Numerical Abstract Domain Using Support Functions

Yassamine Seladji and Olivier Bouissou

CEA Saclay Nano-INNOV Institut CARNOT
91 191 Gif sur Yvette CEDEX, France
{yassamine.seladji,olivier.bouissou}@cea.fr

Abstract. An abstract interpretation based static analyzer depends on the choice of both an abstract domain and a methodology to compute fixpoints of monotonic functions. Abstract domains are almost always representations of convex sets that must provide efficient algorithms to perform both numerical and order-theoretic computations. In this paper, we present a new abstract domain that uses support functions to represent convex sets. We define the order-theoretic operations and, using a predefined set of directions, we define an efficient method to compute the fixpoint of linear and non-linear programs. Experiments show the efficiency and precision of our methods.

1 Introduction

Almost all static analysers rely on a method to efficiently compute numerical invariants. This is particularly true for highly numerical programs like digital filters for which we are interested in computing a possibly tight over-approximation of the range of values the variables can take. The theory of abstract interpretation defines such invariants as the least fixpoint of a system of semantics equations operating on elements of some abstract domain. The quality of the invariant then depends on both the algorithm to compute the least fixpoint and the choice of the abstract domain to encode sets of values.

For this second point, most domains over-approximate the sets of variables values by convex sets, very often using a (sub) polyhedral representation [6,14,9,15]. More recently, new domains were proposed that allow to encode non-convex (even non-connected) sets [3,1] but these are convex sets in another space. So it is clear that the static analyser efficiency relies on a precise and efficient representation of convex sets, that allows for both numeric and order-theoretic transformations. In this paper, we define a new abstract domain based on the support function representation of a convex set and show that this domain allows to efficiently and precisely compute numerical invariants.

Support function is a popular representation of convex sets for numerical analysis [11]: a set S is represented as a function mapping each direction d with the distance between the origin and the supporting hyperplane of S in the direction d. Support functions offer a very compact and precise representation of

G. Brat, N. Rungta, and A. Venet (Eds.): NFM 2013, LNCS 7871, pp. 155–169, 2013.

convex sets and allow for an exact computation of affine transformation of sets (see Section 2). Support functions with finite supports were successfully used in the hybrid systems analysis [10] to represent value sets or in our previous work to speed up the convergence of the Kleene algorithm on general polyhedra [16].

In this article, we present a new abstract domain which is based on a sub-polyhedral representation of convex sets using support functions with finite supports. It allows for a compact representation of sets and we define efficient algorithms to compute the fixpoint of affine and non-linear loops. This domain is similar to the template domain [15] in that it depends on a fixed direction set $\Delta \subseteq \mathbb{R}^n$ (n: the space dimension) and bounds the convex sets in each Δ direction. However, as it benefits from the algorithms on support functions, linear operations are very efficient and do not depend on linear programming solvers.

This article is organized as follows. Section 2 gives some basic definitions and results on support functions. Section 3 formally defines our abstract domain, in particular the order-theoretic operations. Sections 4 and 5 show how to adapt Kleene algorithm to our domain for linear and non-linear loops, respectively. For the non-linear case, the notion of interval based support function is introduced which allows to compute both an over- and under-approximation of the least fixpoint. Section 6 concludes the article with some experimentation.

Notations. We put $\mathbb{R}_\infty = \mathbb{R} \cup \{-\infty, +\infty\}$ and $\mathbb{I}_\mathbb{R} = \{[a, b] | a \leq b : a, b \in \mathbb{R}_\infty\}$. 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. Let $\mathbb{B}^n$ be the unit sphere in $\mathbb{R}^n$.

2 Support Function

In this section, we give the definition of the support function of a convex set and give some usefull properties that show how a support function is modified by set transformations. Given a convex set $S \subseteq \mathbb{R}^n$, the support function of S, denoted δ_S, is a functional representation of S, as stated by Definition 1 and Property 1.

Definition 1 ([11, Def. C.2.1.1]). *Let $S \subseteq \mathbb{R}^n$ be a convex set. The support function δ_S is defined by:*

$$\delta_S : \begin{cases} \mathbb{B}^n \to \mathbb{R}_\infty \\ d \quad \mapsto \sup\{\langle x, d \rangle : x \in S\} \end{cases}$$

Property 1 ([11, Corollary. C.3.1.2]). *Let $S \subseteq \mathbb{R}^n$ be a convex set and let δ_S be its support function. Then, $S = \bigcap_{d \in \mathbb{B}^n} \left\{x \in \mathbb{R}^n \mid \langle x, d \rangle \leq \delta_S(d)\right\}$.*

Property 1 states that a convex set is uniquely determined by its support function. Stated differently, any positively homogeneous function $\delta : \mathbb{B}^n \to \mathbb{R}_\infty$ determines exactly one convex set defined as the intersection of all hyperplanes $\left\{x \in \mathbb{R}^n \mid \langle x, d \rangle \leq \delta(d)\right\}$ for all $d \in \mathbb{B}^n$. We recall that δ is positively homogeneous (of degree 1) if $\forall d \in \mathbb{B}^n, k \in \mathbb{R}, \delta(kd) \leq k\delta(d)$.

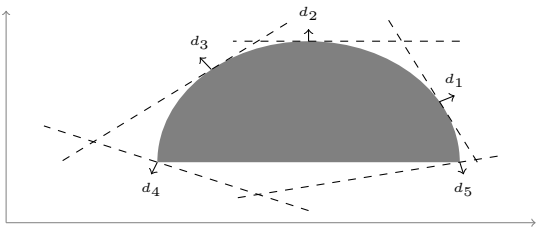


Fig. 1. Graphical representation of the support function of a convex set (in gray). The dashed lines are the lines $\langle x, d\rangle = \delta_S(d)$ for various directions d.

Figure 1 shows a convex set (in gray) and the value of its support function for some directions $d \in \mathbb{B}^n$. It should be clear from Property 1 and Figure 1 that it holds that, for a given direction set $\Delta \subseteq \mathbb{B}^n$, $S \subseteq \bigcap_{d\in\Delta}\{x \in \mathbb{R}^n | \langle x, d\rangle \leq \delta_S(d)\}$. Note that if Δ is a finite set, then $S_\Delta = \bigcap_{d\in\Delta}\{x \in \mathbb{R}^n | \langle x, d\rangle \leq \delta_S(d)\}$ is a convex polyhedron [11, Def. A.4.2.5]. So, the restriction of a support function of a set S over a finite domain Δ defines a convex polyhedron that over-approximates S. The faces of this polyhedron have a pre-defined shape: they are orthogonal to the chosen directions $d \in \Delta$. This is the basic idea behind our abstract domain based on support functions, see Section 3.

