Fixpoint Computation in the Polyhedra Abstract Domain Using Convex and Numerical Analysis Tools

Yassamine Seladji and Olivier Bouissou

CEA LIST
CEA Saclay Nano-INNOV Institut CARNOT
91 191 Gif sur Yvette CEDEX, France
firstname.lastname@cea.fr

Abstract. Polyhedra abstract domain is one of the most expressive and used abstract domains for the static analysis of programs. Together with Kleene algorithm, it computes precise yet costly program invariants. Widening operators speed up this computation and guarantee its termination, but they often induce a loss of precision, especially for numerical programs. In this article, we present a process to accelerate Kleene iteration with a good trade-off between precision and computation time. For that, we use two tools: convex analysis to express the convergence of convex sets using *support functions*, and numerical analysis to accelerate this convergence applying *sequence transformations*. We demonstrate the efficiency of our method on benchmarks.

1 Introduction

Static analysis plays an important role in software engineering. It allows to verify some safety properties on programs, like for example the absence of runtime errors, which is crucial in the verification of embedded programs. Static analysis uses the (concrete) program semantic function F to define a program invariant X, such that $F(X) = X$. This represents the set of reachable program states, but is however not computable. To deal with that, an over-approximation is introduced using abstract interpretation [10]. The main idea is to define a new (abstract) semantic function $F^\sharp$ that computes an abstract program invariant that includes the concrete one. This inclusion guarantees the safety of the result but not its accuracy: the larger the over-approximation is, the higher the probability to add false alarms. This over-approximation is due to the type of elements manipulated by $F^\sharp$; the most common and expressive abstraction is to represent the reachable states of numerical programs as convex polyhedra [9], that represent the linear relations existing between program variables.

The applicability of this analysis is faced with the compromise between precision and complexity: the standard algorithms (union, intersection) for computing with polyhedra are as precise as possible, but are mainly exponential in the number of program variables. A lot of work has been proposed to reduce this complexity, often at the cost of a loss in precision.

R. Giacobazzi, J. Berdine, and I. Mastroeni (Eds.): VMCAI 2013, LNCS 7737, pp. 149–168, 2013.

First, some less expressive domains were defined to decrease computational time, such as intervals, octagons [22] and templates [24]. These domains allow to express only a certain kind of linear relations between variables: for example, the octagon abstract domain encodes relations of the kind $\pm x \pm y \leq c$ for $c \in \mathbb{R}$. In the template abstract domain, the shape of the linear relations is fixed using a predefined matrix (the so-called template), and the efficiency of the analysis lies in the use of linear programming (LP) solvers that efficiently transform a general polyhedron into a template polyhedron. However, each of these domains limits the expressiveness of the computed invariants, which limits their use to analyse complex programs for which the shape of the relations between variables is difficult to establish a priori.

The other tool to reduce the computation time of the analysis is to modify Kleene algorithm to make it faster. The most classical way to do this is to use a widening operator [8] that ensures the termination of the analysis by extrapolating the limits of the iterates. The usual widening in the polyhedra domain removes constraints that are not stable from one iteration to the other. Even if it improves computation time, widening clearly creates a large over-approximation of the result. Some techniques were developed in complement of widening to minimize this loss of accuracy. For example, widening with thresholds [19] predefines a set of thresholds whose elements are used as limit candidates at each iteration. The difficulty lies in the choice of a relevant set of thresholds, which is often computed statically before the analysis. In our previous work [5], we presented a method that dynamically computes thresholds using numerical analysis tools. In [5], we defined this technique for the interval and octagon abstract domains, it can be smoothly extended to the template polyhedra domain. In this paper, we show how it can be extended to general polyhedra.

It can be noted that the solutions proposed to improve the computation time of a polyhedral analysis can be divided into two categories: on the one hand, a restricted representation of data to facilitate their manipulation, and on the other hand techniques that improve fixpoint computation, generally using a good widening operator. In this paper, we propose a novel method to accelerate the fixpoint computation in the polyhedral domain that uses both ideas together: polyhedra are represented using support functions [17,23] and Kleene algorithm is modified using sequence transformation methods [6]. The representation of polyhedra with support functions allows to use the notion of scalar convergence [25] of a sequence of polyhedra. We show that this convergence can be used with sequence transformation methods to accelerate the fixpoint computation with a good accuracy. Our main contribution is this method which offers a good balance between efficiency and precision, depending on our chosen polyhedra representation.

The paper is organized as follows: in Section 2, we present our method on a simple example, in Section 3, we introduce some useful definitions. In Section 4, we formally define our algorithm, and in Section 5, we compare it with the analysis using template polyhedra and present some techniques to improve the

```
Assume: −20 ≤ x ≤ 20
Assume:0 ≤ y ≤ 3
while 1 do
  if y ≥ 50 ∧ x ≥ −20 then
    y=x-0.1y+60
    x=0.2x+40
  else
    x=0.5x-y-2.5
    y=0.9y+10
  end if
end while
```

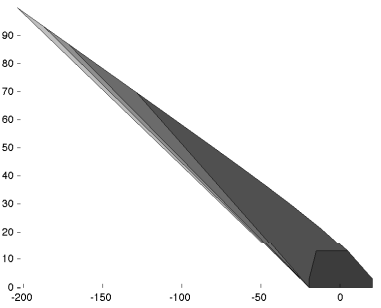


Fig. 1. The `prog` example, left, and some polyhedra obtained using Kleene iteration (iterates 1, 10, 20, 30 and 70, from dark to light), right

performance of our method. Some benchmarks are given in Section 6. Finally, we conclude with related work and some perspectives.

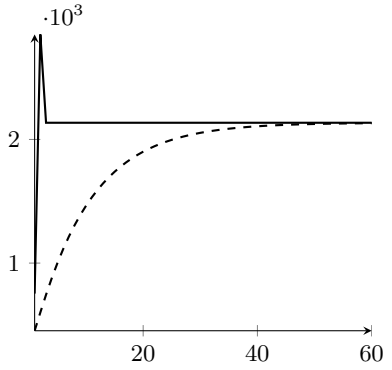
Notations. We define by $M_K(n, m)$ the set of matrices of a field K with n rows and m columns. Given $M \in M_K(n, m)$, $M_{i,j}$ is the element at the i^{th} row and j^{th} column. Given two vectors $v, w \in M_K(n, 1)$, let $\langle v, w \rangle \in K$ be the scalar product of v and w. For a set D, a sequence of elements in D will be noted $(x_k)_{k\in\mathbb{N}}$.

2 A Simple Example

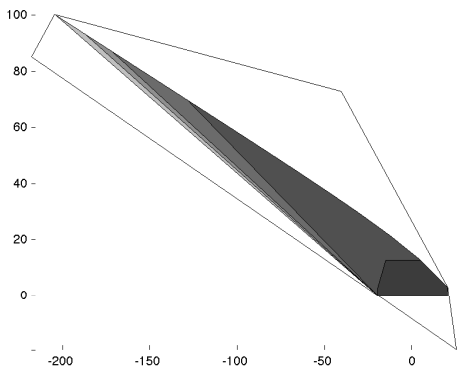
To illustrate our method, we consider the simple program presented in Figure 1 (left). It consists of a loop iterating a linear transformation, with a change if the variables enter the zone $x \geq -20$, $y \geq 50$. It can be shown that with the initial conditions $x \in [-20, 20]$ and $y \in [0, 3]$, the `then` branch of the loop is never taken. We used this condition to demonstrate the precision of our method compared to an analysis using template polyhedra.

A polyhedra based analysis of this program computes a sequence of polyhedra $(\mathbb{P}_k)_{k\in\mathbb{N}}$, some of which are depicted in Figure 1 (right). This sequence does not terminate, and widening does not help here: it discards the lower-left constraints thus computing an unbounded post-fixpoint while the least fixpoint is bounded. Note however that, in this case, the use of widening allows to prove that the program never enters the `then` branch but fails to compute a finite lower bound for variables x and y. To over-approximate the least fixpoint, our method works as follows. First, we chose a fixed set of directions Δ, i.e. a set of vectors in $\mathbb{R}^2$, and for each direction $d \in \Delta$, we extract the sequence of the values of the *support functions* of each polyhedron in this direction. As will be made clear in Section 3.2, the support function δ_P of a convex set P is a function such that $\forall d \in \mathbb{R}^2$, $\delta_P(d) = \sup\{\langle x, d\rangle : x \in P\}$. The property that we will use is that the sequence of support functions of the polyhedra $\mathbb{P}_k$ pointwise converges to the support function of the least fixpoint.

