# Numerical Abstract Domain using Support Function

Yassamine Seladji[⋆] and Olivier Bouissou[⋆⋆]

CEA Saclay Nano-INNOV Institut CARNOT
91 191 Gif sur Yvette CEDEX, France

**Abstract.** In this paper, we present a new abstract domain that uses support functions to represent convex sets. Then, using a predefined set of directions, we can use efficient method to compute the fixpoint of linear programs. We show on a simple example the efficiency of our method.

## 1 Introduction

Abstract interpretation based static analysis [1, 2] of numerical programs aims at computing invariants of the values that variables can take during any execution of the program. Most numerical domains can efficiently represent convex sets using a polyhedral representation [3, 7, 4, 8]. Another popular representation of convex sets is the use of *support functions* [6]. In particular, support functions have been successfully used in the field of hybrid systems analysis [5] to represent sets of values. In this article, we present some ongoing work that shows that it is possible to use support functions to define a new abstract domain that is simpler than the polyhedra domain and allows for efficient computation of linear operations. This domain is similar to the template abstract domain [8] in that it depends on a fixed set of directions $\Delta \subseteq \mathbb{R}^n$ (where $n$ is the space dimension) and bounds the convex sets in each direction $d \in \Delta$. However, as it benefits from the algorithms on support functions, the linear operations are very efficient and do not depend on linear programming solvers.

In the following, we first give some basics results on support functions, and then present our domain and how to compute on it. We consider that we handle sets in $\mathbb{R}^n$ (where $n$ is the number of variables of our program). Our domain depends on a set of directions $\Delta \subseteq \mathbb{R}^n$ which we assume to be finite.

*Notations* We put $\mathbb{R}_\infty = \mathbb{R} \cup \{-\infty, +\infty\}$. Given two vectors $v, w \in \mathbb{R}^n$, let $\langle v, w\rangle \in \mathbb{R}$ be the scalar product of $u$ and $w$

## 2 Support function

In this section, we recall the definition of the support function and some useful properties. In convex analysis, support function is commonly used as a repre-

[⋆] yassamine.seladji@cea.fr
[⋆⋆] olivier.bouissou@cea.fr

sentation of convex sets $S \subseteq \mathbb{R}^n$. The support function $\delta_S$ of $S$ is defined by

$$\delta_S(d) = sup\{\langle x, d\rangle : x \in S\}, \forall d \in \mathbb{R}^n.$$

Using this definition, $S$ can be represented by intersection of half spaces as given in Equation 1.

$$S = \bigcap_{d \in \mathbb{R}^n} \{x \in \mathbb{R}^n | \langle x, d\rangle \leq \delta_S(d)\}. \tag{1}$$

From the computational point of view, it's hard to use Equation 1 as an exact representation of $S$. To deal with that, we can define a polyhedron that over-approximates $S$ as given in Proposition 1.

**Property 1** *Let $S$ be a convex set, and $\Delta \subseteq \mathbb{R}^n$ be a set of directions. We put $P = \bigcap_{d\in\Delta}\{x \in \mathbb{R}^n | \langle x, d\rangle \leq \delta_S(d)\}$. Then $S \subseteq P$.*

Support function of an arbitrary convex set can be computed efficiently using operations of Property 2. For a given matrix $M \in \mathbb{R}^n \times \mathbb{R}^m$ (where $m \in \mathbb{N}$ and $n$ is the space dimension), $MS$ denotes the transformation of the convex set $S$ by $M$, such that $MS = \{Mx | x \in S\}$. Let $S, S' \subseteq \mathbb{R}^n$, $S \oplus S'$ denotes the Minkowski sum: $S \oplus S' = \{x + x' \mid x \in S,\ x' \in S'\}$.