Support function computation. In the rest of this section, we show how the support function of a convex set can be computed efficiently in some cases. Obviously Definition 1 shows that the value of $\delta_S(d)$ for each $d \in \mathbb{B}^n$ can be obtained using a convex optimization problem [2], if an appropriate description of S is known. Property 2 below shows that we can compute efficiently the support function of a transformation of S. In this property, we denote by:

- MS for a given matrix $M \in \mathbb{R}^{n\times m}$ the set $MS = \{Mx | x \in S\}$,
- $S \oplus S'$ given two convex sets S and S' the Minkowski sum of S and S' defined by $S \oplus S' = \{x + x' \mid x \in S,\ x' \in S'\}$,
- λS for $\lambda \in \mathbb{R}$ the set $\lambda S = \{\lambda x \,|\, x \in S\}$.
- $S \cup S'$ the convex hull of convex sets S and S' and $S \cap S'$ their intersection.

Property 2 ([10, Prop. 3]). *Let S, S' be two convex sets. We have:*

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

Note that all relations are exact, except for the computation of the support function of the intersection for which we only have an over-approximation.

Another important case for which we can efficiently compute the support function of a convex set S is when S is a convex polyhedron. Then, the convex optimization problem of Definition 1 becomes a linear programming problem

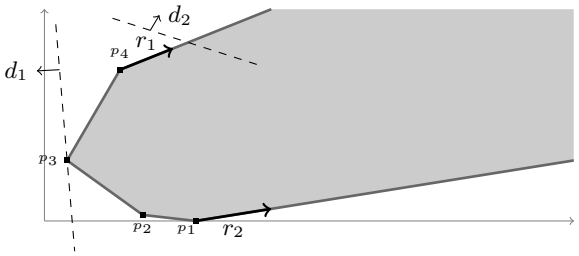


Fig. 2. Support function of a convex polyhedron. In direction d_1 the supremum is realized by a generator, in direction d_2 it is unbounded.

for which we have efficient algorithms (although linear programming may be exponential in the worst case). Moreover, Property 3 below shows that a more efficient method exists if the polyhedron is described by its generators.

Property 3 ([11, Ex. C.3.4.3]). *Let $\mathsf{P} \subseteq \mathbb{R}^n$ be a convex polyhedra generated by the set of generators $v_1, \ldots, v_k$ and rays $r_1, \ldots, r_l$. The support function δ_{P} is defined by:*

$$\forall d \in \mathbb{B}^n,\ \delta_{\mathsf{P}}(d) = \begin{cases} \max\limits_{i \in [1,k]} \langle v_i, d \rangle & \text{if } \forall j \in [1,l],\ \langle r_j, d \rangle \leq 0 \\ +\infty & \text{otherwise} \end{cases}.$$

Property 3 shows that for a convex polyhedron P represented by its generator, the support function δ_{P} in a direction d can be computed in linear time. The condition $\forall j \in [1,l], \langle r_j, d \rangle \leq 0$ in Property 3 allows us to efficiently detect when the polyhedron supremum $\sup_{x \in \mathsf{P}} \langle x, d \rangle$ is finite or not, as illustrated on Figure 2.

3 Abstract Domain

In this section, we formally define our abstract domain based on support functions: we define both the order theoretic operations and the effect of an affine and non-linear affectation. Our domain is an abstraction of convex polyhedra over $\mathbb{R}^n$, where n is the number of variables of the program being analyzed. We denote by $\mathbb{P}$ the abstract domain of convex polyhedra over $\mathbb{R}^n$.

3.1 Lattice Structure

Let $\Delta = \{d_1, \ldots, d_l\}$ be a finite set of directions, i.e. $\Delta \subseteq \mathbb{B}^n$. Our abstract domain $\mathbb{P}^\sharp_\Delta$ is parametrized by this set Δ and is defined in Definition 2.

Definition 2. *Let $\Delta \subseteq \mathbb{B}^n$ be the set of directions. We define $\mathbb{P}^\sharp_\Delta$ as the set of all functions from Δ to $\mathbb{R}_\infty$, i.e. $\mathbb{P}^\sharp_\Delta = \Delta \to \mathbb{R}_\infty$. We denote $\bot_\Delta$ (resp. $\top_\Delta$) the function such that $\forall d \in \Delta,\ \bot_\Delta(d) = -\infty$ (resp. $\top_\Delta(d) = +\infty$).*

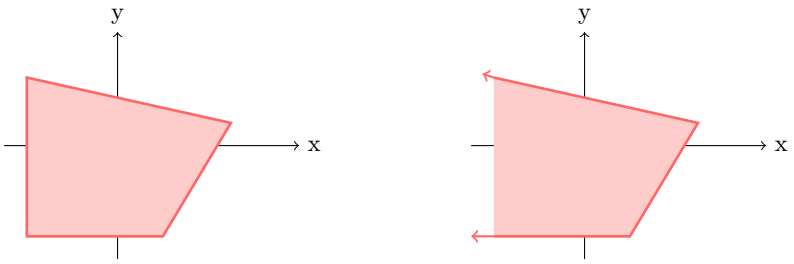


Fig. 3. The geometrical representation of $\gamma_\Delta(\Omega_1)$ (left) and $\gamma_\Delta(\Omega_2)$ (right)

For each $\Omega \in \mathbb{P}^\sharp_\Delta$, we write $\Omega(d)$ the value of Ω in direction $d \in \Delta$. Intuitively, Ω is a support function with finite domain.

The abstraction and concretization functions of $\mathbb{P}^\sharp_\Delta$ are given in Definition 3.

Definition 3. *Let $\Delta \subseteq \mathbb{R}^n$ be the set of directions. We define the concretizatiion function $\gamma_\Delta : \mathbb{P}^\sharp_\Delta \to \mathbb{P}$ by:*

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

The abstraction function $\alpha_\Delta : \mathbb{P} \to \mathbb{P}^\sharp_\Delta$ is defined by:

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

Example 1. *Let $\Delta \subseteq \mathbb{R}^2$ with $\Delta = \{(-3,5),(1,3),(-1,0),(0,-1)\}$. For the abstract element $\Omega_1 = \{3,4,3,2\}$[1]. The result of $\gamma_\Delta(\Omega_1)$ is given in Figure 3(left). The right of the Figure 3 is the result of $\gamma_\Delta(\Omega_2)$, with $\Omega_2 = \{3,3,+\infty,2\}$. In this case, for $d_3 = (-1,0)$, $\Omega_2(d_3) = +\infty$, which means that the resulting polyhedron is unbounded in the direction d_3.*

Definition 3 shows that the concretization of an abstract element of $\mathbb{P}^\sharp_\Delta$ is a polyhedron defined by the intersection of half-spaces, where each one is characterized by its normal vector $d \in \Delta$ and the coefficient $\Omega(d)$. The abstraction function on the other side is the restriction of the support function of the polyhedra on the set of directions Δ. We next define the order, join and meet of $\mathbb{P}^\sharp_\Delta$ and then show that $(\alpha_\Delta, \gamma_\Delta)$ is a Gallois connection.

