The Verified Polyhedron Library: an overview

Sylvain Boulmé, Alexandre Maréchal, David Monniaux, Michaël Périn, Hang Yu

The Verified Polyhedron Library: an overview

Sylvain Boulmé Alexandre Maréchal David Monniaux
Michaël Périn Hang Yu

December 30, 2018

Abstract

The Verified Polyhedra Library operates upon a constraint-only representation of convex polyhedra and provides all common operations (image, pre-image, projection, convex hull, widening, inclusion and equality tests...). Optionally, the soundness of the results is checked by a layer certified in Coq.

1 Introduction and motivation

By *convex polyhedra* we mean the solution sets in $\mathbb{Q}^n$, or equivalently $\mathbb{R}^n$,[1] of systems of linear inequalities with integer (or rational) coefficients. When $n = 2$ they are known as *convex polygons*; the case $n = 3$ is probably the most familiar, but here we are concerned with higher dimensions ($n \geq 10$).

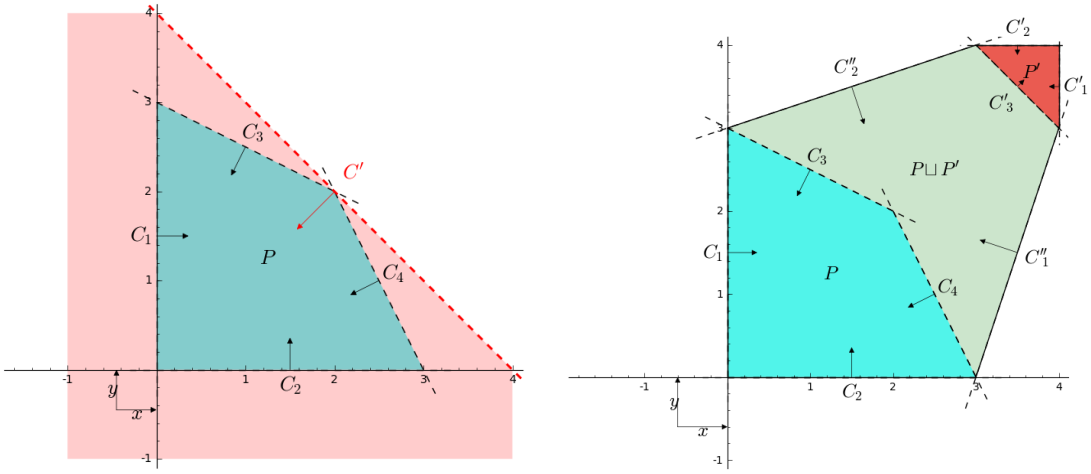
VPL, the *Verified Polyhedron Library*[2] is a library for computing over convex polyhedra in arbitrary dimension. It is implemented in OCaml (though an optional fast simplification procedure is implemented in C++), with an optional layer in Coq for verified computations.

In general, convex polyhedra may be specified by strict (e.g. $x + y < 1$) or non-strict (e.g. $x + y \leq 1$) inequalities. In this paper, we shall cover only the non-strict case, even though VPL also supports strict inequalities. All our polyhedra shall thus be convex and topologically closed.

Our library provides all operations commonly used in static analysis and other applications of polyhedra: projection, inclusion testing, equality testing, convex hull, image and pre-image by a linear map, widening operator

[1] There is no need to distinguish the real and rational cases, since in such a polyhedron any real point will also have rational points in its neighborhood. The integer case is very different and we shall not discuss it here.

[2] Or alternatively the Verimag Polyhedron Library, or the Verasco Polyhedron Library, since it originated in the Verasco project.



(a) (b)

Figure 1: Representation of polyhedron $P : \{C_1 : x \geq 0,\ C_2 : y \geq 0,\ C_3 : x + 2y \leq 6,\ C_4 : 2x + y \leq 6\}$.
(a): Constraint $C' : x + y \leq 4$ includes P, because $C' = \frac{1}{3}C_3 + \frac{1}{3}C_4$.
(b): The convex hull of P and $P' : \{C'_1 : x \leq 4,\ C'_2 : y \leq 4,\ C'_3 : x + y \geq 7\}$ yields new constraints $C''_1 : 3x - y \leq 9$ and $C''_2 : -x + 3y \leq 9$.

[17, §5.3, p. 56] [2] for enforced convergence of ascending iterations. It is available at https://github.com/VERIMAG-Polyhedra/VPL

1.1 Static analysis

Our main target application is *static analysis by abstract interpretation*. In Floyd-Hoare program correctness proofs, a human provides *inductive invariants*, that is, properties that can be shown by induction to hold for all program execution steps, for all iterations of a loop, for all recursive calls of a function, etc. These invariants are then shown to entail the desired *safety properties* (e.g. no runtime error, no assertion violation, etc.).[3] The inductiveness step proof was initially done by paper-and-pencil, which is tedious and error-prone; since then *proof assistants* have appeared, but they make this step only partially automatic. In automated proofs by abstract

[3]There are also *liveness* properties, e.g. the program eventually terminates. They are typically proved by exhibiting a well-founded relation that decreases along iterations. This decrease is typically shown using inductive invariants.

interpretation, an inductive invariant is automatically searched for within a predefined class of invariants known as an *abstract domain*. For instance, for proving that a program will never produce arithmetic overflows, a natural abstraction is to consider one interval per numeric variable.

Interval arithmetic, especially if combined with certain refinements (e.g. partitioning of execution traces according to conditions, some limited symbolic propagation...), proves many properties. It is however quite weak. For instance, after a sequence of instructions $y := x; z := y - x$ ran on the initial condition that $x \in [0, 1]$, interval arithmetic computes $y \in [0, 1]$ and $z \in [-1, 1]$, whereas one would expect $z = 0$. This is inevitable for any *non relational* abstraction: in order to establish $z = 0$ one needs to know a relation between x and y. Relational numeric domains have thus been proposed, among which convex polyhedra [10, 17].

Unfortunately, the most common approaches for computing over convex polyhedra tended to be highly inefficient as the number of dimensions increased ("curse of dimensionality"), which discouraged from using convex polyhedra. This is an issue that we wished to address in VPL.

1.2 Double vs single description

Most libraries for computing over convex polyhedra (NewPolka,[4] Parma Polyhedra Library,[5] PolyLib,[6] CDD...[7]) use a *double description* [32] of a polyhedron both as the solution set of a system of constraints (inequalities and equalities) and as the convex hull of a system of generators — in the case of a bounded polyhedron, the generators are its vertices. These descriptions are dual in the sense of convex duality. Some operations are easier on one description than on the other, and some, such as pruning redundant constraints or generators, are easier if both are available. Conversion between descriptions is done by Chernikova's algorithm [27].

This approach has many advantages but one major weakness: on cases very common in program verification, such as when one interval is known per variable, the generator description has size exponential in the number of variables. In other words, the hypercube $[0, 1]^n$ is defined by $2n$ inequalities ($0 \leq v_i \leq 1$ for each variable v_i) but 2^n vertices (choose each v_i in $\{0, 1\}$). This specific example can be dealt with by detecting that the polyhedron is

[4] Bertrand Jeannet's NewPolka is available as part of the Apron library of abstract domains, http://apron.cri.ensmp.fr/library/ [19]

[5] https://www.bugseng.com/ppl [3]

[6] https://icps.u-strasbg.fr/PolyLib/

[7] https://www.inf.ethz.ch/personal/fukudak/cdd_home/

a Cartesian product of intervals, or more generally of simple polyhedra [18, 34], but this breaks if the polyhedron is *almost* a Cartesian product. This was one motivation for using only constraints.