**Property 2** *Let $S$ be a convex set. We have:*

- $\forall M \in \mathbb{R}^n \times \mathbb{R}^m, \delta_{MS}(d) = \delta_S(M^T d)$.
- $\forall \lambda \geq 0, \delta_{\lambda S}(d) = \lambda \delta_S(d)$.
- $\forall S' \subseteq \mathbb{R}^n, \delta_{S \cup S'}(d) = \max(\delta_S(d), \delta_S(d))$.
- $\forall S' \subseteq \mathbb{R}^n, \delta_{S \cap S'}(d) \leq \min(\delta_S(d), \delta_S(d))$.
- $\forall S' \subseteq \mathbb{R}^n, \delta_{S \oplus S'}(d) = \delta_S(d) + \delta_S(d)$.

In Property 2, $S \cup S'$ is the convex hull of $S$ and $S'$. Note that, all relations are exact, except for the computation of the support function of the intersection of two convex sets for which we only have an over-approximation relation.

In the next section, we shall see how the properties presented in this section allow us to define the Restricted Polyhedra Abstract Domain.

## 3 Abstract Domain

In this section, we formalize our abstract domain using support function. For that we fix the set $\Delta = \{d_1, \ldots, d_l\}$ as a set of $l$ directions. We define the domain $\mathbb{P}^\sharp_\Delta$ in Definition 1.

**Definition 1** *Let $\Delta \subseteq \mathbb{R}^n$ be a set of $l$ directions. Let $\mathbb{P}^\sharp_\Delta$ be our abstract domain such that: $\mathbb{P}^\sharp_\Delta = \Delta \to \mathbb{R}_\infty$, the set of functions from $\Delta$ to $\mathbb{R}_\infty$.*

For each $\Omega \in \mathbb{P}^\sharp_\Delta$, we write $\Omega(d)$ the value of $\Omega$ in direction $d \in \Delta$. So an abstract value of $\mathbb{P}^\sharp_\Delta$ is a mapping from $\Delta$ to $\mathbb{R}_\infty$, such that the concretization function $\gamma_\Delta$ is given in Definition 2

**Definition 2** *Let $\Delta \subseteq \mathbb{R}^n$ be a set of $l$ directions, and $\mathbb{P}^\sharp_\Delta$ be our abstract domain. The concretization function of $\mathbb{P}^\sharp_\Delta$ is noted $\gamma_\Delta$, such that:*

$$\forall \Omega \in \mathbb{P}^\sharp_\Delta, \gamma_\Delta(\Omega) = \bigcap_{d \in \Delta} \{x \in \mathbb{R}^n \mid \langle x, d \rangle \leq \Omega(d)\}$$

Definition 2 shows that the concretization of an abstract element of $\mathbb{P}^\sharp_\Delta$ is a polyhedron. This polyhedron is represented by the intersection of $l$ half-spaces, where each one is characterized by its normal vector $d \in \Delta$ and a coefficient $\Omega(d)c$. We define the inclusion operation $\sqsubseteq$ as follow:

$$(\forall \Omega_1, \Omega_2 \in \mathbb{P}^\sharp_\Delta), \Omega_1 \sqsubseteq \Omega_2 \iff \gamma_\Delta(\Omega_1) \subseteq \gamma_\Delta(\Omega_2),$$

where $\subseteq$ is the inclusion defined for the Polyhedra Abstract Domain. Using this definition, we have that $(\mathbb{P}^\sharp_\Delta, \sqsubseteq)$ forms a complete lattice, such that:

- $\bot = \lambda d. -\infty$.
- $\top = \lambda d. +\infty$.
- $(\forall \Omega_1, \Omega_2 \in \mathbb{P}^\sharp_\Delta), (\Omega_1 \sqcup \Omega_2)(d) = \max(\Omega_1(d), \Omega_2(d))$.
- $(\forall \Omega_1, \Omega_2 \in \mathbb{P}^\sharp_\Delta), (\Omega_1 \sqcap \Omega_2)(d) \sqsubseteq \min(\Omega_1(d), \Omega_2(d))$.