Definition 4 (Order structure of $\mathbb{P}^\sharp_\Delta$). *We define the inclusion operation $\sqsubseteq_\Delta$ as $\forall \Omega_1, \Omega_2 \in \mathbb{P}^\sharp_\Delta, \Omega_1 \sqsubseteq_\Delta \Omega_2 \iff \gamma_\Delta(\Omega_1) \sqsubseteq \gamma_\Delta(\Omega_2)$, where $\sqsubseteq$ is the inclusion on $\mathbb{P}$. The join $\sqcup_\Delta$ and meet $\sqcap_\Delta$ are defined by:*

- $\forall \Omega_1, \Omega_2 \in \mathbb{P}^\sharp_\Delta,\ \Omega_1 \sqcup_\Delta \Omega_2 = \lambda d.\ \max(\Omega_1(d), \Omega_2(d))$;
- $\forall \Omega_1, \Omega_2 \in \mathbb{P}^\sharp_\Delta,\ \Omega_1 \sqcap_\Delta \Omega_2 = \lambda d.\ \min(\Omega_1(d), \Omega_2(d))$.

[1] We identify $\mathbb{P}^\sharp_\Delta = \Delta \to \mathbb{R}_\infty$ with $\mathbb{R}_\infty^{|\Delta|}$, so that $\Omega_1 = \{3,4,3,2\}$ is the function mapping the first direction to 3, the second to 4,..

Note that the join operation is exact:

$$\forall \Omega_1,\ \Omega_2 \in \mathbb{P}^\sharp_\Delta,\ \gamma_\Delta(\Omega_1 \sqcup_\Delta \Omega_2) = \gamma_\Delta(\Omega_1) \sqcup \gamma_\Delta(\Omega_2)$$

while the meet operation is over-approximated:

$$\forall \Omega_1,\ \Omega_2 \in \mathbb{P}^\sharp_\Delta,\ \gamma_\Delta(\Omega_1 \sqcap_\Delta \Omega_2) \supseteq \gamma_\Delta(\Omega_1) \sqcap \gamma_\Delta(\Omega_2)\ .$$

Property 4. *The function pair* $(\alpha_\Delta, \gamma_\Delta)$ *form a Galois connection [5] between* $\mathbb{P}$ *and* $\mathbb{P}^\sharp_\Delta$.

Proof. See our extended version [17]. □

Note that $\forall \Omega \in \mathbb{P}^\sharp_\Delta$, if $\Omega = \alpha_\Delta(\mathsf{P})$, then $\mathsf{P} \subseteq \gamma_\Delta(\Omega)$, and the vertices of the polyhedron P touch the faces of $\gamma_\Delta(\Omega)$. This is stated in Proposition 5.

Property 5 ([10, Prop. 3]). *Let* P *be a polyhedron and* $\Omega \in \mathbb{P}^\sharp_\Delta$ *such that* $\Omega = \alpha_\Delta(\mathsf{P})$. *We have that,* $\mathsf{P} \subseteq \gamma_\Delta(\Omega)$. *This over approximation is tight as the vertices of* P *touch the faces of* $\gamma_\Delta(\Omega)$.

3.2 Affine Transformations

We now explain how an element $\Omega \in \mathbb{P}^\sharp_\Delta$ is modified by an affine transformation of the form $X = AX + b$ where X is the variable set of the program, $A \in \mathbb{R}^{n\times n}$ is a square matrix and $b \in \mathbb{R}^n$ is a vector. Let thus P_0 be a polyhedron and $\Omega_0 = \alpha_\Delta(\mathsf{P}_0)$ be the initial abstract state. Let also $\mathsf{P}_1 = [\![X = AX + b]\!](\mathsf{P}_0)$ be the polyhedron obtained after applying the affine transformation. Our goal is to compute the best possible abstraction Ω_1 of P_1, without computing P_1. Note that, using the operation set defined in Section 2, we have that $\mathsf{P}_1 = A\mathsf{P}_0 \oplus b$. Thus, using Property 2 we have: $\forall d \in \Delta, \delta_{\mathsf{P}_1}(d) = \delta_{A\mathsf{P}_0 \oplus b}(d) = \delta_{A\mathsf{P}_0}(d) + \delta_b(d) = \delta_{\mathsf{P}_0}(A^T d) + \langle b, d\rangle$. So we define Ω_1 as:

$$\forall d \in \Delta,\ \Omega_1(d) = \delta_{\mathsf{P}_0}(A^T d) + \langle b, d\rangle. \tag{1}$$

Note that $\Omega_1 = \alpha_\Delta(\mathsf{P}_1)$, while $\mathsf{P}_1 \subseteq \gamma_\Delta(\Omega_1)$. However, we do not need compute P_1, we only need to evaluate δ_{P_0} on directions $A^T d$, which can be done efficiently if P_0 is described using generators, as stated by Proposition 3. Moreover, Proposition 5 ensures that P_1 vertices touch $\gamma_\Delta(\Omega_1)$ faces. The precision of $\gamma_\Delta(\Omega_1)$ depends strongly on the chosenΔ: more directions we have more precise Ω_1 is.

3.3 Non-linear Transformations

We now deal with non-linear transformation, i.e. we want to apply the transformation $X = f(X)$, where $f : \mathbb{R}^n \to \mathbb{R}^n$ is non-linear. We use the notion of *linearisation* presented in [13] to abstract the transformation into an interval linear form. Interval linear forms are given by $\boldsymbol{i} + \sum_{k=1}^n \boldsymbol{i}_k X_k$, where $\forall k \in [1, n],\ X_k$ is a program variable and $\boldsymbol{i},\ \boldsymbol{i}_k \in \mathbb{I}_\mathbb{R}$. For example, the expression $X_1 \times X_2$ can be transformed into $\boldsymbol{i}_1 \times X_2$ where $\boldsymbol{i}_1$ is the interval concretization of X_1. After

the linearisation process, the transformation $X = f(X)$ can be abstracted by $X = \boldsymbol{A}X + \boldsymbol{b}$, where $\boldsymbol{A} \in \mathbb{I}_{\mathbb{R}}^{n\times n}$ and $\boldsymbol{b} \in \mathbb{I}_{\mathbb{R}}^{n}$.

As for Section 3.2, we want to compute $\Omega \in \mathbb{P}^{\sharp}_{\Delta}$, which is an abstraction of $\mathsf{P}_1 = [\![X = \boldsymbol{A}X + \boldsymbol{b}]\!](\mathsf{P}_0)$ (the semantics of interval linear forms is given in [13]). We cannot use Equation 1 directly because $\forall d \in \Delta,\ \boldsymbol{A}^T d$ is an interval vector and δ_{P_0} is only defined on $\mathbb{R}^n$. To deal with that, we introduce the notion of *interval based support function*. In the rest of this article, $\forall \boldsymbol{i} \in \mathbb{I}_{\mathbb{R}}$, $\overline{\boldsymbol{i}}$ represents the upper bound of $\boldsymbol{i}$ and $\underline{\boldsymbol{i}}$ its lower bound.

