A Linearization Technique for Multivariate Polynomials Using Convex Polyhedra Based on Handelman-Krivine's Theorem

Alexandre Maréchal, Michaël Périn

A Linearization Technique for Multivariate Polynomials Using Convex Polyhedra Based on Handelman-Krivine's Theorem*

A. Maréchal[1,2] & M. Périn[1,3]

1: Verimag, Centre Équation
2, avenue de Vignate 38610 Gières, France
2: alex.marechal@imag.fr
3: michael.perin@imag.fr

Abstract

We present a new linearization method to over-approximate non-linear multivariate polynomials with convex polyhedra. It is based on Handelman-Krivine's theorem and consists in using products of constraints of a polyhedron to over-approximate a polynomial on this polyhedron. We implemented it together with two other linearization methods that we will not detail in this paper, but that we shall use as comparison. Our implementation in OCAML generates certificates that can be verified by a trusted checker, certified in COQ, that guarantees the correctness of our linear approximation.

1. Numerical static analysis using convex polyhedra

Static analyzers are verification tools that automatically discover program properties. They are mainly used to guarantee the absence of runtime error such as divisions by zero or accesses to arrays out of their boundaries. They operate on the source code of the program and associate invariant properties that summarize every possible memory state at a program point. An abstract domain is a class of properties able to simulate by symbolic computations the effect of guards and assignments. For instance, the abstract domain of intervals is used to capture the range of a variable at a program point in any possible execution. When information on relations between variables is needed, *e.g.* $z + 1 \leq x + y \leq 2z$, programs must be analyzed using a relational abstract domain such as convex polyhedra.

Convex polyhedra A convex polyhedron[1] is defined as a conjunction (or a set) of affine constraints of the form $\sum_{i=1}^{n} a_i x_i \geq b$ where x_i are variables, a_i and b are constants in $\mathbb{Q}$. For instance, the polyhedron P defined by the system $\{x \geq 1,\ y \geq -2,\ x - y \geq 0,\ x + y \leq 5\}$ characterizes the geometrical space $\{(x, y) \mid x \geq 1 \wedge y \geq -2 \wedge x - y \geq 0 \wedge x + y \leq 5\}$ represented in Figure 1. A polyhedron which is bounded, meaning that its constraints imply upper and lower bounds on each variable, is called a *polytope*.

*This work was 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.

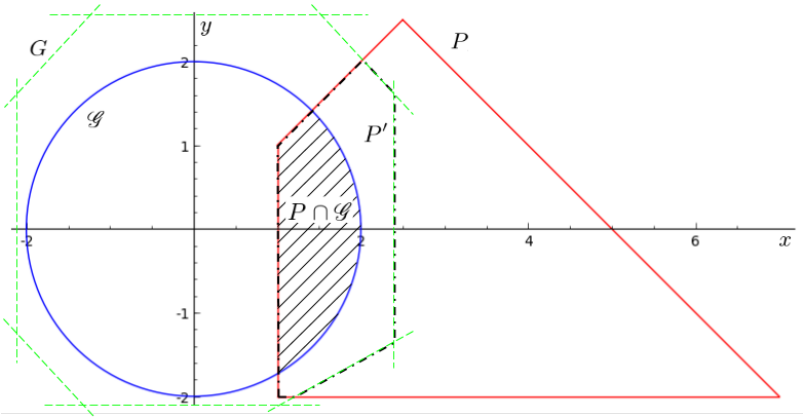


Figure 1: Graphical representations: $P \triangleq \{\, x \geq 1,\ y \geq -2,\ x - y \geq 0,\ x + y \leq 5 \,\}$ *is the initial polyhedron; the disc corresponds to the guard* $\mathscr{G} \triangleq \{\, (x, y) \mid x^2 + y^2 \leq 4 \,\}$*; the octagon* G *is a polyhedral approximation of* $\mathscr{G}$*; the hashed region represents the set* $P \cap \mathscr{G}$*; the desired approximation of* $P \cap \mathscr{G}$ *is the polyhedron* $P' \triangleq P \wedge G$ *represented in dotted lines.*

Dealing with polynomials Polyhedra suffer from an ontological limitation: they cannot deal with non-linear relations, *i.e.* multivariate polynomials containing products of variables. Typically, polynomial expressions appear in computations of physical quantities (distances, surfaces, flows, etc.) and cells coordinates in arrays, *e.g.* `i*j+k` where `i,j,k` are loop indices. A static analyzer using polyhedra cannot infer any information on a variable z assigned with a non-linear expression *e.g.* $z := x * y$. After such an assignment, z is considered to range from $-\infty$ to $+\infty$ and all the previously determined affine constraints containing z do not hold anymore. The consequence is a dramatic loss of precision which propagates along the analysis. Therefore, some *linearization techniques* are necessary to preserve precision in the presence of polynomial expressions.

A certified polyhedra library featuring linearization A Verified Polyhedra Library (VPL) was recently created by Fouilhé *et al.* [4] mixing Ocaml and Coq developments in order to bring reasonable performance and a guarantee of correctness. The polyhedral operators generate certificates which are checked by a Coq-certified validation tool that ensures the correctness of the result. This library is part of the development of a Coq-certified static analyzer [7] for the certified compiler CompCert [8]. The VPL has the particularity to use only the constraint representation of polyhedra which is more convenient for certificate generation, whereas the existing polyhedra libraries constantly need to switch between representations of polyhedra as sets of constraints and as sets of vertices, depending on the operation to perform. Our goal is to extend the VPL with a certified linearization operator that we present in this paper.

2. Affine approximation of polynomials

The goal of linearization is to approximate non-linear relations by linear ones. In this work we do not consider expressions formed of non-algebraic functions[2] like $sin, log, \ldots$ Instead, we focus on linearization methods that address polynomial expressions containing products of variables.

In this paper, the symbol g will represent a polynomial expression on the variables $x_1, \ldots, x_n$ of a program. We only consider constraints in a positive form $g \geq 0$ or $g > 0$: any other forms (including equalities and negation) can be changed into a disjunction of conjunctions of positive constraints, *e.g.*

[2]We could in principle handle analytic functions by considering their Taylor polynomials.

```
1  int x,y,z;
2  if(x >= 1 && y >= -2 && x >= y && x <= 5 - y){
3    if(x*x + y*y <= 4)
4      z = y*x ;
5    else
6      z = 0 ;
7  }
```

Figure 2: A fragment of C program containing two non-linear expressions $x*x+y*y \leq 4$ and $y*x$

$\neg(g_1 = g_2) \equiv (g_1 < g_2 \vee g_1 > g_2) \equiv (g_2 - g_1 > 0 \vee g_1 - g_2 > 0)$ and any polyhedra library is able to treat conjunction (as intersection denoted by $\sqcap$) and disjunction (as convex hull denoted by $\sqcup$).

We will use the program of Figure 2 as a running example: our goal is to compute a polyhedral approximation of the polynomial guard $x^2 + y^2 \leq 4$ on line 3 – which is equivalent to $g \geq 0$ with $g(x, y) \triangleq 4-x^2-y^2$, in the context of the polytope $P \triangleq \{x-1 \geq 0,\ y+2 \geq 0,\ x-y \geq 0,\ 5-x-y \geq 0\}$ that corresponds to the condition on line 2. We start by reducing the handling of non-linear assignment *e.g.* $z := y * x$ on line 4 of Program 2, to that of a guard. Then, we focus on linearization of guards.

The effect of an assignment $x := g$ **on a polyhedron** P corresponds to the intersection of P with a guard G formed of two inequalities $\tilde{x} - g \geq 0 \wedge g - \tilde{x} \geq 0$ encoding the equality $\tilde{x} = g$ where $\tilde{x}$ is a fresh variable that denotes the new value of x. It is renamed into x after elimination of the old value of x by projection. We denote by $P_{/x}$ the polyhedron P where the variable x has been projected. Formally, the effect of $x := g$ on P is the polyhedron $\left((P \cap G)_{/x}\right)[\tilde{x}/x]$. Therefore, if we can approximate the effect of intersection with a non-linear guard G, we can cope with non-linear assignments through projection and renaming.

The effect of a guard $g \geq 0$ **on a polyhedron** P consists in the intersection of the set of points of P with $\mathscr{G} \triangleq \{(x_1, \ldots, x_n) \mid g(x_1, \ldots, x_n) \geq 0\}$. When the guard is linear, say $x - 2y \geq 0$, $P \cap \mathscr{G}$ is simply the conjunction of P and the constraint $x - y \geq 0$; it is a polyhedron. When the guard $\mathscr{G}$ is not linear, we approximate $P \cap \mathscr{G}$ by a polyhedron P' such that $P \cap \mathscr{G} \subseteq P'$.

