Efficient Elimination of Redundancies in Polyhedra by Raytracing *

Alexandre Maréchal and Michaël Périn

Université Grenoble-Alpes, VERIMAG, F-38000 Grenoble, France
alex.marechal@imag.fr, michael.perin@imag.fr

Abstract. A polyhedron can be represented as constraints, generators or both in the double description framework. Whatever the representation, most polyhedral operators spend a significant amount of time to maintain minimal representations. To minimize a polyhedron in constraints-only representation, the redundancy of each constraint must be checked with respect to others by solving a linear programming (LP) problem. We present an algorithm that replaces most LP problem resolutions by distance computations. It consists in launching rays starting from a point within the polyhedron and orthogonal to its bounding hyperplanes. A face first encountered by one of these rays is an irredundant constraint of the polyhedron. Since this procedure is incomplete, LP problem resolutions are required for the remaining undetermined constraints. Experiments show that our algorithm drastically reduces the number of calls to the simplex, resulting in a considerable speed improvement. To follow the geometric interpretation, the algorithm is explained in terms of constraints but it can also be used to minimize generators.

1 Redundancy in Polyhedra

Convex polyhedra are used in static analysis [2] and automatic parallelization [6] to capture linear inequalities of the form $\sum_{i=1}^{n} a_i x_i \leq b$ relating the program variables $x_1, \ldots, x_n$.[1] A polyhedron $\mathscr{P}$ can be defined as the set of points $\boldsymbol{x} = (x_1, \ldots, x_n)$ that satisfy a system of inequalities $\boldsymbol{Ax} \leq \boldsymbol{b}$. The ℓ^{th} row of the augmented matrix $[\boldsymbol{A}|-\boldsymbol{b}]$ is a vector $\boldsymbol{C_\ell} = (a_{\ell 1}, \ldots, a_{\ell n}, -b_\ell)$ which encodes the constraint $\sum_{i=1}^{n} a_{\ell i}\, x_i \leq b_\ell$. A constraint $\boldsymbol{C_\ell}$ defines the bounding hyperplane normal to $(a_{\ell 1}, \ldots, a_{\ell n})$ and shifted by b_ℓ (see Fig. 1). Alternatively the same set of points can be defined as the convex combination of generators (vertices and rays), *i.e.* $\{\boldsymbol{x} \mid \boldsymbol{x} = \sum_{i=1}^{v} \beta_i \boldsymbol{v_i} + \sum_{i=1}^{r} \lambda_i \boldsymbol{R_i},\ \beta_i, \lambda_i \geq 0,\ \sum \beta_i = 1\}$ where $\boldsymbol{R_i}$'s and $\boldsymbol{v_i}$'s denote respectively rays and vertices. Fig. 1 shows two polyhedra $\mathscr{P}_a$ and $\mathscr{P}_b$ defined in double description as the set of constraints $\{\boldsymbol{C_1} : x_2 -$

* This work was partially supported by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement nr. 306595 "STATOR".

[1] We only deal with convex polyhedra. For readability, we will omit the adjective *convex* in the following.

$x_1 \leq 1,\ \boldsymbol{C_2} : x_2 - x_1 \geq -2,\ \boldsymbol{C_3} : x_1 \geq 1,\ \boldsymbol{C_4} : x_1 + x_2 \geq 2\}$ and the set of generators $\{\boldsymbol{v_1} : (1,2),\ \boldsymbol{v_2} : (1,1),\ \boldsymbol{v_3} : (2,0),\ \boldsymbol{R_1} : (1,1)\}$ for $\mathscr{P}_a$; respectively as $\{\boldsymbol{C_1}, \boldsymbol{C_2}, \boldsymbol{C'} : x_1 \geq 3\}$ and $\{\boldsymbol{v'_1} : (3,4),\ \boldsymbol{v'_2} : (3,1),\ \boldsymbol{R_1}\}$ for $\mathscr{P}_b$.

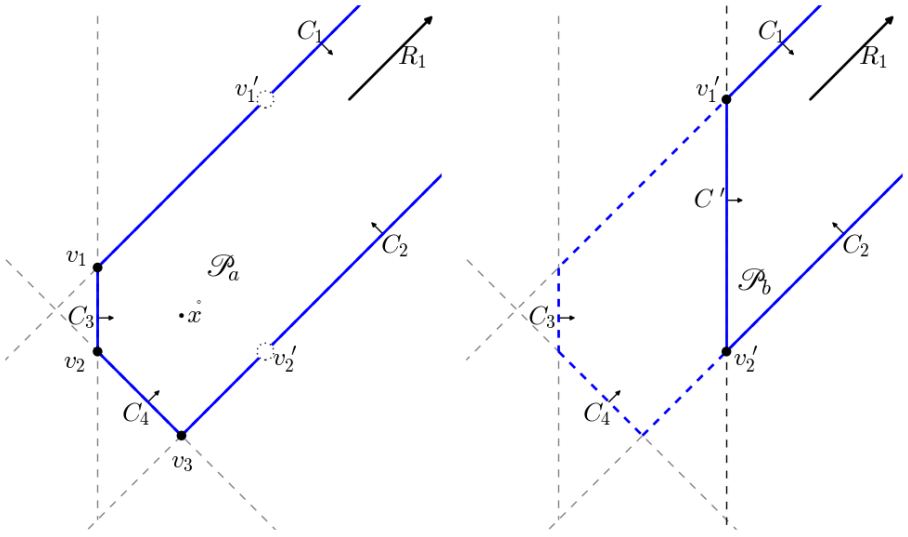


Fig. 1: Emergence of redundant constraints and generators in polyhedra.

The addition of new constraints or generators introduces redundancies which must be removed to reduce memory consumption and avoid useless computations in subsequent operations. In constraints-only representation, redundant constraints tend to grow exponentially during the computation of a projection by Fourier-Motzkin elimination [18]. For a description by generators the same pitfall occurs when a polyhedron is sliced with a constraint [8]. The emergence of redundancies is illustrated by Fig. 1: when constraint $\boldsymbol{C'}$ is added into $\mathscr{P}_a$ to form $\mathscr{P}_b$, constraints $\boldsymbol{C_3}$ and $\boldsymbol{C_4}$ become redundant. Conversely, the addition of points $\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}$ into $\mathscr{P}_b$ generates $\mathscr{P}_a$ and makes $\boldsymbol{v'_1}$ and $\boldsymbol{v'_2}$ redundant.

Characterization of Redundancy. A ray $\boldsymbol{R_k}$ is redundant if it is a nonnegative combination of the other rays and a point $\boldsymbol{v_k}$ is redundant if it is a convex combination of the other generators, *i.e.* $\boldsymbol{v_k} = \sum_i \beta_i \boldsymbol{v_i} + \sum_i \lambda_i \boldsymbol{R_i}$ for some $\beta_i, \lambda_i \geq 0$ with $\sum \beta_i = 1$ and $\beta_k = 0$. Back to our running example, the equations $\boldsymbol{v'_1} = 1 \times \boldsymbol{v_1} + 2 \times \boldsymbol{R_1}$ and $\boldsymbol{v'_2} = 1 \times \boldsymbol{v_3} + 1 \times \boldsymbol{R_1}$ prove the redundancy of $\boldsymbol{v'_1}$ and $\boldsymbol{v'_2}$ in $\mathscr{P}_a$. Therefore, these equations account for *certificates of redundancy.*

Intuitively, a constraint is redundant if it is useless, in the sense that adding it does not change the geometrical space delimited by the polyhedron. Formally, a constraint $\boldsymbol{C_k}$ is redundant if it is a nonnegative combination of other constraints. As we did for generators, we can find equations that prove the redundancy of $\boldsymbol{C_3}$ and $\boldsymbol{C_4}$ in $\mathscr{P}_b$, but we need to consider the tautological constraint $\boldsymbol{C_0} : 1 \geq 0$ as

being part of the system in order to exactly fit the constant b of the redundant constraint.

Example 1. The equations $\boldsymbol{C_3} = 1{\times}\boldsymbol{C'}{\oplus}2{\times}\boldsymbol{C_0}$ and $\boldsymbol{C_4} = 2{\times}\boldsymbol{C'}{\oplus}1{\times}\boldsymbol{C_2}{\oplus}2{\times}\boldsymbol{C_0}$ are called *the Farkas decomposition* of $\boldsymbol{C_3}$ and $\boldsymbol{C_4}$. They act as certificates of redundancy. Indeed, $\boldsymbol{C_3} : x_1 \geq 1 \equiv (x_1 \geq 3) \oplus 2 \times (1 \geq 0)$ and $\boldsymbol{C_4} : x_1 + x_2 \geq 2 \equiv 2{\times}(x_1 \geq 3){\oplus}(x_2 - x_1 \geq -2){\oplus}2{\times}(1 \geq 0)$ where $(l \geq r){\oplus}(l' \geq r') \stackrel{def}{=} l{+}l' \geq r{+}r'$.

