9–12 ensure that O contains at least the first element of I which is known to be in the output $\mathbb{Z}$-polyhedron by the previous argument. The following lines ensure that O only contains inequalities that are non-redundant and that intersect in an integral point. Non-redundancy of ι_1 is ensured by the test in line 16 which holds if ι_1 is the last inequality in I or if it is not entailed by its neighbouring inequalities. Furthermore, integrality is ensured by the test in lines 19, in which case lines 20–21 move the head ι_1 of I to the tail of O. In case the intersection point is not integral, lines 23–24 calculate a cut that has an integral intersection with o_m. However, the new cut may make o_m redundant which is illustrated in the first graph of Fig. 31. Since o_m was appended to O, it is non-redundant with respect to o_{m-1} and ι_1 and there exists an integral point $p = \mathit{intersect}(o_{m-1}, o_m)$. It follows that o_m can only be redundant if $p = \mathit{intersect}(o_m, c)$. In this case, the current o_m becomes ι_1 in the next loop iteration and is removed by lines 16–18. The loop iteration thereafter will find that o_{m-1} and the cut c (now the new ι_1) intersect in the same integral point p and therefore appends the cut to O. Thus, no more than one element of O is ever taken out of O. On the contrary, inserting a new cut c in line 24 may render several of the following inequalities $\iota_1, \iota_2, \ldots$ redundant as shown in the second graph of Fig. 31. These are consecutively removed by lines 16–18. Since the number of possible cuts between two inequalities is bounded and the fact that the other three branches of the loop (lines 10–12, 17–18 and 20–21) reduce the length of I, the loop will terminate eventually. In particular, since each element in O is put back into I at most once, the algorithm is linear in the size of the output set O.

4.2.2.4. *Complexity.* In order to assess the complexity of the above tightening methods, observe that the $O(k)$ redundancy removal algorithm is augmented with the calculation of cuts between interval bounds and the adjacent inequalities. Harvey observes that no more than $O(\log A)$ cuts can exist between any two inequalities whose coefficients are bounded by A. Furthermore, even if inequalities that are adjacent to the bounds become redundant and each rational input inequality were to be removed, the whole redundancy removal will still terminate in $O(k \log A)$. Similarly, calculating cuts in each quadrant terminates after creating at most $O(\log A)$ cuts between each pair of adjacent inequalities, giving an overall running time of $O(k \log A)$. Note that Harvey's algorithm requires only $O(k \log A)$ steps even if the input inequalities are not sorted by angle. However, Alg. 8 can be implemented using a dynamically growing array for the output O. The simpler data structure compensates for the requirement of sorting the input inequalities. This is particularly true in the case of using the TVPI domain for program analysis as the occurring planar polyhedra are usually very small such that the overall running time is dominated by constant overheads, e.g. manipulation a complicated level-linked tree structure.

The next section presents a closure algorithm which builds on the integral hull and the redundancy removal algorithm.

4.3. Planar $\mathbb{Z}$-Polyhedra and Closure

Given an efficient algorithm that shrinks a given planar polyhedron around the integral points that it contains, we now consider the problem of closing a system of planar integral polyhedra. Nelson originally proposed the calculation of the closure of a TVPI system as a way to check satisfiability [50]. However, checking if an arbitrary TVPI system has an integral solution is NP-complete [42]. Hence, although the planar integral hull algorithm is complete, completeness is not preserved when this algorithm is applied during the TVPI closure algorithm of Sect. 3.2.5. In this section we shall explore how this manifests itself in practice.

4.3.1. *Our Implementation of the $\mathbb{Z}$-TVPI Domain*

The complexity of calculating an integral TVPI system could be circumvented by implementing a rational TVPI domain and merely tightening planar projections whenever the value of a variable is queried. However, this approach does not prevent the excessive growth of coefficients and does not fully exploit the precision improvement due to tightening. The latter is illustrated by Fig. 32, which depicts the closure

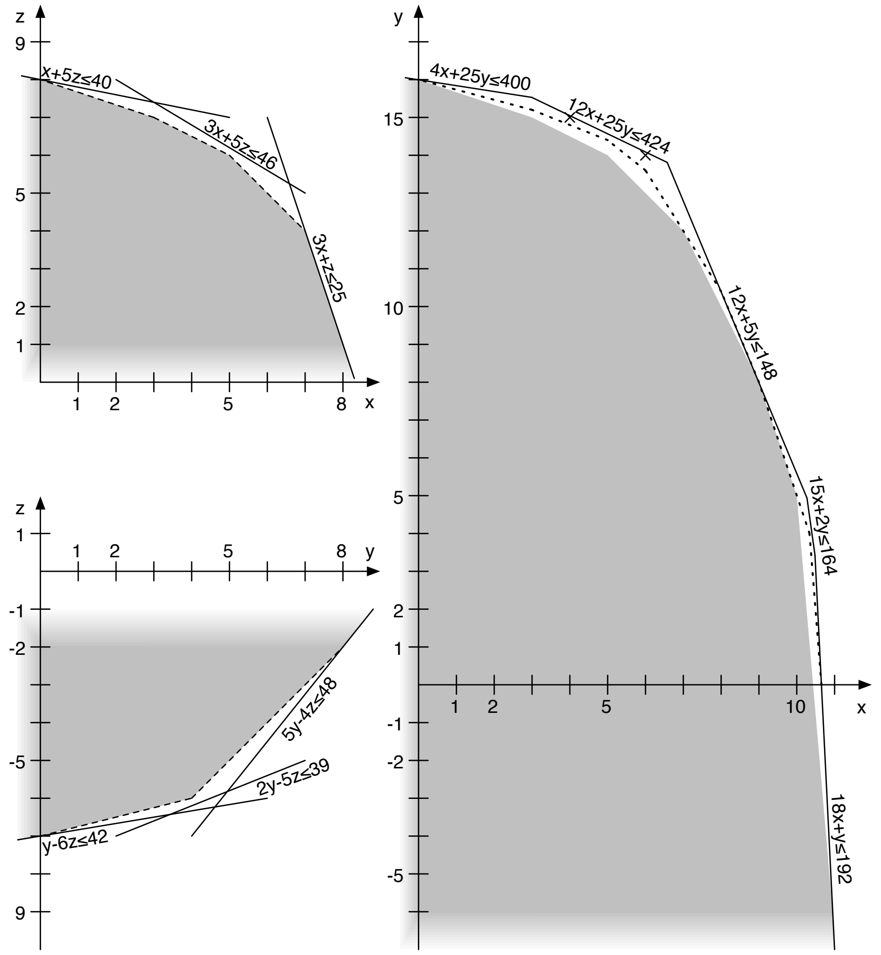


Figure 32. Closing a system over x, z and y, z to yield x, y. Shrinking the initial system around the integral grid removes two integral points in the x, y system.