A naive approach would consist in guessing a polyhedron G that covers the non-linear guard $\mathscr{G}$ and then in computing the intersection $P \sqcap G$ as an approximation of $P \cap \mathscr{G}$. Have a look on Figure 1 for an example, the space $\mathscr{G}$ associated to the non-linear guard $x^2 + y^2 \leq 4$ could be included in an octagon G. Then, the dotted polyhedron $P' \triangleq P \sqcap G$ over-approximates the hashed region $P \cap \mathscr{G}$. This method is inefficient since 5 on the 8 constraints that form G are not used to compute P'. Instead, we look for a linearization algorithm that takes into account the initial polyhedron P to only compute the significant part of the approximation of $\mathscr{G} \cap P$. This paper is a step toward avoiding the calculation of the whole polyhedron G when only a small part of it is finally useful.

Related work Miné proposed two linearization techniques based on variable intervalization. It consists in replacing some variables of the polynomial by intervals to remove products of variables. The first approach is to obtain an affine expression with interval as coefficients which is converted into a polyhedron. An efficient but not very precise conversion is implemented in the APRON polyhedral library [6, 10]: intervals are replaced with their center value and the right-hand side constant of the equality is enlarged accordingly. The second approach is to switch to the abstract domain of polyhedra with interval coefficients [1] to maintain precision at the price of a high algorithmic cost.

A well known linearization method consists in representing polynomials in the Bernstein basis which allows to deduce a bounding polyhedron from its Bernstein coefficients [11]. This approach has the advantage to converge: the higher is the degree of the Bernstein basis considered, the more accurate is the approximation.

Both intervalization and Bernstein's linearization need to know the range of each variable. In

the case of the VPL which uses a constraint representation of polyhedra, getting such an information requires to run the simplex algorithm for each variable to deduce its bounds. Moreover, Bernstein's method returns a polyhedron as a set of vertices that must then be converted into the constraint representation using Chernikova's algorithm, which is expensive and not obvious to certify in Coq.

Our goal is therefore to develop an accurate linearization for the VPL. It means working only on constraints in order to avoid the overhead of range determination and conversion into the constraint representation. The approach that we shall present in this paper is a new linearization method based on Handelman-Krivine's theorem. Given a starting polyhedron P, Handelman's method consists in using products of constraints of P to obtain a polyhedral over-approximation of the polynomial constraint $g \geq 0$.

Overview of the paper We shall begin with a description of the linearization method based on Handelman-Krivine's theorem in Section 3. The parametric simplex algorithm, which is used in Handelman's method, is presented in Section 4. In Section 5, we give a sketch about the Coq certification of this linearization. The three methods are compared in terms of precision in Section 6 and we discuss about future works in Section 7.

3. Linearization on polytopes using Handelman representation

In this section, we explain how to exploit Handelman-Krivine's theorem [5] as a new linearization technique. This theorem gives a characterization of positive polynomials over a compact set.

Suppose that the polyhedron $P = \{\, x - 1 \geq 0,\ y + 2 \geq 0,\ x - y \geq 0,\ -x - y + 5 \geq 0 \,\}$ describes the possible values of (x, y) at a program point before the guard $g \geq 0$ where $g(x, y) = 4 - x^2 - y^2$. We seek to compute a polyhedron that approximates $P \cap (g \geq 0)$. A naive call to the intersection operator of the polyhedral domain would not exploit the constraint $g \geq 0$ which is not affine, and just return P. Our problem is to find a polyhedral constraint $\alpha_0 + \alpha_1 x + \alpha_2 y$, denoted by aff_g, such that $P \Rightarrow \mathit{aff}_g > g$ meaning that aff_g bounds g on the polyhedron P. By transitivity of $\geq$ we will conclude that $P \wedge g \geq 0 \Rightarrow P \wedge \mathit{aff}_g > 0$, which can be expressed in terms of sets as $(P \cap g \geq 0) \subseteq \big(P \sqcap \mathit{aff}_g > 0\big)$. Thus, $P \sqcap \mathit{aff}_g > 0$ will be a polyhedral approximation of the program state after the polynomial guard $g \geq 0$.

3.1. Handelman representation of positive polynomials

Notations Tuples $\boldsymbol{x} = (x_1, \ldots, x_n)$ and multi-indexes $\boldsymbol{I} = (i_1, ..., i_p) \in \mathbb{N}^p$ are typed in boldface. We define a partial order on multi-indexes by $\boldsymbol{I} \leq \boldsymbol{J} \Leftrightarrow \forall k = 1, .., p,\ i_k \leq j_k$. Let us define the set of Handelman products associated to a polyhedron $P \triangleq \{\, C_1 \geq 0, \ldots, C_p \geq 0 \,\}$ as

$$\mathscr{H}_P = \{\, C_1^{i_1} \times \cdots \times C_p^{i_p} \mid (i_1, \ldots, i_p) \in \mathbb{N}^p \,\}$$

The set $\mathscr{H}_P$ contains all products of constraints C_i of P. Given a multi-index $\boldsymbol{I} = (i_1, \ldots, i_p)$, $H^{\boldsymbol{I}} \triangleq C_1^{i_1} \times \ldots \times C_p^{i_p}$ denotes an element of $\mathscr{H}_P$. Note that the $H^{\boldsymbol{I}}$ are positive polynomials on P as products of positive constraints of P.

Example 1. *Considering our running example,* $H^{(0,2,0,0)} = (y+2)^2$, $H^{(1,0,1,0)} = (x-1)(x-y)$ *and* $H^{(1,0,0,3)} = (x-1)(-x-y+5)^3$ *belong to* $\mathscr{H}_P$.

The Handelman representation of a positive polynomial $g(\boldsymbol{x})$ on P is

$$g(\boldsymbol{x}) = \sum_{\boldsymbol{I} \in \mathbb{N}^p} \underbrace{\lambda_I}_{\geq 0} \underbrace{H^{\boldsymbol{I}}}_{\geq 0} \ \textit{with}\ \lambda_{\boldsymbol{I}} \in \mathbb{R}^+$$

This representation is used in mathematics as a certificate, called a positivstellensatz, ensuring that $g(\boldsymbol{x})$ is positive on P. Clearly if a polynomial can be written in this form, it is necessarily positive on P. Handelman-Krivine's theorem [5], that we summarize here, concerns the non-trivial opposite implication:

Theorem 1 (Handelman-Krivine's Theorem, 1988)**.** *Let $P = \{C_1 \geq 0, \ldots, C_p \geq 0\}$ be a compact polytope where each C_i is an affine polynomial over $\boldsymbol{x} = (x_1, \ldots, x_n)$. Let $g(\boldsymbol{x})$ be a positive polynomial on P. Then there exists $\lambda_{\boldsymbol{I}} \in \mathbb{R}^+$ and $H^{\boldsymbol{I}} \in \mathscr{H}_P$ such that $g(\boldsymbol{x}) = \sum_{\boldsymbol{I} \in \mathbb{N}^p} \lambda_{\boldsymbol{I}} H^{\boldsymbol{I}}$*

Usually, the Handelman representation of a polynomial $g(\boldsymbol{x})$ is used to determine a constant lower bound of $g(\boldsymbol{x})$ on P thanks to Schweighofer's algorithm [9] that focuses on iteratively improving the bound by increasing the degree of the $H^{\boldsymbol{I}}$. We present here another use of Handelman-Krivine's theorem: we are not interested in just one bound but in a polyhedron dominating the polynomial $g(\boldsymbol{x})$ on P.

3.2. Handelman linerization as a Parametric Linear Optimization Problem

Remember that we are looking for an affine constraint aff_g that approximates a non-linear guard g, meaning $\mathit{aff}_g > g$ on P. This is equivalent to $\mathit{aff}_g - g > 0$ on P and then, Handelman-Krivine's theorem applies. Note that this theorem requires the polyhedron P to be a polytope. Thus from now, we will consider P as a polytope.