Let P be a polyhedron represented by its generators $v_1, \ldots, v_k$ and rays $r_1, \ldots, r_l$. We define the function $\boldsymbol{\sigma}_{\mathsf{P}}$ by:

$$\boldsymbol{\sigma}_{\mathsf{P}} : \begin{cases} \mathbb{I}_{\mathbb{R}}^n \to \mathbb{R} \\ \boldsymbol{d} \mapsto \begin{cases} \max\limits_{i\in[1,k]} \overline{\langle v_i, \boldsymbol{d}\rangle} & \text{if } \forall j \in [1,l], \langle r_j, \boldsymbol{d}\rangle \cap]0, +\infty[= \emptyset \\ +\infty & \text{otherwise} \end{cases} \end{cases} .$$

In the same way, we define $\boldsymbol{\iota}_{\mathsf{P}}$ by:

$$\boldsymbol{\iota}_{\mathsf{P}} : \begin{cases} \mathbb{I}_{\mathbb{R}}^n \to \mathbb{R} \\ \boldsymbol{d} \mapsto \max_{i\in[1,k]} \underline{\langle v_i, \boldsymbol{d}\rangle} \end{cases} .$$

Property 6 shows that δ_{P}, the support function of P, can be approximated using $\boldsymbol{\iota}_{\mathsf{P}}$ and $\boldsymbol{\sigma}_{\mathsf{P}}$. We call this approximation *interval based support function*.

Property 6 (Interval based support function). *Let P be a polyhedron and δ_{P} be its support function. Let $\boldsymbol{d} \in \mathbb{I}_{\mathbb{R}}^n$ be an interval vector, representing a set of possible directions. We have that:*

$$\forall d \in \boldsymbol{d}, \boldsymbol{\iota}_{\mathsf{P}}(\boldsymbol{d}) \leq \delta_{\mathsf{P}}(d) \leq \boldsymbol{\sigma}_{\mathsf{P}}(\boldsymbol{d})$$

Proof. On the one hand, we have that $\exists v \in \mathsf{P}$ *s.t.* $\boldsymbol{\iota}_{\mathsf{P}}(\boldsymbol{d}) = \underline{\langle v, \boldsymbol{d}\rangle} = b$. So,

$$(\forall d \in \boldsymbol{d}), \delta_{\mathsf{P}}(d) \geq \langle v, d\rangle \geq b. \tag{2}$$

On the other hand, we have that $\exists v \in \mathsf{P}$ *s.t.* $\boldsymbol{\sigma}_{\mathsf{P}}(\boldsymbol{d}) = \overline{\langle v, \boldsymbol{d}\rangle} = b'$. So,

$$\forall d \in \boldsymbol{d}, \exists v_i \in \mathsf{P} \text{ s.t. } \delta_{\mathsf{P}}(d) = \langle v_i, d\rangle \leq \overline{\langle v_i, \boldsymbol{d}\rangle} \leq b'. \tag{3}$$

Thus from Equations 2 and 3, we have that: $b \leq \delta_{\mathsf{P}}(d) \leq b'$. □

Let us now define Ω, abstraction of P_1. We know that, for all $d \in \Delta$ and for all $d' \in \boldsymbol{A}^T d$, $\delta_{\mathsf{P}_0}(d') \leq \boldsymbol{\sigma}_{\mathsf{P}_0}(\boldsymbol{A}^T d)$. We have, $\forall d \in \Delta, \delta_{\mathsf{P}_1}(d) = \delta_{\boldsymbol{A}\mathsf{P}_0 \oplus \boldsymbol{b}}(d)$, so $\delta_{\mathsf{P}_1}(d) \leq \boldsymbol{\sigma}_{\boldsymbol{A}\mathsf{P}_0}(d) + \overline{\langle \boldsymbol{b}, d\rangle}$. So we define Ω as

$$\forall d \in \Delta,\ \Omega(d) = \boldsymbol{\sigma}_{\mathsf{P}_0}(\boldsymbol{A}^T d) + \overline{\langle \boldsymbol{b}, d\rangle} . \tag{4}$$

Note that in this case, Ω is an over-approximation of δ_{P_1}, i.e. $\alpha_{\Delta}(\mathsf{P}_1) \sqsubseteq_{\Delta} \Omega$. In the same way, we can use $\boldsymbol{\iota}_{\mathsf{P}}$ to under approximate δ_{P_1}, i.e. we have that:

$$\forall d \in \Delta,\ \delta_{\mathsf{P}}(d) \geq \boldsymbol{\iota}_{\mathsf{P}_0}(\boldsymbol{A}^T d) + \underline{\langle \boldsymbol{b}, d\rangle} .$$

This under-approximation can be combined with the over-approximation to evaluate the precision of Ω.

4 Fixpoint Computation for Affine Loops

In this section, we present a specialization of Kleene algorithm to compute the fixpoint of an affine loop using our domain $\mathbb{P}^{\sharp}_{\Delta}$. We consider loops of the form:

```
while(C)
  X=AX+b;
```

We suppose that `A` is a real matrix, `b` may be a set of values, given as a polyhedra P_b, and `C` is a guard. Such loops include for example linear filters in which P_b represents the possible values of the new input at each loop iteration. We assume that the program variables lie initially in the polyhedra P_0.

4.1 Loops without Guards

We first consider the case where the loop is not conditioned by a guard, i.e. `C` is `true`. We want to compute an over-approximation in $\mathbb{P}^{\sharp}_{\Delta}$ of P_∞, the loop invariant defined as the least fixpoint of the equation $\mathsf{P} = \mathsf{P}_0 \sqcup \big(A\mathsf{P} + \mathsf{P}_b\big)$. Usually, P_∞ is defined as the limit of the Kleene iterates given by $\mathsf{P}_i = \mathsf{P}_{i-1} \sqcup (A\mathsf{P}_{i-1} + \mathsf{P}_b)$.

Property 7 defines the abstract element Ω_i at each iteration and shows that for all $d \in \Delta, \Omega_i(d) = \delta_{\mathsf{P}_i}(d)$. Thus, we have that $\Omega_i = \alpha_\Delta(\mathsf{P}_i)$, which means that Ω_i is the best abstraction of P_i in $\mathsf{P}^{\sharp}_{\Delta}$.

Property 7. *Let P_i be the polyhedron obtained in the i^{th} iteration using polyhedra abstract domain, then*

$$\delta_{\mathsf{P}_i}(d) = \Omega_i(d) = \max\Big(\delta_{\mathsf{P}_0}(d), \max_{j\in[1,i]}\big(\delta_{\mathsf{P}_0}(A^{T^j}d) + \sum_{k=1}^{j}\delta_{\mathsf{P}_b}(A^{T^{(k-1)}}d)\big)\Big) \tag{5}$$