of a TVPI system containing inequalities over x, z and z, y. Here, rational inequalities are shown as solid lines whereas the contained integral polyhedra are shown as the solid grey area. Calculating the resultants of the inequalities in the two graphs on the left yields five non-redundant inequalities shown in the x, y-projection. The dashed lines in the left two graphs denote the cuts that define the integral polyhedra in the x, z and z, y-projections. Closing the system using these cuts results in the inequalities indicated by the dashed line in the x, y-projection. Note that the two integral points $\langle 4, 15\rangle$ and $\langle 6, 14\rangle$ are no longer part of the feasible state space which shows that tightening every planar projection leads to more precise relationships between other variables of the system. However, the x, y-resultants calculated from the integral x, z and z, y-cuts do not define a $\mathbb{Z}$-polyhedron and tightening these resultants around the integral grid is necessary, which results in the grey polyhedron. The example shows that (1) using a rational TVPI domain and applying integer tightening only when querying the domain is less precise than (2) tightening all projections before closure which, in turn, is less precise than (3) tightening all projections, performing closure and then tightening all updated projections. For precision, tightening before closure is preferable. In order to limit the size of coefficients as discussed in Sect. 4.1.2, it is necessary to perform integral tightening after closure. Hence, our implementation follows (3).

4.3.2. *Tightening Bounds Across Projections*

In order to adapt the incremental closure algorithm to use Harvey's tightening algorithm, recall that closure adds a set of inequalities to a projection, removes redundant inequalities and uses the non-redundant subset of the new inequalities to close the TVPI system. When performing integral tightening after running the redundancy removal algorithm, the set of new, non-redundant inequalities is not a subset of the new inequalities as tightening may have added cuts that are necessary to describe the integral polyhedron. This non-monotone behaviour complicates the data structures that are required in the implementation.

Except for calculating the cuts, performing closure with tightening exhibits the same complexity as the rational closure. However, an integral closure algorithm would provide a way to test for integral satisfiability which, in turn, is NP-complete. In fact, performing tightening, closure and another tightening step does not generally lead to a $\mathbb{Z}$-polyhedron. For instance, consider the TVPI system over the three variables x, y, z shown in Fig. 33. Suppose the initial system consists of $\{x = 2z\}$ and that the inequalities $2x + 3y \leq 27, -2x + 3y \leq 3, -2x - 3y \leq -15, 2x - 3y \leq 9$ are then added as indicated by the solid lines in the upper left system. Applying integral tightening to these inequalities has no effect as the intersection points of the inequalities are already integral. Closing the system calculates the resultants of these inequalities and the empty y, z-projection before combining the x, y-projection with $\{x = 2z\} = \{x - 2z \leq 0, -x + 2z \leq 0\}$ which results in a compressed image of the rhombus in the y, z-plane. During redundancy removal, the interval bounds of z are tightened to $2 \leq z \leq 4$. Now the scaled rhombus $4z + 3y \leq 27, -4z + 3y \leq 3, -4z - 3y \leq -15, 4z - 3y \leq 9$, depicted by the solid lines, has non-integral intersection points with the bounds of z. As the next step, the inequalities are tightened around the integral grid, yielding $2z + y \leq 11, -2z + y \leq -1, -2z - y \leq -7, 2z - y \leq 5$ as

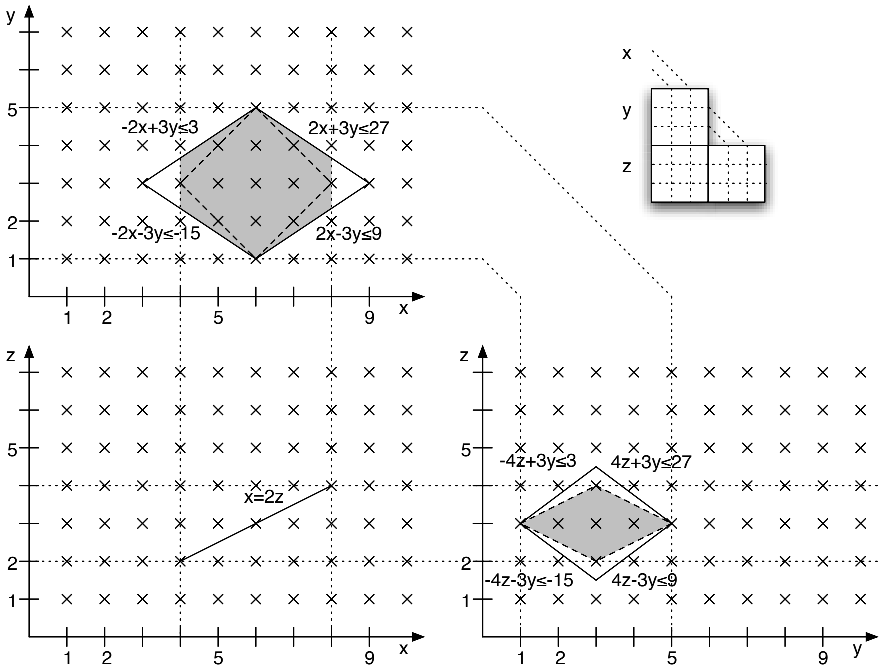


Figure 33. Tightening inequalities around their integral grid can affect interval bounds which requires a new tightening step in previously visited projections

indicated by the dashed lines. This tightening has derived bounds on z, namely $[2, 4]$. Since the bounds of variables are of particular interest, any updated interval is propagated to all other interval bounds using inequalities drawn from any projection that fulfils cases three and four of Fig. 23 on p. 37. Using these propagation rules, the x, z-projection thereby tightens the interval of x to $[4, 8]$. Our implementation stops at this point. Observe that the x, y-projection now has non-integral intersection points since the bounds on z and x were updated after the x, y-projection was tightened.