> *The polynomial $\mathit{aff}_g - g$ which is positive on the polytope P has an* Handelman representation *as a positive linear combination of products of the constraints of P, i.e.*
>
> $$\mathit{aff}_g - g = \sum_{\boldsymbol{I} \in \mathbb{N}^p} \lambda_{\boldsymbol{I}} H^{\boldsymbol{I}},\ \lambda_{\boldsymbol{I}} \in \mathbb{R}^+,\ H^{\boldsymbol{I}} \in \mathscr{H}_P \tag{1}$$

The relation (1) of the theorem ensures that there exists some positive combinations of g and some $H^{\boldsymbol{I}} \in \mathscr{H}_P$ that remove the monomials of total degree >1 and lead to affine forms:

$$\alpha_0 + \alpha_1 x_1 + \ldots + \alpha_n x_n = \mathit{aff}_g = 1 \cdot g + \sum_{\boldsymbol{I} \in \mathbb{N}^p} \lambda_{\boldsymbol{I}} H^{\boldsymbol{I}}$$

The theorem says that the polynomials of $\mathscr{H}_P$ are *generators* of the positive polynomials on P, not a *basis*. Indeed, it is possible to have one $H \in \mathscr{H}_P$ being a positive linear combination of other elements of $\mathscr{H}_P$.

Example 2. *Consider $P = \{(x, y) \mid x \geq 0,\ y \geq 0,\ x - y \geq 0,\ x + y \geq 0\}$. Then, $H^{(2,0,0,0)} \triangleq x^2$, $H^{(1,1,0,0)} = xy$, $H^{(0,2,0,0)} = y^2$, $H^{(0,0,0,2)} = (x + y)^2$ belong to Handelman products and they are not independant. Indeed, $H^{(0,0,0,2)} = x^2 + 2xy + y^2 = H^{(2,0,0,0)} + 2H^{(1,1,0,0)} + H^{(0,2,0,0)}$*

As a consequence, a positive polynomial $\mathit{aff}_g - g$ can have several Handelman representations, even on a given set of Handelman products. Actually, we exploit the non-uniqueness of representation to get a precise approximation of the guard: we look for several affine constraints aff_g bounding g on P. Their conjunction forms a polyhedron that over-approximates g on P.

Polyhedral approximation as solution of a linear problem We now explain how the determination of all polyhedral constraints bounding g can be expressed as a *linear problem*.

Let us denote by $\mathbf{deg}\,(\boldsymbol{x_1^{i_1} \ldots x_n^{i_n}}) = (i_1, ..., i_n)$ the degree of the monomial $x_1^{i_1} \ldots x_n^{i_n}$. Let $\boldsymbol{d_g} \in \mathbb{N}^n$ be the maximal degree of the monomials of g *i.e.* $\mathbf{deg}\,(\boldsymbol{m}) \leq \boldsymbol{d_g}$ for all monomials m in g. We restrict

our search to finding a Handelman representation of $\mathit{aff}_g - g$ on the subset $\{H_1, \ldots, H_q\}$ of all the Handelman products $H^{\boldsymbol{I}}$ such that $\mathbf{deg}\,(\boldsymbol{m}) \leq \boldsymbol{d_g}$ for all monomials m of $H^{\boldsymbol{I}}$, instead of the whole set $\mathscr{H}_P$. If we fail with monomials of degree $\leq \boldsymbol{d_g}$ we increase it ; Handelman-Krivine's theorem ensures that we will eventually succeed.

Example 3. *With $g = 4 - x^2 - y^2$, $\boldsymbol{d_g} = (2,2)$. Therefore, we shall consider the 15 following Handelman products:*

$$
\begin{array}{ll}
H_1 = H^{(0,0,0,0)} = 1 & H_2 = H^{(1,0,0,0)} = x - 1 \\
H_3 = H^{(0,1,0,0)} = y + 2 & H_4 = H^{(0,0,1,0)} = x - y \\
H_5 = H^{(0,0,0,1)} = -x - y + 5 & H_6 = H^{(2,0,0,0)} = (x-1)^2 \\
H_7 = H^{(0,2,0,0)} = (y+2)^2 & H_8 = H^{(0,0,2,0)} = (x-y)^2 \\
H_9 = H^{(0,0,0,2)} = (-x-y+5)^2 & H_{10} = H^{(1,1,0,0)} = (x-1)(y+2) \\
H_{11} = H^{(1,0,1,0)} = (x-1)(x-y) & H_{12} = H^{(1,0,0,1)} = (x-1)(-x-y+5) \\
H_{13} = H^{(0,1,1,0)} = (y+2)(x-y) & H_{14} = H^{(0,1,0,1)} = (y+2)(-x-y+5) \\
H_{15} = H^{(0,0,1,1)} = (x-y)(-x-y+5) &
\end{array}
$$

With the restriction to $\{H_1, \ldots, H_q\}$, finding the $\lambda_{\boldsymbol{I}}$ can be formulated as a linear problem. Relation (1) on $\{H_1, \ldots, H_q\}$ amounts to find positive $\lambda_1, \ldots, \lambda_q \in \mathbb{R}^+$ such that

$$
\begin{array}{ccccc}
\mathit{aff}_g & = & 1 \cdot g + \sum_{i=1}^{i=q} \lambda_i H_i & = & \underbrace{(\lambda_g, \lambda_1, \ldots, \lambda_q)}_{\boldsymbol{\lambda}^\intercal} \cdot \underbrace{(g, H_1, \ldots, H_q)^\intercal}_{\mathcal{H}_g{}^\intercal \cdot \boldsymbol{m}} \\
\| & & \| & & \\
\alpha_0 + \alpha_1 x_1 + \ldots + \alpha_n x_n & & \boldsymbol{\lambda}^\intercal \cdot \mathcal{H}_g{}^\intercal \cdot \boldsymbol{m} & & \\
\| & & \| & & \\
\boldsymbol{m}^\intercal \cdot (\alpha_0, \ldots, \alpha_n, 0, \ldots, 0) & = & \boldsymbol{m}^\intercal \cdot \mathcal{H}_g \cdot \boldsymbol{\lambda} & &
\end{array}
$$

where:

- $\mathcal{H}_g$ is the matrix of the coefficients of g and the H_i in the canonical basis of monomials of degree $\leq \boldsymbol{d_g}$ denoted by $\boldsymbol{m} = (1, x_1, \ldots, x_n, \boldsymbol{x}^{\boldsymbol{d_1}}, \ldots, \boldsymbol{x}^{\boldsymbol{d_g}})^\intercal$
- the column vector $\boldsymbol{\lambda} = (\lambda_g, \lambda_1, \ldots, \lambda_q)^\intercal = (1, \lambda_1, \ldots, \lambda_q)^\intercal$ characterizes the Handelman's positive combination of g and the H_i. We associated a coefficient $\lambda_g = 1$ to g just to get convenient notations.

The result of the matrix-vector product $\mathcal{H}_g \cdot \boldsymbol{\lambda}$ is a vector $\boldsymbol{\alpha} \triangleq (\alpha_0, \alpha_1, \ldots, \alpha_n, \alpha_{\boldsymbol{d_1}}, \ldots, \alpha_{\boldsymbol{d_g}})^\intercal$ that represents the constraint $\alpha_0 + \alpha_1 x_1 + \ldots + \alpha_n x_n + \sum_{I \leq \boldsymbol{d_g}} \alpha_I \boldsymbol{x}^I$ in the $\boldsymbol{m}$ basis. Since we seek an affine constraint aff_g we are finally interested in finding $\boldsymbol{\lambda} \in \{1\} \times (\mathbb{R}^+)^q$ such that

$$\mathcal{H}_g \cdot \boldsymbol{\lambda} = (\alpha_0, \ldots, \alpha_n, 0, \ldots, 0)^\intercal$$

By construction, each such $\boldsymbol{\lambda}$ gives an affine constraint aff_g that bounds g on P. Therefore, the conjunction of all constraints $\mathit{aff}_g \geq 0$ form a polyhedron that approximates the guard $g \geq 0$ on P.