If only one representation is available – as generators or as constraints – discovering redundancy requires solving linear programming (LP) problems of the form "does there exist nonnegative scalars satisfying some linear equations?":

$$\exists \lambda_0, \ldots, \lambda_p \geq 0,\ \boldsymbol{C_k} = \sum_{i=0, i\neq k}^{p} \lambda_i \boldsymbol{C_i} \qquad \textit{(1) for constraints}$$

$$\exists \lambda_1, \ldots, \lambda_r \geq 0,\ \boldsymbol{R_k} = \sum_{i=1, i\neq k}^{r} \lambda_i \boldsymbol{R_i} \qquad \textit{(2) for rays}$$

$$\exists \beta_1, \ldots, \beta_v, \lambda_1, \ldots, \lambda_r \geq 0,\ \boldsymbol{v_k} = \sum_{i=1, i\neq k}^{v} \beta_i \boldsymbol{v_i} + \sum_{i=1}^{r} \lambda_i \boldsymbol{R_i} \quad \textit{(3) for vertices}$$
$$\wedge \sum_{i=1, i\neq k}^{v} \beta_i = 1$$

Polyhedral Cones. The way to reconcile the two definitions of redundancy is to switch to *polyhedral cones* to get a homogeneous system of constraints and only rays as generators. The trick for changing a polyhedron $\mathscr{P}$ into a cone is to associate an extra variable η to the constant term $\boldsymbol{b}$ as follows [19]: $\boldsymbol{Ax} \leq \boldsymbol{b} \equiv \eta(\boldsymbol{Ax}) \leq \eta\boldsymbol{b} \equiv \boldsymbol{A}(\eta\boldsymbol{x}) - \eta\boldsymbol{b} \leq 0 \equiv [\boldsymbol{A}|-\boldsymbol{b}] \begin{pmatrix} \eta\boldsymbol{x} \\ \eta \end{pmatrix} \leq 0$ *for any* $\eta > 0$. It can be proved [9] that $\boldsymbol{x} \in \mathbb{Q}^n$ belongs to $\mathscr{P}$ if and only if $\begin{pmatrix} \boldsymbol{x} \\ 1 \end{pmatrix} \in \mathbb{Q}^{n+1}$ belongs to the cone $\left\{\boldsymbol{x'} \in \mathbb{Q}^{n+1} \mid \boldsymbol{A'x'} \leq 0\right\}$ where $\boldsymbol{A'} = [\boldsymbol{A}|-\boldsymbol{b}]$. Using this transformation, operators on polyhedra can be implemented as computations on their associated cones producing a cone that, once intersected with the hyperplane $\eta = 1$, is the expected polyhedron. We switch back to polyhedra in illustrations as they are easier to draw. Considering cones simplifies the presentation: the constant term of constraints and the vertices disappear from definitions. Then, we end up with the same definition of redundancy for constraints (1) and for generators (2): a vector is redundant if it is a nonnegative combination of the others vectors.

Deciding Redundancy. The redundant/irredundant status of a constraint or a ray depends on the satisfiability of an existential problem *(1,2)* involving linear equations but also inequalities ($\bigwedge_i \lambda_i \geq 0$). Thus, such a problem does not fall within the realm of linear algebra but in that of LP for which the simplex algorithm is a standard solver [5]. In practice, the simplex performs much better than its theoretical exponential complexity – but still remains a costly algorithm. So, much research has been devoted to identifying many cases where the simplex can be avoided. Wilde [19] and Lassez *et al.* [13] suggest several *fast redundancy-detection criteria* before switching to the general LP problem:

– **The quasi-syntactic redundancy test** considers pairs of constraints and looks for single constraint redundancies of the form $\boldsymbol{C}' = \lambda \boldsymbol{C}$ with $\lambda > 0$, e.g. $\boldsymbol{C}' : 4x_1 - 6x_2 \geq 2$ is redundant with respect to $\boldsymbol{C} : x_1 - 3x_2 \geq 1$ since $\boldsymbol{C}' = 2 \times \boldsymbol{C}$.
– **The bound shifting test** exploits the implication $\sum_{i=1}^{n} a_i\, x_i \leq b \implies \sum_{i=1}^{n} a_i\, x_i \leq b'$ if $b \leq b'$. Hence, when the coefficients of two constraints $\boldsymbol{C}$ and $\boldsymbol{C}'$ only differ on b and b' with $b \leq b'$ then $\boldsymbol{C}'$ is redundant and the certificate is $\boldsymbol{C}' = \boldsymbol{C} \oplus (b' - b) \times \boldsymbol{C_0}$ where C_0 is the tautology $0 \leq 1$.
– **The combination of single variable inequalities** such as $x_1 \leq b_1$ and $x_2 \leq b_2$ entails for instance the redundancy of $\boldsymbol{C} : 2x_1 + 3x_2 \leq b$ with $2b_1 + 3b_2 < b$. The corresponding certificate is $\boldsymbol{C} = 2 \times (x_1 \leq b_1) \oplus 3 \times (x_2 \leq b_2) \oplus (2b_1 + 3b_2 - b) \times (0 \leq 1)$.

While these criteria can detect certain redundancies at low-cost, in this paper we investigate the other side of redundancy: we provide a *fast criterion to detect irredundant constraints*. The combination of the two approaches limits the usage of the simplex to constraints that are neither decided by our criteria nor by those of Wilde and Lassez *et al.*

Contributions. We present an algorithm that replaces most LP problem resolutions by distance computations. It is detailed in §4, after introducing useful notations in §2. The geometric intuition of our irredundancy criterion is simple: consider ray traces starting from a point within the polyhedron and orthogonal to its bounding hyperplanes. The hyperplane first encountered by one of these rays is an actual face of the polyhedron. It is therefore an irredundant constraint. Since this procedure is incomplete, LP problem resolutions are required for the remaining undetermined constraints. Experiments of §5 show that our algorithm drastically reduces the number of calls to the simplex, resulting in a considerable speed improvement. In addition, our algorithm generates certificates of correctness, precision and minimality which make it usable in a certified static analyzer. Certificates are presented in §3. To follow the geometric interpretation, the algorithm is explained below in terms of constraints but it can similarly be used to minimize generators. We conclude in §6 by a discussion of the potential benefit of integrating our algorithm in the double description framework.

2 Notations

Vectors and matrices are written in boldface to be distinguished from scalars, e.g. $\mathbf{0}$ is a vector of 0. For clarity and without loss of generality the rest of the paper will focus on *polyhedral cones over rationals*. A polyhedral cone $\mathcal{P}$ of p constraints on n variables $(x_1, \ldots, x_n)$ is a conjunction (written as a set) of homogeneous linear constraints $\{\boldsymbol{C_1}, \ldots, \boldsymbol{C_p}\}$ of the form $\boldsymbol{C_\ell} : \sum_{i=1}^{n} a_{\ell i}\, x_i \leq 0$ where $\boldsymbol{C_\ell} = (a_{\ell 1}, \ldots, a_{\ell n})$. The inner product of vectors offers a convenient notation $\langle \boldsymbol{C_\ell}, \boldsymbol{x} \rangle \leq 0$ for that inequality. Then, the cone $\mathcal{P}$ corresponds to a matrix inequality $\boldsymbol{Ax} \leq \mathbf{0}$ where the rows of $\boldsymbol{A}$ are the vectors $\boldsymbol{C_1}, \ldots, \boldsymbol{C_p}$. Finally, $\{\boldsymbol{C_1}, \ldots, \boldsymbol{C_p}\}$, $\bigwedge_{\ell=1}^{p} \langle \boldsymbol{C_\ell}, \boldsymbol{x} \rangle \leq 0$ or $\boldsymbol{Ax} \leq \mathbf{0}$ are three equivalent ways of

denoting a polyhedral cone $\mathcal{P}$. We use $[\![\mathcal{P}]\!]$ to specifically refer to the set of points defined by $\mathcal{P}$. Given a cone $\mathcal{P} : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{0}$, the same system with a strict inequality defines $\mathring{\mathcal{P}}$, *the interior of* $\mathcal{P}$, and $\mathring{x}$ denotes a point of $[\![\mathring{\mathcal{P}}]\!] \stackrel{def}{=} \{\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x} < \boldsymbol{0}\}$.

3 Certifying a Minimization of Polyhedra

Our minimization algorithm is part of the Verimag Polyhedra Library (VPL) which operates on rational polyhedra in constraints-only representation. It was originally designed by Fouilhé *et al.* [7] as an abstract domain for the VERASCO certified static analyzer whose soundness is proved in COQ [12]. VERASCO can collaborate with an external library in OCAML such as the VPL, provided that it produces certificates of correctness, allowing a COQ-checker to verify the results computed in OCAML. In this section we recall the algorithm used in the original VPL for minimizing a *polyhedral cone represented as a set of constraints.* It is the standard algorithm but extended to produce on-the-fly *certificates of correctness, precision and minimality.* We recall the *fundamental theorem of linear inequalities* due to Farkas (1894) which ensures the existence of such certificates. Revisiting this theorem with a geometrical interpretation reveals an efficient way to determine irredundant constraints, which will be the key of our algorithm (§4).

Minimizing a cone $\mathcal{P}$ consists in removing all redundant constraints such that the result, $\mathcal{P}_M$, represents the same geometrical space, *i.e.* $[\![\mathcal{P}]\!] = [\![\mathcal{P}_M]\!]$. Two certificates are needed to prove that equality: (1) one for the inclusion $[\![\mathcal{P}]\!] \subseteq [\![\mathcal{P}_M]\!]$ which guarantees *the correctness of the minimization* and (2) another one for $[\![\mathcal{P}_M]\!] \subseteq [\![\mathcal{P}]\!]$ which justifies its *precision.* A third certificate (3) ensures *the minimality of the result* showing that all constraints of P_M are irredundant.