Note that in our lattice, the operation of intersection is not exact because we are not sure that $\Omega_1$ and $\Omega_2$ are in a normal form. The abstraction function $\alpha_\Delta$ of $\mathbb{P}^\sharp_\Delta$ is a function which transforms a given polyhedron into a set of coefficients. For that we use the support function.

**Definition 3** *Let $\Delta \subseteq \mathbb{R}^n$ be a set of $l$ directions, and $\mathbb{P}^\sharp_\Delta$ be our abstract domain. The abstraction function of $\mathbb{P}^\sharp_\Delta$ is defined by*

$$\alpha_\Delta(\mathbb{P}) = \begin{cases} \bot & \textit{if } \mathbb{P} = \emptyset \\ \top & \textit{if } \mathbb{P} = \mathbb{R}^n \\ \lambda d. \delta_{\mathbb{P}}(d) & \textit{otherwise} \end{cases}$$

For a given polyhedron $\mathbb{P}$, the result of $\alpha_\Delta$ is the support function of $\mathbb{P}$ applied on each element of $\Delta$. In the case where $\mathbb{P}$ is unbounded in a direction $d$, we put $\delta_{\mathbb{P}}(d) = +\infty$. The support function of $\mathbb{P}$, for each $d \in \Delta$, can be obtained using two methods:

- if $\mathbb{P}$ is represented by a linear system, we use linear programming to find $\sup\langle d, x \rangle$ under the linear system of $\mathbb{P}$,
- if $\mathbb{P}$ is represented by a set of generators, $\delta_{\mathbb{P}}(d) = \max\{\langle g, d \rangle \mid g \in \mathbb{P}\}$.

Note that $\forall \Omega \in \mathbb{P}^\sharp_\Delta$, if $\Omega = \alpha_\Delta(\mathbb{P})$, then $\gamma_\Delta(\Omega) \subseteq \mathbb{P}$, and the generators of the polyhedron $\mathbb{P}$ touch the faces of $\gamma_\Delta(\Omega)$. This is stated in Proposition 3.

**Property 3** *Let $\mathbb{P}$ be a polyhedron and $\Omega \in \mathbb{P}^\sharp_\Delta$ such that $\Omega = \alpha_\Delta(\mathbb{P})$. We have that, $\mathbb{P} \subseteq \gamma_\Delta(\Omega)$ where this over approximation is tight as the generators of $\mathbb{P}$ touch the faces of $\gamma_\Delta(\Omega)$.*

**Example 1** *In this example, we put $\Delta \subseteq \mathbb{R}^2$ such that:*
$\Delta = \{(-3,5),(1,3),(-1,0),(0,-1)\}$. *For the abstract element $\Omega_1 = \{3,4,3,2\}$, the result of its concretization is given in Figure 1(left). The right of the Figure 1 is the result of $\gamma_\Delta(\Omega_2)$, with $\Omega_2 = \{3,3,+\infty,2\}$. In this case, for $c_2 = +\infty$ in $\Omega_2$, the resulted polyhedron is unbounded in the direction $d_2 = (-1,0)$.*

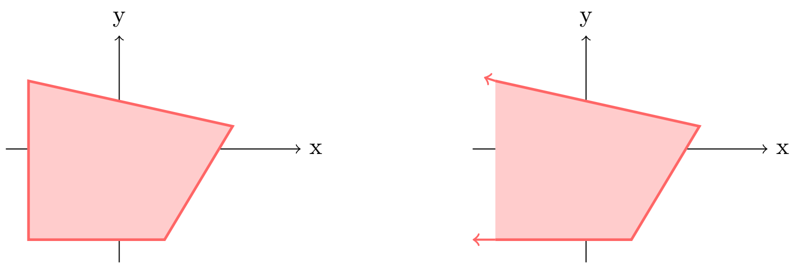

Fig. 1: The geometrical representation of $\gamma_\Delta(\Omega_1)$ (left) and $\gamma_\Delta(\Omega_2)$ (right).