Example 4. *Here is the matrix $\mathcal{H}_g$ of $g \triangleq 4 - x^2 - y^2$ and our 15 Handelman products with respect to the basis $(1, x, y, xy, x^2, y^2)$.*

$$
\begin{array}{c}
 \\
1 \\
x \\
y \\
xy \\
x^2 \\
y^2
\end{array}
\begin{pmatrix}
g & H_1 & H_2 & H_3 & H_4 & H_5 & H_6 & H_7 & H_8 & H_9 & H_{10} & H_{11} & H_{12} & H_{13} & H_{14} & H_{15} \\
4 & 1 & -1 & 2 & 0 & 5 & 1 & 4 & 0 & 25 & -2 & 0 & -5 & 0 & 10 & 0 \\
0 & 0 & 1 & 0 & 1 & -1 & -2 & 0 & 0 & -10 & 2 & -1 & 6 & 2 & -2 & 5 \\
0 & 0 & 0 & 1 & -1 & -1 & 0 & 4 & 0 & -10 & -1 & 1 & 1 & -2 & 3 & -5 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -2 & 2 & 1 & -1 & -1 & 1 & -1 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & -1 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & -1 & -1 & 1
\end{pmatrix}
$$

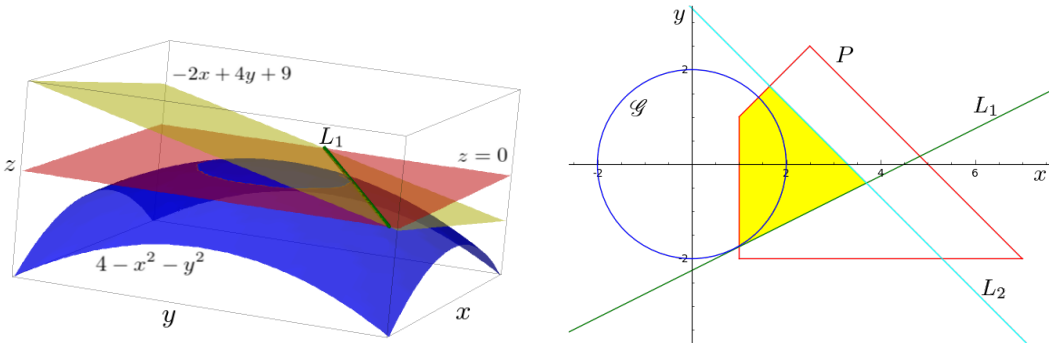


Figure 3: Fig.(b) is the cut at $z=0$ of Fig.(a) in which we added the polyhedron $P \triangleq \{\, x-1 \geq 0,\ y+2 \geq 0,\ x-y \geq 0,\ -x-y+5 \geq 0 \,\}$: The circle $\mathscr{G}$ of Fig.(b) appears in (a) as the intersection of the surface $z = g(x,y) \triangleq 4-x^2-y^2$ with the plane $z=0$. The polyhedral approximation of g is the inclined plane $z = aff_g(x,y) \triangleq -2x+4y+9$ that dominates g. It cuts the plane $z=0$ along the line L_1 in Fig.(a) which is reported in (b). The lines L_1 (resp. L_2) are the frontiers of the aff_g constraints $-2x+4y+9 \geq 0$ (resp. $-5x-5y+\frac{33}{2} \geq 0$). The filled area is the polyhedron $P \wedge -2x+4y+9 \geq 0 \wedge -5x-5y+\frac{33}{2} \geq 0$ that over-approximates $P \cap \{\,(x,y) \mid g(x,y) \geq 0\,\}$.

The choices $\lambda_g = \lambda_6 = \lambda_7 = 1$ and every other $\lambda_i = 0$ is a solution to the problem $\mathcal{H}_g \cdot \boldsymbol{\lambda} = (\alpha_0, \alpha_1, \alpha_2, 0, 0, 0)^\intercal$. We obtain $\mathcal{H}_g \cdot \boldsymbol{\lambda} = (9, -2, 4, 0, 0, 0)$ that corresponds to $9-2x+4y+0\times xy+0\times x^2+0\times y^2$ in the basis $\boldsymbol{m} = (1, x, y, xy, x^2, y^2)$. Thus, $aff_g = 9-2x+4y$ is a constraint that bounds g on P, as shown on the Figure 3(a): in Cartesian coordinates (x,y,z), the plane $z = 9-2x+4y$ dominates the polynomial $z = g(x,y)$. Therefore, the effect of the guard $g \geq 0$ on P is approximated by the affine constraint $9-2x+4y \geq 0$. Figure 3(b) shows its intersection with the plane $z=0$. It is a half-space whose frontier is the line L_1 of equation $9-2x+4y=0$.

Any solution $\boldsymbol{\lambda}$ of the problem $\mathcal{H}_g \cdot \boldsymbol{\lambda} = (\alpha_0, \ldots, \alpha_n, 0, \ldots, 0)$ is a polyhedral constraint *aff* that bounds g on P. Among all these solutions we are only interested in the best approximations. One constraint $aff > g$ is better than another $aff' > g$ at point $(x_1, \ldots, x_n)$ if $aff'(x_1, \ldots, x_n) > aff(x_1, \ldots, x_n)$. It then appears that for a given point $(x_1, \ldots, x_n)$ we are looking for the polyhedral constraint $aff > g$ that minimizes its value on that point. Therefore, we change our linear problem into a linear minimization problem that depends on some parameters: the point $(x_1, \ldots, x_n)$ of evaluation.

Linearization as a Parametric Linear Optimization Problem We now come to the main result of the paper: finding the tightest approximations aff_g that bounds g on P can be expressed as a *Parametric Linear Optimization Problem* (PLOP) which can be solved using our *Simplex Algorithm for Parametric Objective*:

> *Given a set $\{\, H_1, \ldots, H_q \,\} \subseteq \mathscr{H}_P$ of Handelman products,*
> ***minimize*** *aff_g that is, $\alpha_0 + \alpha_1 x_1 + \ldots + \alpha_n x_n$, which is also equal to $g + \sum_{i=1}^{i=q} \lambda_i H_i$*
> ***under the constraints***
>
> $$\begin{cases} \mathcal{H}_g(\lambda_g, \lambda_1, \ldots, \lambda_q)^\intercal = (\alpha_0, \ldots, \alpha_n, 0, \ldots, 0)^\intercal \\ \lambda_g = 1 \\ \lambda_i \geq 0,\ i = 1..q \end{cases} \qquad \text{(H-PLOP)}$$
>
> *where $\lambda_1, \ldots, \lambda_q$ are the decision variables of the PLOP; $x_1, \ldots, x_n$ are the parameters. $\alpha_0, \ldots, \alpha_n$ are kept for the sake of presentation ; in practice they are substituted by their expression issued from $\mathcal{H}_g\boldsymbol{\lambda}$.*

Example 5. *The objective aff_g in the form $g + \sum_{i=1}^{i=q} \lambda_i H_i$ is*

$$4 + \lambda_1 + \lambda_2(x-1) + \lambda_3(2+y) + \lambda_4(x-y) + \lambda_5(5-x-y) + \lambda_6(1-2x) + \lambda_7(4+4y)$$
$$+\lambda_9(25-10x-10y) + \lambda_{10}(2x-y-2) + \lambda_{11}(y-x) + \lambda_{12}(6x+y-5) + \lambda_{13}(2x-2y)$$
$$+\lambda_{14}(10-2x+3y) + \lambda_{15}(5x-5y).$$

This is the representation used in practice as it exhibits the parametric coefficients in x, y of each variable λ. Note that the monomials of degree > 1 do not appear since the problem imposes to cancel the non-linear part of $g + \sum_{i=1}^{i=q} \lambda_i H_i$, i.e. $xy\,(-2\lambda_8+2\lambda_9+\lambda_{10}-\lambda_{11}-\lambda_{12}+\lambda_{13}-\lambda_{14}) + x^2\,(-1+\lambda_6+\lambda_8+\lambda_9+\lambda_{11}-\lambda_{12}-\lambda_{15}) + y^2\,(-1+\lambda_7+\lambda_8+\lambda_9-\lambda_{13}-\lambda_{14}+\lambda_{15})$. The solutions of the problem are the vectors $\boldsymbol{\lambda}$ that minimize the objective and cancel the coefficients of xy, x^2 and y^2. The reader could prefer the objective in a re-factorized form $\alpha_0 + \alpha_1 x + \alpha_2 y$ in order to get the approximation aff_g in terms of $\boldsymbol{\lambda}$, where

$$\alpha_0 = 4+\lambda_1-\lambda_2+2\lambda_3+\lambda_6+4\lambda_7+25\lambda_9-2\lambda_{10}-5\lambda_{12}+10\lambda_{14}$$
$$\alpha_1 = \lambda_2+\lambda_4-\lambda_5-2\lambda_6-10\lambda_9+2\lambda_{10}-\lambda_{11}+6\lambda_{12}+2\lambda_{13}-2\lambda_{14}+5\lambda_{15}$$
$$\alpha_2 = \lambda_3-\lambda_4-\lambda_5+4\lambda_7-10\lambda_9-\lambda_{10}+\lambda_{11}+\lambda_{12}-2\lambda_{13}+3\lambda_{14}-5\lambda_{15}$$

Finally, remark that any instantiation of the parameters x and y with constants would define a standard Linear Optimization Problem *with an objective that is an affine expression $c_0 + \sum_{i=1}^{i=q} \lambda_i c_i$ with constant costs $c_0 = g(x,y)$, $c_i = H_i(x,y)$. It could then be solved by the simplex algorithm, providing the optimal value of the instantiated objective.*

In the next section, we describe our *parametric simplex algorithm* able to discover all optimal solutions of H-PLOP without instantiating the parameters. The result of this algorithm is a pre-computed function $R \mapsto (\alpha_0, \alpha_1, \ldots, \alpha_n)$ that partitions the parameter space into regions $R_1, \ldots, R_s$ represented by polyhedra, and that associates an optimal affine forms $aff_g = \alpha_0 + \alpha_1 x_1 + \ldots + \alpha_n x_n$ to each region.