Certificate (1) must prove that each point of $[\![\mathcal{P}]\!]$ belongs to $[\![\mathcal{P}_M]\!]$. In the particular case of minimization, inclusion (1) is trivial because $\mathcal{P}_M$ is obtained by only removing constraints from $\mathcal{P}$, which necessarily leads to a larger set of points. By contrast, the existence of certificates (2) and (3) is not straightforward but the consequence of the following theorem, which we rephrased in our constraint terminology to ease its interpretation.

Theorem 1 (Fundamental theorem of linear inequalities [17, 7.1 p.85]). *Let* $\boldsymbol{C_1}, \ldots, \boldsymbol{C_p}$ *and* $\boldsymbol{C'}$ *be vectors in a n-dimensional space. Then,*

(I) either $\boldsymbol{C'}$ ***is redundant*** *and there exists a* Farkas decomposition *of* $\boldsymbol{C'}$ *that is a nonnegative linear combination of linearly independent vectors from* $\boldsymbol{C_1}, \ldots, \boldsymbol{C_p}$*, i.e.* $\boldsymbol{C'} = \lambda_1 \boldsymbol{C_1} + \ldots + \lambda_p \boldsymbol{C_p}$ *for some scalars* $\lambda_1, \ldots, \lambda_p \geq 0$.

(II) or $\boldsymbol{C'}$ ***is irredundant*** *and there exists a n-dimensional vector* $\boldsymbol{w}$ *such that* $\langle \boldsymbol{C'}, \boldsymbol{w} \rangle > 0$ *and* $\langle \boldsymbol{C_1}, \boldsymbol{w} \rangle, \ldots, \langle \boldsymbol{C_p}, \boldsymbol{w} \rangle \leq 0$.

The standard algorithm (Algorithm 1) exploits the redundancy criterion (I) of the theorem which was already illustrated in §1 Example 1. The existence of a *Farkas decomposition* of $\boldsymbol{C'}$ is decided by solving a LP problem. If the simplex algorithm returns a solution $\boldsymbol{\lambda}$ then the pair $(\boldsymbol{C'}, \boldsymbol{\lambda})$ is recorded as a *certificate of precision* (2) which proves that the removed constraint was indeed redundant. To

get rid of all the redundancies, Algorithm 1 needs one execution of the simplex algorithm for each constraint.

Given an existential LP problem, the simplex can return either a solution or an explanation of the lack of solution. The proof of Theorem 1 and the simplex algorithm have strong connections which result in an interesting feature of the VPL simplex: calling $\mathsf{simplex}(\exists \lambda_i \geq 0,\ \boldsymbol{C'} = \sum_i \lambda_i \boldsymbol{C_i})$ returns either SUCCESS($\boldsymbol{\lambda}$) or FAILURE($\boldsymbol{w}$) such that $\langle \boldsymbol{C'}, \boldsymbol{w} \rangle > 0 \bigwedge_i \langle \boldsymbol{C_i}, \boldsymbol{w} \rangle \leq 0$.[2] This feature is a consequence of Theorem 1 and requires no additional computation.

When the simplex returns FAILURE($\boldsymbol{w}$), the irredundancy criterion (II) of the theorem tells that $\boldsymbol{C'}$ is irredundant and must be kept in the set of constraints. Algorithm 1 builds the *certificate of minimality* (3) by associating a *witness point* to each constraint of the minimized polyhedron $\mathcal{P}_M$.

While the standard algorithm focuses on criterion (I), we revisit the theorem paying attention to the geometrical interpretation of criterion (II): when a constraint $\boldsymbol{C'}$ is irredundant, its associated bounding hyperplane is *a frontier* of the polyhedron separating the inside from the outside. Part (II) of the theorem ensures that we can exhibit a witness point $\boldsymbol{w}$, outside of $[\![\mathcal{P}]\!]$, satisfying all constraints of $\mathcal{P}$ except $\boldsymbol{C'}$. The rest of the paper is dedicated to an algorithm that efficiently discovers such witness points.

Algorithm 1: The standard minimization algorithm (used in VPL 1.0)

Input : A set of constraints $\{\boldsymbol{C_1}, \ldots, \boldsymbol{C_p}\}$.
Output: $\mathcal{P}_M$ = the irredundant constraints of $\{\boldsymbol{C_1}, \ldots, \boldsymbol{C_p}\}$
(R, I) = the redundancy and irredundancy certificates

$\mathcal{P}_M \leftarrow \{\boldsymbol{C_1}, \ldots, \boldsymbol{C_p}\}$
for $\boldsymbol{C'}$ *in* $\{\boldsymbol{C_1}, \ldots, \boldsymbol{C_p}\}$ **do**
  switch `simplex` $\left(\exists \lambda_i \geq 0,\ \boldsymbol{C'} = \sum_{\boldsymbol{C_i} \in \mathcal{P}_M \setminus \boldsymbol{C'}} \lambda_i \boldsymbol{C_i}\right)$ **do**
    case SUCCESS $(\boldsymbol{\lambda})$: $R \leftarrow R \cup \{(\boldsymbol{C'}, \boldsymbol{\lambda})\}$; $\mathcal{P}_M \leftarrow \mathcal{P}_M \setminus \boldsymbol{C'}$
    case FAILURE $(\boldsymbol{w})$: $I \leftarrow I \cup \{(\boldsymbol{C'}, \boldsymbol{w})\}$
return $(\mathcal{P}_M, R, I)$

4 An Efficient Minimization Algorithm

Building up on the geometric interpretation of Theorem 1, we present a new minimization algorithm for polyhedral cones that brings two major improvements: it reduces the number of calls to the simplex algorithm and limits the constraints they involve. The key idea of the algorithm is to trace rays starting from a point in the interior of the cone. The first hyperplane encountered by

[2] Conversely, $\mathsf{simplex}\big(\exists \boldsymbol{w}, \langle \boldsymbol{C'}, \boldsymbol{w} \rangle > 0 \bigwedge_i \langle \boldsymbol{C_i}, \boldsymbol{w} \rangle \leq 0\big)$ returns either SUCCESS($\boldsymbol{w}$) or FAILURE($\boldsymbol{\lambda}$) such that $\boldsymbol{C'} = \sum_i \lambda_i \boldsymbol{C_i}$.

a ray is a frontier of the polyhedron, thus an irredundant constraint. Unfortunately, with a limited number of rays, some frontiers can be missed depending on the cone and the position of the interior point. This *raytracing procedure* is thus incomplete and LP problem resolutions are still required for the remaining undetermined constraints.

While the simplex algorithm is used in the standard minimization to discover Farkas decompositions, we rather use it to get closer to a witness point, and only when all previous rays failed to prove the irredundancy of a constraint. Of course, if the constraint is redundant, the simplex algorithm returns no point at all but an explanation of its failure which is nothing else than a Farkas decomposition proving the redundancy.

4.1 The Frontier Detection Criterion

We now detail the process of finding witness points by raytracing. We consider a cone $\mathcal{P}$ with a nonempty interior. Then, there exists a point $\mathring{\boldsymbol{x}}$ in $\mathring{\mathcal{P}}$. The basic operation of our algorithm consists in sorting the constraints of $\mathcal{P}$ with respect to the order in which they are hit by a *ray*, *i.e.* a half-line starting at the interior point $\mathring{\boldsymbol{x}}$ and extending along a given direction $\boldsymbol{d}$.

Consider the constraint $\langle \boldsymbol{C}, \boldsymbol{x} \rangle \leq 0$. The hyperplane of the constraint is $\{\boldsymbol{x} \mid \langle \boldsymbol{C}, \boldsymbol{x} \rangle = 0\}$, *i.e.* the set of points orthogonal to vector $\boldsymbol{C}$. The ray starting at $\mathring{\boldsymbol{x}}$ and extending in direction $\boldsymbol{d}$ is the set of points $\{\boldsymbol{x}(t) \mid \boldsymbol{x}(t) = \mathring{\boldsymbol{x}} + t \times \boldsymbol{d},\ t \geq 0\}$. Let us assume that the ray hits the $\boldsymbol{C}$-hyperplane at point $\boldsymbol{x_c}$. Then, there exists $t_c \geq 0$ such that $\boldsymbol{x_c} = \mathring{\boldsymbol{x}} + t_c \times \boldsymbol{d}$ and so, $\boldsymbol{x_c} - \mathring{\boldsymbol{x}} = t_c \times \boldsymbol{d}$. Therefore, the distance $\|\mathring{\boldsymbol{x}} - \boldsymbol{x_c}\|$ is just a scaling by $|t_c|$ of the norm $\|\boldsymbol{d}\|$ which does not depend on $\boldsymbol{C}$. Hence, *by computing $|t_c|$ for each constraint we will be able to know in which order the constraints are hit by the ray.* Prior to computing t_c we check if the ray can hit the constraint, *i.e.* $\langle \boldsymbol{C}, \boldsymbol{d} \rangle \neq 0$. Then, we use the fact that $\boldsymbol{x_c} \in \{\boldsymbol{x} \mid \langle \boldsymbol{C}, \boldsymbol{x} \rangle = 0\}$ to get $t_c = -\frac{\langle \boldsymbol{C}, \mathring{\boldsymbol{x}} \rangle}{\langle \boldsymbol{C}, \boldsymbol{d} \rangle}$. Indeed,

$$0 = \langle \boldsymbol{C}, \boldsymbol{x_c} \rangle = \langle \boldsymbol{C}, \mathring{\boldsymbol{x}} + t_c \times \boldsymbol{d} \rangle = \langle \boldsymbol{C}, \mathring{\boldsymbol{x}} \rangle + t_c \times \langle \boldsymbol{C}, \boldsymbol{d} \rangle \,.$$

Hence, the basic operation of our raytracing algorithm consists in two evaluations of each constraint $\boldsymbol{C}$ of $\mathcal{P}$ at $\mathring{\boldsymbol{x}}$ and $\boldsymbol{d}$ in order to compute the scalar t_c. Let us explain how we exploit this information to discover actual frontiers of $\mathcal{P}$.