Proof. The proof runs by induction on i. We begin by $i = 1$. $\forall d \in \Delta$, we have, using Property 2, that $\delta_{\mathsf{P}_1}(d) = \delta_{\mathsf{P}_0 \cup A\mathsf{P}_0 + \mathsf{P}_b}(d) = \max\big(\delta_{\mathsf{P}_0}(d), \delta_{\mathsf{P}_0}(A^T d) + \delta_{\mathsf{P}_b}(d)\big)$. Let now $i \geq 1$ such that Equation (5) is true.
Then, we have, $\forall d \in \Delta$: $\delta_{\mathsf{P}_{i+1}} = \delta_{\mathsf{P}_i \sqcup (A\mathsf{P}_i + \mathsf{P}_b)}(d) = \max\big(\delta_{\mathsf{P}_i}(d), \delta_{\mathsf{P}_i}(A^T d) + \delta_{\mathsf{P}_b}(d)\big)$. Now, we have that:

$$\begin{aligned}
\delta_{\mathsf{P}_i}(A^T d) &= \max\Big(\delta_{\mathsf{P}_0}(A^T d), \max_{j\in[1,i]}\big(\delta_{\mathsf{P}_0}(A^{T^j}A^T d) + \sum_{k=1}^{j}\delta_{\mathsf{P}_b}(A^{T^{(k-1)}}A^T d)\big)\Big) \\
&= \max\Big(\delta_{\mathsf{P}_0}(A^T d), \max_{j\in[1,i]}\big(\delta_{\mathsf{P}_0}(A^{T^{(j+1)}} d) + \sum_{k=1}^{j}\delta_{\mathsf{P}_b}(A^{T^k} d)\big)\Big) \\
&= \max\Big(\delta_{\mathsf{P}_0}(A^T d), \max_{j\in[2,i+1]}\big(\delta_{\mathsf{P}_0}(A^{T^{(j)}} d) + \sum_{k=2}^{j}\delta_{\mathsf{P}_b}(A^{T^{(k-1)}} d)\big)\Big)
\end{aligned}$$

We can deduce with a case analysis that:

$$\delta_{\mathsf{P}_i}(A^T d) + \delta_{\mathsf{P}_b}(d) = \max_{j\in[1,i+1]}\big(\delta_{\mathsf{P}_0}(A^{T^j} d) + \sum_{k=1}^{j}\delta_{\mathsf{P}_b}(A^{T^{(k-1)}} d)\big)$$

From that we deduce Equation 5 for $i + 1$. □

Algorithm 1. Kleene Algorithm using support function

Input: $\Delta \subset \mathbb{R}^n$, set of l directions
Input: P_0, The initial polyhedron
Input: $A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n$
1: $D = \Delta$
2: $\Omega = \delta_{\mathsf{P}_0}(\Delta)$
3: **repeat**
4: $\quad \Omega' = \Omega$
5: $\quad$ **for all** $i = 0, \ldots, (l-1)$ **do**
6: $\quad\quad \Theta[i] = \Theta[i] + \delta_{\mathsf{P}_b}(D[i])$
7: $\quad\quad D[i] = A^T D[i]$
8: $\quad\quad \Upsilon[i] = \delta_{\mathsf{P}_0}(D[i]) + \Theta[i]$
9: $\quad\quad \Omega[i] = \max(\Omega[i], \Upsilon[i])$
10: $\quad$ **end for**
11: **until** $\Omega \sqsubseteq_\Delta \Omega'$

Property 7 defines a normal form of Ω_i *i.e.* $\Omega_i = \alpha_\Delta(\gamma_\Delta(\Omega_i))$. From that, we have that $\sqsubseteq_\Delta$ can be performed in linear time, such that:

$$\forall \Omega_1, \Omega_2 \in \mathbb{P}^\sharp_\Delta, \Omega_1 \sqsubseteq_\Delta \Omega_2 \iff \forall d \in \Delta, \Omega_1(d) \leq \Omega_2(d).$$

In Algorithm 1, the computation of the abstract element Ω depends on the computation of δ_{P_0}, Θ and D. We know that P_0 represents the polyhedron of the initial condition of the analysed program, so its representation, in general, is quite simple. In particular, the number of its generators is usually small. This means that the computation of δ_{P_0} does not require LP solvers. So, what changes in each iteration is the direction set in which δ_{P_0} is computed. Thus, Algorithm 1 has a polynomial complexity in the number of iteration and linear in the number of directions in Δ. In addition, its result is as accurate as possible: at each iterate, we have that $\Omega_i = \alpha_\Delta(\mathsf{P}_i)$. So $\Omega_\infty = \alpha_\Delta(\mathsf{P}_\infty)$, with Ω_∞ is the fixpoint obtained in our analysis and P_∞ is the one obtained using polyhedra domain. Note that, $\gamma_\Delta(\Omega_\infty)$ can have redundant constraints *i.e.* $\exists d \in \Delta$ *s.t.* $\gamma_{\Delta\setminus\{d\}}(\Omega_\infty) = \gamma_\Delta(\Omega_\infty)$. However, $\gamma_\Delta(\Omega_\infty)$ can be used for another analysis, so a redundancy removal method is needed. The one defined on polyhedra domain is time consuming, so we want to develop an efficient redundancy removal method based on our domain, which is the subject of our ongoing work.

Remark 1. *1) Like in the standard Kleene algorithm, Algorithm 1 does not guarantee the termination of the analysis. To handle this problem, we can use a widening operator on support functions which is very easy to define: if the support function in a given direction increases, we set it to* $+\infty$*. Of course using threshold [12] can help to limit this over-approximation. Another solution to speed-up the convergence is the use of the acceleration method presented in our previous work [16]. For that, we construct for each* $d \in \Delta$ *the numerical sequence* $S_d = (\Omega_i(d))_{i\in\mathbb{N}}$*, and then use acceleration methods to compute its limit.*

2) Our results are more precise than those obtained using template domain [4] with the chosen direction set Δ *as a TCM [16, Sect. 5.1]. The difference is*

that, in $\mathbb{P}^{\sharp}_{\Delta}$ the analysis is done with the precision of the polyhedra domain and the over-approximation is done only, at the end, in the concretization function. When with template domain, all the analysis is done in a less expressive domain.

4.2 Loops with Linear Guard

We now consider the case when the loop has a guard of the form $\langle X, c\rangle \leq l$, with $c \in \mathbb{R}^n$ and $l \in \mathbb{R}$. Let H be the half space $H = \{x \in \mathbb{R}^n | \langle x, c\rangle \leq l\}$. In this case, the polyhedra P_i is defined as: $\mathsf{P}_{i+1} = \mathsf{P}_i \cup \big((A_i\mathsf{P}_i \oplus \mathsf{P}_b) \cap H\big)$. $\forall d \in \Delta$, we have $\delta_{\mathsf{P}_{i+1}}(d) = \delta_{\mathsf{P}_i \cup ((A_{i+1}\mathsf{P}_i \oplus \mathsf{P}_b)\cap H)}(d)$ so:

$$\delta_{\mathsf{P}_{i+1}}(d) \leq \max(\delta_{\mathsf{P}_i}(d), \min(\delta_{\mathsf{P}_i}(A^T d) + \delta_{\mathsf{P}_b}(d), \delta_H(d)))$$