Before developing our own algorithm we experimented PIP – Feautrier's parametric simplex [3] which was designed for PLOP with parameters in the right-hand side of constraints, not in the objective. We tried using it by dualizing our problem: this classical transformation moves the objective to the right-hand side of the dual problem constraints. However, it was not obvious how to get the polyhedral constraint aff_g from the parametric solution of the dual problem. Finally, we kept the idea of Feautrier's parametric simplex and we implemented a version for parametric objectives.

4. A simplex algorithm with parametric objective

The classical simplex algorithm is used to find the optimal solution to a linear problem of the form

$$\textbf{\textit{minimize the objective}} \sum_{1 \le i \le q} c_i \lambda_i \textbf{\textit{ under the constraints }} A\boldsymbol{\lambda} \ge \boldsymbol{b},\ \boldsymbol{\lambda} \ge 0$$

where $\boldsymbol{b} \in \mathbb{Q}^p$, $A \in M_{p,q}(\mathbb{Q})$ and $(c_1, \ldots, c_q) \in \mathbb{Q}^q$ defines the cost associated to each decision variable of $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_q)$. We assume the reader is familiar with the standard simplex algorithm which is clearly explained in [2].

Algorithm 1 presents our parametric simplex, *i.e.* a simplex able to find the optimal solution of linear programming problems when the objective cost coefficients c_i are linear forms of some parameters $(x_1, \ldots, x_n)$. The differences with respect to the standard simplex algorithm are enlightened by boxes. The condition $R \sqsubseteq c_j > 0$ that appear in the algorithm checks if the constraints of the polyhedron R entail the constraint $c_j > 0$. These tests are performed by a decision procedure of the VPL for testing polyhedra inclusion. We now illustrate the differences between the standard and the parametric simplex on an execution of the algorithm.

Let us consider (PLOP 1) whose objective contains two parametric coefficients c_1 and c_2. For simplicity we consider the case where the affine forms are reduced to parameters, *i.e.* $c_1(x_1, x_2) = x_1$

and $c_2(x_1, x_2) = x_2$.

$$\begin{array}{l} \textbf{\textit{minimize}}\ \ x_1\lambda_1 + x_2\lambda_2\ \ \textbf{\textit{under the constraints}} \\ \lambda_1 + \lambda_2 \leq 5 \qquad -\lambda_2 \leq 2 \quad \lambda_1, \lambda_2 \geq 0 \\ \quad -\lambda_1 \leq 1 \quad -\lambda_1 + \lambda_2 \leq 0 \end{array} \tag{PLOP 1}$$

Finding a feasible solution The first step is a normalization: it changes inequalities into equalities at the price of introducing a new positive variable s_i per constraint. For instance, the first constraint $\lambda_1 + \lambda_2 \leq 5$ of PLOP 1 is replaced by the equation $s_1 = 5 - \lambda_1 - \lambda_2$ and the constraint $s_1 \geq 0$. The variable s_1 measures the slack between the left-hand side term $\lambda_1 + \lambda_2$ and the right-hand side constant 5. Repeating this for all constraints, PLOP 1 can be rewritten into

$$\begin{array}{l} \textbf{\textit{minimize}}\ \ x_1\lambda_1 + x_2\lambda_2\ \ \textbf{\textit{under the constraints}} \\ s_1 = 5 - \lambda_1 - \lambda_2 \quad s_3 = 2 + \lambda_2 \qquad s_1, s_2, s_3, s_4, \lambda_1, \lambda_2 \geq 0 \\ s_2 = 1 + \lambda_1 \qquad\ \ s_4 = \lambda_1 - \lambda_2 \end{array} \tag{PLOP 2}$$

Note that every solution (λ_1, λ_2) of PLOP 1 leads to a unique solution of PLOP 2 thanks to the equations. Reciprocally, from a solution of PLOP 2, a solution of PLOP 1 can be deduced by removing the slack variables. A point satisfying the constraints of PLOP 2 can now be found easily in our example, setting λ_1 and λ_2 at 0. We obtain the solution ($\lambda_1 = 0,\ \lambda_2 = 0,\ s_1 = 5,\ s_2 = 1,\ s_3 = 2,\ s_4 = 0$). The value of the objective associated to this point is $x_1 \times 0 + x_2 \times 0 = 0$. If PLOP 2 is feasible but the solution obtained by setting λ_1 and λ_2 at 0 is not (*i.e.* $s_i < 0$ for some i), there exists an *auxiliary* problem able to find a feasible solution. This process is detailed in [2].

Vocabulary The system composed of the equations and the objective function is called a *dictionary*. A *feasible point* is a valuation of the decision and slack variables $(\lambda_1, \ldots, \lambda_p, s_1, \ldots, s_q)$ that satisfies the equations. The variables appearing on the left-hand side of the equations are usually called *basic variables*. They are defined in terms of the *nonbasic variables* appearing on the right-hand side, also called *null variables* since we obtain a feasible point by setting those variables to 0 and computing the values of the basic variables accordingly. For an homogeneous presentation, one can think of the objective being just another equation $z = \sum x_i\lambda_i$ with a fresh variable z. A minimization (resp. maximization) problem looks for the lower (resp. upper) bound of z.

Finding better solutions Once we have a feasible point, the next step is to find another one that has a smaller objective value. The idea is to pick a variable that has a negative coefficient in the current objective $x_1\lambda_1 + x_2\lambda_2$ and to increase it as much as possible, such that the dictionary remains satisfiable. Thus, we look for the sign of the coefficients x_i. Due to its parametric nature, in general the sign of x_i cannot be decided. So, we build an exploration tree which considers two cases $x_i < 0$ and $x_i \geq 0$. First, looking at the coefficient x_1, we shall assume $x_1 < 0$ and obtain a branch leading to one or several optimal solutions depending on the sign of x_2. Second, we shall assume $x_1 \geq 0$ and explore a second branch.

Let us execute the algorithm for the branch $x_1 < 0$ to clearly see what type of results appears. The variable λ_1 that was previously null is now entering the basis. It will be then defined in terms of the nonbasic variables and, as in the standard simplex algorithm, we must find the constraint bounding the value of λ_1 the most. This will determine the variable leaving the basis, *i.e.* becoming null. The limiting constraint is $s_1 = 5 - \lambda_1 - \lambda_2$ which prevents λ_1 to exceed 5, reached when $s_1 = \lambda_2 = 0$. A pivot operation similar to the one used in Gaussian elimination substitutes λ_1 by its definition $5 - s_1 - \lambda_2$ in every other equations, including the objective. The pivot of λ_1 with s_1 gives a new dictionary PLOP 3 equivalent to PLOP 2, a new basis $\mathscr{B} = \{\lambda_1, s_2, s_3, s_4\}$, a new feasible point

$(\lambda_1 = 5,\ \lambda_2 = 0,\ s_1 = 0,\ s_2 = 6,\ s_3 = 2,\ s_4 = 5)$ and the minimal objective value is now $5x_1$. A crucial point of the simplex algorithm is that each pivot preserves the following invariants: each variable in the basis $\mathscr{B}$ has a null coefficient in the objective. Therefore, the minimum value of the objective is obtained by setting to their lower bound (*i.e.* 0) the *non basic* variables that still appear in the objective. This process allows to move from a feasible point to another.

$$\begin{array}{lll} \textbf{\textit{minimize}} & -x_1 s_1 + (x_2 - x_1)\lambda_2 + 5x_1 & \textbf{\textit{under the constraints}} \\ \lambda_1 = -s_1 - \lambda_2 + 5 & s_3 = 2 + \lambda_2 & s_1, s_2, s_3, s_4, \lambda_1, \lambda_2 \geq 0 \\ s_2 = -s_1 - \lambda_2 + 6 & s_4 = -2\lambda_2 - s_1 + 5 & \end{array} \quad \text{(PLOP 3)}$$