## 4 Fixpoint computation

In this section, we show how to manipulate abstract elements of Section 3 to analyze numerical programs. For that, we define techniques that allow us to use support functions to compute the fixpoint of programs to analyze. We show that these results have a good accuracy. We assume that all programs we analyze are of the form $X' = AX + b$, where $A \in \mathbb{R}^n \times \mathbb{R}^n$ and $b \in \mathbb{R}^n$. Let $\mathbb{P}_0$ be the polyhedron that represents the initial condition of the program, and $F^\sharp$ its semantic function. We put $\Omega_0 = \alpha_\Delta(\mathbb{P}_0)$. For the simple case of programs without loops, the fixpoint, noted $\Omega$, is obtained by computing the support function of the polyhedron $\mathbb{P}$ in directions $d \in \Delta$, where $\mathbb{P} = F^\sharp(\mathbb{P}_0)$. We have that $F^\sharp(\mathbb{P}_0) = A\mathbb{P}_0 \oplus b$. Using Property 2 we have:

$$\begin{aligned}\forall d \in \Delta, \delta_{\mathbb{P}}(d) &= \delta_{F^\sharp(\mathbb{P}_0)}(d)\\ &= \delta_{A\mathbb{P}_0 \oplus b}(d)\\ &= \delta_{A\mathbb{P}_0}(d) + \delta_b(d)\\ &= \delta_{\mathbb{P}_0}(A^T d) + \langle b, d\rangle\ .\end{aligned}$$

So we define $\Omega$ as

$$\forall d \in \Delta,\ \Omega(d) = \delta_{\mathbb{P}_0}(A^T d) + \langle b, d\rangle .$$

Note that $\Omega = \alpha_\Delta(\mathbb{P})$, while $\mathbb{P} \subseteq \gamma_\Delta(\Omega)$. Proposition 3 ensures that the generators of $\mathbb{P}$ touch the faces of $\gamma_\Delta(\Omega)$. The precision of $\gamma_\Delta(\Omega)$ depends strongly on the choice of $\Delta$. We assume that $\mathbb{P}_0$ is bounded, so we represent it by its generators, such that: $\mathbb{P}_0 = \{g_1, \ldots, g_j\}$. We can define $\delta_{\mathbb{P}_0}(d)$ for all $d \in \Delta$ by $\delta_{\mathbb{P}_0}(d) = sup\{\langle d, v_i\rangle | v_i \in \mathbb{P}_0\}$. In this case, the computation of $\delta_{\mathbb{P}_0}(d)$ is done without using linear programming. So, this technique is efficient.

Now, we extends our class of programs to programs with loops. For that, we distinguish two kinds of loops: loops with guard and loops without guard.

*Loops without guard.* Here, we consider the case of programs with loops, where the loop is not conditioned by a guard. To compute the abstract element $\Omega_i$ of each iteration, we use support function to ensure that $\Omega_i = \alpha_\Delta(\mathbb{P}_i)$, where $\mathbb{P}_i$ is the result of the $i^{th}$ Kleene iteration using the Polyhedra Abstract Domain. This means that $\Omega_i$ is the best abstraction, in $\mathbb{P}^\sharp_\Delta$, for $\mathbb{P}_i$. The computation of $\Omega_i$ uses the support function of $\mathbb{P}_0$ as given in Equation 2.

$$\forall d \in \Delta,\ \Omega_i(d) = \max\Big(\delta_{\mathbb{P}_0}(d), \delta_{\mathbb{P}_0}(A^{Tj}d) + \sum_{k=1}^{j}\langle b, A^{T(k-1)}d\rangle, j = 1, \dots, i\Big) \quad (2)$$

Property 4 shows that for all $d \in \Delta, \Omega_i(d) = \delta_{\mathbb{P}_i}(d)$. Thus, we have that $\Omega_i = \alpha_\Delta(\mathbb{P}_i)$. So $\Omega_i$ is the best abstraction of $\mathbb{P}_i$ in $\mathbb{P}^\sharp_\Delta$.