1.3 Certified computation

Static analysis and verification tools may be used for proving properties of safety-critical software — as an example, the Astrée static analyzer was aimed at fly-by-wire aircraft controls. Can the tools themselves be trusted?

The same question was asked of compilers. Designers of safety-critical systems and certification authorities may be unwilling to trust complex optimizing compilers; in fact some disable all optimizations so as to be able to easily match the source and assembly codes. One more satisfying answer is CompCert,[8] a compiler for a large subset of the C programming language such that there are mathematical definitions of the semantics of the source and target languages and a theorem that the compiler always preserves the semantics. This theorem is verified in the Coq proof assistant,[9] meaning that the compiler and the theorem statements are described in mathematical terms, then given to Coq along with all steps in the proof, for verification.

It was then decided to experiment implementing a certified static analysis tool over CompCert, called Verasco [23, 22, 26]. This implied that all algorithms whose correctness was relied upon for proving the overall correctness of the analysis tool must be proved correct, thus the need for a verified library for computing over convex polyhedra used as an abstract domain. More specifically, it was necessary to prove that the result of each polyhedron operation includes the ideal result that should be computed, which is used for showing that the analysis does not "forget" reachable states of the program.

It seemed difficult to use the double description in this context. In order to show that a constraint representation includes the ideal result that should be computed, it is sufficient to show that each constraint produced includes that ideal result; this property may be established separately for each constraint. In contrast, in order to show that a generator representation includes the ideal result, one must somehow show that the set of generators produced does not miss any; this is a global property of the generator representation. One way would have been to produce a certified implementation of Chernikova's algorithm, but it seemed likely that an implementation of

[8] http://compcert.inria.fr/ https://www.absint.com/compcert/
[9] https://coq.inria.fr/

this algorithm in Coq's programming language, which suffers from several limitations compared to usual programming languages, would be inefficient. Also, the proof effort seemed important.

Another possible approach would have been to use an efficient, but unverified, implementation and check *a posteriori* each result: whether a given generator representation and a given constraint representation define the same polyhedron — where the hard part is checking that there is no missing generator. Unfortunately, this problem is hard [25].

Our choice was thus to compute only on the constraint representation. Each constraint produced in a result polyhedron would come with a certificate that this constraint was truly correct in the result. The results and certificates would be then fed to a simple verification procedure, proved correct in Coq.

2 Farkas' lemma and certificates

It is well known that if one has a system of linear inequalities, one obtains consequences by multiplying both sides of each inequality by the same nonnegative coefficient and summing all resulting inequalities. The converse is known as Farkas' lemma[10] : any linear inequality (over the rational or the reals) that is a consequence of a system of linear inequalities can be expressed as a nonnegative combination of these inequalities, plus possibly some relaxation of the constant term, corresponding to a combination with the trivial inequality $T : 0 \leq 1$.

Example 1. Consider the polyhedron P on Figure 1(a). All points in P satisfy $x+y \leq 4$. This inequality can be obtained by summing $\frac{1}{3}$ of C_3 and $\frac{1}{3}$ of C_4. In other words, the vector of Farkas coefficients w.r.t. (C_1, C_2, C_3, C_4) is $\Lambda = \left(0, 0, \frac{1}{3}, \frac{1}{3}\right)$.

Thus, to show that a polyhedron Q includes a polyhedron P, it is sufficient to exhibit for each constraint C of Q a vector of nonnegative coefficients such that applying these coefficients to the constraints of P yields C. These constitute a *certificate of inclusion*.

In order to show that a polyhedron Q includes the projection of a polyhedron P parallel to variables V, it is sufficient to exhibit such a certificate of inclusion and check that the constraints of Q do not refer to the variables in V.

[10] also known as the strong duality property of linear programming.

Example 1 (continuing from p. 5)**.** The projection of P onto the x axis parallel to y is the segment $[0, 3]$. Inequality $x \geq 0$ is exactly C_1, thus is obtained by Farkas vector $(1, 0, 0, 0)$. Equality C_2 being the same as $-y \leq 0$, inequality $x \leq 3$ is obtained by summing $\frac{1}{2}$ times C_2 and $\frac{1}{2}$ times C_4, thus by Farkas vector $\left(0, \frac{1}{2}, 0, \frac{1}{2}\right)$. Note how the coefficients for y vanish for projection parallel to y.

In order to show that a polyhedron Q includes the convex hull of a family P_i of polyhedra, it is sufficient to exhibit for each P_i a certificate of inclusion of P_i in Q.

Example 1 (continuing from p. 5)**.** Consider the polyhedron P' defined by $C'_1 : x \leq 4$, $C'_2 : y \leq 4$, $C'_3 : x + y \geq 7$. The convex hull of P and P', shown on Figure 1(b), is defined by $P \sqcup P' = \{C_1,\ C_2,\ C'_1,\ C'_2,\ C''_1,\ C''_2\}$ where $C''_1 : 3x - y \leq 9$ and $C''_2 : -x + 3y \leq 9$.

One can check that $P \sqcup P'$ includes both P and P'. For instance, C''_1 is expressed as $\frac{5}{2}C_2 + \frac{3}{2}C_4$, which means that C''_1 includes P. C''_1 is also expressed as $C'_3 + 4C'_1$, hence C''_1 also includes P'. Similarly C''_2 includes both P and P', and so do C_1, C_2, C'_1 and C'_2.

Some operations do only need very trivial Farkas certificates. In order to show that a polyhedron Q includes the intersection of a family of polyhedra P_i, it is sufficient to check that each constraint of Q is among the constraints of one of the P_i. Similarly, in order to show that one polyhedron is included in its "simplification" by removal of redundant constraints, it is sufficient to check that each constraint in the simplified polyhedron was in the original polyhedron.

Example 2. Consider the polyhedra P_1 and P_2 on Figure 2. Consider also $Q = \{C_1,\ C'_2,\ C_3,\ C'_4\}$. Q includes the intersection of P_1 and P_2 (denoted $P_1 \sqcap P_2$), since all its constraints belong to P_1 or to P_2. Similarly, the actual intersection $\{C'_1,\ C_2,\ C'_3,\ C_4\}$ includes $P_1 \sqcap P_2$.

The image of a polyhedron P over variables X by an affine linear map $X \mapsto AX + B$ is easily computed by projecting the intersection of P with the polyhedron $X' = AX + B$ onto the variables X' (and renaming X' into X).

Example 3. Let P be defined by $0 \leq x \leq 1 \wedge 0 \leq y \leq 1$. Its image by $(x, y) \mapsto (x, x + y)$ is obtained by projecting $x' = x + y \wedge y' = y \wedge 0 \leq x \leq 1 \wedge 0 \leq y \leq 1$ onto x' and y' parallel to x and y.