For the next iteration, we look again for a variable whose coefficient is negative in the objective: $-x_1$ cannot be negative because we made the assumption $x_1 < 0$. However, none of our assumptions prevents $x_2 - x_1$ from being negative. Again, we explore two branches where the first one shall assume $x_1 < 0 \wedge x_2 - x_1 < 0$ whereas the other one shall assume $x_1 < 0 \wedge x_2 - x_1 \geq 0$. In the first of these two branches, the variable entering the basis is λ_2, and the most limiting constraint is $s_4 = -2\lambda_2 - s_1 + 5$ – it imposes the upper bound $\frac{5}{2}$ on λ_2 – thus the leaving variable is s_4. The new feasible solution is $\big(\lambda_1 = \frac{5}{2},\ \lambda_2 = \frac{5}{2},\ s_1 = 0,\ s_2 = \frac{7}{2},\ s_3 = \frac{9}{2},\ s_4 = 0\big)$, $\mathscr{B}$ becomes $\{\lambda_1, \lambda_2, s_2, s_3\}$, the minimal value of the objective is $\frac{5x_1+5x_2}{2}$ and we end with the dictionary:

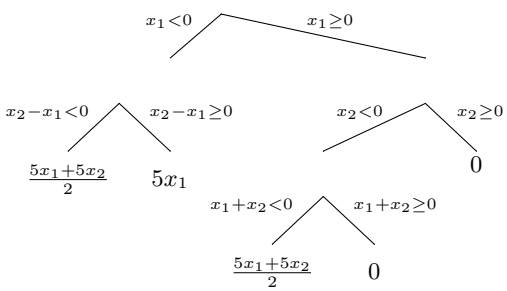
$$\begin{array}{lll} \textbf{\textit{minimize}} & \frac{-x_1-x_2}{2} s_1 + \frac{x_1-x_2}{2} s_4 + \frac{5x_1+5x_2}{2} & \textbf{\textit{under the constraints}} \\ \lambda_1 = -\frac{s_1}{2} + \frac{s_4}{2} + \frac{5}{2} & s_3 = -\frac{s_4}{2} - \frac{s_1}{2} + \frac{9}{2} & s_1, s_2, s_3, s_4, \lambda_1, \lambda_2 \geq 0 \\ s_2 = \frac{s_4}{2} - \frac{s_1}{2} + \frac{7}{2} & \lambda_2 = -\frac{s_4}{2} - \frac{s_1}{2} + \frac{5}{2} & \end{array} \quad \text{(PLOP 4)}$$

Recall that our current region is $x_1 < 0 \wedge x_2 - x_1 < 0$. With these assumptions, neither $\frac{-x_1-x_2}{2}$ nor $\frac{x_1-x_2}{2}$ can be negative. Thus, there is no more way to improve the current objective meaning that the optimum has been found *for this region of the parameter space*. For the second branch, going back to PLOP 3 and assuming $x_1 < 0 \wedge x_2 - x_1 \geq 0$, there is no remaining variable with a negative coefficient in the objective. Thus, the optimum for this region is $5x_1$. The region where $x_1 < 0$ has been fully explored and leads to two different optimal solutions depending on x_2. At this point, the space where $x_1 \geq 0$ still needs to be traveled, so the next iteration starts from PLOP 1 with the assumption $x_1 \geq 0$.

Solutions of the parametric simplex

The parametric simplex returns a function z^* of the parameters $x_1, \ldots, x_n$ that associates an optimum to each region of the parameter space. In practice, it is represented as a decision tree. For instance, solving PLOP 1 we obtain the following optimum function and decision tree:

$$z^*(x_1, x_2) = \begin{cases} \frac{5x_1+5x_2}{2} & \text{if } x_1 < 0 \wedge x_2 - x_1 < 0 \\ 5x_1 & \text{if } x_1 < 0 \wedge x_2 - x_1 \geq 0 \\ \frac{5x_1+5x_2}{2} & \text{if } x_1 \geq 0 \wedge x_2 < 0 \wedge x_1 + x_2 < 0 \\ 0 & \text{if } x_1 \geq 0 \wedge x_2 < 0 \wedge x_1 + x_2 \geq 0 \\ 0 & \text{if } x_1 \geq 0 \wedge x_2 \geq 0 \end{cases}$$



The tree summarizes the optimal solutions depending on the different regions encountered throughout the algorithm. Note that the leaves are parametric optima: they depend on the parameters (x_1, x_2).

Algorithm 1: Parametric Simplex Algorithm

Input : **A polyhedron** as a set of constraint $\mathscr{C} \triangleq \{C_1, \ldots, C_q\}$ where $C_i \triangleq \sum_{1 \leq j \leq p} a_{i,j} v_j \geq b_i$, the $v_1, \ldots, v_p$ are the decision variables and $a_{i,j}, b_i \in \mathbb{Q}$.
A parametric objective $O \triangleq c_0 + \sum_{1 \leq j \leq p} c_j v_j$ where $c_j(x_1, \ldots, x_n)$ and $c_0(x_1, \ldots, x_n)$ are affine forms on the parameters $x_1, \ldots, x_n$; we omit the parameters for readability.
An optional region R which is a set of polyhedral constraints on the parameters. By default, $R \triangleq \emptyset$.

Output: A binary tree of type tree = LEAF(objectif, witness) | BRANCH(tree,constraint,tree) where a witness is a set of couples $(\lambda_i, i) \in \mathbb{Q}^+ \times \mathbb{N}$

Data : Let $\mathscr{E}$ denote the set of equations associated to $\mathscr{C}$ with q additional slack variables, and $\mathcal{V}$ be the set of variables of $\mathscr{E}$, that is $\{v_1, \ldots, v_{p+q}\}$. $\mathscr{B}$ denotes a subset of $\mathcal{V}$. Any variable in $\mathscr{B}$ is defined by an equality of $\mathscr{E}$ in terms of the variables not in $\mathscr{B}$.

Function `getColPivot` (R : region, O : objective) : set of int
  return $\left\{ i \mid c_i \in \left\{ c_j \mid \boxed{R \not\sqsubseteq c_j > 0} \right\} \right\}$ **where** O **is** $\sum_{1 \leq j \leq p} c_j v_j + c_0$

Function `getRowPivot` ($\mathscr{E} = \{E_1, \ldots, E_q\}$: set of equalities, col : int) : int
  return row *such that* $\frac{b_{row}}{a_{row,col}}$ *is the upper bound of* v_{col} *i.e.*
  where $\frac{b_{row}}{a_{row,col}} = \min_{1 \leq i \leq q} \left\{ \frac{b_i}{a_{i,col}} \;\middle|\; a_{i,col} < 0, \left(v_i = \left(\sum_{1 \leq j \leq p,\ j \neq i,col} a_{i,j} v_j \right) + a_{i,col} v_{col} + b_i \right) \in \mathscr{E} \right\}$

Function `pivot` ($\mathscr{B}$: basis, O : objective, $\{E_1, \ldots, E_q\}$: set of equalities, col : int) : objective $\times$ set of equalities
  $row \leftarrow$ `getRowPivot`$(\{E_1, \ldots, E_q\}, col)$
  $E'_{row} \leftarrow \frac{1}{a_{row,col}} E_{row}$
  $O' \leftarrow O - c_{col} E_{row}$
  for $i = 1, \ldots, q,\ i \neq row$ **do** $E'_i \leftarrow E_i - a_{i,col} E_{row}$
  $\mathscr{B}' \leftarrow (\mathscr{B} \setminus \{v_{row}\}) \cup \{v_{col}\}$
  return $(\mathscr{B}',\ O',\ \{E'_1, \ldots, E'_q\})$

Function `explore` (R : region, $\mathscr{B}$: basis, O : objective, $\mathscr{E}$: set of equalities) : tree
  $K \leftarrow$ `getColPivot`(R, O)
  if $K = \emptyset$ **then return** LEAF(O, `witness`$(\mathscr{B}, \mathscr{E})$)
  else
    $col \leftarrow min(K)$
    if $\boxed{R \sqsubseteq c_{col} < 0}$ **then return** `explore`(R, `pivot`($\mathscr{B}$, O, $\mathscr{C}$, k))
    else *Both* $c_{col} > 0$ *and* $c_{col} < 0$ *can be true, we explore both cases*
      negativeCase $\leftarrow$ `explore` $(R \cup \{c_{col} > 0\}, \mathscr{B}, O, \mathscr{C})$
      positiveCase $\leftarrow$ `explore`$(R \cup \{c_{col} < 0\}$, `pivot`$(\mathscr{B}, O, \mathscr{C}, k))$
      return BRANCH(negativeCase, c_{col}, positiveCase)