Now, let us study the values of the support functions in direction $X + 6Y$ (for example). We present in Figure 2(a) their values w.r.t. the number of iteration (dashed line). We see that this sequence slowly converges to its limit, and actually never reaches it. There exist many methods that help to quickly compute the limit of such numerical sequences. Among these methods we can use *sequence transformations* which automatically compute a new sequence that converges faster towards the same limit (even after finitely many steps in some cases). We depict in Figure 2(a) this new sequence (bold line); we clearly see that this sequence converges much faster towards its limit, after some irrelevant values due to the initialization of the process. Our method uses such a transformation to compute the limit l_d of the sequence of support functions in each direction $d \in \Delta$. Once all the limits are computed, we construct a new polyhedron using the constraints $\langle x, d \rangle \leq l_d$ in each direction and insert it into Kleene iteration. This result is shown in Figure 2(b) and is a post-fixpoint (named $\mathbb{P}^\sharp$) of the program. Remark that this post-fixpoint is precise in the sense that the vertices of the least-fixpoint touch the faces of $\mathbb{P}^\sharp$. A more precise post-fixpoint can be computed using more directions, as will be shown in Section 6. Let us also note that this post-fixpoint is sufficiently precise to show that the `then` branch of the loop is never taken, while a template based analysis with the same directions could not prove that.



(a) The sequence of value of the support function in direction $X + 6Y$ (dashed) and the accelerated sequence (thick line).



(b) The polyhedra of Kleene algorithm (in dark) and the post-fixpoint computed using our method (in white).

Fig. 2. Results of our method on the example of Figure 1

3 Backgrounds

In this article, we are interested in computing invariants for programs with n variables using Kleene iteration on the polyhedra abstract domain. We denote the variables by $x_1, \ldots, x_n$ and we let $X = (x_1, \ldots, x_n)^t$ be the vector of all variables. We thus need to compute the limit of a sequence of convex subsets of $\mathbb{R}^n$.

We start by explaining how this sequence is defined, then we show that another notion of convergence (namely the scalar convergence) can be used to compute its limit. Finally, we present some methods that work well for accelerating the convergence of sequences of real values.

3.1 Polyhedra Abstract Domain

A convex polyhedron is a subset of $\mathbb{R}^n$ defined as the intersection of finitely many half-spaces, each half-space being given as a constraint of the form $\sum_{i=1}^{n} \alpha_i x_i \leq c$ where $\forall i \in [1, n],\ \alpha_i \in \mathbb{R}$ and $c \in \mathbb{R}$. We here adopt the constraint representation of polyhedron, which is best suited for static analysis. The set of all convex polyhedra has a lattice structure with an exact intersection operator and the convex hull as union. This was one of the first numerical abstract domains used in abstract interpretation [9]. Obviously, for implementation issues, the coefficients of the constraints are usually chosen as rational numbers and the set of constraints is encoded using a matrix representation.

Definition 1 (Polyhedra Abstract Domain). *The abstract domain of convex polyhedra is the set of all pairs (A, b) where $A \in M_{\mathbb{Q}}(k, n)$ is a matrix and $b \in \mathbb{Q}^k$ is a vector of dimension k for some $k \in \mathbb{N}$. A polyhedron $\mathbb{P}$ given by the pair (A, b) represents the set of all points $(x_1, \ldots, x_n) \in \mathbb{R}^n$ such that*

$$\forall j \in [1, k],\ \sum_{i=1}^{n} A_{j,i} x_i \leq b_j \ .$$

Let C^n be the abstract domain of convex polyhedra in $\mathbb{R}^n$.

Given a polyhedron $\mathbb{P} = (A, b)$, the i^{th} constraint of $\mathbb{P}$ is $\langle A_i, X \rangle \leq b_i$ where X is the vector of variables and A_i is the i^{th} line of A.

The static analysis of a program with the polyhedra domain consists in computing the least fixpoint $\mathbb{P}_\infty$ of a monotone map $F : C^n \to C^n$ given by the program semantics. To do so, the most used method is Kleene algorithm that uses the equation $\mathbb{P}_\infty = \bigsqcup_{k \in \mathbb{N}} F^k(\bot)$, where $\bot$ is the least element of C^n. Kleene iteration can be summarized by the following algorithm:

1: $\mathbb{P}_0 := \bot$
2: **repeat**
3: $\quad \mathbb{P}_i := \mathbb{P}_{i-1} \sqcup F(\mathbb{P}_{i-1})$
4: **until** $\mathbb{P}_i \sqsubseteq \mathbb{P}_{i-1}$

So we see that an abstract interpretation based static analysis of a program consists of defining a sequence of polyhedra $(\mathbb{P}_k)_{k \in \mathbb{N}}$ and computing its order-theoretic limit $\mathbb{P}_\infty = \bigsqcup_{k \in \mathbb{N}} \mathbb{P}_k$. As Kleene algorithm may not terminate, a widening operator is used to compute an over-approximation $\mathbb{P}_w \sqsupseteq \mathbb{P}_\infty$. In this article, we show that $\mathbb{P}_\infty$ can also be computed using scalar convergence and that numerical acceleration methods can be used to quickly compute an approximation of the scalar limit.

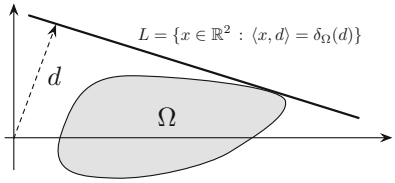


Fig. 3. Geometrical interpretation of the support function: the line L represents the limit of the half-space $H = \left\{x \in \mathbb{R}^2 : \langle x, d\rangle \leq \delta_\Omega(d)\right\}$, in which Ω lies

3.2 Convex Analysis Tools

A convenient representation of a convex set is the use of its support function [17,23]. In this section, we define the notion of support function and present how they can be computed in the case of convex polyhedron. We also introduce the notion of scalar convergence.

Definition 1 (Support Function). *Let $\Omega \subseteq \mathbb{R}^n$ be a non-empty convex set. The support function $\delta_\Omega : \mathbb{R}^n \mapsto \mathbb{R} \cup \{+\infty\}$ of Ω is given by*

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

As stated by Definition 1, the support function is defined only for the non-empty convex set. In the following, we consider only non-empty convex sets. The support function of a convex set Ω associates to each *direction* $d \in \mathbb{R}^n$ the biggest value of the scalar product $\langle x, d\rangle$ for $x \in \Omega$. The interest of this notion is that any convex set Ω is uniquely and completely determined by the values of its support function for all $d \in \mathbb{R}^n$ (see Property 1). So the set of all support functions (i.e. the set of positive homogeneous, convex real valued functions over $\mathbb{R}^n$) is isomorphic to the set of convex sets over $\mathbb{R}^n$.

Property 1. *Let Ω be a non-empty convex set and δ_Ω be its support function. We have:*

$$\Omega = \bigcap_{d \in \mathbb{R}^n} \{x \in \mathbb{R}^n : \langle x, d\rangle \leq \delta_\Omega(d)\}\ .$$

The value of δ_Ω in a direction $d \in \mathbb{R}^n$ might be infinite depending on whether Ω is bounded in this direction. Figure 3 gives a geometrical interpretation of δ_Ω: in the direction d, $\delta_\Omega(d)$ defines the smallest half-space (delimited by the thick line L) that contains Ω. When Ω is a polyhedron represented by its constraints system, its support function can be obtained using *linear programming*.