**Property 4** *Let $\mathbb{P}_i$ be the $i^{th}$ iteration in the polyhedra abstract domain, then*

$$\delta_{\mathbb{P}_i}(d) = \max\Big(\delta_{\mathbb{P}_0}(d), \delta_{\mathbb{P}_0}(A^{Tj}d) + \sum_{k=1}^{j}\langle b, A^{T(k-1)}d\rangle, j = 1, \dots, i\Big).$$

Proof. We show that the Property 4 is true for $i = 1$.

$$\begin{aligned}(\forall d \in \Delta), \delta_{\mathbb{P}_1}(d) &= \delta_{\mathbb{P}_0 \cup F^\sharp(\mathbb{P}_0)}(d)\\ &= \max(\delta_{\mathbb{P}_0}(d), \delta_{A\mathbb{P}_0 \oplus b}(d))\\ &= \max(\delta_{\mathbb{P}_0}(d), \delta_{A\mathbb{P}_0}(d) + \delta_b(d))\\ &= \max(\delta_{\mathbb{P}_0}(d), \delta_{\mathbb{P}_0}(A^T d) + \langle b, d\rangle)\ .\end{aligned}$$

We assume that the Property 4 is true for rank $i$, and we prove that it's true for rank $i + 1$.

$$\begin{aligned}\delta_{\mathbb{P}_{i+1}}(d) &= \delta_{\mathbb{P}_i \cup F^\sharp(\mathbb{P}_i)}(d)\\ &= \max(\delta_{\mathbb{P}_i}(d), \delta_{A\mathbb{P}_i \oplus b}(d))\\ &= \max(\delta_{\mathbb{P}_i}(d), \delta_{\mathbb{P}_i}(A^T d) + \langle b, d\rangle)\end{aligned}$$

We have that :

$$\begin{aligned}\delta_{\mathbb{P}_i}(A^T d) + \langle b, d\rangle &= \max(\delta_{\mathbb{P}_0}(A^T d) + \langle b, d\rangle, \dots, \delta_{\mathbb{P}_0}(A^{Ti}d) + \textstyle\sum_{k=1}^{i-1}\langle b, A^{Tk}d\rangle + \langle b, d\rangle,\\ &\quad \delta_{\mathbb{P}_0}(A^{T(i+1)}d) + \textstyle\sum_{k=1}^{i}\langle b, A^{Tk}d\rangle + \langle b, d\rangle)\\ &= \max(\delta_{\mathbb{P}_0}(A^T d) + \langle b, d\rangle, \dots, \delta_{\mathbb{P}_0}(A^{Ti}d) + \textstyle\sum_{k=1}^{i}\langle b, A^{T(k-1)}d\rangle,\\ &\quad \delta_{\mathbb{P}_0}(A^{T(i+1)}d) + \textstyle\sum_{k=1}^{i+1}\langle b, A^{T(k-1)}d\rangle)\end{aligned}$$

Note that,

$\max(\delta_{\mathbb{P}_0}(A^T d) + \langle b, d\rangle, \dots, \delta_{\mathbb{P}_0}(A^{Ti}d) + \sum_{k=1}^{i}\langle b, A^{T(k-1)}d\rangle) \leq \delta_{\mathbb{P}_i}(d)$.