The shown dashed rhombus of the x, y-plane can be obtained by either tightening the existing inequalities around the integral grid or by calculating new resultants from the rhombus in the y, z-projection and the line segment in the x, z-projection. In order to ensure that a TVPI system is closed, closure and tightening has to be applied repeatedly until a fixpoint is reached. To ensure efficiency, our implementation stops with the state space shown in grey and thereby admits non-integral intersection points with the interval bounds. This implies that vertex-based algorithms such as the planar convex hull may operate on non-integral vertices and may thus create inequalities that intersect in

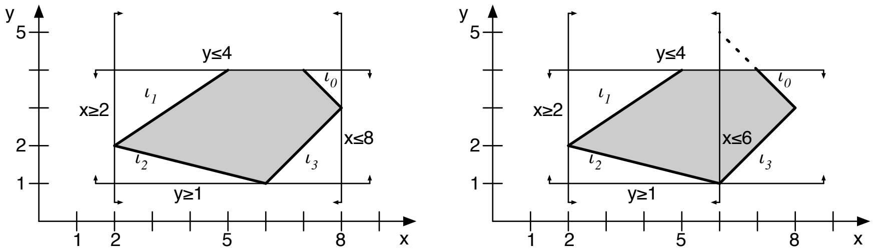


Figure 34. Allowing interval bounds to tighten after calculating the integral hull can lead to intersection points outside the original polyhedron

non-integral points. For instance, the polyhedra in the second system of Fig. 1.3 on p. 6 can arise by tightening the bounds of the polyhedron in the first system, namely once to $x_2 \leq 4$ and once to $x_2 \geq 6$. Any rational intersection point between the updated bound and the existing inequalities are only removed if new inequalities are added.

4.3.3. *Discussion and Implementation*

Rather unexpectedly, note that even repeated application of integral closure does not constitute a decision procedure for integral TVPI satisfiability. Consider the system comprised of the inequalities $0 \leq 4y - 7z \leq 1$ for the y, z-projection and $6 \leq -4x + 7z \leq 7$ for the x, z-projection. Closing this TVPI system, which corresponds to adding the two equations, adds $6 \leq -4x + 4y \leq 8$ to the x, y-projection. The latter can be tightened without losing integral solutions to $\lceil \frac{6}{4} \rceil \leq x - y \leq \lfloor \frac{8}{4} \rfloor$ which is equivalent to $x - y = 2$. The resulting system is, in fact, closed and all projections are integral. This is peculiar, since the TVPI system is actually unsatisfiable in $\mathbb{Z}$. For this reason, we deemed repeated application of closure to be excessive as, even then, unsatisfiability cannot be detected in all cases.

Another consequence of bounds being tightened after a projection has been shrunk around the integral grid is that inequalities might become redundant. Suppose the integral polyhedron in the left graph of Fig. 34 has just been tightened around the integral grid. If tightening in a different projection reduces the upper bound on x to $x \leq 6$, the polyhedron will contain two redundant inequalities ι_0 and ι_3 as shown in the right graph. Applying the convex hull algorithm to this system of inequalities will calculate an intersection point between the upper bound on y and the boundary of ι_0, as these are angle-wise adjacent. However, the resulting point lies outside the polyhedron and the convex hull algorithm calculates a result that is incorrect. Thus, inequalities that are redundant due to tightened bounds have to be removed before applying the convex hull or other planar algorithms that require a non-redundant input system. However, note that these excess inequalities can be removed on-the-fly by merely using case two of the tests in Fig. 22 on p. 36, rather than by applying the full redundancy removal algorithm.

Working on non-closed TVPI systems implies that the analysis is not as precise as possible. Worse, since the closure calculation has to be stopped at some point, the specific implementation of the TVPI domain determines the precision of the analysis. While this is an argument against integer tightening, observe that the integral TVPI domain is always more precise than its rational counterpart. Furthermore, since coefficients in the inequalities of a rational TVPI system can grow excessively, inequalities have to be removed in order to ensure scalability which inevitably leads to a non-closed and possibly imprecise system. Tightening the planar polyhedron ensures that the coefficients in the inequalities remain small so that the removal of inequalities with large coefficients is unnecessary. Hence, implementing the TVPI domain over planar $\mathbb{Z}$-polyhedra seems to be the only way to implement the rational TVPI domain efficiently.

We conclude this section by reviewing work on rational and integral satisfiability of the TVPI domain.

4.4. Related Work

Inequalities with at most two variables have given rise to much research in recent decades, not least due to the fact that general network flow problems can be expressed using TVPI systems. Integer TVPI systems describe a special class of flow problems where flows consists of discrete units.

The closure operation presented in Sect. 3.2 stems from an idea of Nelson to check for satisfiability of a rational TVPI system [50]. It turns out that more efficient methods exist for this task [36]. However, Shostak used closure algorithms to check for satisfiability of integer TVPI problems [63] although his procedure is not guaranteed to either terminate nor detect satisfiability. In the context of weaker TVPI classes, Jaffar et al. [39] show that satisfiability of two-variables per inequality constraints with unit coefficients can be solved in polynomial time and that this domain supports efficient entailment checking and projection. More recently, Harvey and Stuckey [34] have shown how to reformulate this solver to formally argue completeness, which gave rise to the planar integer hull algorithm [33]. Su and Wagner [73] present an algorithm for calculating the least integer solution of a system of two variable inequalities. They claim that their algorithm is polynomial, however, it turns out that solving integer two variable per inequality constraints is NP-complete [42]. However, checking integral satisfiability of a TVPI system is polynomial if all inequalities are monotone [35], that is, if all inequalities have the form $ax - by \leq c$ where $a, b \in \mathbb{N}$. A practical implementation of the integer satisfiability for general polyhedra is the Omega test [55], an extension of Fourier-Motzkin variable elimination that is complete over the integers, but that might not terminate in general. Other classic integer decision procedures include the SUP-INF algorithm [62] and Cooper's algorithm [19]. These techniques are widely applied in verification but do not provide the operations necessary for domains used in abstract interpretation.