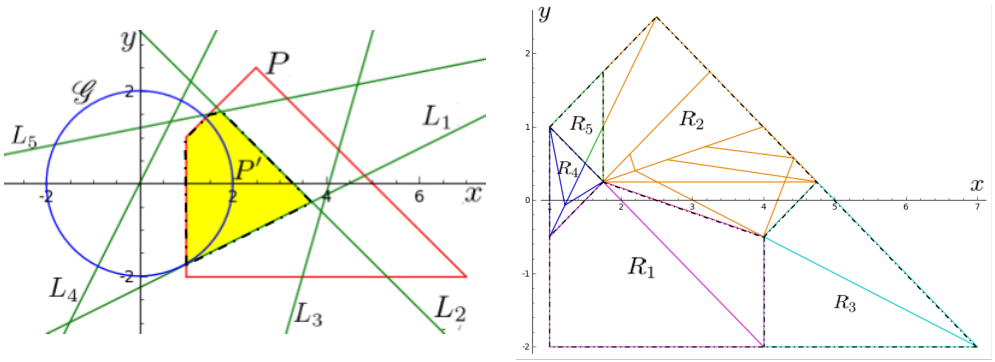
Function `witness` ($\mathscr{B}$: basis, $\{E_1, \ldots, E_q\}$: set of equalities) : witness
  for $v \notin \mathscr{B}$ **do** $v \leftarrow 0$
  return $\bigcup_i \{(b_i, j)$ **where** E_i **is** $v_j = b_i\}$

Function `parametricSimplex` (R : region, O : objective, $\mathscr{C}$: set of constraints) : tree
  Objective and constraints are initialized with a feasible basis
  $(\mathscr{B},\ O',\ \{E_1, \ldots, E_q\}) \leftarrow$ `initilization`$(O,\ \{C_1, \ldots, C_q\})$
  return `explore` $(R,\ \mathscr{B},\ O',\ \{E_1, \ldots, E_q\})$

Each of these optima is associated to a region – the conjunction of the constraints along the branch – that defines a polyhedral region of the parameter space.

Application to Handelman's linearization Back to our running example, we obtain the best polyhedral approximations of g by running our parametric simplex on H-PLOP where the decision variables are the λ_i, the parametric coefficients are the $H_i(x_1, \ldots, x_n)$ and the parameters are the x_i. Moreover, we use the polyhedron P as initial region meaning that we only consider $(x_1, \ldots, x_n) \in P$.

We obtain a decision tree with optimal affine forms at the leaves, interpreted as constraints $\mathit{aff}_g(x_1, \ldots, x_n) \geq 0$. These constraints appear on Figure 4(a) as the lines L_1 to L_5. Their conjunction with P forms the polyhedron P' which over-approximates $P \cap (g \geq 0)$. The figure points out two weaknesses of our linearization: first, Figure 4(a) shows that L_3 and L_4 are useless since they do not intersect P'. This is not due to the parametric simplex: it happens when a constraint aff_g does not intersect the plane $z = 0$ on its region of relevance R. We try to exploit this remark for an early detection of useless contraints. Second, the spider web of Figure 4(b) illustrates the fact that our parametric simplex generates many sub-regions leading to the same constraint L_i and are finally merged to form R_i. For instance, the solution z^* of PLOP 1 associates the same constraint at different leaves. We are working on improving our heuristics for the pivot selection in our parametric simplex.



(a) (b)

Figure 4: (a) The polyhedron $P' = P \sqcap \{L_1 \geq 0, \ldots, L_5 \geq 0\}$ is the over-approximation of $P \cap (g \geq 0)$ computed by our linearization. It is delimited by the constraints L_1 to L_5 returned by the parametric simplex. The constraints L_3 and L_4 are clearly redundant. (b) Each constraint L_i is the optimum associated to a region R_i of P. The borders of a region are the dotted lines. The regions contain subdivisions which correspond to pivots in the simplex algorithm that finally lead to the same optimum.

5. Certification in Coq

The linearization process is divided in two parts: an Ocaml oracle uses heuristics to select the most promising Handelman products $H^{\boldsymbol{I_1}}, \ldots, H^{\boldsymbol{I_q}}$, then it runs the parametric simplex to find the coefficients $\lambda_{\boldsymbol{I_1}}, \ldots, \lambda_{\boldsymbol{I_q}}$ such that $g + \sum \lambda_{\boldsymbol{I_i}} H^{\boldsymbol{I_i}}$ is affine. It feeds a Coq-certified checker

$$\mathsf{affine}\ (P : \text{VPL.polyhedra})\ (g : \text{VPL.}\mathbb{Q}[\boldsymbol{x}])\ (\mathit{aff}_g : \text{VPL.}\mathbb{Q}[\boldsymbol{x}])\ (w : \mathsf{witness})\ :\ \mathsf{option}\ \text{VPL.}\mathit{Aff}(\mathbb{Q}, \boldsymbol{x})$$

with a certificate formed of the polyhedron P, the polynomial g, our candidate aff_g and the witness provided by the parametric simplex which corresponds to Handelman coefficients $[(\lambda_{\boldsymbol{I_i}}, \boldsymbol{I_i}) \mid 1 \leq i \leq q,\ \lambda_{\boldsymbol{I_i}} > 0]$. In Example 4, the certificate of the affine approximation $\mathit{aff}_g = 9 - 2x + 4y = g + H^{(2,0,0,0)} + H^{(0,2,0,0)}$ is $[(1, (2,0,0,0)),\ (1, (0,2,0,0))]$. The affine checker returns $\text{SOME}(\mathit{aff})$ if the

certificate is correct and NONE otherwise. It does not only guarantee that $\mathit{aff} = g + \sum \lambda_{\boldsymbol{I_i}} H^{\boldsymbol{I_i}}$, it also checks that *aff* is affine and converts it from polynomial to an affine term of the VPL. Finally it proves the implication $P \wedge g \geq 0 \Longrightarrow P \wedge \mathit{aff}_g \geq 0$ which guarantee that the polyhedron $P \wedge \mathit{aff}_g \geq 0$ is an over-approximation of the reachable states of the program after the polynomial guard $g \geq 0$.

The checker handles two different types of polynomials: VPL.$\mathbb{Q}[\boldsymbol{x}]$ is the type of general expressions within the VPL. It provides very few operators on non-linear expressions. Thus, for computations, we switch to the type RING.$\mathbb{Q}[\boldsymbol{x}]$ of a COQ library coming with the `ring` tactic. We denote by $\overset{ring}{=}$ the polynomial equality provided by this library. We first proved the semantic preservation of the translation of polynomials from VPL.$\mathbb{Q}[\boldsymbol{x}]$ to RING.$\mathbb{Q}[\boldsymbol{x}]$. It is used to certify the correctness of the checker, specified by: $\forall P\ g\ \mathit{aff}_g\ w$,

$$P = \bigwedge_{1 \leq i \leq q} C_i \geq 0 \Longrightarrow (\mathsf{affine}\ P\ g\ \mathit{aff}_g\ w) = \text{SOME}(\mathit{aff}) \Longrightarrow P \wedge (g \geq 0) \Longrightarrow \textit{properties (1) to (4)}$$

The implications introduce the semantics of P, *i.e.* $\bigwedge_{1 \leq i \leq q} C_i \geq 0$ and $g \geq 0$ as hypothesis, then the proof is divided in four steps:

(1) $g + \sum \lambda_{\boldsymbol{I_i}} H^{\boldsymbol{I_i}} \geq 0$ on P. We use some auxiliary functions that use g and the witness $w = [(\lambda_{\boldsymbol{I_i}}, \boldsymbol{I_i}) \mid 1 \leq i \leq q,\ \lambda_{\boldsymbol{I_i}} > 0]$ to build a representation of this polynomial in RING.$\mathbb{Q}[\boldsymbol{x}]$. Exploiting the positivity of g, every C_i and $\lambda_{\boldsymbol{I_i}}$, these functions are proved to construct positive polynomials on P as a positive combination of product of positive polynomials. We paid attention to get an efficient computation of the $H^{\boldsymbol{I_i}}$: we use memoization to store the previously computed products of C_i. The optimal order to compute these products is provided (as part of the witness) by an external Ocaml oracle.