Note that $\delta_H(d) = l$ if $d = \lambda c$ for some $\lambda \geq 0$ and $\delta_H(d) = +\infty$ otherwise. We thus distinguish two cases: $d = \lambda c$ with $(\lambda \geq 0)$ or $\nexists \lambda \geq 0,\ d = \lambda c$. Let us thus put $\Delta_1 = \{d \in \Delta | d = \lambda c, \lambda \geq 0\}$, and $\Delta_2 = \Delta \setminus \Delta_1$, if Δ_1 is empty we put $\Delta_1 = \{c\}$. Note that Δ is defined such that its elements are not two per two parallel i.e. $\forall d \in \Delta, \nexists d' \in \Delta \setminus \{d\} : d = \lambda.d'(\lambda \geq 0)$. So, the cardinality of Δ_1 is 1. For the fixpoint computation, we separate the two cases. If $d \in \Delta_2$, as $\delta_H(d) = +\infty$, we have the same relation between $\delta_{\mathsf{P}_{i+1}}$ and δ_{P_i} as for the case of loops without guards, so $\Omega_i(d)$ defined as in Property 7.
Now, for $d \in \Delta_1$, we put $\Omega_i(d) = \max(\delta_{\gamma_\Delta(\Omega_{i-1})}(d), \min(\delta_{\gamma_\Delta(\Omega_{i-1})}(A^T d) + \delta_{\mathsf{P}_b}(d), l))$ which is an over approximation of $\delta_{\mathsf{P}_i}(d)$:

$$\begin{aligned}
\delta_{\mathsf{P}_i}(d) &= \delta_{\mathsf{P}_{i-1} \cup ((A\mathsf{P}_{i-1} \oplus \mathsf{P}_b)\cap H)}(d) \\
&\leq \max(\delta_{\mathsf{P}_{i-1}}(d), \min(\delta_{\mathsf{P}_{i-1}}(A^T d) + \delta_{\mathsf{P}_b}(d), \delta_H(d))) \\
&\leq \max(\delta_{\gamma_\Delta(\Omega_{i-1})}(d), \min(\delta_{\gamma_\Delta(\Omega_{i-1})}(A^T d) + \delta_{\mathsf{P}_b}(d), l)) \\
&\leq \Omega_i(d)
\end{aligned}$$

To compute $\Omega_i(d)$, we use $\delta_{\gamma_\Delta(\Omega_{i-1})}(d)$ and $\delta_{\gamma_\Delta(\Omega_{i-1})}(A^T d)$, which are obtained using linear programming. This does not affect a lot our method efficiency, because it is applied at most for one direction in Δ. So, in the case of affine loops with linear guard $\langle X, c\rangle \leq l$, we use Algorithm 1 but distinguish when $d \in \Delta_1$ from $d \in \Delta_2$.Then, we have that $\alpha_\Delta(\mathsf{P}_i) \sqsubseteq_\Delta \Omega_i$ such that the P_i vertices touch $\gamma_\Delta(\Omega_i)$ faces, except for the face of $\gamma_\Delta(\Omega_i)$ whose normal vector belongs to Δ_1.

5 Fixpoint Computation for Non-linear Loops

Let us now handle the case of non-linear loop, i.e. we consider a loop whose body is $X = f(X)$, f being a (possibly) non-linear function of the program variables. We again compute over-approximations of P_i, the Kleene algorithm iterates over the polyhedra domain. Basically, we apply, at each iteration, a linearisation of the function f and then use the interval based support function to compute Ω_i.

Let us denote $\boldsymbol{A}_i X + \boldsymbol{b} = L(f, \Omega_i)$ the interval linear form produced by linearisation of f when the value of variables X are in $\gamma_\Delta(\Omega_i)$. Note that this means that the matrix A of Algorithm 1 is now an interval matrix which may change

from one iteration to the other. We compute $\boldsymbol{A}_i$ using techniques from [13], which requires that we have bounds on variables $X_k \in X$. Such bounds are very easy to get in our case: we assume that in Δ we added each direction $\pm X_k$, for every variable X_k. Then, the bounds on X_k are given by $\Omega_i(X_k)$ and $\Omega_i(-X_k)$[2].

To compute Ω_{i+1} from Ω_i, we will thus: first compute $\boldsymbol{A}_i$ using the linearisation process, and then apply the interval linear transformation $\boldsymbol{A}_i X + b$ to Ω_i. Using functions $\boldsymbol{\sigma}$ and $\boldsymbol{\iota}$ defined in Section 3.3, we can have bounds for Ω_{i+1}. In the polyhedra abstract domain, let P_i be the polyhedron obtained in the i^{th} Kleene iteration. Property 8 below shows that we can compute bounds on δ_{P_i}.

Property 8. *Let $d \in \Delta$. For all $i \in \mathbb{N}$, we define $\boldsymbol{d_i}^\star = \prod_{k=1}^{i} \boldsymbol{A}_k^T d$ and :*

$$\overline{\Psi_i} = \sum_{k=1}^{i-1} \boldsymbol{\sigma}_{\mathsf{P}_b}\Big(\prod_{q=k+1}^{i} \boldsymbol{A}_q^T d\Big) + \delta_{\mathsf{P}_b}(d), \quad \underline{\Psi_i} = \sum_{k=1}^{i-1} \boldsymbol{\iota}_{\mathsf{P}_b}\Big(\prod_{q=k+1}^{i} \boldsymbol{A}_q^T d\Big) + \delta_{\mathsf{P}_b}(d)\ .$$

We have that:

$$\begin{cases} \delta_{\mathsf{P}_i}(d) \geq \max\Big(\delta_{\mathsf{P}_0}(d), \max_{j\in[1,i]}\big(\boldsymbol{\iota}_{\mathsf{P}_0}(\boldsymbol{d_j}^\star) + \underline{\Psi_j}\big)\Big) \\ \delta_{\mathsf{P}_i}(d) \leq \max\Big(\delta_{\mathsf{P}_0}(d), \max_{j\in[1,i]}\big(\boldsymbol{\sigma}_{\mathsf{P}_0}(\boldsymbol{d_j}^\star) + \overline{\Psi_j}\big)\Big) \end{cases}$$

Proof. We do not give the whole proof as it is technical and long, but rather show how it runs for $i = 1, 2$. The general case is then a generalization of this.

Case $i = 1$. We know that $\mathsf{P}_1 = \mathsf{P}_0 \sqcup \boldsymbol{A_1}\mathsf{P}_0 + \mathsf{P}_b$. This means that

$$\mathsf{P}_1 = \mathsf{P}_0 \sqcup \Big(\bigsqcup_{A_1 \in \boldsymbol{A_1}} A_1\mathsf{P}_0 + \mathsf{P}_b\Big)\ .$$

The property of $\boldsymbol{\sigma}$ proves that $\forall A_1 \in \boldsymbol{A_1}$, we have $\delta_{A_1\mathsf{P}_1}(d) \leq \boldsymbol{\sigma}_{\mathsf{P}_0}(\boldsymbol{A_1^T} d)$, and equivalently for $\boldsymbol{\iota}$. This proves the property for $i = 1$.