5. Applications

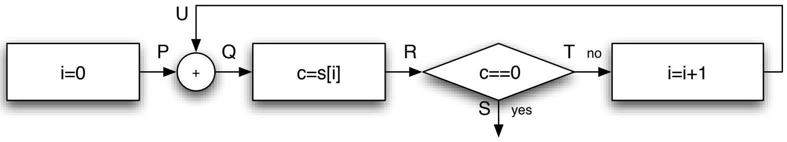
When coupled with integral tightening, the TVPI domain can improve both efficiency and precision of an analysis. This section presents example applications of the TVPI domain that illustrate both aspects.

5.1. String Buffer Analysis

We demonstrate the use of the TVPI domain in the context of an example drawn from string buffer analysis. In particular, the example below shows how integral tightening reduces the number of iterations to calculate a fixpoint. Furthermore, this verification problem requires inequalities with arbitrary coefficients and therefore beyond the precision of the Octagon domain whose coefficients are unitary. The verification task is to check that no memory buffer is accessed out-of-bounds. In C, this analysis is complicated by the fact that the end of a string is marked by a zero character (the so-called NUL character). One common idea of string buffer analysis [28, 67, 76] is to not track the contents of a buffer, but only the position of the first NUL character. Consider the following loop which is naturally produced by a C compiler translating `while (*s) s++;`.

```
char s[32] = "the␣string";
int i = 0;
while (true) {
  c = s[i];
  if (c==0) break;
  i = i+1;
};
```

The task is to check that the string buffer `s` is only accessed within bounds. This program is challenging for automatic verification because the loop invariant is always satisfied and the extra exit condition within the loop does not mention the loop counter `i`. In C, a string is merely an array of bytes, in this case `s` is an array of 32 bytes. The string literal initialises the first ten characters whilst the eleventh position is set to 0 (the NUL character). A single polyhedral variable per array suffices to express the position of the first NUL character. Specifically, let n represent the index of the NUL position in `s`. The control flow graph of the string buffer example is decorated with polyhedra P, Q, R, S, T, U as follows:



The initial state of the program is described by $P = [\![\{i = 0, n = 10\}]\!]$. The merge of this polyhedron and the one on the back edge, U, defines $Q = P \sqcup_P U$. To verify that the array access `s[i]` is within bounds, we compute $Q' = Q \sqcap_P [\![\{0 \leq i \leq 31\}]\!]$ and issue a warning if $Q' \neq Q$. The analysis continues under the premise that the access was within bounds and hence R is defined in terms of Q' rather than Q as follows:

$$\begin{array}{rcl} R & = & (\exists_c(Q') \sqcap_P [\![\{i < n, 1 \leq c \leq 255\}]\!]) \\ & \sqcup_P & (\exists_c(Q') \sqcap_P [\![\{i = n, c = 0\}]\!]) \\ & \sqcup_P & (\exists_c(Q') \sqcap_P [\![\{i > n, 0 \leq c \leq 255\}]\!]) \end{array}$$

The projection operator $\exists_c$ removes all information pertaining to c in Q' so that c can be updated. Since the contents of `s` are ignored in our model, the new value of c only depends on the relationship between the index i and n which describes the position of the *first* NUL character. The value of c is restricted to $[1, 255]$ if $i < n$, it is set to 0 if $i = n$ and to $[0, 255]$ if $i > n$. Note that this model is valid for platforms where the C `char` type is unsigned. The last three equations that comprise the system are given by the following:

$$\begin{array}{rcl} S & = & R \sqcap_P [\![\{c = 0\}]\!] \\ T & = & (R \sqcap_P [\![\{c \leq -1\}]\!]) \sqcup_P (R \sqcap_P [\![\{c \geq 1\}]\!]) \\ U & = & \{\langle n, i+1, c\rangle \mid \langle n, i, c\rangle \in T\} \end{array}$$

The affine transformation in the last equation defining U assumes that the variables in the polyhedron are ordered as in the sequence

n, i, c. Figure 35 details the calculation of a fixpoint using Jacobi iteration [23], where widening is applied in Q at iteration 10. Specifically, we use widening with landmarks [70] that tracks the development of the upper bound on i and extrapolates this bound to 10 since at $i = 10$ the second case of R first makes a contribution to R.

The development of state R_{11} is depicted in Fig. 36. The first graph shows the contributions of the first and second cases in the definition of R; the second graph shows their join. The states S_{12} and T_{12} are derived from R_{11} by applying the test `c==0` and its negation. Observe that i takes on its maximal value in the state $R_{11} \sqcap_P [\![c \geq 1]\!]$ at the point $\langle 10, \frac{2549}{255}, 1\rangle$, that is, $i < 9.996$. Integral tightening ensures that $i \leq 9$ in T_{12}. This is important since $i \leq 10$ in U_{13} and since $U_{13} \sqsubseteq_P Q_{13}$, a fixpoint has been reached. Additional iterations are required to reach a fixpoint without integral tightening. In order to illustrate this, suppose that no integral tightening is performed so that $U_{13} = [\![\{1 \leq i < 10.996, 1 \leq c \leq 255, n = 10\}]\!]$. Since i exceeds 10, another loop iteration is calculated, leading to $Q_{14} = [\![\{0 \leq i < 10.966, n = 10\}]\!]$. Next R_{15} is calculated. Note that the guards $i < n$, $i = n$ and $i > n$ in the definition of R merely abbreviate $i \leq n - 1$, $n \leq i \leq n$ and $i \geq n + 1$. Hence, since $n = 10$, these guards amount to intersection Q_{14} with the ranges $0 \leq i \leq 10$ and $i = 10$ and $i \geq 11$. Since the latter guard is unsatisfiable, the upper bound of i in R_{15} is 10 which (fortuitously) recovers integrality so that the same fixpoint is reached, albeit after more iterations. Observe that in order to express the fixpoint, non-unitary coefficients are required in the inequalities that describe the relation between the index i and the character c. Hence, this verification problem is beyond the reach of other weakly relational domains such as the Octagon [47] or the Octahedral domain [17].