Let now $(\Omega_k)_{k\in\mathbb{N}}$ be a sequence of convex sets, with $\Omega_k \subseteq \mathbb{R}^n$ for all $k \in \mathbb{N}$. From a topological point of view, there are many ways to define the convergence of the sequence $(\Omega_k)_{k\in\mathbb{N}}$ [21], each of them leading to a (possibly different) limit. We choose to use the notion of scalar-convergence because it is based on support function and because it corresponds to the notion of limit used in abstract interpretation when the sequence $(\Omega_k)_{k\in\mathbb{N}}$ is monotone.

Definition 2 (Scalar-convergence). *Let $(\Omega_k)_{k\in\mathbb{N}}$ be a sequence of convex sets. For each $k \in \mathbb{N}$, let δ_{Ω_k} be the support function of Ω_k. Let Ω be a convex set and let δ_Ω be its support function. We say that $(\Omega_k)_{k\in\mathbb{N}}$ scalar-converges (or S-converges) to Ω, denoted by* $\text{s-lim}_{k\to+\infty}\,\Omega_k = \Omega$, *iff*

$$\forall d \in \mathbb{R}^n,\ \lim_{k\to+\infty}(\delta_{\Omega_k}(d)) = \delta_\Omega(d)\ .$$

The S-convergence defines the limit of a sequence of convex sets $(\Omega_k)_{k\in\mathbb{N}}$ via infinitely many limits of numerical sequences $(\delta_{\Omega_k}(d))_{k\in\mathbb{N}}$, for all $d \in \mathbb{R}^n$. Property 2 shows that the S-convergence of an increasing sequence is the supremum of its elements.

Property 2. *Let $(\Omega_k)_{k\in\mathbb{N}}$ be a sequence of closed convex sets and let cl be the convex closure function. If we have that $\forall k \in \mathbb{N},\ \Omega_k \subseteq \Omega_{k+1}$, then*

$$\text{s-lim}_{k\to\infty}\,\Omega_k = cl(\bigcup_{k\geq 0}\Omega_k)\ .$$

Recall that, as defined in Section 3.1, Kleene algorithm computes program invariants as the union of a sequence of convex polyhedra. These polyhedra form an increasing sequence, so Property 2 shows that the S-convergence can be used to compute the result of Kleene iteration. We use this idea in Section 4 to define an accelerated version of Kleene algorithm for convex polyhedra.

3.3 Numerical Analysis Tools

In this section, we present techniques called sequence transformations that are used to quickly compute the limit of a numerical sequence of real numbers. These techniques were already used in [5] to accelerate the convergence of the fixpoint computation for the box or octagon abstract domains. We recall the basic notions of sequence transformation that are needed to understand our framework, a good review on the theory and applications of these methods can be found in [6].

We equip $\mathbb{R}$ with the euclidean distance and define $\mathbf{Seq}(\mathbb{R})$ as the set of all sequences of real numbers (i.e. functions from $\mathbb{N}$ to $\mathbb{R}$). We say that a sequence $(x_k)_{k\in\mathbb{N}} \in \mathbf{Seq}(\mathbb{R})$ converges towards $x \in \mathbb{R}$, denoted by $\lim_{k\to\infty} x_k = x$, if $\lim_{k\to\infty} |x_k - x| = 0$. More formally, we have:

$$\lim_{k\to\infty} x_k = x \Leftrightarrow \forall\varepsilon > 0,\ \exists K \in \mathbb{N} : \forall k \geq K,\ |x_k - x| \leq \varepsilon\ .$$

Given two sequences $(x_k)_{k\in\mathbb{N}}$ and $(y_k)_{k\in\mathbb{N}}$ with the same limit ℓ, we say that (y_k) converges *faster* to ℓ than (x_k) if $\lim_{k\to\infty}\left(\frac{y_k-l}{x_k-l}\right) = 0$. The goal of sequence transformations is to automatically compute, from a slowly converging sequence (x_k), a sequence (y_k) that converges towards the same limit faster than (x_k). In this way, we can use (y_k) to quickly obtain an approximation of the limit of (x_k). This is formally stated in Definition 3.

Definition 3. *A* sequence transformation *is a function* $T : \mathbf{Seq}(\mathbb{R}) \to \mathbf{Seq}(\mathbb{R})$ *such that, for all* converging *sequences* $(x_k)_{k\in\mathbb{N}} \in \mathbf{Seq}(\mathbb{R})$*, the sequence* (y_k) *defined by* $(y_k) = T(x_k)$ *is convergent with* $\lim_{k\to\infty} y_k = \lim_{k\to\infty} x_k$*. The sequence* (y_k) *is said to be* accelerated *if* $\lim_{k\to\infty} \frac{y_k - l}{x_k - l} = 0$*.*

An example of a sequence transformation is the well-known Δ^2-Aitken transformation $\Theta : \mathbf{Seq}(\mathbb{R}) \to \mathbf{Seq}(\mathbb{R})$, defined by:

$$\forall \mathbf{x} \in \mathbf{Seq}(\mathbb{R}),\ \forall k \in \mathbb{N},\ \Theta(\mathbf{x})_k = x_k - \frac{(x_{k+1} - x_k)^2}{x_k - 2x_{k+1} + x_{k+2}}\ .$$

We apply this transformation method to $\mathbf{x}_k = 1 + \frac{1}{k+1}$ $(\forall k \in \mathbb{N})$, a sequence that converges to 1. The result is given in the following table, where $(\Theta(\mathbf{x})_k)_{k\in\mathbb{N}}$ converges faster than $(\mathbf{x}_k)_{k\in\mathbb{N}}$ toward 1.

$\mathbf{x}_k$	2.00	1.5	1.33	1.25	1.2	1.16	1.14	1.125	1.11
$\Theta(\mathbf{x})_k$		1.25	1.16	1.12	1.10	1.08	1.07	1.06	

For more details on Δ^2-Aitken transformation see [7].

Obviously, a sequence transformation does not accelerate all converging sequences, i.e. $(T(x_k))_{k\in\mathbb{N}}$ does not always converge faster than $(x_k)_{k\in\mathbb{N}}$. Remark however that it is required that $T(x_k)$ still converges towards the same limit. An important notion is the kernel of a sequence transformation, which is the set of all sequences $\mathbf{x} \in \mathbf{Seq}(\mathbb{R})$ such that $T(\mathbf{x})$ is ultimately constant, see Definition 4.

Definition 4. *Let* $T : \mathbf{Seq}(\mathbb{R}) \to \mathbf{Seq}(\mathbb{R})$ *be a sequence transformation. The* kernel *of* T*, denoted by* $K(T) \subseteq \mathbf{Seq}(\mathbb{R})$*, is the set of sequences defined by:*

$$\forall \mathbf{x} \in \mathbf{Seq}(\mathbb{R}),\ \mathbf{x} \in K(T) \Leftrightarrow \mathbf{y} = T(\mathbf{x})\ \textit{and}\ \exists n \in \mathbb{N} : \forall k \geq n,\ y_k = y_n\ .$$

So clearly, sequences in the kernel of a transformation T are very interesting because we can compute their limit in a finite time by looking at the elements of the accelerated sequence. However, computing exactly the kernel of a sequence transformation is very complicated [7]. In our experimentations, we used the ϵ-algorithm [26], which is often cited as the best general purpose acceleration algorithm. The ϵ-algorithm is a generalization of the Δ^2-algorithm that is less sensible to numerical instability. For the sake of conciseness, we do not present it, let us just mention that its kernel contains a large class of sequences like convergent linear sequences and *totally monotonic* ones [6, Chap. 2, pp. 85–91].

4 The Acceleration Process

In this section, we present our main contribution which is a new fixpoint algorithm on the polyhedra abstract domain. For that, we define in Section 4.1 an accelerated version of the S-convergence, called *the accelerated S-convergence*. In section 4.2, we propose an abstract version of the accelerated S-convergence which is used in Section 4.3 for the fixpoint computation.

4.1 The Accelerated S-Convergence

In this section, we show how support function can be combined with sequence transformations to accelerate the s-convergence of convex polyhedra sequences. The method we develop is called the *accelerated S-convergence*.