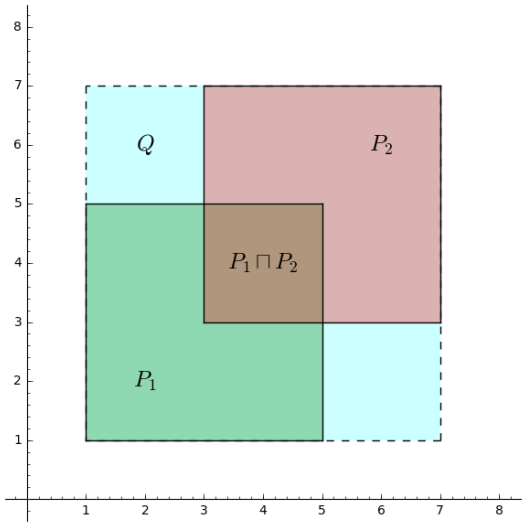


Figure 2: Intersection $P_1 \sqcap P_2$ of
polyhedra $P_1 : \{C_1 : x \geq 1,\ C_2 : x \leq 5,\ C_3 : y \geq 1,\ C_4 : y \leq 5\}$
and $P_2 : \{C'_1 : x \geq 3,\ C'_2 : x \leq 7,\ C'_3 : y \geq 3,\ C'_4 : y \leq 7\}$.

The inverse image of a polyhedron P by an affine linear map $X \mapsto AX + B$ is obtained by substituting $AX + B$ into X in the constraints defining P.

Example 3 (continuing from p. 6)**.** The reverse image of P by $(x, y) \mapsto (x, x + y)$ is $0 \leq x \leq 1 \wedge 0 \leq x + y \leq 1$.

3 Canonical form

3.1 Empty interior

The *affine span* of a polyhedron is the least affine subspace containing it. For instance, that of the polyhedron defined by

$$x \leq y + 1 \wedge y \leq z - 1 \wedge z \leq x \wedge 0 \leq x \wedge y \leq 0 \tag{1}$$

is the line defined by $x = y + 1 = z$. The polyhedron has *nonempty interior* (meaning it contains an open ball) if and only its affine span is the full ambient space.

Convex polyhedra with nonempty interior have a canonical representation as a list of irredundant inequalities (see 3.2 for more about redundancy) corresponding to the faces of the polyhedron. This representation is unique up to reordering and scaling of the constraints (that is, $2x + 2y \leq 2$ is the same as $x + y - 1 \leq 0$). However, this is not true of polyhedra with empty interior, such as the one defined by (1), which could be also defined by the system

$$y \leq x \wedge x \leq y + 1 \wedge y + 1 \leq z \wedge -1 \leq y \wedge z \leq 1. \tag{2}$$

In order to recover canonicity, we first echelonize the system of equalities defining the affine span: we get x and y as a function of z, $x = z$ and $y = z - 1$. Echelon form assumes an ordering of the variables (here, $x \prec y \prec z$), and canonicity is relative to a specific ordering. We get $n - s$ equations if n is the dimension of the ambient space and s that of the affine span. Then, we use these equations as rewriting rules (the variable on the left-hand side is replaced by the right-hand side): system 1 becomes

$$z \leq z \wedge z \leq z \wedge z \leq z \wedge 0 \leq z \wedge z \leq 1, \tag{3}$$

thus, after discarding trivial inequalities and adding the equalities, the canonical representation

$$0 \leq z \wedge z \leq 1 \wedge x = z \wedge y = z - 1. \tag{4}$$

Note that the system of inequalities (without equalities) defines a polyhedron with nonempty interior with respect to a reduced number of variables. Since several of our algorithms require an interior point, equations must be extracted prior to running them. We thus maintain a representation of polyhedra as a system of equalities and a system of inequalities defining a polyhedron with nonempty interior.

How do we detect the implicit equalities arising from a system of inequalities? If we have an inequality $l(x, y, \dots) \leq b$ and $C_1, \dots, C_n$ the other inequalities or equalities, then this inequality is actually an equality if and only if the lower bound on $l(x, y, \dots)$ on $C_1 \wedge \cdots \wedge C_n$ is b, or, equivalently, if $l(x, y, \dots) < b \wedge C_1 \wedge \cdots \wedge C_n$ has no solution. Equivalently, during the search for an interior point, away from the constraints, the constraints that stay "stuck" are identified as equalities.

For operations such as projection and convex hull, it is possible to compute in advance the affine span of the result. The affine span of the projection is the projection of the affine span, which is computed by echelonizing with the variables to be projected being first in the ordering. For instance, (4) is

echelonized for $x \prec y \prec z$, so if we wish to project parallel to x onto y and z we just discard the equation defining x.

The affine span of the convex hull of polyhedra of affine spans A_1 and A_2 can be obtained by posing $x = x_1 + x_2 \wedge x_1 \in A_1 \wedge x_2 \in A_2$ and projecting out x_1 and x_2 as above.

3.2 Redundancy elimination

A system of inequalities defining a convex polyhedron may contain redundant inequalities. Some of these are syntactically redundant: trivially true ones ($0 \leq 1 \ldots$) and those that are immediately subsumed by another (e.g. $x + y \leq 3$ is subsumed by $x + y \leq 1$; $2x + 2y \leq 2$ is the same as $x + y \leq 1$). Such redundancy may be easily eliminated:

1. Discard all trivially true inequalities (and if a trivially false inequality, e.g. $0 \leq -1$, appears, return that the polyhedron is empty).
2. Move all linear factors to the left-hand side and constants to the right-hand side.
3. Remove denominators from the linear part of other inequalities, e.g. $\frac{1}{2}x + \frac{1}{3}y \leq \frac{1}{5}$ becomes $3x + 2y \leq \frac{6}{5}$.
4. Remove common factors from the left hand side, e.g. $2x + 2y \leq 1$ becomes $x + y \leq \frac{1}{2}$.
5. Put the inequalities in a map from left-hand side to right-hand side, when an existing mapping is encountered, replace it if it is strengthened: if $x + y \leq 1$ is present and $x + y \leq 2$ is encountered, ignore the latter; if $x + y \leq 2$ is present and $x + y \leq 1$ is encountered, strengthen.

The general case, where an inequality is redundant with respect to several others, is more difficult. In order to know if $Ax \leq b$ is redundant with respect to other inequalities $C_1, \ldots, C_n$ one can maximize Ax subject to $C_1, \ldots, C_n$ (linear programming): the inequality $Ax \leq b$ is redundant if and only if the resulting bound b' is less than or equal to b. One can also solve for a solution of $Ax > b \wedge C_1 \wedge \cdots \wedge C_n$ (linear programming with no objective): $Ax \leq b$ is redundant if an only if this system has no solution. Equivalently, by Farkas' lemma, one may dually search for nonnegative coefficients such that $Ax \leq b$ is obtained from $C_1, \ldots, C_n$.