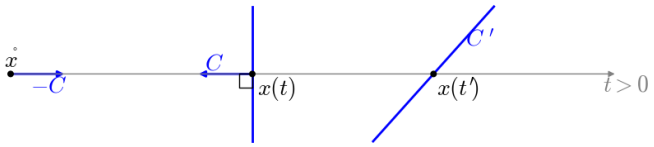


Fig. 2: The ray starting at the interior point $\mathring{\boldsymbol{x}}$ and orthogonal to a constraint $\boldsymbol{C}$ meets $\boldsymbol{C}$ and possibly others constraints.

Note that any direction could be used to sort the constraints with respect to the order of intersection by a ray. We choose successively for $\boldsymbol{d}$ the opposite

direction of the normal vector of each bounding hyperplane of $\mathcal{P}$. This heuristic ensures that each hyperplane will be hit by at least one ray. As illustrated by Fig. 2, a direction $\boldsymbol{d} \stackrel{def}{=} -\boldsymbol{C}$ necessarily intersects the $\boldsymbol{C}$-hyperplane and may potentially cross many other constraints for some values of t. Considering a

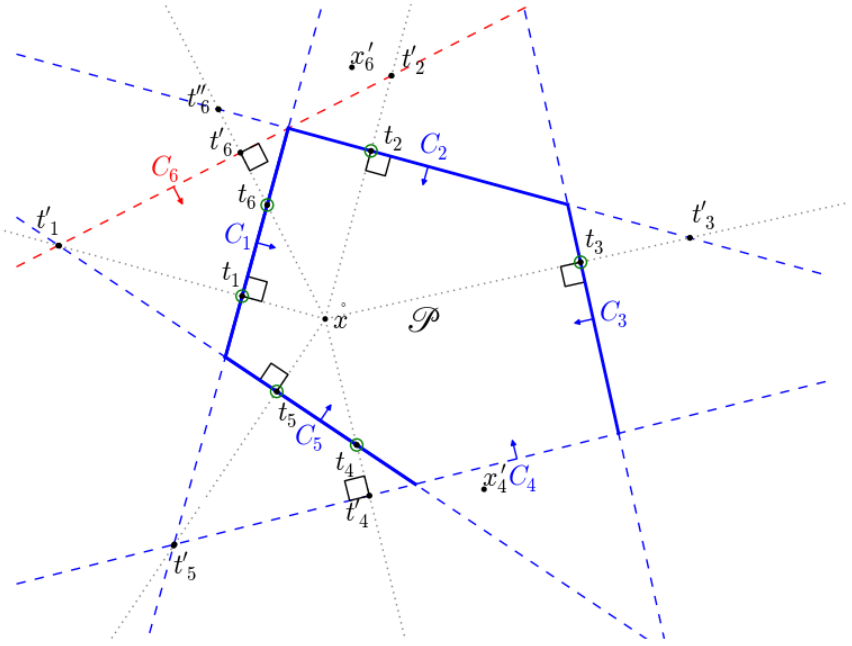


Fig. 3: Detection of some frontiers of a polyhedron by looking at their intersections with rays starting from an interior point $\mathring{\boldsymbol{x}}$ and orthogonal to a constraint. The thick lines are the discovered frontiers, confirmed by the doubly-circled intersection points.

direction $\boldsymbol{d_i} = -\boldsymbol{C_i}$, we sort the intersected hyperplanes with respect to the increasing order of the scalar t, which is proportional to the distance between the interior point $\mathring{\boldsymbol{x}}$ and the intersection point $\boldsymbol{x}(t)$ of an hyperplane and the ray $\text{RAY}(\mathring{\boldsymbol{x}}, \boldsymbol{d}_i)$. We obtain a sorted *intersection list* of pairs (t, S_t) where S_t is the set of the (possibly many) constraints vanishing at $\boldsymbol{x}(t)$. If a constraint $\boldsymbol{C}$ is not hit by the ray (because $\langle \boldsymbol{C}, \boldsymbol{d}_i \rangle = 0$), then $\boldsymbol{C}$ is not added to the intersection list. The head pair provides the constraints which are encountered first by the ray. At the heart of our algorithm is the following proposition: "If the head of an intersection list is a pair $(t, \{\boldsymbol{C}\})$ with *a single constraint*, then $\boldsymbol{C}$ is a frontier of $\mathcal{P}$; otherwise we cannot conclude from this list." This will be proved in §4.2 (Proposition 1) when we will come to the generation of witness points.

Example 2. Here are the sorted intersection lists obtained for the 6-constraints polyhedron of Fig. 3. The list I_i records the constraints met along $\text{RAY}(\mathring{\boldsymbol{x}}, -\boldsymbol{C_i})$ from $\mathring{\boldsymbol{x}}$ orthogonally to the hyperplane of $\boldsymbol{C_i}$. It satisfies $t_i < t'_i < t''_i < t'''_i$.

$$I_1 = [\ (t_1, \{\boldsymbol{C_1}\});\ (t'_1, \{\boldsymbol{C_5}, \boldsymbol{C_6}\});\ (t''_1, \{\boldsymbol{C_2}\})\ ]$$
$$I_2 = [\ (t_2, \{\boldsymbol{C_2}\});\ (t'_2, \{\boldsymbol{C_6}\});\ (t''_2, \{\boldsymbol{C_3}\});\ (t'''_2, \{\boldsymbol{C_1}\})\ ]$$
$$I_3 = [\ (t_3, \{\boldsymbol{C_3}\});\ (t'_3, \{\boldsymbol{C_2}\});\ (t''_3, \{\boldsymbol{C_4}\})\ ] \quad I_4 = [\ (t_4, \{\boldsymbol{C_5}\});\ (t'_4, \{\boldsymbol{C_4}\});\ (t''_4, \{\boldsymbol{C_3}\})\ ]$$
$$I_5 = [\ (t_5, \{\boldsymbol{C_5}\});\ (t'_5, \{\boldsymbol{C_1}, \boldsymbol{C_4}\})\ ] \quad I_6 = [\ (t_6, \{\boldsymbol{C_1}\});\ (t'_6, \{\boldsymbol{C_6}\});\ (t''_6, \{\boldsymbol{C_2}\})\ ]$$

These lists reveal that $\boldsymbol{C_1}$, $\boldsymbol{C_2}$, $\boldsymbol{C_3}$ and $\boldsymbol{C_5}$ are frontiers of $\mathcal{P}$; $\boldsymbol{C_1}$ and $\boldsymbol{C_5}$ are even confirmed twice. Our criterion fails to decide the status of $\boldsymbol{C_4}$ and $\boldsymbol{C_6}$ because, in any of the considered directions, they are never encountered first. This situation is legitimate for the redundant constraint $\boldsymbol{C_6}$ but also happens for $\boldsymbol{C_4}$ even if it is a frontier of $\mathcal{P}$.

At this point (line 10 of Algorithm 2), we run the simplex to determine the irredundancy of the remaining constraints. In order to keep LP problems as small as possible, we build them incrementally as follows. Consider an undetermined constraint $\boldsymbol{C_i}$ and let I_i be the intersection list resulting from the direction $\boldsymbol{d}_i = -\boldsymbol{C_i}$. We pose a LP problem to find a point $\boldsymbol{x'_i}$ satisfying $\langle \boldsymbol{C_i}, \boldsymbol{x'_i} \rangle > 0 \wedge \langle \boldsymbol{C'}, \boldsymbol{x'_i} \rangle \leq 0$, where $\boldsymbol{C'}$ is the single constraint that appears at the head of I_i. As said earlier, $\boldsymbol{C'}$ is a frontier because it is the first hyperplane encountered by the ray. We illustrate the algorithm on the case of a single head constraint as it is the most frequent one. If the head set contains several constraints we cannot know which one is a frontier, thus we add all of them in the LP problem (lines 13-14 of Algorithm 2). We distinguish two cases depending on the satisfiability of the existential LP problem: If the problem of line 15 is unsatisfiable, the simplex returns FAILURE(λ), $\boldsymbol{C_i}$ is redundant with respect to $\boldsymbol{C'}$ and the Farkas decomposition of $\boldsymbol{C_i}$ is $\lambda \times \boldsymbol{C'}$. Otherwise, the simplex exhibits a point $\boldsymbol{x'_i}$ which satisfies $\langle \boldsymbol{C_i}, \boldsymbol{x'_i} \rangle > 0 \wedge \langle \boldsymbol{C'}, \boldsymbol{x'_i} \rangle \leq 0$. Here, we cannot conclude on $\boldsymbol{C_i}$'s redundancy since $\boldsymbol{x'_i}$ is a witness showing that $\boldsymbol{C_i}$ is irredundant with respect to $\boldsymbol{C'}$ alone, but $\boldsymbol{C_i}$ could still be redundant with respect to the other constraints.