Now, let Ω be a convex set and $(\mathbb{P}_k)_{k\in\mathbb{N}}$ be a sequence of polyhedra such that $\text{s-lim}_{k\to+\infty}\mathbb{P}_k = \Omega$. We want to compute Ω in a fast way. Let $\delta_{\mathbb{P}_k}$ be the support functions of $\mathbb{P}_k$ for all $k \in \mathbb{N}$. We put $\forall d \in \mathbb{R}^n,\ \forall k \in \mathbb{N},\ S_k^d = \delta_{\mathbb{P}_k}(d)$. From Definition 2, we have that if $\lim_{k\to+\infty} S_k^d = S_d$, then $S_d = \delta_\Omega(d)$. It means that $\Omega = \bigcap_{d\in\mathbb{R}^n}\{x \in \mathbb{R}^n : \langle x, d\rangle \leq S_d\}$. So, the S-limit of $(\mathbb{P}_k)_{k\in\mathbb{N}}$ can be defined using the limit of numerical sequences $(S_k^d)_{k\in\mathbb{N}}$, for all $d \in \mathbb{R}^n$.

Property 3. *Let $(\mathbb{P}_k)_{k\in\mathbb{N}}$ be a convex polyhedra sequence, and $\delta_{\mathbb{P}_k}$ be the support function of each $\mathbb{P}_k$.*

$$\textit{If } (\forall d \in \mathbb{R}^n), \lim_{k\to+\infty}\delta_{\mathbb{P}_k}(d) = S_d \textit{ then } \text{s-lim}_{k\to+\infty}\mathbb{P}_k = \bigcap_{d\in\mathbb{R}^n} H_d$$

where $H_d = \{x \in \mathbb{R}^n : \langle x, d\rangle \leq S_d\}$ is a supporting hyperplane of the S-limit of $(\mathbb{P}_k)_{k\in\mathbb{N}}$.

Property 3 shows that it is possible to use numerical methods of Section 3.3 to accelerate the computation of the S-limit of $(\mathbb{P}_k)_{k\in\mathbb{N}}$. Let T be a sequence transformation as presented in Section 3.3. We compute the sequence $(T(S_k^d))_{k\in\mathbb{N}}$ for all $d \in \mathbb{R}^n$, we assume that the sequence $(S_k^d)_{k\in\mathbb{N}}$ belongs to the kernel of T (see Definition 4). So $(T(S_k^d))_{k\in\mathbb{N}}$ converges faster than $(S_k^d)_{k\in\mathbb{N}}$ towards S_d, thus accelerating the computation of Ω. This is stated by Definition 5.

Definition 5 (Accelerated S-convergence). *Let $(\mathbb{P}_k)_{k\in\mathbb{N}}$ be a convex polyhedra sequence. For each $k \in \mathbb{N}$, let $\delta_{\mathbb{P}_k}$ be the support function of $\mathbb{P}_k$, and let T be a sequence transformation. The accelerated S-convergence, noted* $\text{s}_\text{A}\text{-lim}$*, is defined as:*

$$\text{s}_\text{A}\text{-lim}_{k\to\infty}\mathbb{P}_k = \bigcap_{d\in\mathbb{R}^n}\left\{x \in \mathbb{R}^n \,:\, \langle x, d\rangle \leq \lim_{k\to+\infty} T(\delta_{\mathbb{P}_k}(d))\right\}\ .$$

In particular, we have $\text{s-lim}_{k\to+\infty}\mathbb{P}_k = \text{s}_\text{A}\text{-lim}_{k\to\infty}\mathbb{P}_k$.

In practice, the s_A-lim of $(\mathbb{P}_k)_{k\in\mathbb{N}}$ cannot be used because we must compute $\lim_{k\to+\infty} T(\delta_{\mathbb{P}_k}(d))$ for all d in $\mathbb{R}^n$. We can easily prove that this set can be restricted to directions in the unit ball $\mathbb{B}^n$, but then the accelerated S-limit still required infinitely many limit computations. In Section 4.2, a finite abstraction of the accelerated S-convergence, called *the abstract S-convergence*, is defined.

4.2 The Abstract S-Convergence

Let Ω be a convex set, and $\mathbb{B}^n \subseteq \mathbb{R}^n$ be the unit ball. Using support function properties, we have that $\forall d \in \mathbb{B}^n,\ \Omega \subseteq H_d$ with $H_d = \{x \in \mathbb{R}^n : \langle x, d\rangle \leq \delta_\Omega(d)\}$. In particular, Ω is included in every intersection of finitely many supporting

hyperplanes. So we can over-approximate Ω using a finite set of directions and computing each supporting hyperplane in these directions. Property 4 presents that.

Property 4. *For a convex set Ω, we have :*

- $(\forall \Delta \subseteq \mathbb{B}^n), \Omega \subseteq \bigcap_{d_i \in \Delta} \{x \in \mathbb{R}^n : \langle x, d_i \rangle \leq \delta_\Omega(d_i)\}$.
- *If* $(\Delta = \mathbb{B}^n)$ *then* $\Omega = \bigcap_{d_i \in \Delta} \{x \in \mathbb{R}^n : \langle x, d_i \rangle \leq \delta_\Omega(d_i)\}$.

In the sequel, we define the set $\Lambda = \mathcal{P}(\mathbb{B}^n)$, Λ is the power set of $\mathbb{B}^n$. $(\Lambda, \subseteq_\Lambda)$ forms a complete lattice with $\bot = \emptyset$, $\top = \mathbb{B}^n$, $\subseteq_\Lambda, \cup_\Lambda$ and $\cap_\Lambda$ being the usual set operations.

The abstract S-convergence applies the accelerated S-convergence on an element Δ of the lattice $(\Lambda, \subseteq_\Lambda)$ to compute an over-approximation of the limit of $(\mathbb{P}_k)_{k \in \mathbb{N}}$. The idea is to apply the accelerated S-convergence partially using directions in Δ. This is defined in Definition 6.

Definition 6 (The Abstract S-convergence). *Let $(\Lambda, \subseteq_\Lambda)$ be the lattice of direction sets. The abstract S-convergence of a sequence $(\mathbb{P}_k)_{k \in \mathbb{N}} \subseteq C^n$, noted $\mathrm{s}^\sharp_\mathrm{A}$-lim, is a function from $\Lambda \times (\mathbb{N} \to C^n)$ to C^n, such that:*

$$\forall \Delta \in \Lambda,\ \mathrm{s}^\sharp_\mathrm{A}\text{-lim}(\Delta, \mathbb{P}_k) = \bigcap_{d_i \in \Delta} \left\{x \in \mathbb{R}^n : \langle x, d_i \rangle \leq \lim_{k \to +\infty} T(\delta_{\mathbb{P}_k}(d_i))\right\}$$

where T is a sequence transformation and $\delta_{\mathbb{P}_k}$ are the support functions of $\mathbb{P}_k$.

Now, we can consider *the abstract S-convergence* as a finite approximation of the accelerated S-convergence, if the chosen direction set Δ is finite. As stated by Property 4, it computes an over-approximation of the S-limit of a polyhedra sequence.

Property 5. *For a sequence $(\mathbb{P}_k)_{k \in \mathbb{N}} \subseteq C^n$, we have that, if $\mathrm{s}_\mathrm{A}\text{-lim}_{k \to +\infty}(\mathbb{P}_k) = \Omega$ then:*

- $(\forall \Delta \in \Lambda), \Omega \subseteq \mathrm{s}^\sharp_\mathrm{A}\text{-lim}(\Delta, \mathbb{P}_k) = \Omega^\sharp$, *where $\Omega^\sharp$ is the best abstraction of Ω using the direction set Δ, i.e.* $\Omega^\sharp = \bigcap_{d_i \in \Delta} \left\{x \in \mathbb{R}^n : \langle x, d_i \rangle \leq \delta_\Omega(d_i)\right\}$.
- $(\forall \Delta_1, \Delta_2 \in \Lambda)$, *if* $\Delta_1 \subseteq_\Lambda \Delta_2$ *then* $\Omega \subseteq \mathrm{s}^\sharp_\mathrm{A}\text{-lim}(\Delta_2, \mathbb{P}_k) \subseteq \mathrm{s}^\sharp_\mathrm{A}\text{-lim}(\Delta_1, \mathbb{P}_k)$

Informally Property 5 says that the more directions we have, the more precise the result will be. In the case where Ω is a polyhedron, there exists a minimal set $\Delta_\Omega \subseteq \Lambda$, such that $\Omega = \cap_{d \in \Delta_\Omega} \{x \in \mathbb{R}^n : \langle x, d \rangle \leq \delta_\Omega(d)\}$. This Δ_Ω represents the set of all normal vectors of the constraints of Ω. However, these constraints are generally unknown, so we do not know Δ_Ω. Even worse, when Ω is not a polyhedron, there is no finite set Δ which is optimal to compute $\mathrm{s}^\sharp_\mathrm{A}\text{-lim}(\Delta, \mathbb{P}_k)$ (because there is no best abstraction of a general convex set into the abstract domain of polyhedra). The efficiency and precision of our method depends on the choice of a relevant set Δ. We discuss this choice in Section 5.2.