In any case, these methods solve n linear programs in order to check whether each of the constraints is redundant with respect to the $n - 1$ others, which is expensive. We thus searched for a cheaper method for solving the "easy" cases. Our insight was that if one has a point x_0 in the interior of a polyhedron, and an arbitrary nonzero vector v, then the first constraint hyperplane encountered by the ray $\{x_0 + tv \mid t > 0\}$ is a

face of the polyhedron — meaning this constraint is irredundant (the case where several faces are encountered for the same minimal t is exceptional and we shall not discuss it here). By trying several v's our *ray-tracing* [29, 28] algorithm often establishes quickly that a number of constraints are irredundant. Our scheme then establishes redundancy or irredundancy of other constraints by solving mostly small linear programs that involve only irredundant constraints, though in the worst case it boils down to the naive algorithm. We have a new implementation of this scheme where the redundancy of each constrained is tested in parallel.

This redundancy elimination may be used to implement an inclusion test: $P \leq Q$ if and only if all constraints in Q are redundant with respect to P.

4 Algorithms based on projection

Our first-generation algorithms were based on Fourier-Motzkin projection and elimination of redundant constraints. *These algorithms are described in more detail, with proofs, in Alexis Fouilhé's thesis [12] and paper [14].*

4.1 Fourier-Motzkin elimination

The Fourier-Motzkin projection [15, 24] of a polyhedron P parallel to a variable x consists in collecting into a set C_+ all constraints of the form $x \leq f_+$ where f_+ does not involve x, into a set C_- all constraints of the form $x \geq f_-$ where f_- does not involve x, and into a set C_0 all constraints where x does not appear. Fourier-Motzkin projection produces the constraints $f_- \leq f_+$ where f_- and f_+ range in C_+ and C_- respectively, to which are added the constraints from C_0.

Example 1 (continuing from p. 5)**.** Let us eliminate y, that is, project onto x parallel to y, in the system $C_1 \wedge C_2 \wedge C_3 \wedge C_4$ defining P. $C_+ = \{C_3, C_4\}$, $C_- = \{C_2\}$, $C_0 = \{C_1\}$. $C_4 : y \leq -2x + 6$ and $C_2 : y \geq 0$ yield $0 \leq -2x + 6$, which simplifies into $x \leq 3$. $C_3 : y \leq -\frac{1}{2}x + 3$ and $C_2 : y \geq 0$ yield $0 \leq -\frac{1}{2}x + 3$, which simplifies into $x \leq 6$. Obviously this last constraint is redundant.

By instrumenting the elimination process, one can attach to each generated constraint some Farkas coefficients in terms of the original constraints, which will ultimately constitute a certificate that the result of the projection process includes the projection. In fact, this is a way of proving Farkas' lemma.

In the worst case, the set of n constraints of P splits evenly into C_+ and C_- each of size $n/2$, and thus there are $|C_+|.|C_-| = n^2/4$ constraints in the output. Fourier-Motzkin elimination tends to generate many redundant constraints. Some of them, which boil down to $0 \leq 1$, may be eliminated straight away, but for most of them general redundancy elimination is needed.

When projecting out several variables, one may choose their ordering arbitrarily. A common heuristic is to project out first the variable that minimizes $|C_+|.|C_-|$, to eliminate redundant constraints, and to continue the process until all variables to be projected are eliminated.

Since the elimination of a single variable may yield $n^2/4$ constraints from a polyhedron with n constraints, thus iterating the process over p variables to project can produce at most $n^{2^p}/4^{2^p-1}$ constraints. There exist examples that exhibit such double exponential behavior [24, ex. 1]. However, this double exponential relies on redundant inequalities. If redundant inequalities are eliminated, only single exponential growth is possible [24], due to the upper bound theorem [30].

4.2 Convex hull by projection

Let P_1 and P_2 be two polyhedra defined by $A_1X_1 \leq B_1$ and $A_2X_2 \leq B_2$. X is in the convex hull of P_1 and P_2 if and only if there exists $0 \leq \alpha_1, \alpha_2 \leq 1$, $\alpha_1 + \alpha_2 = 1$, X_1 and X_2 such that $X = \alpha_1 X_1 + \alpha_2 X_2$, $A_1X_1 \leq B_1$ and $A_2X_2 \leq B_2$.

Assume now $0 < \alpha_1, \alpha_2 < 1$. By posing $Y_1 = \alpha_1 X_1$ and $Y_2 = \alpha_2 X_2$ we get the equivalent conditions $X = Y_1 + Y_2$, $\alpha_1 + \alpha_2 = 1$, $A_1Y_1 \leq \alpha_1 B_1$ and $A_1Y_1 \leq \alpha_2 B_2$. These inequalities are linear in α_1, α_2 and the components of X, Y_1 and Y_2, thus it is possible to apply polyhedral projection parallel to α_1, α_2 and the components of Y_1 and Y_2 to obtain a polyhedron over X. This polyhedron is the least closed convex polyhedron including P_1 and P_2 [4].[11]

Note that this approach for computing the convex hull of two polyhedra in dimension d poses a projection problem from dimension $3d + 1$ into dimension d. It is thus particularly sensitive to the cost of projection in high dimension.

[11] In the case of unbounded polyhedra this may be strictly greater than the convex hull of P_1 and P_2, for instance if $P_1 = \{(0, 1)\}$ and $P_2 = \{(x, 0) \mid x \in \mathbb{R}\}$, whose convex hull is $\mathbb{R} \times [0, 1) \cup \{(0, 1)\}$, whereas our algorithm produces $\mathbb{R} \times [0, 1]$.

5 Parametric linear programming

We developed algorithms for projection, convex hull and thus all operations based on parametric linear programming [20, 21]. *These algorithms are described in more detail, with proofs, in Alexandre Maréchal's thesis [28] and paper [1].*

5.1 Algorithms using parametric linear programming

5.1.1 Projection

Let P be a polyhedron defined by $AX + BY \leq C$. If Λ is a nonnegative column vector, then all points (X, Y) in P satisfy $\Lambda AX + \Lambda BY \leq \Lambda C$. If Λ is chosen so that $\Lambda B = 0$, then all points X in the projection of P parallel to Y satisfy $\Lambda AX \leq \Lambda C$.

Assume P has a point (X_0, Y_0) in its interior, and let $X \neq X_0$. Find Λ such that $\Lambda(C - AX_0) = 1$ and $\Lambda B = 0$ such that $\Lambda(C - AX)$ is minimal. Then, $(\Lambda A)X \leq \Lambda C$ defines the face of the projection of P onto X parallel to Y that one encounters on an infinite ray starting in X_0 and pointed towards X.[12]

By varying X, we obtain all the faces. Geometrically, there is one polyhedral cone per face, pointed at X_0, such that if X lies in the cone attached to a face then the associated optimum described the face. These cones are quasi-disjoint: they overlap only at their boundaries. They describe the results of parametric linear programming where X is the parameter, occurring in the objective function only.

Example 4. Consider P defined by $-y \leq -1$, $-x \leq -1$, $y \leq 3$, $x - y \leq 2$, $z - x \leq 5$, $-z \leq -3$ with $X = (x, y)^T$ and $Y = (z)$. Let the interior point be $(2, 2, 4)$, and then $\Lambda(1, 1, 1, 2, 3, 1)^T = 1$, $\Lambda(0, 0, 0, 0, 1, -1)^T = 0$. By minimizing $\Lambda(y - 1, x - 1, -y + 3, -x + y + 2, x, 0)^T$ for all values of the parameters (x, y), we will obtain the projection of P parallel to z: $y - 1 \geq 0, -x + y + 2 \geq 0, -y + 3 \geq 0, x - 1 \geq 0$.