5.2. Trace Partitioning

While the previous section showed how integral tightening can reduce the number of iterations that arise during fixpoint calculations, this section presents an example of how the precision improvement due to integral tightening can be crucial to verify a program property. The example builds on the observation that a Boolean flag can be added to a polyhedral domain in order to distinguish two sub-polyhedra that characterise different paths through the program [66]. In order to illustrate control-flow splitting, consider the task of proving the absence of a division by zero in the following C fragment:

```
        int r=MAX_INT;
```

j	Q_j	R_j	S_j	T_j	U_j
1	$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
2	$\mathbf{0 \leq i \leq 0}$	$\bot$	$\bot$	$\bot$	$\bot$
3	$0 \leq i \leq 0$	$\mathbf{0 \leq i \leq 0}$, $\mathbf{1 \leq c \leq 255}$	$\bot$	$\bot$	$\bot$
4	$0 \leq i \leq 0$	$0 \leq i \leq 0$, $1 \leq c \leq 255$	$\bot$	$\mathbf{0 \leq i \leq 0}$, $\mathbf{1 \leq c \leq 255}$	$\bot$
5	$0 \leq i \leq 0$	$0 \leq i \leq 0$, $1 \leq c \leq 255$	$\bot$	$0 \leq i \leq 0$, $1 \leq c \leq 255$	$\mathbf{1 \leq i \leq 1}$, $\mathbf{1 \leq c \leq 255}$
6	$\mathbf{0 \leq i \leq 1}$	$0 \leq i \leq 0$, $1 \leq c \leq 255$	$\bot$	$0 \leq i \leq 0$, $1 \leq c \leq 255$	$1 \leq i \leq 1$, $1 \leq c \leq 255$
7	$0 \leq i \leq 1$	$\mathbf{0 \leq i \leq 1}$, $\mathbf{1 \leq c \leq 255}$	$\bot$	$0 \leq i \leq 0$, $1 \leq c \leq 255$	$1 \leq i \leq 1$, $1 \leq c \leq 255$
8	$0 \leq i \leq 1$	$0 \leq i \leq 1$, $1 \leq c \leq 255$	$\bot$	$\mathbf{0 \leq i \leq 1}$, $\mathbf{1 \leq c \leq 255}$	$1 \leq i \leq 1$, $1 \leq c \leq 255$
9	$0 \leq i \leq 1$	$0 \leq i \leq 1$, $1 \leq c \leq 255$	$\bot$	$0 \leq i \leq 1$, $1 \leq c \leq 255$	$\mathbf{1 \leq i \leq 2}$, $\mathbf{1 \leq c \leq 255}$
10	$\mathbf{0 \leq i \leq 10}$	$0 \leq i \leq 1$, $1 \leq c \leq 255$	$\bot$	$0 \leq i \leq 1$, $1 \leq c \leq 255$	$1 \leq i \leq 2$, $1 \leq c \leq 255$
11	$0 \leq i \leq 10$	$\mathbf{0 \leq i, c \leq 255}$, $\mathbf{255i + c \leq 2550}$, $\mathbf{-i - 10c \leq -10}$	$\bot$	$0 \leq i \leq 1$, $1 \leq c \leq 255$	$1 \leq i \leq 2$, $1 \leq c \leq 255$
12	$0 \leq i \leq 10$	$0 \leq i, c \leq 255$, $255i + c \leq 2550$, $-i - 10c \leq -10$	$\mathbf{i = 10}$, $\mathbf{c = 0}$	$\mathbf{0 \leq i \leq 9}$, $\mathbf{1 \leq c \leq 255}$	$1 \leq i \leq 2$, $1 \leq c \leq 255$
13	$0 \leq i \leq 10$	$0 \leq i, c \leq 255$, $255i + c \leq 2550$, $-i - 10c \leq -10$	$i = 10$, $c = 0$	$0 \leq i \leq 9$, $1 \leq c \leq 255$	$\mathbf{1 \leq i \leq 10}$, $\mathbf{1 \leq c \leq 255}$
14	$0 \leq i \leq 10$	$0 \leq i, c \leq 255$, $255i + c \leq 2550$, $-i - 10c \leq -10$	$i = 10$, $c = 0$	$0 \leq i \leq 9$, $1 \leq c \leq 255$	$1 \leq i \leq 10$, $1 \leq c \leq 255$

Figure 35. Fixpoint calculation of the string loop. A polyhedron $[\![S]\!]$ is abbreviated to S and $\bot$ denotes an unsatisfiable set of inequalities. The column P_j is omitted since $P_j = [\![\{i = 0\}]\!]$ for all iterations j. Further we omit $n = 10$ from all polyhedra.

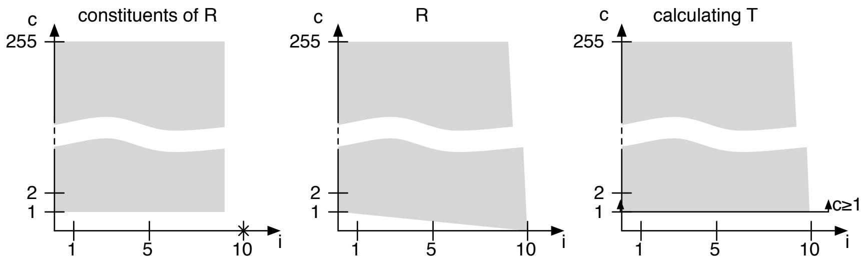


Figure 36. Performing integer tightening during the fixpoint calculation.

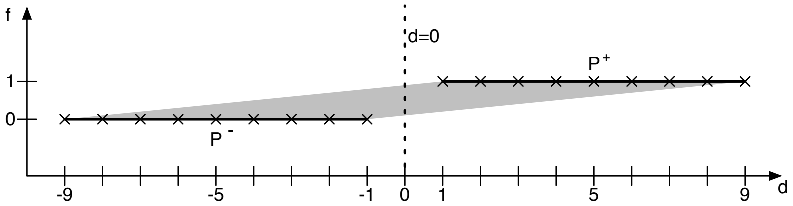


Figure 37. Using a Boolean flag to perform control-flow splitting. Feasible integral points are indicated by crosses, the dashed line indicating the polyhedron $[\![d = 0]\!]$.

```
        if (d!=0) r=v/d;
```

The test $d \neq 0$ is modelled by intersecting the current state, say P, with $d > 0$ and $d < 0$, resulting in $P^+ = P \sqcap_P [\![d > 0]\!]$ and $P^- = P \sqcap_P [\![d < 0]\!]$. The state with which the division operation is analysed is approximated by $P' = P^+ \sqcup_P P^-$. However, the convex hull reintroduces the state $P \sqcap_P [\![d = 0]\!]$ for which the division `r=v/d` is erroneous. In fact, it turns out that $P' = P$ whenever P^+ and P^- are non-empty.