Algorithm 1. Accelerated Kleene Algorithm.

Input: $\Delta \in \Lambda$, finite
Input: $\varepsilon > 0$
1: $\mathbb{P}_0 := \bot$
2: $i := 1$
3: $\mathbb{P}^\sharp := \{\}$ *//Initialize $\mathbb{P}^\sharp$ to the empty sequence*
4: **repeat**
5: $\quad \mathbb{P}_i := \mathbb{P}_{i-1} \sqcup F^\sharp(\mathbb{P}_{i-1})$
6: $\quad \mathbb{P}^\sharp = \mathbb{P}^\sharp \cup \{\mathbb{P}_i\}$ *//Add the result of the iteration $\mathbb{P}_i$ to $\mathbb{P}^\sharp$*
7: $\quad \mathbb{P}^\sharp_{\infty(i)} := \mathrm{s}^\sharp_{\mathrm{A}}\text{-lim}\left(\Delta, \mathbb{P}^\sharp\right)$
8: $\quad$ **if** $(\text{distance}(\mathbb{P}^\sharp_{\infty(i)}, \mathbb{P}^\sharp_{\infty(i-1)}) \leq \varepsilon)$ **then**
9: $\quad\quad \mathbb{P}_i := \mathbb{P}^\sharp_{\infty(i)}$
10: $\quad$ **end if**
11: $\quad i := i + 1$
12: **until** $\mathbb{P}_i \sqsubseteq \mathbb{P}_{i-1}$

4.3 The Accelerated Kleene Iteration Algorithm

In this section, we use *the abstract S-convergence* with Kleene iteration to accelerate fixpoint computation. This improvement proposes a trade-off between precision and computation time by including more directions in the set Δ used for the abstract S-convergence. If we run the Kleene algorithm with the polyhedra abstract domain, the collection of successive iterates forms a sequence of convex polyhedra, noted $(\mathbb{P}_k)_{k\in\mathbb{N}}$, such that:

$$\begin{cases} \mathbb{P}_0 & = \bot \\ \mathbb{P}_{k+1} & = \mathbb{P}_k \sqcup F^\sharp(\mathbb{P}_k), (\forall k \in \mathbb{N}) \end{cases}$$

We assume that $\forall k \geq 1, \mathbb{P}_k \neq \bot$.

The abstract semantic function $F^\sharp$ is monotone by definition and $(\mathbb{P}_k)_{k\in\mathbb{N}}$ is an increasing sequence, i.e. $\forall k \in \mathbb{N},\ \mathbb{P}_k \sqsubseteq \mathbb{P}_{k+1}$. As stated in Property 2, the S-limit of an increasing sequence is the convex hull of its elements, so for the sequence of Kleene iterates, the S-limit of $(\mathbb{P}_k)_{k\in\mathbb{N}}$ is the least fixpoint of $F^\sharp$, denoted $\mathbb{P}_\infty$. So we have:

$$\begin{aligned} & \mathbb{P}_\infty = \ \text{s-lim}_{k\to+\infty}\, \mathbb{P}_k \\ \text{So} \quad & \mathbb{P}_\infty = \mathrm{s}_{\mathrm{A}}\text{-lim}_{k\to+\infty}\, \mathbb{P}_k \quad \text{(By transitivity).} \\ \text{Thus } (\forall \Delta \in \Lambda), & \mathbb{P}_\infty \sqsubseteq \ \mathrm{s}^\sharp_{\mathrm{A}}\text{-lim}(\Delta, \mathbb{P}_k) \quad \text{(Using Property 5).} \end{aligned}$$

This shows that we can compute an over-approximation of $\mathbb{P}_\infty$ using the new notion of convergence introduced in Section 4.2. Note that the quality of the over-approximation depends on the choice of the direction set Δ.

In Algorithm 1, we define a new accelerated Kleene algorithm, which is the standard Kleene algorithm combined with the abstract S-convergence. The main idea is to compute in parallel the sequence $(\mathbb{P}_k)_{k\in\mathbb{N}}$ and its $\mathrm{s}^\sharp_{\mathrm{A}}$-lim. Once this limit is computed, we use it as a fixpoint candidate.

Using a direction set Δ given as an input, in each iteration Algorithm 1 computes the abstract element $\mathbb{P}_i$ and puts it as a new element of $\mathbb{P}^\sharp$. We have that $\mathbb{P}^\sharp_{\infty(i)}$ is the result of the abstract S-convergence applied on $\mathbb{P}^\sharp$ and Δ. Collecting these results, we obtain the accelerated sequence, called $(\mathbb{P}^\sharp_{\infty(k)})_{k\in\mathbb{N}}$. So we construct simultaneously $\mathbb{P}^\sharp$ and $(\mathbb{P}^\sharp_{\infty(k)})_{k\in\mathbb{N}}$. When the algorithm detects that the sequence $(\mathbb{P}^\sharp_{\infty(k)})_{k\in\mathbb{N}}$ stabilizes, we assume that it is close to its limit and thus we use the last element of $(\mathbb{P}^\sharp_{\infty(k)})_{k\in\mathbb{N}}$, noted $\mathbb{P}^\sharp_{\infty(i)}$, as a relevant threshold, i.e. we modify the current Kleene iterate $\mathbb{P}_i$ to be $\mathbb{P}^\sharp_{\infty(i)}$. Thanks to the properties of the abstract S-convergence, this threshold is obtained after a few iterations, and it is a good approximation of the fixpoint.

Algorithm 1 detects the stabilization of the sequence $(\mathbb{P}^\sharp_{\infty(k)})_{k\in\mathbb{N}}$ by computing the distance between two successive elements. The distance d_Δ we use is given by:

$$\forall \mathbb{P}_1, \mathbb{P}_2 \in C^n,\ d_\Delta(\mathbb{P}_1, \mathbb{P}_2) = \sup_{d_i \in \Delta} |\delta_{\mathbb{P}_1}(d_i) - \delta_{\mathbb{P}_2}(d_i)| \ .$$

Clearly if Δ is finite (or $\Delta \subset \mathbb{B}^n$), d_Δ is not a distance on C^n. In particular, there exist infinitely many pairs $(\mathbb{P}, \mathbb{P}')$, $\mathbb{P} \neq \mathbb{P}'$, with $d_\Delta(\mathbb{P}, \mathbb{P}') = 0$. However, the sequence $(\mathbb{P}^\sharp_{\infty(k)})_{k\in\mathbb{N}}$ is made of polyhedra whose directions are given by the set Δ, i.e. they are template polyhedra. The function d_Δ is a distance between template polyhedra and can be used to detect the stabilization of the sequence. Moreover, it can be computed in linear time as the support function in a direction d_i are just the inhomogeneous term of the associated constraint. So we say that the sequence $(\mathbb{P}^\sharp_{\infty(k)})_{k\in\mathbb{N}}$ has stabilized when $d_\Delta(\mathbb{P}^\sharp_{\infty(i)}, \mathbb{P}^\sharp_{\infty(i-1)}) \leq \varepsilon$, where $\varepsilon > 0$ is a user-defined parameter (usually around 10^{-3}).