So

$$\begin{aligned}\delta_{\mathbb{P}_{i+1}}(d) &= \max(\delta_{\mathbb{P}_i}(d), \delta_{\mathbb{P}_0}(A^{T(i+1)}d) + \textstyle\sum_{k=1}^{i+1}\langle b, A^{T(k-1)}d\rangle)\\ &= \max(\delta_{\mathbb{P}_0}(d), \delta_{\mathbb{P}_0}(A^{Tj}d) + \textstyle\sum_{k=1}^{j}\langle b, A^{T(k-1)}d\rangle, j = 1, \dots, i+1).\end{aligned}$$

Thus,$(\forall d \in \Delta, \forall i \geq 1)$,

$$\delta_{\mathbb{P}_i}(d) = \max(\delta_{\mathbb{P}_0}(d), \delta_{\mathbb{P}_0}(A^{Tj}d) + \sum_{k=1}^{j}\langle b, A^{T(k-1)}d\rangle, j = 1, \dots, i).$$

$\square$

**Algorithm 1** Kleene Algorithm using support function.

```
Input: $\Delta \subset \mathbb{R}^n$, set of $l$ directions
Input: $\mathbb{P}_0$, The initial polyhedron
Input: $A \in \mathbb{R}^n \times \mathbb{R}^m, b \in \mathbb{R}^m$
 1: $d = \Delta$
 2: $\Omega = \bot$
 3: repeat
 4:   $\Omega' = \Omega$
 5:   for all $i = 0, \dots, (l-1)$ do
 6:     $\Theta[i] = \Theta[i] + \langle b, d[i] \rangle$
 7:     $d[i] = A^T d[i]$
 8:     $\Upsilon[i] = \delta_{\mathbb{P}_0}(d[i]) + \Theta[i]$
 9:     $\Omega[i] = \max(\Omega[i], \Upsilon[i])$
10:   end for
11: until $\Omega \sqsubseteq \Omega'$
```

Using Equation 2, we define an efficient algorithm to compute the fixpoint. In Algorithm 1, the computation of the abstract element $\Omega_i$ depends on the computation of $\delta_{\mathbb{P}_0}$ and $\Theta_i^k$ for each $d'_k$. We assume that $\mathbb{P}_0$ is a bounded polyhedron, which is represented by its generators. To compute $\delta_{\mathbb{P}_0}$ in each direction, we put $\mathbb{P}_0 = \{v_1, \dots, v_j\}$. So, we have that $\delta_{\mathbb{P}_0}(d) = sup\{\langle d, v_i \rangle : v_i \in \mathbb{P}_0\}$. Indeed, we know that $\mathbb{P}_0$ represents the polyhedron of the initial condition of the program to analyze, so its representation is quite simple. In particular, the number of its vertices is usually small. This means that the computation of $\delta_{\mathbb{P}_0}$ does not require LP solvers, what changes in each iteration is the set of directions in which we compute $\delta_{\mathbb{P}_0}$. Thus, Algorithm 1 has a polynomial complexity, and in addition its results is as accurate as possible.

*Loops with guard :* Generally in programs with loops, the loops have a condition. To treat this case, we assume that the condition of the loop is represented by a guard of the form $\langle X, c \rangle \leq l$, such that $c \in \mathbb{R}^n$ and $l \in \mathbb{R}$. Let $H$ be the half space, such that: $H = \{x \in \mathbb{R}^n | \langle x, c \rangle \leq l\}$. In this case, the abstract semantic function is defined by: $F^\sharp(\mathbb{P}) = (A\mathbb{P} \oplus b) \cap H$, with $\mathbb{P}$ a polyhedron. We put $\mathbb{P}' = \mathbb{P}_0 \cup F^\sharp(\mathbb{P}_0)$. So $(\forall d \in \Delta), \delta_{\mathbb{P}'}(d) = \max(\delta_{\mathbb{P}_0}(d), \delta_{F^\sharp(\mathbb{P}_0)}(d))$. We have that:

$$\begin{aligned}\delta_{F^\sharp(\mathbb{P}_0)}(d) &= \delta_{(A\mathbb{P}_0 \oplus b) \cap H}(d) \\ &\leq \min(\delta_{\mathbb{P}_0}(A^T d) + \langle b, d \rangle, \delta_H(d))\end{aligned}$$