Suppose this block of code is executed with a value of $[-9, 9]$ for `d`. Rather than analysing the division twice, once with positive values of d, once with negative values of d, Fig. 37 shows how the states P^+ and P^- can be stored in a single state without introducing an integral point where $d = 0$. Specifically, the figure shows the state $P' = (P^- \sqcap_P [\![f = 0]\!]) \sqcup_P (P^+ \sqcap_P [\![f = 1]\!])$. Suppose that the division operations on polyhedra flags a warning whenever the denominator d can be zero in P'. Indeed, while the state space $P' \sqcap_P [\![\{d = 0\}]\!]$ is non-empty, it contains no integer point, thereby indicating that an integer division by zero is not possible. This can be proved by performing integral tightening on P' which will collapse P' to the empty state.

5.3. Related Work

Quite apart from the applications discussed above, weakly relational domains have found application in many areas. Difference-bounded matrices that track the maximal difference between any two variables have been used to reason about systems of constraints [5], clocks in timed automata [14], and parallelisation [7]. More generally, TVPI inequalities with coefficients -1,0 or 1 were proposed for constraint logic programming [34, 38] and were subsequently generalised to the Octagon abstract domain [47] for the purpose of verifying C code. Elsewhere, Octagons have been used to detect memory leaks in Java programs [61]. Two variable inequalities were relaxed to arbitrary coefficients in the context of satisfiability testing in theorem proving [50, 63]. As far as we are aware, our work was the first to define an abstract domain based on TVPI inequalities with arbitrary coefficients [72].

6. Conclusion

This paper presented the Two Variables Per Inequality abstract domain. Specifically, we presented abstract operations on planar polyhedra which we lifted to several variables using a closure (information propagation) algorithm. We then presented techniques to perform integral tightening. Overall, the TVPI domain, whether integral or not, represents an attractive trade-off between tractability and expressiveness. Moreover, the TVPI domain provides polynomial runtime for all its operations whilst being the most expressive domain over two variables. Furthermore, the ability to perform integral tightening may be important to an analysis focussing solely on integer properties.

Acknowledgements. This work was supported, in part, by EPSRC grants EP/E033105, EP/E034519, EP/F012896, the INRIA project “Abstraction” funded by CNRS and ENS, and the DFG EN SI 1579/1. We would like to thank the anonymous reviewers for their comments; in particular one reviewer who alerted us to Prop. 4 which incorrectly stated in [72].

References

1. Akl, S. G. and G. T. Toussaint: 1978, ‘A fast convex hull algorithm’. *Information Processing Letters* **7**(5), 219–222.
2. Anderson, K. R.: 1978, ‘A Reevaluation of an Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set’. *Information Processing Letters* **7**(1), 53–55.

3. Andrews, A. M.: 1979, 'Another efficient algorithm for convex hulls in two dimensions'. *Information Processing Letters* **9**(5), 216–219.
4. Avis, D. and K. Fukuda: 1992, 'A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra'. *Discrete & Computational Geometry* **8**, 295–313.
5. Bagnara, R.: 1997, 'Data-Flow Analysis for Constraint Logic-Based Languages'. Ph.D. thesis, Università di Pisa, Dipartimento di Informatica, Pisa, Italy.
6. Bagnara, R., E. Ricci, E. Zaffanella, and P. M. Hill: 2002, 'Possibly Not Closed Convex Polyhedra and the Parma Polyhedra Library'. In: M. V. Hermenegildo and G. Puebla (eds.): *Static Analysis Symposium*, Vol. 2477 of *LNCS*. Madrid, Spain, pp. 213–229.
7. Balasundaram, V. and K. Kennedy: 1989, 'A Technique for Summarizing Data Access and Its Use in Parallelism Enhancing Transformations'. In: *Programming Language Design and Implementation*. Snowbird, Utah, USA, pp. 41–53.
8. Banterle, F. and R. Giacobazzi: 2007, 'A Fast Implementation of the Octagon Abstract Domain on Graphics Hardware'. In: H. R. Nielson and G. Filé (eds.): *Static Analysis Symposium*, Vol. 4634 of *LNCS*. Kongens Lyngby, Denmark, pp. 315–332.
9. Bentley, J. L., H. T. Kung, M. Schkolnick, and C. D. Thompson: 1978, 'On the Average Number of Maxima in a Set of Vectors'. *Journal of the ACM* **25**, 536–543.
10. Bertrand, J.: 2005, 'NewPolka'. `http://www.irisa.fr/prive/bjeannet/newpolka.html`.
11. Blanchet, B., P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival: 2002, 'Design and Implementation of a Special-Purpose Static Program Analyzer for Safety-Critical Real-Time Embedded Software'. In: T. Æ. Mogensen, D. A. Schmidt, and I. H. Sudborough (eds.): *The Essence of Computation: Complexity, Analysis, Transformation. Essays Dedicated to Neil D. Jones*, Vol. 2566 of *LNCS*. pp. 85–108.
12. Blanchet, B., P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival: 2003, 'A Static Analyzer for Large Safety-Critical Software'. In: *Programming Language Design and Implementation*. San Diego, California, USA.
13. Bournez, O., O. Maler, and A. Pnueli: 1999, 'Orthogonal Polyhedra: Representation and Computation'. In: F. W. Vaandrager and J. Schuppen (eds.): *Hybrid Systems: Computation and Control*, Vol. 1569 of *LNCS*. Berg en Dal, The Netherlands, pp. 46–60.
14. Bowman, H. and R. Gomez: 2006, *Concurrency Theory, Calculi and Automata for Modelling Untimed and Timed Concurrent Systems*. Springer.
15. Brönnimann, H., J. Iacono, J. Katajainen, P. Morin, J. Morrison, and G. T. Toussaint: 2002, 'In-Place Planar Convex Hull Algorithms'. In: *Latin American Symposium on Theoretical Informatics*, Vol. 2286 of *LNCS*. Cancun, Mexico, pp. 494–507.
16. Chernikova, N. V.: 1968, 'Algorithm for Discovering the Set of All Solutions of a Linear Programming Problem'. *USSR Computational Mathematics and Mathematical Physics* **8**(6), 282–293.
17. Clariso, R. and J. Cortadella: 2004, 'The Octahedron Abstract Domain'. In: R. Giacobazzi (ed.): *Static Analysis Symposium*, Vol. 3148 of *LNCS*. Verona, Italy.