We can prove that Algorithm 1 terminates when $\mathbb{P}^\sharp$ can be accelerated by $\mathrm{s}^\sharp_\mathrm{A}$-lim, i.e. when $\mathbb{P}^\sharp$ belongs to the kernel of the sequence transformation used to compute $\mathrm{s}^\sharp_\mathrm{A}$-lim. Note that practical experiments show that many sequences are accelerated even if they are not in the kernel, so we believe that Algorithm 1 can be efficiently used for many types of programs. However, it's hard to establish a priori if a sequence will be accelerated, and we know that no method accelerates all convergent sequences [12]. To overcome this, we combine our method with widening by replacing lines 6 to 10 of Algorithm 1 by:

if $i \leq nbDelay$ **then**
  $\mathbb{P}^\sharp = \mathbb{P}^\sharp \cup \{\mathbb{P}_i\}$
  $\mathbb{P}^\sharp_{\infty(i)} := \mathrm{s}^\sharp_\mathrm{A}\text{-lim}\left(\Delta, \mathbb{P}^\sharp\right)$
  if (distance$(\mathbb{P}^\sharp_{\infty(i)}, \mathbb{P}^\sharp_{\infty(i-1)}) \leq \varepsilon$) **then**
    $\mathbb{P}_i := \mathbb{P}^\sharp_{\infty(i)}$
  end if
else
  $\mathbb{P}_i := \mathbb{P}_{i-1} \nabla \mathbb{P}_i$ *// The widening operator.*
end if

The idea is similar to widening with delay: we apply the accelerated S-convergence during the first $nbDelay$ iterations. If the computation doesn't terminate we use the widening to force the termination. In our experiments, however, we did not need to use the widening to make the analysis terminate. Note that in the case where $s^\sharp_A$-lim accelerates sequences obtained in some directions and not the others, a way to improve this algorithm is to use the polyhedron defined by the accelerated directions as a threshold for widening. This allows to keep the information obtained by $s^\sharp_A$-lim and thus improves the precision of the widening result. Note that other widening techniques, as defined in [2], can also be smoothly combined with our technique.

5 Performance

The performance of the method presented in this paper mainly depends on two parameters. First, the choice of the transformation method to accelerate the sequences convergence is important. Clearly, each transformation accelerates very well sequences in its kernel (see definition in Section 3.3), so we must choose an algorithm with a large kernel. In our experimentations, we used the ε-algorithm. Second and mainly, our method depends on the choice of directions used to compute the abstract S-limit. We discuss this choice in Section 5.2. The direction set used in our technique can be seen as a template defined in [24]. We next emphasize this comparison and the differences between both methods.

5.1 Comparison with Template Abstract Domain

The template abstract domain [24] represents a derivation of the polyhedra domain in which the polyhedra are restricted to have a fixed shape. This shape is defined by a $n \times m$ matrix, called template constraint matrix (TCM), where n is the number of program variables and m the number of constraints used to define the polyhedra shape. The analogue of the TCM in our method is the direction set Δ: each line of a TCM is a direction, so clearly the set of all TCMs is equivalent to the lattice Λ of direction sets. Given a TCM Γ, we denote by T_Γ the template domain with template Γ and by $\Delta_\Gamma \in \Lambda$ the equivalent direction set; we want to compare the fixpoint we obtain using Kleene iteration in the T_Γ and $s^\sharp_A$-lim$(\Delta_\Gamma, \mathbb{P}_k)$.

Let Γ be a TCM, and $\alpha_\Gamma : C^n \to T_\Gamma$ be the abstraction function for the template domain, such that $\forall \mathbb{P} \in C^n$ represented as a conjunction of constraints of the form (A,b) (see Definition 1), $\alpha_\Gamma(\mathbb{P}) = \cap_{d_i \in \Gamma}\{x |\, \langle x, d_i\rangle \leq c_i\}$, where c_i is the solution of the following problem: min $b^T\lambda$ s.t $\lambda \geq 0 \wedge A^T\lambda = d_i$. Note that $c_i = \delta_\mathbb{P}(d_i)$, so $\alpha_\Gamma(\mathbb{P})$ can be defined as:

$$\alpha_\Gamma(\mathbb{P}) = \cap_{d_i \in \Gamma}\{x |\, \langle x, d_i\rangle \leq \delta_\mathbb{P}(d_i)\} \ .$$

Let now P be a program and $F^\sharp$ be its abstract semantic function in the polyhedra domain. An analysis of P in the template domain computes the invariant $\mathbb{P}^t_\infty = \sqcup^t \mathbb{P}^t_k$ such that:

$$\begin{cases} \mathbb{P}^t_0 & = \alpha_\Gamma(\mathbb{P}_0) \\ \mathbb{P}^t_{k+1} & = \mathbb{P}^t_k \sqcup^t \alpha_\Gamma(F^\sharp(\mathbb{P}_k)),\ \forall k \in \mathbb{N} \end{cases} .$$

In other words, it performs the standard Kleene iteration, but abstracts each intermediate value into the template abstract domain. Here $\sqcup^t$ is the union in T_Γ which is very easy to compute [24].

Let $(\mathbb{P}_k)_{k\in\mathbb{N}}$ be the sequence computed by Kleene iteration in the polyhedra domain. It is easy to prove by induction that $\forall k \in \mathbb{N},\ \alpha_\Gamma(\mathbb{P}_k) \sqsubseteq \mathbb{P}^t_k$. So $\sqcup^t_{k\in\mathbb{N}}\alpha_\Gamma(\mathbb{P}_k) \sqsubseteq \sqcup_{k\in\mathbb{N}}\mathbb{P}^t_k = \mathbb{P}^t_\infty$. As α_Γ is continuous, we know that:

$$\alpha_\Gamma\big(\sqcup_{k\in\mathbb{N}}\mathbb{P}_k\big) \sqsubseteq \mathbb{P}^t_\infty. \tag{1}$$

Let $\mathbb{P}_\infty = \sqcup_{k\in\mathbb{N}}\mathbb{P}_k$ be the least fixpoint of $F^\sharp$ in the polyhedra domain. From the definitions of α_Γ and $\text{s}^\sharp_\text{A}\text{-lim}(\Delta_\Gamma, \mathbb{P}_k)$, we can easily prove that:

$$\text{s}^\sharp_\text{A}\text{-lim}(\Delta_\Gamma, \mathbb{P}_k) = \alpha_\Gamma(\mathbb{P}_\infty)\ . \tag{2}$$

From Equation 1 and 2, we obtain that $\text{s}^\sharp_\text{A}\text{-lim}(\Delta_\Gamma, \mathbb{P}_k) \sqsubseteq \mathbb{P}^t_\infty$.

It means that, using the same TCM, the result of our method is more precise than the one obtained with template abstract domain. The cause is that, in the template case, all the analysis is made in a less expressive domain, so some over-approximation is added at each iteration. In our method, the over-approximation is done once, when the result of the abstract S-convergence is injected in the Kleene iteration to accelerate its termination. From Equation 2, our method automatically computes the best abstraction of the fixpoint in the template domain. The use of numerical acceleration methods allows to compute it without having to compute the fixpoint itself.

5.2 Discussion on Direction Set

Given a direction set Δ, the abstract S-convergence computes the template abstraction of the least fixpoint computed by Kleene iterates. So clearly, the choice of Δ has a major influence on the quality of the abstraction. Moreover, as we want to stabilize every sequence $\delta_{\mathbb{P}_k}(d)$ for all $d \in \Delta$, the choice of Δ also influences the performance of the algorithm as some sequences will be less likely to be accelerated. However, there is no best direction set and choosing a good one is a difficult problem [24]. We mainly have two methods to choose the directions.

Uniformly Distributed Directions. First, one can choose directions that are uniformly distributed on a surface of the n-dimensional sphere (n represents the program dimension). This technique guarantees that the entire space is covered, so that if the limit of Kleene iterates is bounded, our method computes a bounded polyhedra as well. This technique is also used in [14], where support functions are used to represent convex sets for hybrid systems analysis. This technique however does not consider the dynamics of the program to choose the directions and is thus often not optimal.