Thus, $\delta_{\mathbb{P}'}(d) \leq \max(\delta_{\mathbb{P}_0}(d), \min(\delta_{\mathbb{P}_0}(A^T d) + \langle b, d \rangle, \delta_H(d)))$. We can generalize this to $\delta_{\mathbb{P}_i}(d)$, such that $\mathbb{P}_i$ is the polyhedron obtained in the $i^{th}$ iteration. This generalization needs to distinguish two cases: the first case is where $d = \lambda c$ with $(\lambda \geq 0)$, and the second is where $d \neq \lambda c$. We separate these cases because:

$$\delta_H(d) = \begin{cases} l & \text{if } d = \lambda c, (\lambda \geq 0) \\ +\infty & \text{otherwise} \end{cases} .$$

In this case, for a better precision we add the normal vector of $H$ to $\Delta$. we put $\Delta_1 = \{c\} \cup \{d \in \Delta | d = \lambda c, \lambda \geq 0\}$, and $\Delta_2 = \Delta \setminus \Delta_1$. Note that, $\Delta$ is defined such that its elements are not two per two parallel. So, the cardinality of $\Delta_1$ does not exceed 2. For the fixpoint computation, we separate the case where $d \in \Delta_1$ from when $d \in \Delta_2$. Let us detail the computation of $\Omega_i(d)$ for both cases.

– For $d \in \Delta_2$, we have $\Omega_i(d)$ defined as in Equation 2. Let us show that. We have that, $\Omega_i(d) = \delta_{\mathbb{P}_i}(d)$ with:
$$\begin{aligned}\delta_{\mathbb{P}_i}(d) &= \delta_{\mathbb{P}_{i-1} \cup F^\sharp(\mathbb{P}_{i-1})}(d)\\ &= \max(\delta_{\mathbb{P}_{i-1}}(d), \delta_{(A\mathbb{P}_{i-1} \oplus b) \cap H}(d))\\ &\leq \max(\delta_{\mathbb{P}_{i-1}}(d), \min(\delta_{\mathbb{P}_{i-1}}(A^T d) + \langle b, d \rangle, \delta_H(d))\\ &= \max(\Delta_{\gamma_\Delta(\Omega_{i-1})}(d), \delta_{\gamma_\Delta(\Omega_{i-1})}(A^T d) + \langle b, d \rangle (\delta_H(d) = +\infty)\\ &= \max(\delta_{\mathbb{P}_0}(d), \delta_{\mathbb{P}_0}(A^{Tj} d) + \textstyle\sum_{k=1}^{j} \langle b, A^{T(k-1)} d \rangle, j = 1, \ldots, i).\end{aligned}$$
So, in this case, the effect of the guard doesn't appear.

– For $d \in \Delta_2$, we put $\Omega_i(d) = \max(\Delta_{\gamma_\Delta(\Omega_{i-1})}(d), \min(\delta_{\gamma_\Delta(\Omega_{i-1})}(A^T d) + \langle b, d \rangle, l)$, which is an over approximation of $\delta_{\mathbb{P}_i}(d)$, such that :
$$\begin{aligned}\delta_{\mathbb{P}_i}(d) &= \delta_{\mathbb{P}_{i-1} \cup F^\sharp(\mathbb{P}_{i-1})}(d)\\ &= \max(\delta_{\mathbb{P}_{i-1}}(d), \delta_{(A\mathbb{P}_{i-1} \oplus b) \cap H}(d))\\ &\leq \max(\delta_{\mathbb{P}_{i-1}}(d), \min(\delta_{\mathbb{P}_{i-1}}(A^T d) + \langle b, d \rangle, \delta_H(d))\\ &\leq \max(\Delta_{\gamma_\Delta(\Omega_{i-1})}(d), \min(\delta_{\gamma_\Delta(\Omega_{i-1})}(A^T d) + \langle b, d \rangle, l)\\ &\leq \Omega_i(d)\end{aligned}$$
To compute $\Omega_i(d)$, we have to determine $\delta_{\gamma_\Delta(\Omega_{i-1})}(d)$ and $\delta_{\gamma_\Delta(\Omega_{i-1})}(A^T d)$, which are obtained using linear programming. This choice doesn't affect a lot the efficiency of our method, because it's applied at most for two directions in $\Delta \cup \{c\}$. Here, $\alpha_\Delta(\mathbb{P}_i) \sqsubseteq \Omega_i$ such that the generators of $\mathbb{P}_i$ touch all the faces of $\gamma_\Delta(\Omega_i)$, except for those whose normal vector belongs to $\Delta_1$.