5.2 Convex hull

Let P_1 be the polyhedron defined by $A_1X_1 \leq B_1$, P_2 the polyhedron defined by $A_2X_2 \leq B_2$. If Λ_1 and Λ_2 are nonnegative column vectors, then all points

[12] There is also a λ_0 for the constraint $0 \leq 1$, used when the projection is unbounded in the selected direction. The same applies to convex hull.

X in the convex hull of P_1 and P_2 satisfy $\Lambda_1 A_1 X \leq \Lambda_1 B_1$ and $\Lambda_2 A_2 X \leq \Lambda_2 B_2$.

Assume the convex hull has a point X_0 in its interior, and let $X \neq X_0$. Find Λ_1, Λ_2 such that $\Lambda_1 A_1 = \Lambda_2 A_2$, $\Lambda_1 B_1 = \Lambda_2 B_2$, $\Lambda_1(B_1 - A_1 X_0) = 1$ such that $\Lambda_1(B_1 - A_1 X)$ is minimal. Then, $(\Lambda_1 A_1)X \leq \Lambda_1 B_1$ (or equivalently $(\Lambda_2 A_2)X \leq \Lambda_2 B_2$) defines the face of the convex hull of that one encounters on an infinite ray starting in X_0 and pointed towards X. As above, by varying X we obtain all the faces of the convex hull.

5.3 Solving the parametric linear program

We consider linear programs where the objective function depends linearly on parameters, e.g. $o(x, y) = x(1 + \alpha) + y(\alpha + \beta)$ where α and β are parameters (in a dual variant, the parameters are in the constant part of the inequalities).

5.3.1 The simplex algorithm by example

We shall not describe in detail the simplex algorithm, since there exist excellent textbooks covering it [11, 33, 9]. Instead we shall demonstrate it working on a simple example.

Example 5. Pose $z = 6 - x - 2y$, $t = 6 - 2x - y$. Then P from Example 1 is defined by $x, y, z, t \geq 0$ and a system of two equalities, to which we add an equality defining our objective $o = x + y$.

$$\begin{cases} z = 6 & -x - 2y \\ t = 6 & -2x - y \\ o = & x + y \end{cases} \tag{5}$$

The variables on the left-hand side are called *basic* and those on the right-hand side are called *nonbasic*; the partition of variables into basic and nonbasic is called a *basis*. Nonbasic variables x, y are considered to take 0 as a value[13] and thus $z = 6$, $t = 6$ and $o = 0$. Here, by chance, our initial basis yields a solution of the problem (z and t are also nonnegative) but in general we would have to run phase I of the simplex algorithm. However, this solution is nonoptimal: x and y are at their lower bound and by increasing either of them we can increase the objective from its current value of 0.

[13] Here we assume all variables to be nonnegative. In generalized versions of the simplex algorithm, variables may have both a lower and an upper bound, or even be unconstrained.

In order to increase x we have to make it basic, and make one of the basic variables nonbasic, an operation known as *pivoting*. Let us pivot x with t: we express $x = 3 - \frac{1}{2}y - \frac{1}{2}t$ and replace into the system

$$\begin{cases} x = 3 & -\frac{1}{2}y - \frac{1}{2}t \\ z = 3 & -\frac{3}{2}y + \frac{1}{2}t \\ o = 3 & +\frac{1}{2}y - \frac{1}{2}t \end{cases} \tag{6}$$

The objective now has value 3. It can still be improved by increasing y.

Let us pivot y with z: we express $y = 2 - \frac{2}{3}z + \frac{1}{3}t$ and obtain

$$\begin{cases} y = 2 & -\frac{2}{3}z + \frac{1}{3}t \\ x = 2 & +\frac{1}{3}z - \frac{2}{3}t \\ o = 4 & -\frac{1}{3}z - \frac{1}{3}t \end{cases} \tag{7}$$

At this point $(x, y) = (2, 2)$ and $o = 4$. The last line shows that o cannot be improved upon: one would need to decrease either z or t and both are already at their lower bound 0. That is, the final basis yields both a solution point and a proof that it is optimal.

5.3.2 Sign splitting

The simplex algorithm takes decisions according to the signs of coefficients in the line for the objective in its tableau. If the objective function is made parametric, then the signs of its coefficients may be indeterminate; the algorithm is then modified to branch on sign conditions. This modified simplex algorithm then explores a search tree with edges adorned with conditions (linear inequalities) on the parameters, closing branches when the conditions accumulated from the tree root are inconsistent (meaning there is no value of the parameters that can lead to that branch).

We initially experimented that approach, but opted against it since it tended to branch too much and was thus inefficient.

5.3.3 Generalization

Recall how in Example 5 the final basis yielded both a solution point and a proof of optimality with respect to a given non-parametric objective. We shall see here how to generalize this proof to a whole polyhedron of possible parameters.

Consider a parametric objective $C(\Lambda)$, which we aim to maximize over $AX \leq B$, meaning we maximize the product $C(\Lambda).X$. Pick a Λ_0 and run the

simplex algorithm. It will eventually stop with an indication that either the problem is unsolvable (and will be unsolvable for any value of Λ), or it has an optimal solution, or it is unbounded. In the last two cases, the result can be verified by examining the signs of the coefficients of the objective function in the simplex tableau, that is, the expression of the objective function (with parameter $\Lambda = \Lambda_0$) in the last basis explored by the simplex algorithm.

If we now consider C with arbitrary Λ, then these sign conditions become linear inequalities over Λ, which are met for $\Lambda = \Lambda_0$ but also for other values. The Λ meeting these conditions form a closed convex cone. By eliminating the redundant constraints from these conditions we obtain a description of the cone of parameters that lead to an optimum at this particular basis.

Example 5 (continuing from p. 13)**.** Assume now that $o = \lambda_1 x + \lambda_2 y$. In the last basis, o is to be expressed as a function of z and t, thus

$$o = 4 + (\lambda_1 - 2\lambda_2)z + (-2\lambda_1 + \lambda_2)t \tag{8}$$

The point $(x, y) = (2, 2)$ is optimal for any (λ_1, λ_2) satisfying

$$\begin{cases} \lambda_1 - 2\lambda_2 & \leq 0 \\ -2\lambda_1 + \lambda_2 & \leq 0 \end{cases} \tag{9}$$

We then explore the space of parameters by choosing Λ_1, Λ_2 etc. outside of the cones explored so far, until no Λ left uncovered by a cone is left.

For a parametric linear programming problem in a "general position", the cones produced by this algorithm are exactly the description of the parametric solution: one cone per optimum, and the cones overlap only at their boundaries. There exist however two kinds of degeneracy: (a) there may be two optimal vertices for the same objective function; (b) there may be two bases describing the same optimal vertex. Both kinds of degeneracy result in cones overlapping non trivially and in inefficiency. Several solutions are being investigated for this problem.