To check the irredundancy of $\boldsymbol{C_i}$, we launch a new ray RAY($\mathring{\boldsymbol{x}}, \boldsymbol{x'_i} - \mathring{\boldsymbol{x}}$) from $\mathring{\boldsymbol{x}}$ to $\boldsymbol{x'_i}$ in direction $\boldsymbol{d} = \boldsymbol{x'_i} - \mathring{\boldsymbol{x}}$. As before, we compute the intersection list of this ray with all the constraints but this time we know for sure that $\boldsymbol{C_i}$ will precede $\boldsymbol{C'}$ in the list.[3] Then, we analyze the head of the list: if $\boldsymbol{C_i}$ is the *single* first element, then it is a frontier. Otherwise the first element, say $\boldsymbol{C''}$, is added to the LP problem, which is now asked for a point $\boldsymbol{x''_i}$ such that $\langle \boldsymbol{C_i}, \boldsymbol{x''_i} \rangle > 0 \wedge \langle \boldsymbol{C'}, \boldsymbol{x''_i} \rangle \leq 0 \wedge \langle \boldsymbol{C''}, \boldsymbol{x''_i} \rangle \leq 0$ resulting in a new RAY($\mathring{\boldsymbol{x}}, \boldsymbol{x''_i} - \mathring{\boldsymbol{x}}$). The way we choose rays guarantees that the previous constraints $\boldsymbol{C'}, \boldsymbol{C''}, ...$ will always be hit after $\boldsymbol{C_i}$ by the next ray. Therefore, ultimately the constraint $\boldsymbol{C_i}$ will be hit first by a ray, or it will be proved redundant. Termination is guaranteed because the first constraint struck by the new ray is either $\boldsymbol{C_i}$ and we are done, or a not already considered constraint and there is a finite number of constraints in $\mathcal{P}$. Observe that this algorithm builds incremental LP problems which contain only frontiers that were between $\mathring{\boldsymbol{x}}$ and the hyperplane of $\boldsymbol{C_i}$ at some step.

Example 2 (continued). In the above example, we found out that $\boldsymbol{C_1}$, $\boldsymbol{C_2}$, $\boldsymbol{C_3}$ and $\boldsymbol{C_5}$ were frontiers. To determine the status of $\boldsymbol{C_4}$, we solve the LP problem

[3] As this property is a pure technicality it is not given here but is available in [15].

$\exists \boldsymbol{x'_4},\ \langle \boldsymbol{C_4}, \boldsymbol{x'_4} \rangle > 0 \wedge \langle \boldsymbol{C_5}, \boldsymbol{x'_4} \rangle \leq 0$ because $\boldsymbol{C_5}$ is the head of I_4. The simplex finds such a point $\boldsymbol{x'_4}$ and the next step is to compute the intersection list corresponding to $\text{RAY}(\mathring{\boldsymbol{x}}, \boldsymbol{x'_4} - \mathring{\boldsymbol{x}})$. This list will reveal $\boldsymbol{C_4}$ as an actual frontier.

Similarly, the intersection list I_6 of the example suggests to solve the LP problem $\exists \boldsymbol{x'_6},\ \langle \boldsymbol{C_6}, \boldsymbol{x'_6} \rangle > 0 \wedge \langle \boldsymbol{C_1}, \boldsymbol{x'_6} \rangle \leq 0$ to launch a new ray toward $\boldsymbol{C_6}$. This problem is satisfiable and the simplex returns $\text{SUCCESS}(\boldsymbol{x'_6})$. Then, we compute the intersection list corresponding to $\text{RAY}(\mathring{\boldsymbol{x}}, \boldsymbol{x'_6} - \mathring{\boldsymbol{x}})$ and this time the head of the list is $\boldsymbol{C_2}$. We thus add $\boldsymbol{C_2}$ to the previous LP problem and call the simplex on $\exists \boldsymbol{x''_6},\ \langle \boldsymbol{C_6}, \boldsymbol{x''_6} \rangle > 0 \wedge \langle \boldsymbol{C_1}, \boldsymbol{x''_6} \rangle \leq 0 \wedge \langle \boldsymbol{C_2}, \boldsymbol{x''_6} \rangle \leq 0$. This problem has no solution: the simplex returns $\text{FAILURE}(\boldsymbol{\lambda} = (1, 1))$ showing that $\boldsymbol{C_6}$ is redundant and its Farkas decomposition is $\boldsymbol{C_6} = 1 \times \boldsymbol{C_1} + 1 \times \boldsymbol{C_2}$.

Algorithm 2: Raytracing algorithm

Input : A set of constraints $\mathcal{P} = \{\boldsymbol{C_1}, \ldots, \boldsymbol{C_p}\}$; a point $\mathring{\boldsymbol{x}} \in \mathring{\mathcal{P}}$
Output : $\mathcal{P}_M$: minimized version of $\mathcal{P}$
Data : $LP[i]$: LP problem associated to $\boldsymbol{C_i}$; $I[i]$: intersection list of $\boldsymbol{C_i}$
Function: `intersectionList`($\boldsymbol{d}$, $\{\boldsymbol{C_1}, \ldots, \boldsymbol{C_q}\}$) returns the intersection list obtained by intersecting $\{\boldsymbol{C_1}, \ldots, \boldsymbol{C_q}\}$ with ray $\boldsymbol{d}$

```
1  Function updateFrontiers (I[i], P_M, P)
2      if head (I[i]) = (t_F, {F}) then
3          P_M ← P_M ∪ {F}
4          P ← P \ F
5      return (P_M, P)
6  P_M ← ∅ ; LP ← arrayOfSize(p) ; I ← arrayOfSize(p)
7  for C_i in P do      /* First step of raytracing with orthogonal rays */
8      I[i] ← intersectionList (RAY(x̊, −C_i), P)
9      (P_M, P) ← updateFrontiers (I[i], P_M, P)
10 while P ≠ ∅ do
11     for C_i in P do
12         (t, S) ← head(I[i])
13         for C in S do
14             LP[i] ← LP[i] ∧ ⟨C, x'_i⟩ ≤ 0
15         switch simplex (∃x'_i, ⟨C_i, x'_i⟩ > 0 ∧ LP[i]) do
16             case SUCCESS (x'_i):
17                 I[i] ← intersectionList (RAY(x̊, x'_i − x̊), P ∪ P_M)
18                 (P_M, P) ← updateFrontiers (I[i], P_M, P)
19             case FAILURE (λ): P ← P \ C_i          /* C_i is redundant */
20
21 return P_M
```

4.2 Irredundancy Certificates

Let us explain how we compute witness points from the intersection lists defined in the previous section. From now on, we will denote a constraint by $\boldsymbol{F}$ if it is a frontier and by $\boldsymbol{C}$ when we do not know if it is a redundant constraint or an actual frontier. Let us come back to the list of the intersections of constraints of $\mathcal{P}$ with a ray $\{\boldsymbol{x}(t) \mid \boldsymbol{x}(t) = \mathring{\boldsymbol{x}} + t \times \boldsymbol{d},\ t \geq 0\}$ for a direction $\boldsymbol{d}$.

Proposition 1. *If the head of an intersection list contains a single constraint $\boldsymbol{F}$, then we can build a* witness point *satisfying the irredundancy criterion of Theorem 1 which proves that $\boldsymbol{F}$ is a frontier:*

(*a*) *For a list* $[(t_{\boldsymbol{F}}, \{\boldsymbol{F}\})]$, *we take the witness* $\boldsymbol{w}_a = \mathring{\boldsymbol{x}} + (t_{\boldsymbol{F}} + 1) \times \boldsymbol{d}$
(*b*) *For a list* $[(t_{\boldsymbol{F}}, \{\boldsymbol{F}\})\ ;\ (t', S')\ ;\ \ldots]$ *with at least two pairs, we define the witness* $\boldsymbol{w}_b = \mathring{\boldsymbol{x}} + \frac{t_{\boldsymbol{F}}+t'}{2} \times \boldsymbol{d}$.

Proof. Let us prove that these witness points attest that $\boldsymbol{F}$ is an irredundant constraint. According to Theorem 1, it amounts to proving that $\bigwedge_{\boldsymbol{C} \in \mathcal{P} \setminus \boldsymbol{F}} \langle \boldsymbol{C}, \boldsymbol{w} \rangle \leq 0 \wedge \langle \boldsymbol{F}, \boldsymbol{w} \rangle > 0$ for $\boldsymbol{w}_a$ (resp. $\boldsymbol{w}_b$).