## 5 Experimentation

To show the efficiency of our abstract domain we did some preliminary test. We implemented Algorithm 1 using the R language, and we tested it on a simple program given in the left of Figure 2. The experimentation is done on $2.4GHz$ Intel Core2 Duo laptop, with $8Gb$ of RAM. The set of directions $\Delta$ is chosen in a random way. The result of the analysis is illustrated in the right part of Figure 2, which is obtained after only 0.046 second. This result is obtained using $\Delta = \{(9,2); (1,6); (-7,7); (8,7); (-3,-7); (3,2); (-8,7); (4,1)\}$ , where the filled polyhedrons represent the result obtained in each Kleene iteration using polyhedra abstract domain, in an increasing way from the dark to the light. And the result of Algorithm 1 is given by the transparent(dotted) polyhedron. The time execution of this analysis is of 0.046 second, which is a good trade of between precision and time execution.

## 6 Conclusion

We showed a new abstract domain that uses support functions to represent convex sets. Depending on the chosen set of directions, our domain $\mathbb{P}^\sharp_\Delta$ holds an over-approximation of the support functions of the set in each direction. Clearly,

```
begin
   x = -2.0;   y = 1.0;
   xn = yn= 0.0;
   while (0<=10) do
      xn = 0.5 *x - y - 2.5;
      yn = 0.9 *y + 10;
      x = xn; y = yn;
   done;
end
```

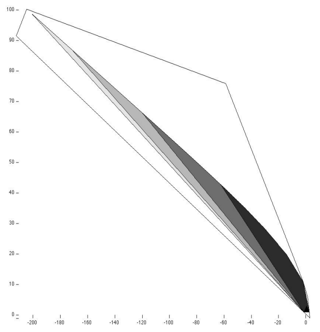

Fig. 2: The result of our experimentation. The program to analyze in left and the geometrical interpretation of the result in right.

both the definition and the order defined in our domain are the same as for the template abstract domain. However, the linear affectations are very different as we can always rely on the support function of the initial polyhedron which is easily computed. Using this technique, we showed that our domain is very precise: for a loop, the $i^{\text{th}}$ iterate is the best abstraction in $\mathbb{P}^{\sharp}_{\Delta}$ of the $i^{\text{th}}$ iterate one would have computed using the polyhedra domain.

As already stated, the precision of our domain is theoretically good and must now be tested on various numerical programs. We are currently working on an implementation of this domain to compare it with the results of [8].

## References

1. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM Press, 1977.
2. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.
3. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Conference Record of the Fifth ACM Symposium on Principles of Programming Languages (POPL'78)*, pages 84–97. ACM Press, 1978.
4. Eric Goubault and Sylvie Putot. Perturbed affine arithmetic for invariant computation in numerical program analysis. *CoRR*, abs/0807.2961, 2008.
5. Colas Le Guernic and Antoine Girard. Reachability analysis of hybrid systems using support functions. In *CAV*, pages 540–554, 2009.
6. Jean-Baptiste Hiriart-Urrut and Claude Lemaréchal. *Fundamentals of Convex Analysis.* Springer, 2004.
7. A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
8. S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI.* Springer, 2005.