Case $i = 2$. We notice that $\mathsf{P}_2 = \mathsf{P}_0 \sqcup (\boldsymbol{A_1}\mathsf{P}_0 + \mathsf{P}_b) \sqcup (\boldsymbol{A_2}\boldsymbol{A_1}\mathsf{P}_0 + \boldsymbol{A_2}\mathsf{P}_b + \mathsf{P}_b)$. And then, for all $d \in \Delta$:

$$\begin{aligned} \forall A_1 \in \boldsymbol{A_1},\ A_2 \in \boldsymbol{A_2},\ \delta_{A_2A_1\mathsf{P}_0 + A_2\mathsf{P}_b}(d) &= \delta_{\mathsf{P}_0}(A_1^T A_2^T d) + \delta_{\mathsf{P}_b}(A_2^T d) \\ &\leq \boldsymbol{\sigma}_{\mathsf{P}_0}(\boldsymbol{A_1}^T \boldsymbol{A_2}^T d) + \boldsymbol{\sigma}_{\mathsf{P}_b}(\boldsymbol{A_2}^T d) \end{aligned}$$

We equivalently get the lower bound using the $\boldsymbol{\iota}$ function, and using the fact that $\delta_{S\cup S'}(d) = \max(\delta_S(d), \delta_{S'}(d))$, we get the result for $i = 2$. □

Let $\Delta \subseteq \mathbb{R}^n$ be a set of directions. Property 8 allows us to define the abstract element $\Omega_i \in \mathsf{P}^\sharp_\Delta$ as given in Definition 5.

Definition 5. *Let $\boldsymbol{d_i}^\star$ and Ψ_i be defined as in Property 8. The abstract element $\Omega_i \in \mathsf{P}^\sharp_\Delta$ obtained in the i^{th} Kleene iteration is given by:*

$$\forall d \in \Delta,\ \Omega_i(d) = \max\Big(\delta_{\mathsf{P}_0}(d), \max_{j\in[1,i]}\big(\boldsymbol{\sigma}_{\mathsf{P}_0}(d_j^\star) + \overline{\Psi_j}\big)\Big)$$

[2] We let X_k denote the vector of $\mathbb{B}^n$ with a 1 in the dimension corresponding to X_k.

We have that $\mathsf{P}_i \subseteq \gamma_\Delta(\Omega_i)$, so Ω_i of Definition 5 is sound. Note that we are no-longer guaranteed to have the best abstraction, i.e. we only have $\alpha_\Delta(\mathsf{P}_i) \sqsubseteq_\Delta \Omega_i$ (compared to the linear case in which we had an equality). We can also compute an under-approximation of the fixpoint using the lower bound of Property 8.
We can modify Algorithm 1 for non-linear loops. Due to lack of place, we can not present this new algorithm (see the extended version [17]). We here explain the main differences with Algorithm 1 and why the complexity is increased. The main difference is that we now need to keep track of the list of all matrices $\prod_{k=1}^{i} \boldsymbol{A}_k$ and $\prod_{q=k+1}^{i} \boldsymbol{A}_q$. This list, called θ in the algorithm, is used as follows:

1. we extend it at the beginning of each iteration by computing the linearisation matrix $\boldsymbol{A}_{i+1}$ and multiplying each term of the list by $\boldsymbol{A}_{i+1}$.
2. we iterate on it to compute $\overline{\Psi}$ and thus the upper bound of $\Omega(d)$.

The θ list length at iteration i is i, so we must make i call to σ_{P_0} or σ_{P_b} at iteration i, which makes the complexity of this algorithm quadratic in the number of generators of P_0 and P_b, while the algorithm for the linear case is linear.

Now let us extend this method to loops with non-linear guard. Again, we linearize at each iteration the guard and thus get a guard of the form $\langle X, C_i \rangle \leq L$, where $C_i \in \mathbb{I}_{\mathbb{R}}^n$ and $L \in \mathbb{I}_{\mathbb{R}}$. In this case, C_i changes in each iteration. To compute $\Omega_i \in \mathsf{P}_\Delta^\sharp$, we distinguish two cases, $\forall d \in \Delta$:

- If $d \notin C_i$ then Property 8 is used to compute $\Omega_i(d)$.
- If $d \in C_i$, $\Omega_i(d) = \max(\delta_{\gamma_\Delta(\Omega_{i-1})}(d), \min(\boldsymbol{\sigma}_{\gamma_\Delta(\Omega_{i-1})}(A_i^T d) + \overline{\langle b, d \rangle}, \overline{L}))$.

Here more than one direction in each iteration may belong to C_i, so we may need to perform many calls to a linear programming solver to compute Ω_i.

Remark 2. *To analyse programs with floating point numbers, we can use the same technique as in the octagon domain with floating point [13]. The idea is to use the interval analysis and so we can use the interval based support function.*

6 Experimentation

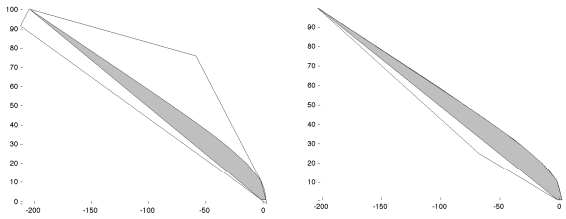
To show the efficiency of our abstract domain, we use it to analyze different numerical programs. The implementation is done using the PPL[3]. The experimentation was done on a computer with 4 $2.0GHz$ processors and $8Gb$ of RAM. The linear programs that we analyze are digital filters [4] of order 2 to 10, to show the impact of the number of variables on the efficiency of the analysis. These filters are taken from the tests of Filter Verification Framework [7]. Note that Kleene iteration using the polyhedra abstract domain without widening fails to analyze these programs, i.e. the analysis does not terminate. So to compare our results, we use polyhedra abstract domain with widening with delay (15 iterations). The table of Figure 4 shows that our domain, using the octagonal

[3] Parma Polyhedra Library : `http://bugseng.com/products/ppl/`
[4] Programs are at `http://www.lix.polytechnique.fr/~bouissou/filters/`

Program		Polyhedra		$\mathbb{P}^\sharp_\Delta$		
Name	$\lvert V\rvert$	t(s)	Bounded	t(s)	Iteration	$\lvert y_n\rvert$
lp_iir_9600_2	6	0.12	No	0.023	47	19.16
lp_iir_9600_4	10	TO	-	0.186	100	2.96
lp_iir_9600_4_elliptic	10	TO	-	0.471	276	3.74
lp_iir_9600_6_elliptic	14	TO	-	3.636	702	4.89
bs_iir_9600_12000_10_chebyshev	22	TO	-	53.986	2391	7.93
non_linear_ODE	3	TO	-	0.059	13	8.047

Fig. 4. Results of analysis obtained using different methods

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



8 directions 200 directions

Fig. 5. A simple program (left) and the obtained post-fixpoints (right)

direction set and without widening, allows us to obtain a good fixpoint quickly, when the analysis using polyhedra abstract domain returns $\top$ in the best cases. The column $|y_n|$ is the width of the bounding box for the output of the filter, it is computed as $|y_n| = |\Omega(y_n) - \Omega(-y_n)|$, where Ω is the obtained post-fixpoint. $|V|$ is the number of variables, columns labeled t are the execution time (in seconds), the value TO meaning that the execution took more than 10 minutes, the column "Bounded" tells whether the polyhedra analysis could compute a bounded post-fixpoint. For most programs the analysis with widening did not terminate before the time-out. Note that increasing the delay did not help in getting a bounded fixpoint for the polyhedra domain. Note also that we did not use thresholds for the widening because our programs contain infinite loops, which means without guard, so it is hard to define relevant thresholds statically. This table thus shows the efficiency of our algorithm for linear loops. Remember that the computed post-fixpoint is also precise: it is the abstraction, in $\mathbb{P}^\sharp_\Delta$, of the least fixpoint obtained with polyhedra domain.