Using Partial Traces. A better solution is to use and extend the work of [1] where statistical tools, namely *principal component analysis* (or PCA), are used to refine the box abstract domain by changing the axis in which boxes are defined. PCA is a commonly used tool in statistics for analysing data. Given a set of points in a n-dimensional space, PCA computes an orthogonal coordinate system that better represents the points, i.e. the new axes maximize the variance of the projection of the initial values. Then, we can remove the most extreme points and we obtain new axes, that are not orthogonal to the first, and that best represent the reduced set of points. Iterating this idea, we obtain a set of directions that contain many information on how the points are distributed in space.

In our case, we generate points by computing partial evaluation traces of the program and collecting all the values of the variables. After the PCA analysis, we obtain a direction set Δ with which we can perform our static analysis as in Algorithm 1. We present in Section 6 the results we obtain using this technique, which are very encouraging and our future work will consist in applying this PCA analysis dynamically to discover relevant directions.

5.3 Case of Affine Programs

The main bottleneck of our algorithm is the fact that we must compute the polyhedra given by Kleene iteration before computing the support functions in the chosen directions. For programs with many variables, this quickly becomes impossible as Kleene iteration is very time consuming. When the program iterates an affine transformation (i.e. when the semantic function F is of the form $F(X) = AX + b$, with A a matrix and b a vector), we can overcome this problem by directly computing the value of the support function of $\mathbb{P}_i$ in each direction without computing $\mathbb{P}_i$, using ideas from [4]. We briefly describe this method here. On the one hand, using Kleene algorithm and the semantic function F, $\mathbb{P}_i$ is obtained by :

$$\mathbb{P}_i = \mathbb{P}_{i-1} \sqcup F(\mathbb{P}_{i-1}). \tag{3}$$

On the other hand, the support functions of convex sets can be computed efficiently using the following operations. For any convex sets $S, S' \subseteq C^n$, we have:

- $\forall M \in M_{\mathbb{R}}(n,m),\ \delta_{MS}(d) = \delta_S(M^T d)$.
- $\delta_{S \sqcup S'}(d) = \max(\delta_S(d), \delta_{S'}(d))$.
- $\delta_{S \oplus S'}(d) = \delta_S(d) + \delta_{S'}(d)$.

In these formula, MS denotes the transformation of S by M, such that $MS = \{Mx | x \in S\}$ and $S \oplus S'$ denotes the Minkowski sum: $S \oplus S' = \{x + x' \mid x \in S,\ x' \in S'\}$. Using these properties and Equation 3, the support function of $\mathbb{P}_i$, for a given direction set Δ, can be computed as follow:

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

This can be generalized to:

$$\delta_{\mathbb{P}_i}(d) = \max\left(\delta_{\mathbb{P}_{init}}(d), \delta_{\mathbb{P}_{init}}(A^{Tj}d) + \sum_{k=1}^{j}\langle b, A^{T(k-1)}d\rangle, j = 1, \ldots, i\right). \quad (4)$$

Note that in Equation 4, the support function of $\mathbb{P}_i$ are obtained using only support function of $\mathbb{P}_{init}$, which is the polyhedron obtained just before the execution of the loop, i.e. it is often the polyhedron representing the initialized variables. This allows us to compute efficiently $\delta_{\mathbb{P}_i}$ without having to deal with the complexity of the polyhedra abstract domain operations.

Remark that the technique presented in this section allows for a very efficient fixpoint computation for affine loops. Such loops are very common in embedded systems (for example linear filters are such loops) and are difficult to analyze. In particular, other acceleration techniques such as [15] are not able to handle affine loops, they can only compute the fixpoint for translations loop only. Our technique is much more general.

6 Experimentation

In this section, we apply Algorithm 1, presented so far, on some benchmark programs. We have implemented our framework on top of **Apron** [18] using the Parma Polyhedra Library (**PPL**) [3]. The experimentations are done on 2.4GHz Intel Core2 Duo laptop, with 8Gb of RAM.

Benchmarks Programs. Next, we present the results of our method on a collection of programs[1] implementing digital filters that are known to be hard to analyse using the standard polyhedra analysis. We used two filters inspired by [13] (named `filter1` and `filter2`) and five filters from the tests of the "Filter Verification Framework" software [11]. We choose filters of order 2 (`lp_iir_9600_2`) to 10 (`bs_iir_9600_12000_10_chebyshev`) to study how our method scales with the number of variables (a filter of order n has $2n + 2$ variables).

Comparison with Classical Kleene Iteration. Our benchmarks contain infinite loops without guards: in this case, it is hard to define thresholds for widening statically as techniques defined in [19] for example do not apply. So we analyse these programs using widening with delay (and no thresholds) on polyhedra abstract domain, with a delay of 15 iterations. The results are compared with ones obtained with our method. These results are given in Figure 4. In the "Program" column, $|V|$ denotes the number of variables of the program and $|\Delta|$ the number of chosen directions. In this table, "Yes" means that the analysis reaches a bounded fixpoint and $\top$ means an unbounded fixpoint. In this case, we give the execution time t. The sign $-$ means that the analysis did not terminate (with a time-out after 5 minutes). The results of Figure 4 shows that our method converges for all programs

[1] All programs can be found at
`http://www.lix.polytechnique.fr/~{}bouissou/vmcai13`

Program			Widening		Our method	
Name	$\|V\|$	$\|\Delta\|$	Converging	t(s)	Converging	t(s)
`prog`	2	8	$\top$	0.091	Yes	0.029
`filter1`	6	24	$\top$	0.156	Yes	0.316
`filter2`	4	48	$\top$	0.053	Yes	0.672
`lp_iir_9600_2`	6	72	$\top$	0.208	Yes	0.049
`lp_iir_9600_4`	10	200	$\top$	6.563	Yes	0.167
`lp_iir_9600_4_elliptic`	10	200	–	–	Yes	0.308
`lp_iir_9600_6_elliptic`	14	392	–	–	Yes	2.564
`bs_iir_9600_12000_10_chebyshev`	22	968	–	–	Yes	19.780

Fig. 4. Results of analysis obtained using different methods

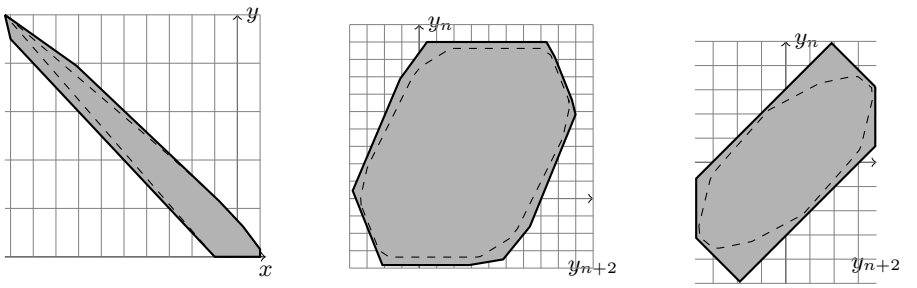
with a good execution time, where the widening fails to compute a bounded post-fixpoint. For these experiments, the direction sets we used are:

- `prog`, a randomly chosen direction set of 8 vectors.
- for `filter1` and `filter2`, we used the PCA analysis to determine a good directions set. We used 6 directions for `filter1` and 20 directions for `filter2`. Note that we also added to the directions set the box directions (i.e. $\pm X$ for each variable X) and for each direction d given by the PCA analysis, we added $-d$ to have a better coverage of the unit ball. The PCA analysis is done before programs analysis, and it is not taken into account in the execution time t.
- the octagonal directions for other programs.

Note that we also tried an analysis of these programs using the template domain, with a fixed template given by the directions set we used. In all cases, the analysis without widening did not terminate and widening with delay converged to $\top$.

In Figure 5, we show the post fixpoint we obtain for `prog` (with 100 directions), `filter2` and `lp_iir_9600_4`. We also show an under-approximation of the fixpoint, so the actual fixpoint is between both. This shows the quality of the invariant we compute.