Let us first study the sign of $\langle \boldsymbol{F}, \boldsymbol{x}(t) \rangle$ at point $\boldsymbol{x}(t) = \mathring{\boldsymbol{x}} + t \times \boldsymbol{d}$. Note that $\langle \boldsymbol{F}, \boldsymbol{x}(t) \rangle = \langle \boldsymbol{F}, \mathring{\boldsymbol{x}} + t \times \boldsymbol{d} \rangle \overset{(\dagger)}{=} \langle \boldsymbol{F}, \mathring{\boldsymbol{x}} \rangle + t \times \langle \boldsymbol{F}, \boldsymbol{d} \rangle$. By construction, $\langle \boldsymbol{F}, \boldsymbol{x}(t_{\boldsymbol{F}}) \rangle = 0$ then, by equation $(\dagger)$, $-\langle \boldsymbol{F}, \mathring{\boldsymbol{x}} \rangle = t_{\boldsymbol{F}} \times \langle \boldsymbol{F}, \boldsymbol{d} \rangle$. Recall that $t_{\boldsymbol{F}} \geq 0$, $\langle \boldsymbol{F}, \boldsymbol{d} \rangle \neq 0$ since the ray hits $\boldsymbol{F}$ and $\langle \boldsymbol{F}, \mathring{\boldsymbol{x}} \rangle < 0$ because $\mathring{\boldsymbol{x}} \in \mathring{\mathcal{P}}$. Thus, $\langle \boldsymbol{F}, \boldsymbol{d} \rangle$ and $t_{\boldsymbol{F}}$ are necessarily positive. Consequently, for a frontier $\boldsymbol{F}$ found in a direction $\boldsymbol{d}$, $\langle \boldsymbol{F}, \boldsymbol{x}(t) \rangle = \langle \boldsymbol{F}, \mathring{\boldsymbol{x}} \rangle + t \times \langle \boldsymbol{F}, \boldsymbol{d} \rangle$ is positive for any $t > t_{\boldsymbol{F}}$. Hence, in case (a) $\langle \boldsymbol{F}, \boldsymbol{w}_a \rangle \overset{def}{=} \langle \boldsymbol{F}, \boldsymbol{x}(t_{\boldsymbol{F}} + 1) \rangle > 0$ and in case (b) $\langle \boldsymbol{F}, \boldsymbol{w}_b \rangle \overset{def}{=} \left\langle \boldsymbol{F}, \boldsymbol{x}(\frac{t_{\boldsymbol{F}}+t'}{2}) \right\rangle > 0$ since $t_{\boldsymbol{F}} < \frac{t_{\boldsymbol{F}}+t'}{2} < t'$.

Let us now study the sign of $\langle \boldsymbol{C}, \boldsymbol{x}(t) \rangle$ for constraints other than $\boldsymbol{F}$:

(*a*) Consider the list $[(t_{\boldsymbol{F}}, \{\boldsymbol{F}\})]$. By construction, it means that no other constraint $\boldsymbol{C}$ of $\mathcal{P}$ is struck by the RAY$(\mathring{\boldsymbol{x}}, \boldsymbol{d})$, *i.e.* whatever the value $t \geq 0$, the sign of $\langle \boldsymbol{C}, \boldsymbol{x}(t) \rangle = \langle \boldsymbol{C}, \mathring{\boldsymbol{x}} \rangle + t \times \langle \boldsymbol{C}, \boldsymbol{d} \rangle$ does not change. As $\langle \boldsymbol{C}, \boldsymbol{x}(t=0) \rangle = \langle \boldsymbol{C}, \mathring{\boldsymbol{x}} \rangle < 0$ because $\mathring{\boldsymbol{x}} \in \mathring{\mathcal{P}}$, we can conclude that $\forall t \geq 0,\ \langle \boldsymbol{C}, \boldsymbol{x}(t) \rangle < 0$. Thus, in particular, $\langle \boldsymbol{C}, \boldsymbol{w}_a \rangle \overset{def}{=} \langle \boldsymbol{C}, \boldsymbol{x}(t_{\boldsymbol{F}} + 1) \rangle < 0$ for any $\boldsymbol{C} \in \mathcal{P} \setminus \boldsymbol{F}$.
(*b*) Consider now the list $[(t_{\boldsymbol{F}}, \{\boldsymbol{F}\});\ (t', S'); \ldots]$. A constraint $\boldsymbol{C}$ that appears in the set S' vanishes at point $\boldsymbol{x}(t')$ with $t' > t_{\boldsymbol{F}} \geq 0$. The previous reasoning $(\dagger)$ (on $\boldsymbol{F}$) based on equation $\langle \boldsymbol{C}, \boldsymbol{x}(t) \rangle = \langle \boldsymbol{C}, \mathring{\boldsymbol{x}} \rangle + t \times \langle \boldsymbol{C}, \boldsymbol{d} \rangle$ is valid for $\boldsymbol{C}$, hence proving $\langle \boldsymbol{C}, \boldsymbol{d} \rangle > 0$. Thus, $\langle \boldsymbol{C}, \boldsymbol{x}(t) \rangle$ is negative for $t < t'$ (zero for $t = t'$ and positive for $t' < t$). Finally, $\langle \boldsymbol{C}, \boldsymbol{w}_b \rangle \overset{def}{=} \left\langle \boldsymbol{C}, \boldsymbol{x}(\frac{t_{\boldsymbol{F}}+t'}{2}) \right\rangle < 0$ since $\frac{t_{\boldsymbol{F}}+t'}{2} < t'$. The same reasoning applies to any other pair (t, S_t) in the tail of the list.

Fig. 4($\mathscr{P}_b$) shows the irredundancy witness points w_1, w_2, w_1', w_2' of constraints $\boldsymbol{C_1}, \boldsymbol{C_2}$ and $\boldsymbol{C}'$. The irredundancy of $\boldsymbol{C}'$ is confirmed three times by different rays respectively orthogonal to $\boldsymbol{C}'$, $\boldsymbol{C_3}$ and $\boldsymbol{C_4}$, leading to witnesses w_1' (twice) and w_2'.

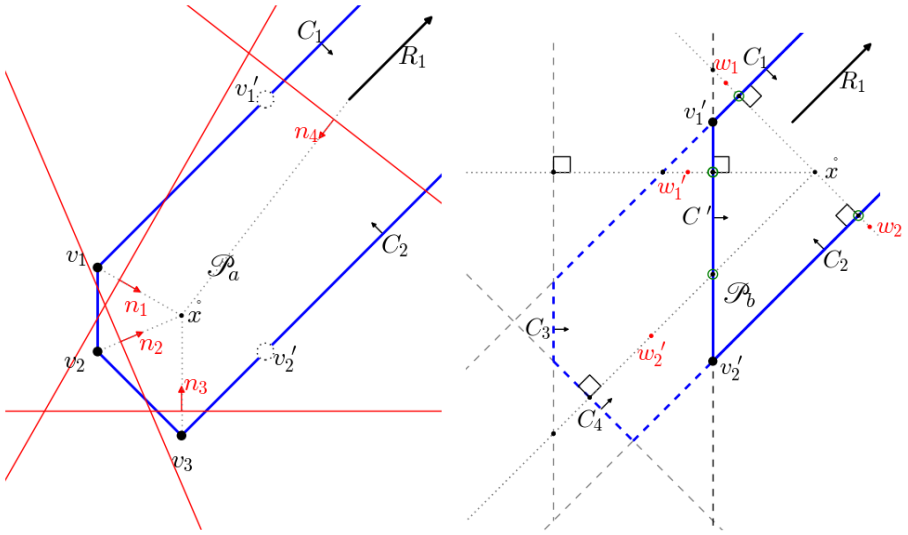


Fig. 4: Irredundancy witnesses for generators of $\mathscr{P}_a$ and constraints of $\mathscr{P}_b$.

4.3 Minimizing Generators

So far, to ease the understanding, we presented the raytracing for constraints-only polyhedra, but it works as well for generators. Indeed, we manipulated constraints as vectors and all our explanations and proofs are based on inner product. Moreover, Theorem 1 is not limited to constraints, it holds for any vector space and can be rephrased for generators. This time the irredundancy certificate for a generator $\boldsymbol{g'}$ is a vector $\boldsymbol{n}$ such that $\langle \boldsymbol{g_1}, \boldsymbol{n} \rangle, \ldots, \langle \boldsymbol{g_p}, \boldsymbol{n} \rangle \leq 0$ and $\langle \boldsymbol{g'}, \boldsymbol{n} \rangle > 0$. Such a vector defines a hyperplane orthogonal to $\boldsymbol{n}$, *i.e.* $\{\boldsymbol{x} \mid \langle \boldsymbol{n}, \boldsymbol{x} \rangle = 0\}$. It is called a *separating hyperplane* because it isolates generator $\boldsymbol{g'}$ from the other ones. Fig. 4($\mathscr{P}_a$) shows the separating hyperplanes defined by $\boldsymbol{n_1}, \boldsymbol{n_2}, \boldsymbol{n_3}$ and $\boldsymbol{n_4}$. They respectively justify the irredundancy of $\boldsymbol{v_1}, \boldsymbol{v_2}, \boldsymbol{v_3}$ and $\boldsymbol{R_1}$ in $\mathscr{P}_a$.

4.4 Using Floating Points in Raytracing

It is possible to make raytracing even more efficient by using floating points instead of rationals. Thereby, we experimented floating points in both LP problem resolutions and distance computations. The rational coefficients of constraints are translated into floating points. It introduces a loss in precision which does not jeopardize the result because the certificate checking controls the minimization process. However, we must pay attention to the generation of *exact* (*i.e.* rational) certificates from floating point computations. The solution we propose differs depending on the kind of certificate.

Witness Points. Checking a certificate of irredundancy consists in evaluating the sign of $\langle \boldsymbol{C_i}, \boldsymbol{w} \rangle$ for all constraints $\boldsymbol{C_i}$ of $\mathcal{P}$ with the provided witness point

$\boldsymbol{w}$. A witness point $\boldsymbol{w}$ must then be given with rational coefficients to avoid sign errors if $\langle \boldsymbol{C_i}, \boldsymbol{w} \rangle$ is too close to 0. Thus, the witness point $\boldsymbol{w}^{\mathbb{F}}$ obtained with floating point computations is translated into a rational one $\boldsymbol{w}^{\mathbb{Q}}$, without loss of precision (each floating point $0.d_1 \ldots d_m 10^e$ is changed into a rational $\frac{d_1 \ldots d_m . 10^e}{10^m}$). Then we check the irredundancy certificate with $\boldsymbol{w}^{\mathbb{Q}}$ and the rational version of the constraints. If the verification passes, then $\boldsymbol{w}^{\mathbb{Q}}$ is indeed a witness point. In the rare case of failure, using the exact simplex of the VPL on the LP problem will fix the approximation error by providing a rational witness point.