For the experimentation, we took also a non-linear program, which represents the Euler scheme to solve a non-linear ODE given by the formulas:

$$x_1 = x_1 + dt \times (-(1 + \gamma \times x_2^2) \times x_1)$$
$$x_2 = x_2 + dt \times (-0.5 \times x_2 \times (1 - \gamma \times x_1^2) + 2 \times x_3)$$
$$x_3 = x_3 + dt \times (-(1 - \gamma \times x_1) \times 2x_2 - 0.5 \times x_3)$$

where $dt = 0.01$, $\gamma = 0.1$ and the initial variables values are in $[-2, 2]$. Its analysis using polyhedra[5] could not give a result in a reasonable time.

[5] Using INTERPROC analyser
http://pop-art.inrialpes.fr/interproc/interprocwebf.cgi

We know that the precision of our analysis result depends strongly on the chosen direction set. To show that, we analyse the program given in the left of Figure 5 using $\mathbb{P}^\sharp_{\Delta_1}$ and $\mathbb{P}^\sharp_{\Delta_2}$, *s.t.* Δ_1 and Δ_2 represent, respectively, set of 8 and 200 random directions. We display on Figure 5 (right) our analysis result (in white) and the polyhedron obtained using Kleene iteration on the polyhedra domain (filled in gray). Note that, the polyhedron is obtained after 200 iterations and is not the least fixpoint in the polyhedra domain, which is contained in the white polyhedron obtained with our method. Execution time using Δ_1 is $0.046s$ and $3.15s$ using Δ_2, which shows our method scalability in the directions number.

Finally, as mentionned in Section 4, our algorithm combines easily with widening: we just set $\Omega_i(d) = +\infty$ if $\Omega_i(d) > \Omega_{i-1}(d)$. Using this widening, we can compute post-fixpoints of unbounded programs. For example, the simple translation $x = x + 1 \wedge y = y + 1$ starting from $x \in [0, 1]$ and $y \in [0, 1]$, we could compute the fixpoint $x \geq 0 \wedge y \geq 0 \wedge -y \geq x - 1 \wedge y \leq x + 1$ in 3 iterations.

7 Conclusion

In this article, we showed a new abstract domain that uses support functions to represent convex sets. Depending on the chosen set of directions, our domain $\mathsf{P}^\sharp_\Delta$ holds an over-approximation of the support functions of the set in each direction. Clearly, both the definition and the order defined in our domain are the same as for the template abstract domain. However, the linear assignments 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^{th} iterate is the best abstraction in $\mathsf{P}^\sharp_\Delta$ of the i^{th} iterate one would have computed using the polyhedra domain.

As already stated, the precision of our domain depends on the relevancy of the used direction set. Our analysis, in most cases, is not time consuming, so we can get a precise post-fixpoint using a large number N of random directions. The problem is that the resulting polyhedron contains a lot of constraints, and is thus hard to be, eventually, re-used as an entry of another analysis. We plan to develop a minimization method, that allows us to keep only $K \leq N$ relevant directions. For that, we are looking to apply pruning methods developed in [8], which allow to keep K linear functions from a set of N templates in order to best approximate the value function of an optimal control problem. We believe that the use of support functions to represent a polyhedron will allow us to use efficient methods to compute the importance of one constraint of the polyhedron, which is an apriori to the algorithm of [8]. Clearly, the choice of random directions is not optimal, so we are also interested in adapting the techniques of parametrized templates used in [4] to define the set of directions we use. In this way, we believe we could change it during the analysis and thus gain in precision. These ideas are the subject of our ongoing works.

Acknowledgement. We want to thank A. Adjé, E. Goubault and the anonymous reviewers for their helpful comments, and precious advices.

References

1. Allamigeon, X., Gaubert, S., Goubault, É.: Inferring Min and Max Invariants Using Max-plus Polyhedra. In: Alpuente, M., Vidal, G. (eds.) SAS 2008. LNCS, vol. 5079, pp. 189–204. Springer, Heidelberg (2008)
2. Bertsekas, D.P., Nedić, A., Ozdaglar, A.E.: Convex Analysis and Optimization. Athena Scientific Series in Optimization and Neural Computation. Athena Scientific (2003)
3. Chen, L., Miné, A., Wang, J., Cousot, P.: Interval polyhedra: An abstract domain to infer interval linear relationships. In: Palsberg, J., Su, Z. (eds.) SAS 2009. LNCS, vol. 5673, pp. 309–325. Springer, Heidelberg (2009)
4. Colón, M.A., Sankaranarayanan, S.: Generalizing the template polyhedral domain. In: Barthe, G. (ed.) ESOP 2011. LNCS, vol. 6602, pp. 176–195. Springer, Heidelberg (2011)
5. Cousot, P., Cousot, R.: 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 1977), pp. 238–252. ACM Press (1977)
6. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Conference Record of the Fifth ACM Symposium on Principles of Programming Languages (POPL 1978), pp. 84–97. ACM Press (1978)
7. Cox, A., Sankaranarayanan, S., Chang, B.-Y.E.: A bit too precise? Bounded verification of quantized digital filters. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 33–47. Springer, Heidelberg (2012)
8. Gaubert, S., McEneaney, W.M., Qu, Z.: Curse of dimensionality reduction in max-plus based approximation methods: Theoretical estimates and improved pruning algorithms. In: CDC-ECE (2011)
9. Goubault, E., Putot, S.: Perturbed affine arithmetic for invariant computation in numerical program analysis. CoRR, abs/0807.2961 (2008)
10. Le Guernic, C., Girard, A.: Reachability analysis of hybrid systems using support functions. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 540–554. Springer, Heidelberg (2009)
11. Hiriart-Urrut, J.-B., Lemaréchal, C.: Fundamentals of Convex Analysis. Springer (2004)
12. Lakhdar-Chaouch, L., Jeannet, B., Girault, A.: Widening with thresholds for programs with complex control graphs. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 492–502. Springer, Heidelberg (2011)
13. Miné, A.: Weakly relational numerical abstract domains. PhD thesis, École Polytechnique (2004), `http://www.di.ens.fr/~mine/these/these-color.pdf`
14. Miné, A.: The octagon abstract domain. Higher-Order and Symbolic Computation 19 (2006)
15. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable analysis of linear systems using mathematical programming. In: Cousot, R. (ed.) VMCAI 2005. LNCS, vol. 3385, pp. 25–41. Springer, Heidelberg (2005)
16. Seladji, Y., Bouissou, O.: Fixpoint computation in the polyhedra abstract domain using convex and numerical analysis tools. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) VMCAI 2013. LNCS, vol. 7737, pp. 149–168. Springer, Heidelberg (2013)
17. Seladji, Y., Bouissou, O.: Numerical abstract domain using support functions (extended version) (2013), `http://www.lix.polytechnique.fr/~bouissou/pdf/publications/NFM13_extended.pdf`