(2) $\mathit{aff}_g \overset{ring}{=} g + \sum \lambda_{\boldsymbol{I_i}} H^{\boldsymbol{I_i}}$ is checked by the equality of RING.$\mathbb{Q}[\boldsymbol{x}]$.

(3) The property $\mathit{aff}_g \geq 0$ which guarantees the soundness of our linearization is direct from (1,2).

(4) The fact that the term aff_g of the certificate is affine is proven by application of the VPL function $\mathsf{split} : \text{VPL.}\mathbb{Q}[\boldsymbol{x}] \rightarrow \text{VPL.}\mathit{Aff}(\mathbb{Q}, \boldsymbol{x}) \times \text{VPL.}\mathbb{Q}[\boldsymbol{x}]$ to aff_g. The function split is proven to extract the affine part *aff* and the reminder part $Q(\boldsymbol{x})$ of an expression, *i.e.* $\mathit{aff}_g \overset{vpl}{=} \mathit{aff} + Q(\boldsymbol{x})$. Then, the reminder is translated into a polynomial of type RING.$\mathbb{Q}[\boldsymbol{x}]$ to check that $Q(\boldsymbol{x}) \overset{ring}{=} 0$.

If the construction (1) and the verifications (2) and (4) succeed, the checker returns the affine term SOME(*aff*) directly usable by the VPL as a polyhedral constraint.

6. Implementation and experiments

We developed the parametric simplex algorithm in Ocaml as part of the Verimag Verified Polyhedra Library. In order to get comparison elements, we also developed in SAGE [12] prototypes for the other linearization techniques (intervalization and Bernstein's approximation), and a simple analyzer that is able to handle C programs containing guards, assignments and if-then-else but no function call nor loop. Given a starting polyhedron P and a list of statements, the analyzer computes its effect on P with the three techniques.

Shape of real-life non linear expressions We tested the linearization techniques on statements taken from the benchmarks *debie1*, based on a satellite control software, and *papabench* which is a flight control code. In general, polynomials of such programs contain less than four variables and their power rarely exceed two. Indeed, most non-linear expressions appear in computation of Euclidian distances, that's why we encounter square roots as well. The guiding example $x^2 + y^2 \leq 4$ is an example taken from this code.

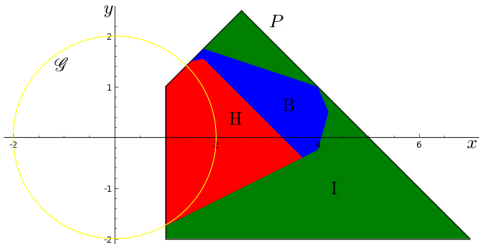


Figure 5: Representation of the effect of the guard $\mathscr{G} = \{(x,y) \mid x^2 + y^2 \leq 4\}$ *on the polyhedron* $P = \{(x,y) \mid x-1 \geq 0,\ y+2 \geq 0,\ x-y \geq 0,\ -x-y+5 \geq 0\}$. *The surfaces* I, B *and* H *are respectively the result of the linearization with intervalization, Bernstein and Handelman.*

Comparison As expected, intervalization is the fastest but the less accurate of the three methods. Bernstein and Handelman's method can be as accurate as needed, but at the price of an high algorithmic cost. We show on Figure 5 the results of the three methods to approximate the guard $\{(x,y) \mid x^2+y^2 \leq 4\}$ on $P \triangleq \{(x,y) \mid x-1 \geq 0,\ y+2 \geq 0,\ x-y \geq 0,\ -x-y+5 \geq 0\}$. We can see that intervalization is not precise enough to approximate the guard. Indeed, the resulting polyhedron is the same as the initial one, and the guard does not add any information in the analysis. We compute Bernstein's method without any interval splitting or degree elevation. Even without any refinement process, Bernstein is more accurate than intervalization but slower. Handelman's polyhedron is the most precise of the three techniques in this example. We chose as subset $\{H_1, \ldots, H_q\}$ the 15 possible products of constraints of P of degree ≤ 2, meaning that we are faced with a 15-variables LP. Industrial linear solvers are able to deal with hundreds of variables, but this is obviously the shortfall of Handelman's linearization.

The Bernstein approximation requires knowledge of each variable range. Extracting the interval of a variable from a polyhedron represented as a set of constraints in the VPL requires to solve two linear optimization problems to get the lower and upper bound of the variable. This overhead is avoided in Handelman's method which reasons directly on the constraints representation of the polyhedron. Hence, Bernstein's method is convenient when the polyhedron is an hypercube – that is the product of the interval of each variable – whereas the Handelman's method is promising with a general polyhedron. Specifically, we think that Handelman's method can be more suitable in terms of precision, even in complexity, in case of successive linearizations. Indeed, where the Bernstein's method stacks approximation errors at each new linearization due to conversion into intervals, the Handelman one does not degrade.

In practice, the three linearization methods can be combined: analysis is an iterative process that switches to finer methods when the cheapest ones failed to prove correctness of the program. We can imagine starting an analysis with intervals which are cheap and cope with non-linear expressions. Then, switching to the domain of polyhedra if more precision is required. This second phase can reuse the intervals computed by the first one and apply intervalization or Bernstein's linearization without paying the overhead of extracting intervals. This time the analysis associates polyhedra to program points. Then, to gain more precision, a third phase can run the analysis with Handelman's linearization. Other combinations are possible and Handelman can be directly used at any phase since a product of bounded intervals is a polyhedra.

7. Future work

Along the document we identified several points that still need work. We review them quickly and sketch directions of improvement: we presented a simplex algorithm with parametric objective. Figure 4 revealed that its exploration scheme is quite naive. We are working on a new exploration

scheme that should reduce region splitting. Another topics is the choice of of the subset $\{H_1, \ldots, H_q\}$: On the one hand, considering a lot of H_i allows lots of Handelman's representations, therefore an improved accuracy. On the other hand, each new H_i adds a variable λ_i in the simplex. In order to minimize the number of H_i to consider, starting with a small subset $\{H_1, \ldots, H_q\}$, we could imagine an incremental approach that adds new H_i when no solution is found. We must pay attention to the algorithm in order to exploit the computations of the previous attempt. Then, we will be able to perform large-scale experiments.

Acknowledgements

The authors thank David Monniaux for having pointed out Handelman-Krivine's theorem as a way to linearize polynomials and Alexis Fouilhé and Sylvain Boulmé for their help during development of the parametric simplex, the certified checker and the linking with the COQ part of the Verimag Verified Polyhedra Library.

References

[1] L. Chen, A. Miné, J. Wang, and P. Cousot. Interval polyhedra: An abstract domain to infer interval linear relationships. *the 16th International Static Analysis Symposium*, 5673:309–325, December 2009.

[2] V. Chvatal. *Linear Programming*. Series of books in the mathematical sciences. W. H. Freeman, 1983.

[3] P. Feautrier. Parametric integer programming. *RAIRO Recherche opérationnelle*, 22(3):243–268, 1988.

[4] A. Fouilhé, D. Monniaux, and M. Périn. Efficient generation of correctness certificates for the abstract domain of polyhedra. In *Static analysis (SAS)*, 2013.

[5] D. Handelman. Representing polynomials by positive linear functions on compact convex polyhedra. *Pac. J. Math*, 132(1):35–62, 1988.

[6] B. Jeannet and A. Miné. APRON: A library of numerical abstract domains for static analysis. In *Computer Aided Verification, CAV'2009*, volume 5643 of *LNCS*, pages 661–667, 2009.

[7] J.-H. Jourdan, V. Laporte, S. Blazy, X. Leroy, and D. Pichardie. Formal verification of a C static analyzer. In *POPL (Principles Of Programming Languages)*, 2015. To appear.

[8] X. Leroy and S. Blazy. Formal verification of a C-like memory model and its uses for verifying program transformations. *Journal of Automated Reasoning*, 41(1):1–31, 2008.

[9] Markus Schweighofer. An algorithmic approach to Schmüdgen's positivstellensatz. *Elsevier Preprint*, June 2001.

[10] A. Miné. Symbolic methods to enhance the precision of numerical abtract domains. *the 7th International Conference on Verification, Model Checking and Abstract Interpretation*, 3855:348–363, January 2006.

[11] C. Muñoz and A. Narkawicz. Formalization of a representation of Bernstein polynomials and applications to global optimization. *Journal of Automated Reasoning*, 51(2):151–196, August 2013.

[12] W. Stein et al. *Sage Mathematics Software (Version 6.2)*. The Sage Development Team, 2014. `http://www.sagemath.org`.