Example 6. Consider the pyramidal cone defined by $x+z \leq 1$, $-x+z \leq 1$, $y + z \leq 1$, $-y + z \leq 1$ and the parametric optimization direction $o(\lambda, \mu) = \lambda x + \mu y + z$. In other words, we are maximizing o such that $a, b, c, d \geq 0$ and

$$\begin{cases} a = 1 & -x & & -z \\ b = 1 & +x & & -z \\ c = 1 & & -y & -z \\ d = 1 & & +y & -z \\ o = & & \lambda x + \mu y & +z \end{cases} \tag{10}$$

There are four different bases that define the vertex $(x, y, z) = (0, 0, 1)$, obtained by picking three nonbasic variables out of a, b, c, d. In the basis where a, b, c are nonbasic,

$$o = (1 - \tfrac{1}{2}a - \tfrac{1}{2}b) + (-\tfrac{1}{2}a + \tfrac{1}{2}b)\lambda + (\tfrac{1}{2}a + \tfrac{1}{2}b - c)\mu \tag{11}$$

Thus, this basis is optimal for any λ, μ such that $\mu \geq 0$, $-\lambda + \mu \leq 1$, $\lambda + \mu \leq 1$. [14] The other three bases are obtained symmetrically and have regions of optimality

- $\mu \leq 0$, $-\lambda - \mu \leq 1$, $\lambda - \mu \leq 1$;
- $\lambda \geq 0$, $-\mu + \lambda \leq 1$, $\mu + \lambda \leq 1$;
- $\lambda \leq 0$, $-\mu - \lambda \leq 1$, $\mu - \lambda \leq 1$.

Almost all points in the $\pm\lambda \pm \mu \leq 1$ square belong to the interior of two such regions.

6 Certified results

Our library provides two kinds of certified results:

1. VPL provides a certified layer in Coq, but for an execution outside of Coq: this layer is extracted to OCaml and linked to the OCaml VPL code. It provides certified services: each function on polyhedra comes with a proof of soundness, e.g. the polyhedron result of the convex hull operators is shown to contain all points in the polyhedron operands. This Coq layer can thus be applied to certify in Coq a larger software package including VPL, as it was done for the Verasco certified static analyzer [23, 22].
2. VPL provides a tactic for interactive proofs in Coq, aimed at simplifying goals involving equalities and inequalities over $\mathbb{Q}$, sometimes completely proving them. The (possibly non-linear) equalities and inequalities over $\mathbb{Q}$ are "reified" into linear ones, where non-linear subterms are abstracted by variables. The resulting polyhedron is converted into canonical form using VPL, and the constraints produced by this simplification replace the original ones in the Coq goal. In particular, if the equalities and inequalities in the hypotheses form an empty polyhedron, that is, are contradictory, the goal is proved (*ex falso quodlibet*). See details in [6].

[14] This is a polyhedral region of optimality but not a cone because o is not homogeneous due to the constant coefficient on z.

Let us point out a major difference between these two. In the second kind, *for each polyhedral computation*, VPL builds a full Coq proof that the computation is correct (by computational reflection from Farkas certificates). On the contrary, in the first kind, VPL provides a Coq proof that *each polyhedra computation* using the Coq layer is correct (when it does not abort). This generic Coq proof is not built/instantiated at each run, since the computations are performed outside of Coq.

Below, we mainly focus on the first kind of certified results, but we also mention how our mechanisms for the first kind are reused for the second kind.

6.1 Explicit Farkas certificates

In its first release, VPL introduced an *abstract syntax* of certificates inspired by Farkas coefficients for inclusion proofs, inspired by [5]. For each computation of a polyhedron P, the untrusted OCaml operators were returning a certificate allowing the Coq layer to rebuild a correct-by-construction version of P (modulo some additional defensive checks)[15]. Such a certificate for building P corresponds in theory to a "Farkas matrix" (e.g. one Farkas vector for each constraint of P). However, we needed special support for equalities, in order to keep the polyhedra of the Coq layer in canonical form – even if the canonicity is not formally proved. Thus, our certificates were actually defined as ad-hoc expressions with some let-binders that allow to share intermediate derived equalities between computations of P constraints.

The benefit of this approach was to avoid converting both P and the Farkas matrix from the OCaml layer to Coq. Because Coq and OCaml used different representation for numbers, such conversion of whole polyhedra at each operation would induce a significant overhead. See [13, 12] for details.

This approach involved tedious renumbering of constraint names in certificates (while handling of our let-binders). Hence, lighter approaches were investigated, as detailed below.

6.2 LCF-style certificates

The OCaml code attaches to each constraint a certificate ensuring that it is a consequence of a given polyhedron (or several certificates for several polyhedra, as in convex hull). A certificate encodes the fact that

[15]For the projection P eliminating a variable y from a polyhedron P_0, the Coq layer builds P from P_0 using Farkas vectors, hence ensuring that P includes P_0, and checks that y does not appear in P.

$H_1, \ldots, H_n \vdash C$, which means "constraint C is a consequence of constraints $H_1, \ldots, H_n$". From the Coq point of view, the certificate includes a proof of $H_1, \ldots, H_n \vdash C$. This proof is a term belonging to a type in the Prop universe of propositions.

The Coq code then provides to the OCaml code functions for deduction steps, capable of operating over certificates, i.e.:
Plus takes certificates for $H \vdash C_1$ and $H \vdash C_2$, outputs one for $H \vdash C_1 + C_2$
Scale takes a certificate for $H \vdash C$ and $\lambda \geq 0$, outputs one for $H \vdash \lambda C$

The Coq certificates are seen from the OCaml library as belonging to an opaque type: this kind of certificates is known as "LCF-style" (ie the style of the LCF proof assistant). The only way the OCaml code may create new certificates is by calling the deduction steps provided from Coq. This enforces that the OCaml code may only perform legitimate deduction steps.[16]

One weakness of this approach is that H (the set of initial polyhedral constraints) must be carried throughout, and that for the plus operator it is necessary to check that the two sets H match. It would be more efficient to simply carry the information about C. Unfortunately, doing it without precaution would allow mixing between different operations: a bug in the OCaml library could possibly lead it to consider $H_1 \vdash C_1$ and $H_2 \vdash C_2$ where H_1 and H_2 are two distinct sets of initial constraints and to attempt deducing $C_1 + C_2$. The way we avoid such "cheating" is by making the OCaml code polymorphic in the type of the certificate.

6.3 Polymorphic LCF-style certificates

In our final implementation, the OCaml code sees the certificates through polymorphic types and operations supplied for performing deduction steps. Intuitively, a certificate for $H \vdash C$ just contains the constraint C, but its Coq type – depending on H – is generalized for the OCaml layer into a (universally quantified) type variable α.

Example 7. Assume an OCaml library that computes over an internal representation of inequalities. It handles certificates polymorphically, through functions provided to it by a record:

```
type inequality          type rational
type α ops = {
```

[16] This, as well as everything following in this section, assumes that the OCaml code does not call primitive functions that examine internal structures and allow distinguishing them by type, e.g. in the Obj module.

```
  plus : α → α → α;
  scale : α → rational → α;
  extract : α → inequality; }
val projection : α ops → α list → int → α list
```

An OCaml code with such an interface can handle the certificates only through the functions provided to it. Moreover, this code cannot store a certificate for $H \vdash C$, into a reference (e.g. inside a hash table) and extracting it later in another context H', for showing $H' \vdash C$. The typing of polymorphic references [16] forbids such an unsoundness.