Farkas Decompositions. To prove a redundancy we need to exhibit the Farkas decomposition of the redundant constraint. To obtain an exact decomposition from the floating LP solution, we record which constraint is actually part of the decomposition.[4] Then, we run the exact simplex on a LP problem involving only those constraints to retrieve the exact Farkas decomposition.

5 Experiments

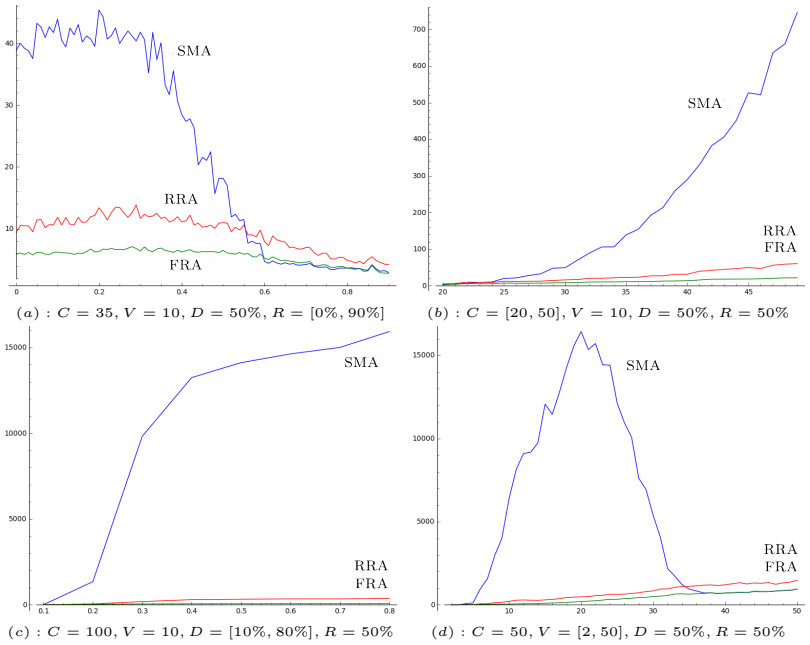


Fig. 5: Execution time in milliseconds of SMA *(blue),* RRA *(red) and* FRA *(green) depending on respectively* (a) *redundancy,* (b) *number of constraints,* (c) *density and* (d) *number of variables.*

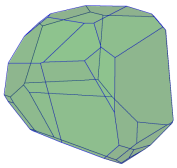
[4] What is needed from the floating point solution is the set of basic variables and an ordering of the nonnull λ_i coefficients to speed up the search in exact simplex.

This section is devoted to the comparison of three minimization algorithms:

- **The Standard Minimization Algorithm (SMA).** The standard Algorithm 1 of §3 is available in the VPL since version 1.0. It works on rationals and can generate certificates of precision, minimality and correctness.
- **The Rational Raytracing Algorithm (RRA).** RRA and SMA use the same LP solver, thus comparing their running time is relevant to estimate the efficiency of raytracing with respect to the standard algorithm.
- **The Floating point Raytracing Algorithm (FRA).** FRA implements raytracing with floating points as explained in §4.4. LP problems are solved by the GNU LP Kit which provides a simplex algorithm on floating points.

These three algorithms are all implemented in the current version (2.0) of the VPL.[5] For computing the exact Farkas decomposition that proves a constraint's irredundancy, the three algorithms ultimately rely on the VPL simplex in rational. They use the same datastructures (e.g. for constraints), allowing more reliable timing comparisons between them. Moreover, they share the same preprocessing step of finding a point within the polyhedron interior. This point is obtained by solving a LP problem checking the emptiness of the polyhedron with strict inequalities. The time measurements given below include this step but not the reconstruction of exact certificates from floating point ones.

Benchmarks. Throughout the paper, we focused on cones to simplify both notations and explanations. However, our algorithm works for general convex polyhedra and we build our experiments as follows. To compare the three algorithms, we asked them to minimize polyhedra that were generated randomly from four parameters that will be detailed further: the number of variables ($V \in [2, 50]$), the number of constraints ($C \in [2, 50]$), the redundancy rate ($R \in [0\%, 90\%]$) and the density rate ($D \in [10\%, 80\%]$). Each constraint is created by giving a random integer between -100 and 100 to the coefficient of each variable, within the density rate. All constraints are attached the same constant bound ≤ 20. Such polyhedra have a convex potatoid shape, shown on the right hand side. We do not directly control the number of generators but we count them using the APRON interface [11] to polyhedral libraries in double description. Among all our measurements, the number of generators ranged from 10 to 6400 and this number grows polynomially in the number of constraints. This covers a wide variety of polyhedra and our experiments show that raytracing is always more efficient.



Redundancy Rate. The effect of redundancy on execution time is displayed on Fig. 5(a). These measures come from the minimization of polyhedra with 10 variables and 35 constraints, and a redundancy rate ranging from 0% to 90% of the number of constraints. To generate a redundant constraint, we randomly

[5] `https://github.com/VERIMAG-Polyhedra`

pick two constraints and produce a nonnegative combination of them. We took care of avoiding redundancies that can be discarded by the fast detection criteria of §1. The graph clearly shows that raytracing has a big advantage on polyhedra with few redundancies. This phenomenon was expected: raytracing is good at detecting irredundancy at low-cost. SMA becomes similar to raytracing when the redundancy rate is high. This is explained by the implementation details given in previous paragraphs: when a redundant constraint is found, it is removed from the LP problem. Thus, if the redundancy rate reaches a very high level, the LP problem becomes smaller and smaller at each iteration, lowering the impact of using floating points. Moreover, the heuristic used by our algorithm never hits if almost all constraints are redundant, which makes the raytracing computations useless. To be fair between raytracing and the standard algorithm, we set the redundancy rate at 50% in the other experiments.

Number of Constraints. Fig. 5(b) measures the minimization time depending on the number of constraints for polyhedra with 10 variables. FRA and RRA scale better with respect to the number of constraints than SMA: experiments show that when C ranges from 20 to 50 constraints, SMA has a quadratic evolution compared to raytracing algorithms.

Density Rate. The density of a polyhedron is the (average) rate of nonnull coefficients within a constraint. For instance, a density of 60% with 10 variables means that on average, constraints have 6 nonnull coefficients. Fig. 5(c) shows the execution time for 10-dimensional polyhedra with 100 constraints, where the density rate D goes from 10% to 80%. The raytracing algorithms are almost insensitive to density, whereas the execution time of the standard algorithm blows up with density. Actually, having a lot of nonnull coefficients in constraints tends to create huge numerators and denominators because a pivot in the simplex performs many combinations of constraints. The blow up does not happen in RRA because LP problems are much smaller in the raytracing algorithms.

Number of Variables. The effect of the dimension on execution time is shown on Fig. 5(d). Whereas raytracing seems linearly impacted by the dimension, SMA has a behaviour that may look a bit strange. After a dramatic increase of execution time, the curve falls down when the dimension reaches about half the number of constraints. It finally joins and sticks to FRA curve. This phenomenon may be explained by the number of pivots needed to solve the LP problem. The closer the dimension is to the number of constraints, the fewer pivots are needed, thus making SMA competitive even with more LP problems to solve.

Table 1 shows results for several values of dimension and number of constraints. Again, each cell of this table gives the average values resulting from the minimization of 50 convex potatoids, with a density and a redundancy both fixed at 50%. For each pair (number of variables $\times$ number of constraints), Table 1 gives the number of LP problems that were solved and their size (*i.e.* the number of constraints they involve) on average. It contains also the computation time of the minimization in milliseconds and the speed up of raytracing compared to SMA. Results of Table 1 show that for small polyhedra, either in dimension

Table 1: Time measures of the three minimization algorithms SMA, RRA *and* FRA *for different values of variables and constraints.*

Var		5 constraints			10 constraints			25 constraints			50 constraints			100 constraints		
		SMA	RRA	FRA	SMA	RRA	FRA	SMA	RRA	FRA	SMA	RRA	FRA	SMA	RRA	FRA
2	# lp	2	0	0	3	1	1	6	2	2	9	5	5	15	11	11
	lp size	3	3	3	4	3	4	6	3	4	8	3	4	12	3	5
	time(ms)	0.05	0.03	0.02	0.04	0.05	0.09	0.10	0.14	0.14	0.27	0.30	0.34	0.83	**0.71**	0.80
	speed up	-	1.8	1.9	-	0.77	0.48	-	0.74	0.73	-	0.90	0.79	-	**1.2**	1.0
5	# lp	5	2	2	10	6	6	24	16	16	46	34	34	90	72	73
	lp size	5	3	4	9	3	5	20	4	7	36	5	8	65	5	9
	time(ms)	0.09	0.15	0.18	0.29	0.45	0.50	3.8	**2.0**	**1.8**	29.8	**7.1**	**5.4**	178	**26.0**	**18.8**
	speed up	-	0.61	0.51	-	0.65	0.59	-	**1.9**	**2.1**	-	**4.2**	**5.5**	-	**6.8**	**9.5**
10	# lp	5	2	2	10	5	5	25	13	13	50	28	28	100	58	58
	lp size	5	3	4	9	4	5	22	7	7	44	10	8	87	13	11
	time(ms)	0.18	0.30	0.32	0.60	1.1	0.86	17.6	**10.8**	**6.0**	811	**65.9**	**23.5**	14936	**336**	**64.0**
	speed up	-	0.59	0.55	-	0.57	0.70	-	**1.6**	**2.9**	-	**12.3**	**34.5**	-	**44.5**	**233**

or in number constraints, raytracing does not help. Indeed, for such small LP problems, the overhead of our algorithm is unnecessary and leads to time losses. Raytracing becomes interesting for larger polyhedra, where the speed improvement is significant. For instance, FRA is 44.5 times faster with 10 variables and 100 constraints than SMA. The gain can be explained by the number of LP problems solved and their average size, noticeably smaller in raytracing than in SMA. As expected, raytracing is faster with floating points.