Impact of the Acceleration Method. Finally, we want to stress out the importance of using an acceleration method to speed up the convergence of the algorithm. To do so, we compare the computation time and number of iterations of our algorithm with the same algorithm but with the identity as acceleration method, i.e. we stop when the sequence of support functions reaches its limit and not the accelerated sequence. As shown by the table of Figure 6, the use of a transformation method greatly improves the performance of the algorithm. We also compare two acceleration methods: the Δ^2-algorithm presented in Section 3.3 and the ε-algorithm. We see that both methods work well, the computation time being smaller for Δ^2 while the number of iterations needed to reach a post-fixpoint is smaller for the ε-algorithm. This was expected: the ε-algorithm is known to compute sequences that converge faster than Δ^2-algorithm, but its principle is that it repeatedly applies Δ^2-algorithm to the accelerated sequences. So its time and memory complexity are quadratic in the number of iteration,



(a) `prog`. (b) `filter2`. (c) `lp_iir_9600_4`.

Fig. 5. Results of the benchmarks programs. The filled polyhedron is the post fixpoint we obtain, the dashed on is the under-approximation of the fixpoint after, respectively, 100, 80 and 15 iterations of Kleene algorithm.

	Without acceleration		With Δ^2 method		With ε-method	
	t (s)	n_i	t (s)	n_i	t (s)	n_i
`lp_iir_9600_2`	0.058	47	0.037	16	0.047	17
`lp_iir_9600_4`	0.258	100	0.159	31	0.197	27
`lp_iir_9600_4_elliptic`	0.726	276	0.255	71	0.311	40
`lp_iir_9600_6_elliptic`	5.119	702	1.552	172	2.553	91
`bs_iir_9600_12000_10_chebyshev`	104.325	2391	19.873	524	-	-

Fig. 6. Influence of the acceleration method on the performance on execution time (t) and number of iterations (n_i)

while the complexity of Δ^2 is linear. This is the reason why the computation of the `bs_iir_9600_12000_10_chebyshev` program with the ε-algorithm timed out. Figure 6 shows that, even if theoretically neither the termination nor the acceleration is guaranteed, acceleration methods are in practice very useful for computing fixpoint of numerical programs.

7 Conclusion

Related work Improving the fixpoint computation in abstract interpretation has been a major concern since the beginning. Many techniques were proposed to improve Kleene iteration, like widening *with thresholds* [19], guided static analysis [16] or the use of acceleration techniques inspired from model checking [15]. Our method is complementary to these as it works well for numerical programs with a complex dynamics and for which no static constraints can be used to define relevant thresholds. As stated in the article, our technique finds its inspiration in the template abstract domain [24] and the use of support functions for hybrid systems analysis [20]. It however differs from an application of the abstract acceleration of [5] on the template domain as we follow the dynamics of the polyhedral analysis, which makes the result more precise than with the template domain. This was shown by our program `prog` in which our analysis

was able to prove that some part of the code is never reached while a template based analysis could not.

Conclusion and Future Work. In this article, we proposed a novel method to over-approximate the least fixpoint of the program semantics in the polyhedra domain. Our method uses a novel notion of convergence of convex sets, the abstract S-convergence, which is based on numerical and convex analysis tools. The algorithm we propose is very close to Kleene algorithm and its implementation does not require many changes in existing fixpoint solvers. One of the strengths of this method is that it can be tuned using many parameters (number and choice of the directions, ...) and thus offers a good trade-off between precision and computation time. Our experiments show that this method improves standard widening techniques on representative examples, as our prototype was able to compute a bounded post fixpoint for programs that the widening method was unable to bound.

Clearly, this method strongly depends on the choice of the directions to accelerate. We saw that the use of an apriori analysis using PCA and partial execution traces can help to determine relevant directions, the generalization and automatization of this process will be investigated. Also, our implementation is currently based on the ε-algorithm for accelerating the convergence of numerical sequences, we must consider other techniques in order to efficiently treat more kinds of programs. Our method is efficient to analyse affine programs, to generalize it to the non affine ones, we would like to experiment techniques of program linearisation. Finally, we would like to investigate the use of our technique to extrapolate on the dynamics of hybrid systems as defined in [20]: we believe that the abstract S-convergence can be used to approximate the limit of a dynamical system with unbounded horizon.

Acknowledgement. We want to thank A. Adjé and E. Goubault for their helpful suggestions and precious advices. The authors are also thankful to the anonymous reviewers for their helpful comments.

References

1. Amato, G., Parton, M., Scozzari, F.: Discovering invariants via simple component analysis. J. Symb. Comput. 47(12), 1533–1560 (2012)
2. Bagnara, R., Hill, P.M., Ricci, E., Zaffanella, E.: Precise widening operators for convex polyhedra. Sci. Comput. Program. 58(1-2), 28–56 (2005), http://dx.doi.org/10.1016/j.scico.2005.02.003
3. Bagnara, R., Hill, P.M., Zaffanella, E.: Not necessarily closed convex polyhedra and the double description method. Formal Asp. Comput. 17(2), 222–257 (2005)
4. Bouissou, O., Seladji, Y.: Numerical abstract domain using support function. Presented at the Fifth International Workshop on Numerical Software Verification, http://www.lix.polytechnique.fr/ bouissou/pdf/bouissou_seladji_nsv_12 .pdf
5. Bouissou, O., Seladji, Y., Chapoutot, A.: Acceleration of the abstract fixpoint computation in numerical program analysis. J. Symb. Comput. 47(12), 1479–1511 (2012)

6. Brezinski, C., Redivo Zaglia, M.: Extrapolation Methods-Theory and Practice. North-Holland (1991)
7. Brezinski, C., Redivo Zaglia, M.: Generalizations of Aitken's process for accelerating the convergence of sequences. Comp. and Applied Math. 26(2) (2007)
8. Cousot, P., Cousot, R.: Comparing the Galois Connection and Widening/Narrowing Approaches to Abstract Interpretation, Invited Paper. In: Bruynooghe, M., Wirsing, M. (eds.) PLILP 1992. LNCS, vol. 631, pp. 269–295. Springer, Heidelberg (1992)
9. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: POPL, pp. 84–97. ACM Press (1978)
10. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL, pp. 238–252. ACM Press (1977)
11. 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)
12. Delahaye, J.P., Germain-Bonne, B.: Résultats négatifs en accélération de la convergence. Numerische Mathematik 35, 443–457 (1980)
13. Feret, J.: Static Analysis of Digital Filters. In: Schmidt, D. (ed.) ESOP 2004. LNCS, vol. 2986, pp. 33–48. Springer, Heidelberg (2004)
14. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable Verification of Hybrid Systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011)
15. Gonnord, L., Halbwachs, N.: Combining Widening and Acceleration in Linear Relation Analysis. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, pp. 144–160. Springer, Heidelberg (2006)
16. Gopan, D., Reps, T.W.: Guided Static Analysis. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 349–365. Springer, Heidelberg (2007)
17. Hiriart-Urrut, J.B., Lemaréchal, C.: Fundamentals of Convex Analysis. Springer (2004)
18. Jeannet, B., Miné, A.: Apron: A Library of Numerical Abstract Domains for Static Analysis. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 661–667. Springer, Heidelberg (2009)
19. 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)
20. Le Guernic, C., Girard, A.: Reachability analysis of linear systems using support functions. Nonlinear Analysis: Hybrid Systems (2010)
21. Löhne, A., Zălinescu, C.: On convergence of closed convex sets. Journal of Mathematical Analysis and Applications 319(2), 617–634 (2006)
22. Miné, A.: The octagon abstract domain. Higher-Order and Symbolic Computation 19(1), 31–100 (2006)
23. Rockafellar, R.: Convex analysis, vol. 28. Princeton Univ. Pr. (1997)
24. 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)
25. Sonntag, Y., Zălinescu, C.: Scalar convergence of convex sets. J. Math. Anal. Appl. 164(1), 219–241 (1992)
26. Wynn, P.: The epsilon algorithm and operational formulas of numerical analysis. Mathematics of Computation 15(74), 151–158 (1961)