18. Codish, M., A. Mulkers, M. Bruynooghe, M. García de la Banda, and M. Hermenegildo: 1995, 'Improving Abstract Interpretations by Combining Domains'. *ACM Transactions on Programming Languages and Systems* **17**(1), 28–44.
19. Cooper, D. C.: 1972, 'Theorem proving in arithmetic without manipulation'. *Machine Intelligence* **7**, 91–99.
20. Cormen, T. H., C. Stein, R. L. Rivest, and C. E. Leiserson: 2001, *Introduction to Algorithms*. McGraw-Hill.
21. Cousot, P. and R. Cousot: 1977, 'Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints'. In: *Principles of Programming Languages*. Los Angeles, California, USA, pp. 238–252.
22. Cousot, P. and R. Cousot: 1979, 'Systematic Design of Program Analysis Frameworks'. In: *Principles of Programming Languages*. San Antonio, Texas, USA, pp. 269–282.
23. Cousot, P. and R. Cousot: 1992a, 'Abstract Interpretation and Application to Logic Programs'. *Journal of Logic Programming* **13**(2–3), 103–179.
24. Cousot, P. and R. Cousot: 1992b, 'Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation'. In: M. Bruynooghe and M. Wirsing (eds.): *Programming Language Implementation and Logic Programming*, Vol. 631 of *LNCS*. Leuven, Belgium, pp. 269–295.
25. Cousot, P. and N. Halbwachs: 1978, 'Automatic Discovery of Linear Constraints among Variables of a Program'. In: *Principles of Programming Languages*. Tucson, Arizona, USA, pp. 84–97.
26. Davenport, H.: 1952, *The Higher Arithmetic*. Cambridge University Press, 7th edition.
27. Day, A. M.: 1990, 'The implementation of a 2D convex hull algorithm using perturbation'. *Computer Graphics Forum* **9**(4), 309–316.
28. Dor, N., M. Rodeh, and M. Sagiv: 2001, 'Cleanness Checking of String Manipulations in C Programs via Integer Analysis'. In: P. Cousot (ed.): *Static Analysis Symposium*, Vol. 2126 of *LNCS*. Paris, France, pp. 194–212.
29. Graham, R. L.: 1972, 'An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set'. *Information Processing Letters* **1**(4), 132–133.
30. Gries, D. and I. Stojmenović: 1987, 'A note on Graham's convex hull algorithm'. *Information Processing Letters* **25**(5), 323–327.
31. Halbwachs, N., D. Merchat, and L. Gonnord: 2006, 'Some ways to reduce the space dimension in polyhedra computations'. *Formal Methods in System Design* **29**(1), 79–95.
32. Halbwachs, N., Y.-E. Proy, and P. Roumanoff: 1997, 'Verification of Real-Time Systems using Linear Relation Analysis'. *Formal Methods in System Design* **11**(2), 157–185.
33. Harvey, W.: 1999, 'Computing Two-Dimensional Integer Hulls'. *SIAM Journal on Computing* **28**(6), 2285–2299.
34. Harvey, W. and P. J. Stuckey: 1997, 'A Unit Two Variable per Inequality Integer Constraint Solver for Constraint Logic Programming'. *Australian Computer Science Communications* **19**(1), 102–111.
35. Hochbaum, D. S.: 2004, 'Monotonizing linear programs with up to two nonzeroes per column'. *Operations Research Letters* **32**(1), 49–58.
36. Hochbaum, D. S. and J. Naor: 1994, 'Simple and Fast Algorithms for Linear and Integer Programs with Two Variables per Inequality'. *SIAM Journal on Computing* **23**(6), 1179–1192.

37. Imbert, J.-L.: 1993, 'Fourier's Elimination: Which to Choose?'. In: *Principles and Practice of Constraint Programming*. pp. 117–129.
38. Jaffar, J. and M. J. Maher: 1994, 'Constraint Logic Programming: A Survey'. *Journal of Logic Programming* **19/20**, 503–581.
39. Jaffar, J., M. J. Maher, P. J. Stuckey, and R. H. C. Yap: 1994, 'Beyond Finite Domains'. In: A. Borning (ed.): *Principles and Practice of Constraint Programming*, Vol. 874 of *LNCS*. Rosario, Orcas Island, Washington, USA, pp. 86–94.
40. Knuth, D. E.: 1973, *The Art of Computer Programming: Fundamental Algorithms*, Vol. 1. Addison-Wesley, 2nd edition.
41. Koplowitz, J. and D. Jouppi: 1978, 'A more efficient convex hull algorithm'. *Information Processing Letters* **7**(1), 56–57.
42. Lagarias, J. C.: 1985, 'The Computational Complexity of Simultaneous Diophantine Approximation Problems'. *SIAM Journal on Computing* **14**(1), 196–209.
43. Le Verge, H.: 1992, 'A Note on Chernikova's Algorithm'. Technical Report 1662, Campus Universitaire de Beaulieu, Institut de Recherche en Informatique, Beaulieu, France.
44. Loechner, V.: 2005, 'PolyLib'. `http://icps.u-strasbg.fr/polylib/`.
45. Mehlhorn, K.: 1984, *Sorting and Searching*, Vol. 1 of *ETACS Monographs*. Springer.
46. Miné, A.: 2001, 'A New Numerical Abstract Domain Based on Difference-Bound Matrices'. In: O. Danvy and A. Filinski (eds.): *Programs as Data Objects*, Vol. 2053 of *LNCS*. Aarhus, Denmark, pp. 155–172.
47. Miné, A.: 2006a, 'The Octagon Abstract Domain'. *Higher-Order and Symbolic Computation* **19**, 31–100.
48. Miné, A.: 2006b, 'Symbolic Methods to Enhance the Precision of Numerical Abstract Domains'. In: E. A. Emerson and K. S. Namjoshi (eds.): *Verification, Model Checking and Abstract Interpretation*, Vol. 3855 of *LNCS*. Charleston, South Carolina, USA, pp. 348–363.
49. Motzkin, T. S., H. Raiffa, G. L. Thompson, and R. M. Thrall: 1953, 'The Double Description Method'. In: H. W. Kuhn and A. W. Tucker (eds.): *Contributions to the Theory of Games*.
50. Nelson, C. G.: 1978, 'An $n^{\log(n)}$ Algorithm for the Two-Variable-Per-Constraint Linear Programming Satisfiability Problem'. Technical Report STAN-CS-78-689, Stanford University.
51. Nelson, C. G.: 1981, 'Techniques for Program Verification'. Ph.D. thesis, Palo Alto Research Center, Palo Alto, California, USA.
52. Pratt, V. R.: 1977, 'Two Easy Theories Whose Combination is Hard'. `http://boole.stanford.edu/pub/sefnp.pdf`.
53. Preparata, F. P. and S. J. Hong: 1977, 'Convex Hulls of Finite Sets of Points in Two and Three Dimensions'. *Communications of the ACM* **20**(2), 87–93.
54. Preparata, F. P. and M. I. Shamos: 1985, *Computational Geometry*, Texts and Monographs in Computer Science. Springer.
55. Pugh, W.: 1992, 'The Omega test: a fast and practical integer programming algorithm for dependence analysis'. *Communications of the ACM* **8**, 102–114.
56. Sankaranarayanan, S., M. Colón, H. B. Sipma, and Z. Manna: 2006, 'Efficient Strongly Relational Polyhedral Analysis.'. In: E. A. Emerson and K. S. Namjoshi (eds.): *Verification, Model Checking and Abstract Interpretation*, Vol. 3855 of *LNCS*. Charleston, South Carolina, USA, pp. 111–125.