We also compare through APRON with libraries in double description, the results are available in [15]. Not surprisingly the minimization time is significantly larger than that of SMA, FRA and RRA, as they must first compute all generators using Chernikova's algorithm.

6 Conclusion & Future Work

In this paper, we present a new algorithm to minimize the representation of a polyhedron, available in the VPL. It is based on raytracing and provides an efficient irredundancy check in which LP executions are replaced by distance computations. The *raytracing procedure* is incomplete and LP problem resolutions are still required for deciding the redundancy of the remaining constraints. However, our algorithm reduces not only the number of LP problems solved along the minimization, but also their size by an incremental approach. Moreover, it is usable for polyhedra in single representation, as constraints or as generators. It can be used either with rational or floating coefficients. In both cases, it can produce certificates of correctness, precision and minimality.

Parallelizing. Our raytracing algorithm is well-suited to parallelization: computing the intersection lists could be done by as many threads as rays. These computations boil down to matrix multiplications for which there exist efficient libraries, e.g. the LAPACK library [1]. Actually, to fully benefit from parallelism, the algorithm should be implemented in C because OCAML does not support native concurrency yet. Exploiting multi-cores, the number of ray traces could

be greatly increased, and applying the raytracing principle from several interior points would allow us to discover frontiers even more easily.

*Redundancy in the Double Description Framework (*DDF*).* Our algorithm has been designed to minimize polyhedra in single representation, but the principle of raytracing can be reused in the double description framework, where it could *quickly detect irredundant constraints*. Redundancy is easier to detect when the two representations of a polyhedron are available. Let the pair $(\mathscr{C}, \mathscr{G})$ denote the set of constraints and the set of generators of a polyhedron in $\mathbb{Q}^n$ and $(\mathscr{C}_M, \mathscr{G}_M)$ be its minimal version. A constraint $\boldsymbol{C} \in \mathscr{C}$ is irredundant if it is saturated by at least n irredundant generators, *i.e.* $\exists \boldsymbol{g_1}, \ldots, \boldsymbol{g_n} \in \mathscr{G}_M, \langle \boldsymbol{C}, \boldsymbol{g_i} \rangle = 0$. Similarly, a generator $\boldsymbol{g} \in \mathscr{G}$ is irredundant if it is the intersection of at least n irredundant constraints *i.e.* $\exists \boldsymbol{C_1}, \ldots, \boldsymbol{C_n} \in \mathscr{C}_M, \langle \boldsymbol{C_i}, \boldsymbol{g} \rangle = 0$. Think for instance of a line in 2D being defined by two points and a point being the intersection of at least two lines. The principle of the minimization algorithm is the following [10]: build the *boolean saturation matrix* $\boldsymbol{S}$ of size $|\mathscr{C}| \times |\mathscr{G}|$ defined by $\boldsymbol{S}[\boldsymbol{C}][\boldsymbol{g}] := (\langle \boldsymbol{C}, \boldsymbol{g} \rangle = 0)$, then iteratively remove constraints (and the corresponding rows of $\boldsymbol{S}$) which are insufficiently saturated and do the same for generators (and columns of $\boldsymbol{S}$) until reaching a stable matrix. The remaining constraints and generators form the minimal version $(\mathscr{C}_M, \mathscr{G}_M)$ which mutually justify the irredundancy of each other. This algorithm is appealing compared to its counterpart in single representation but the number of evaluation of $\langle \boldsymbol{C}, \boldsymbol{g} \rangle$ is huge when each variable x_i ranges in an interval $[l_i, u_i]$. Such a product of intervals can be represented by $2n$ constraints (two inequalities $l_i \leq x_i \wedge x_i \leq u_i$ per variable) which corresponds to 2^n vertices [3].[6] Therefore, the size of $\boldsymbol{S}$ is $n2^{n+1}$. To limit the computations, the saturation matrix is not fully constructed. Let us summarize the improved algorithm [19]: (1) Some constraints are removed by the *fast redundancy detection* recalled in §1. (2) The irredundant generators of $\mathscr{G}_M$ are constructed from the remaining constraints using Chernikova's algorithm [4] with some optimized adjacency criteria [14,8,20]. The adjacency criterion ensures that the construction cannot produce redundant generators [16]. (3) Finally, the saturation matrix is built to remove the constraint redundancies but a row is only completed if the constraint never finds enough saturating generators, otherwise the computation of the row is interrupted.

We believe that our orthogonal raytracing phase can be used at Step (3) to *quickly discover irredundant constraints*, which therefore do not have to be confirmed by the saturation matrix. The cost of this initial raytracing is reasonable: $\mathscr{C}$ rays and $2 \times |\mathscr{C}|$ evaluations per ray resulting in $2 \times |\mathscr{C}|^2$ computations of inner products. It could therefore benefit to minimization in the DDF especially when $|\mathscr{C}| \ll |\mathscr{G}|$ as in hypercubes.

[6] The opposite phenomena ($2n$ vertices corresponding to 2^n constraints) also exists but hardly ever occurs in practice [3].

References

1. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
2. R. Bagnara, P. M. Hill, and E. Zaffanella. Applications of polyhedral computations to the analysis and verification of hardware and software systems. *Theoretical Computer Science*, 410(46):4672–4691, 2009.
3. F. Benoy, A. King, and F. Mesnard. Computing convex hulls with a linear solver. *TPLP:Theory and Practice of Logic Programming*, 5(1-2):259–271, 2005.
4. N. V. Chernikova. Algorithm for discovering the set of all the solutions of a linear programming problem. *USSR Computational Mathematics and Mathematical Physics*, 1968.
5. V. Chvatal. *Linear Programming*. Series of books in the Mathematical Sciences. W. H. Freeman, 1983.
6. P. Feautrier and C. Lengauer. Polyhedron model. In *Encyclopedia of Parallel Computing*, volume 1, pages 1581–1592. Springer, 2011.
7. A. Fouilhé, D. Monniaux, and M. Périn. Efficient certificate generation for the abstract domain of polyhedra. In *Static Analysis Symposium*, 2013.
8. K. Fukuda and A. Prodon. Double description method revisited. In *Combinatorics and Computer Science*, volume 1120 of *LNCS*, page 91–111. Springer, 1996.
9. A. J. Goldman and A. W. Tucker. Polyhedral convex cones. In *Linear inequalities and related systems*, volume 38 of *Annals of Mathematics Studies*, pages 19–40. Princeton University Press, 1956.
10. N. Halbwachs. *Détermination automatique de relations linéaires vérifiées par les variables d'un programme*. PhD thesis, Université scientifique et médicale de Grenoble, 1979. (in french).
11. B. Jeannet and A. Miné. APRON: A library of numerical abstract domains for static analysis. In *Computer Aided Verification (CAV)*, volume 5643 of *LNCS*, pages 661–667, 2009.
12. J.-H. Jourdan, V. Laporte, S. Blazy, X. Leroy, and D. Pichardie. A formally-verified C static analyzer. In *ACM Principles of Programming Languages (POPL)*, pages 247–259. ACM Press, January 2015.
13. J.-L. Lassez, T. Huynh, and K. McAloon. Simplification and elimination of redundant linear arithmetic constraints. In *Constraint Logic Programming*, pages 73–87. MIT Press, 1993.
14. H. Le Verge. A note on Chernikova's algorithm. Research Report RR-1662, INRIA, 1992.
15. A. Maréchal and M. Périn. Efficient elimination of redundancies in polyhedra using raytracing. Technical Report TR-2016-6, Verimag, Université Grenoble-Alpes, October 2016.
16. T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. In *Contributions to the theory of games*, volume 2 of *Annals of Mathematics Studies*, pages 51–73. Princeton University Press, 1953.
17. A. Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.
18. A. Simon and A. King. Exploiting Sparsity in Polyhedral Analysis. In *Static Analysis Symposium (SAS)*, volume 3672 of *LNCS*, pages 336–351. Springer-Verlag, 2005.

19. D. K. Wilde. A library for doing polyhedral operations. Master's thesis, Oregon State University, Corvallis, Oregon, December 1993. Also published as IRISA Technical Report PI 785, Rennes, France (1993).
20. N. Y. Zolotykh. New modification of the double description method for constructing the skeleton of a polyhedral cone. *Computational Mathematics and Mathematical Physics*, 52(1):146–156, 2012.