That approach proved much simpler to debug than computing Farkas certificates as terms, both in the OCaml and Coq layers. Furthermore, it is still possible to rebuild the certificates as terms using the polymorphic oracles if needed (e.g. for our Coq tactic). See [8, 28] for details.

6.4 Formal Reasoning on Imperative Abstract Domains

While our OCaml code is meant to give deterministic results, it seems undesirable to rely on this unchecked property for soundness: a bug in the handling of their hidden state could make VPL functions appear as non-deterministic. Furthermore, in our current developments the OCaml code can call high performance C++ code for which we have no such guarantee. We have thus developed approaches for formally reasoning on such non-deterministic functions in Coq. In particular, the theory of abstract interpretation handles operators of abstract domains as pure functions. We have thus proposed a relaxed framework – based on data-refinement diagrams – for compositional reasoning on non-deterministic operators of a given abstract domain. *See [7] for more details.*

7 Linear approximations of semi-algebraic sets

Let us consider the solution set S of a system of polynomial (linear or non-linear) constraints $P_1(X) \geq 0, \ldots, P_m(X) \geq 0$ where X lies in $\mathbb{R}^n$. Any product $P_1^{k_1} \cdot \cdots \cdot P_m^{k_m} \geq 0$ is also a valid constraint over S, thus is also any nonnegative linear combination of these products.

We consider a heuristically selected collection of these products. Non-linear monomials are then considered as fresh variables; e.g. for $n = 4$, a monomial $X_1^3 X_2 X_4^1$ is replaced by a variable $M_{3,2,0,1}$ (the subscript is the vector of degrees of the monomial in the X variables). This relaxes the problem by discarding the nonlinear relations between the monomials. The

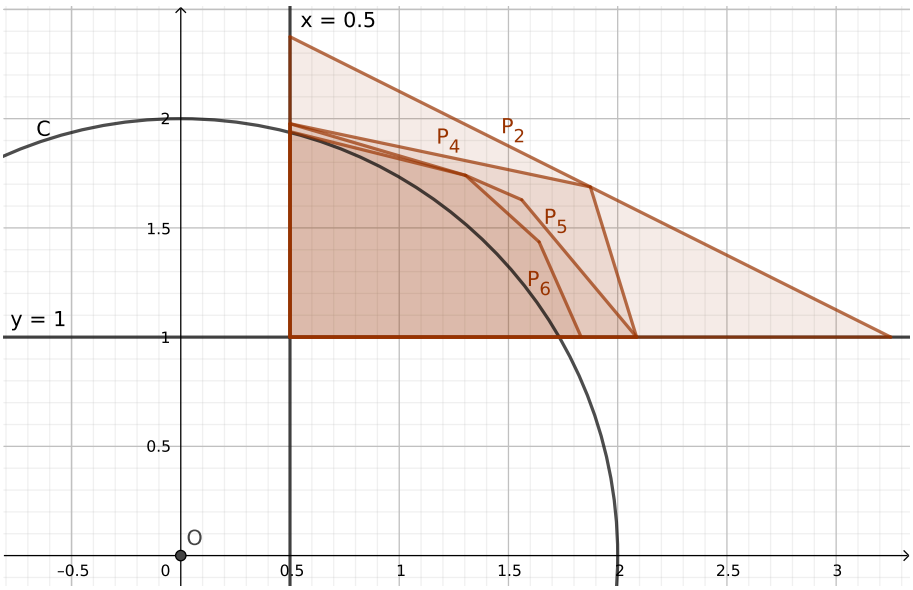


Figure 3: Over-approximations of the intersection of the quarter plane $x - 1/2 \geq 0$, $y - 1 \geq 0$ and the disc $4 - x^2 - y^2 \geq 0$: each P_d is obtained by truncating all polynomials involved to degree d.

system of constraints then describes a convex polyhedron in higher dimension.

This polyhedron is then projected onto the variables X, parallel to the variables M. This projection is an over-approximation of S (Figure 3).

8 Recent work and perspectives

We have worked on improving the performance of the parametric linear programming algorithm. We currently use the simplex algorithm implemented in arbitrary precision rational arithmetic. In contrast, most highly optimized implementations of that algorithm work using floating-point arithmetic. We have implemented a system that computes a solution using such a floating-point implementation, then reconstructs both the solution vertex and the objective function in exact arithmetic. If the solution is incorrect or nonoptimal, an exact implementation of the simplex algorithm is then run.

Furthermore, in the VPL, the exploration of the regions of the parametric linear program is serial. We have designed a parallel algorithm.

References

[1] David Monniaux Alexandre Maréchal and Michaël Périn. "Scalable Minimizing-Operators on Polyhedra via Parametric Linear Programming". In: *Static analysis (SAS)*. Ed. by Francesco Ranzato. Springer, 2017. DOI: 978-3-319-66706-5_11. HAL: hal-01555998.

[2] Roberto Bagnara, Patricia M. Hill, Elisa Ricci, and Enea Zaffanella. "Precise widening operators for convex polyhedra". In: *Sci. Comput. Program.* 58.1-2 (2005), pp. 28–56. DOI: 10.1016/j.scico.2005.02.003.

[3] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. "The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems". In: *Science of Computer Programming* 72.1–2 (2008), pp. 3–21. DOI: 10.1016/j.scico.2007.08.001. arXiv: cs/0612085.

[4] Florence Benoy, Andy King, and Frédéric Mesnard. "Computing Convex Hulls with a Linear Solver". In: *Theory and Practice of Logic Programming* 5.1-2 (2005). arXiv: cs/0311002.

[5] Frédéric Besson, Thomas P. Jensen, David Pichardie, and Tiphaine Turpin. "Certified Result Checking for Polyhedral Analysis of Bytecode Programs". In: *Trustworthy Global Computing (TGC)*. Vol. 6084. Lecture Notes in Computer Science. Springer, 2010, pp. 253–267. HAL: inria-00537816.

[6] Sylvain Boulmé and Alexandre Maréchal. "A Coq Tactic for Equality Learning in Linear Arithmetic". In: *Interactive Theorem Proving (ITP)*. Vol. 10895. Lecture Notes in Computer Science. Springer, 2018, pp. 108–125. DOI: 10.1007/978-3-319-94821-8_7. HAL: hal-01505598.

[7] Sylvain Boulmé and Alexandre Maréchal. "Refinement to Certify Abstract Interpretations, Illustrated on Linearization for Polyhedra". In: *J. Automated Reasoning* (Nov. 2018). DOI: 10.1007/s10817-018-9492-2. HAL: hal-01133865. URL: https://hal.archives-ouvertes.fr/hal-01133865.

[8] Sylvain Boulmé and Alexandre Maréchal. *Toward Certification for Free!* Preprint. July 2017. HAL: hal-01558252.

[9] Vašek Chvátal. *Linear Programming*. Series of books in the Mathematical Sciences. W. H. Freeman, 1983.