57. Sankaranarayanan, S., F. Ivančić, and A. Gupta: 2007, 'Program Analysis Using Symbolic Ranges'. In: H. R. Nielson and G. Filé (eds.): *Static Analysis Symposium*, Vol. 4634 of *LNCS*. Kongens Lyngby, Denmark, pp. 366–383.
58. Schrijver, A.: 1998, *Theory of Linear and Integer Programming*. John Wiley & Sons.
59. Sedgewick, R.: 1988, *Algorithms in C*. Addison-Wesley.
60. Seidel, R.: 1997, *Convex Hull Computations*, pp. 361–376. In: J. E. Goodman and J. O'Rourke (eds.): *Handbook of Discrete and Computational Geometry*. CRC Press.
61. Shaham, R., H. Kolodner, and M. Sagiv: 2000, 'Automatic Removal of Array Memory Leaks in Java'. In: D. A. Watt (ed.): *Compiler Construction*, Vol. 1781 of *LNCS*. Berlin, Germany, pp. 50–66.
62. Shostak, R.: 1977, 'On the SUP-INF method for proving Presburger formulas'. *Journal of the ACM* **24**(4), 529–543.
63. Shostak, R.: 1981, 'Deciding Linear Inequalities by Computing Loop Residues'. *Journal of the ACM* **28**(4), 769–779.
64. Simon, A.: 2005, 'Relational Analysis of Floating-Point Arithmetic'. In: *Workshop on Numerical and Symbolic Abstract Domains*. Paris, France.
65. Simon, A.: 2006, 'Value-Range Analysis of C Programs with Focus on Finding Buffer Overflow Vulnerabilities'. Ph.D. thesis, Computing Laboratory, University of Kent.
66. Simon, A.: 2008, 'Splitting the Control Flow with Boolean Flags'. In: M. Alpuente and G. Vidal (eds.): *Static Analysis Symposium*, Vol. 5079 of *LNCS*. Valencia, Spain, pp. 315–331.
67. Simon, A. and A. King: 2002, 'Analyzing String Buffers in C'. In: H. Kirchner and C. Ringeissen (eds.): *Algebraic Methodology and Software Technology*, Vol. 2422 of *LNCS*. Reunion Island, France, pp. 365–379.
68. Simon, A. and A. King: 2004, 'Convex Hull of Planar H-Polyhedra'. *International Journal of Computer Mathematics* **81**(4), 259–271.
69. Simon, A. and A. King: 2005, 'Exploiting Sparsity in Polyhedral Analysis'. In: C. Hankin and I. Siveroni (eds.): *Static Analysis Symposium*, Vol. 3672 of *LNCS*. London, UK, pp. 336–351.
70. Simon, A. and A. King: 2006, 'Widening Polyhedra with Landmarks'. In: N. Kobayashi (ed.): *Asian Symposium on Programming Languages and Systems*, Vol. 4279 of *LNCS*. Sydney, Australia, pp. 166–182.
71. Simon, A., A. King and J. M. Howe: 2010, 'The Two Variable Per Inequality Abstract Domain: Proofs'. Technical report.
72. Simon, A., A. King, and J. M. Howe: 2003, 'Two Variables per Linear Inequality as an Abstract Domain'. In: M. Leuschel (ed.): *Logic-Based Program Synthesis and Transformation*, Vol. 2664 of *LNCS*. Madrid, Spain, pp. 71–89.
73. Su, Z. and D. Wagner: 2005, 'A Class of Polynomially Solvable Range Constraints for Interval Analysis without Widenings'. *Theoretical Computer Science* **345**(1), 122–138.
74. Toussaint, G. T. and D. Avis: 1982, 'On a convex hull algorithm for polygons and its application to triangulation problems'. *Pattern Recognition Letters* **15**(1), 23–29.
75. Toussaint, G. T. and H. E. Gindy: 1983, 'A counterexample to an algorithm for computing monotone hulls of simple polygons'. *Pattern Recognition Letters* **1**, 219–222.
76. Wagner, D.: 2000, 'Static analysis and computer security: New techniques for software assurance'. Ph.D. thesis, University of California at Berkeley.

77. Wayne, K. D.: 1999, ‘A polynomial combinatorial algorithm for generalized minimum cost flow’. In: *Theory of Computing*. Atlanta, Georgia, USA, pp. 11–18.