[10] Patrick Cousot and Nicolas Halbwachs. "Automatic Discovery of Linear Restraints Among Variables of a Program". In: *ACM SIGACT-SIGPLAN Symposium on Principles of programming languages (POPL)*. Ed. by Alfred V. Aho, Stephen N. Zilles, and Thomas G. Szymanski. ACM Press, 1978, pp. 84–96. DOI: 10.1145/512760.512770.

[11] George Dantzig and Munkund N. Thapa. *Linear Programming. Introduction*. Vol. 1. 2 vols. Springer, 1997. 480 pp. ISBN: 0-387-94833-3.

[12] Alexis Fouilhé. "Revisiting the abstract domain of polyhedra : constraints-only representation and formal proof". PhD thesis. Grenoble Alpes University, France, 2015. TEL: tel-01286086.

[13] Alexis Fouilhé and Sylvain Boulmé. "A Certifying Frontend for (Sub)polyhedral Abstract Domains". In: *Verified Software: Theories, Tools and Experiments (VSTTE)*. Vol. 8471. Lecture Notes in Computer Science. Springer, 2014, pp. 200–215. DOI: 10.1007/978-3-319-12154-3_13. HAL: hal-00991853.

[14] Alexis Fouilhé, David Monniaux, and Michaël Périn. "Efficient Generation of Correctness Certificates for the Abstract Domain of Polyhedra". In: *Static analysis (SAS)*. 2013. ISBN: 978-3-642-38855-2. DOI: 10.1007/978-3-642-38856-9_19. HAL: hal-00806990.

[15] Joseph Fourier. "Histoire de l'Académie, partie mathématique (1824)". In: *Mémoires de l'Académie des sciences de l'Institut de France*. Vol. 7. Gauthier-Villars, 1827, xlvij–lv. URL: https://gallica.bnf.fr/ark:/12148/bpt6k32227/f53.

[16] Jacques Garrigue. "Relaxing the Value Restriction". In: *Functional and Logic Programming, 7th International Symposium (FLOPS)*. Ed. by Yukiyoshi Kameyama and Peter J. Stuckey. Vol. 2998. Lecture Notes in Computer Science. Springer, 2004, pp. 196–213. DOI: 10.1007/978-3-540-24754-8_15. URL: https://caml.inria.fr/pub/papers/garrigue-value_restriction-fiwflp04.pdf.

[17] Nicolas Halbwachs. "Détermination automatique de relations linéaires vérifiées par les variables d'un programme". French. PhD thesis. Université Scientifique et Médicale de Grenoble & Institut National Polytechnique de Grenoble, Mar. 1979. HAL: tel-00288805.

[18] Nicolas Halbwachs, David Merchat, and Laure Gonnord. "Some ways to reduce the space dimension in polyhedra computations". In: *Formal Methods in System Design* 29.1 (2006), pp. 79–95. DOI: 10.1007/s10703-006-0013-2. URL: https://doi.org/10.1007/s10703-006-0013-2.

[19] Bertrand Jeannet and Antoine Miné. "Apron: a Library of Numerical Abstract Domains for Static Analysis". In: *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*. Ed. by Ahmed Bouajjani and Oded Maler. Vol. 5643. Lecture Notes in Computer Science. Springer, 2009, pp. 661–667. DOI: 10.1007/978-3-642-02658-4_52. URL: https://hal.archives-ouvertes.fr/hal-00786354.

[20] Colin. Jones, N., Eric C. Kerrigan, and Jan M. Maciejowski. "On Polyhedral Projections and Parametric Programming". In: *J. Optimization Theory and Applications* 138.2 (2008), pp. 207–220.

[21] Colin N. Jones, Eric C. Kerrigan, and Jan M. Maciejowski. "Lexicographic perturbation for multiparametric linear programming with applications to control". In: *Automatica* (43 2007). DOI: 10.1016/j.automatica.2007.03.008.

[22] Jacques-Henri Jourdan. "Verasco: a Formally Verified C Static Analyzer". PhD thesis. Université Paris Diderot - Paris VII, May 2016. TEL: tel-01327023.

[23] Jacques-Henri Jourdan, Vincent Laporte, Sandrine Blazy, Xavier Leroy, and David Pichardie. "A Formally-Verified C Static Analyzer". In: *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*. ACM, 2015, pp. 247–259. DOI: 10.1145/2676726.2676966. URL: https://doi.org/10.1145/2676726.2676966.

[24] Leonid Khachiyan. "Fourier–Motzkin Elimination Method". In: *Encyclopedia of Optimization*. Ed. by Christodoulos A. Floudas and Panos M Pardalos. 2nd ed. Springer, 2009, pp. 1074–1076. ISBN: 978-0-387-74760-6.

[25] Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled M. Elbassioni, and Vladimir Gurvich. "Generating All Vertices of a Polyhedron Is Hard". In: *Discrete & Computational Geometry* 39.1-3 (2008). Also as DIMACS TR 2005-21 http://archive.dimacs.rutgers.edu/pub/dimacs/TechnicalReports/TechReports/2005/2005-21.pdf, pp. 174–190. DOI: 10.1007/s00454-008-9050-5.

[26] Vincent Laporte. "Verified static analyzes for low-level languages". Theses. Université Rennes 1, Nov. 2015. TEL: tel-01285624.

[27] Hervé Le Verge. *A note on Chernikova's Algorithm*. Tech. rep. 635. IRISA, 1992. HAL: inria-00074895.

[28] Alexandre Maréchal. "New Algorithmics for Polyhedral Calculus via Parametric Linear Programming. (Nouvelle Algorithmique pour le Calcul Polyédral via Programmation Linéaire Paramétrique)". PhD thesis. Grenoble Alpes University, France, 2017. TEL: tel-01695086.

[29] Alexandre Maréchal and Michaël Périn. "Efficient Elimination of Redundancies in Polyhedra by Raytracing". In: *Verification, Model Checking, and Abstract Interpretation (VMCAI)*. Ed. by Ahmed Bouajjani and David Monniaux. Vol. 10145. Lecture Notes in Computer Science. Springer, 2017, pp. 367–385. DOI: 10.1007/978-3-319-52234-0_20. HAL: hal-01385653.

[30] Peter McMullen and Geoffrey C. Shepard. *Convex polytopes and the upper bound conjecture*. Vol. 3. London Mathematical Society lecture note series. Cambridge University Press, 1971. ISBN: 0-521-08017-7.

[31] Theodore S. Motzkin. *Selected papers*. Ed. by David Cantor, Basil Gordon, and Bruce L. Rothschild. Birkhäuser, 1983. ISBN: 3-7643-3087-2.

[32] Theodore S. Motzkin, Howard Raiffa, Gerald L. Thompson, and Robert M. Thrall. "The double description method". In: *Contributions to the theory of games*. Ed. by Harold W. Kuhn and Albert W. Tucker. Vol. 2. Annals of Mathematics Studies 28. Princeton University Press, 1953, pp. 51–74. ISBN: 0691079358. Reprinted as [31, Ch. 2].

[33] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1998. ISBN: 0-471-98232-6.

[34] Gagandeep Singh, Markus Püschel, and Martin T. Vechev. "A practical construction for decomposing numerical abstract domains". In: *Proceedings of the ACM on Programming Languages (PACMPL)* 2.POPL (2018), 55:1–55:28. DOI: 10.1145/3158143.