Smart Widening for the Polyhedral Analysis

Yassamine Seladji
Abou Bekr Belkaid university, Tlemcen, Algeria

Keywords: Abstract Interpretation, Static Analysis, Polyhedra Abstract Domain.

Abstract: The polyhedron abstract domain is a cornerstone of static program analysis, providing a powerful mathematical framework for reasoning about numerical properties of programs. It can express a large number of properties. This makes it complex and time-consuming. In the polyhedral analysis, the widening operator with threshold proposes a good compromise between precision and time execution. However, ensuring both termination and precision in the analysis process remains a challenge, particularly when dealing with complex programs and high-dimensional state spaces. This paper proposes a novel approach to improve polyhedral analysis by dynamically computing relevant widening thresholds, thereby improving both the termination and precision of the analysis. We demonstrate the effectiveness of our approach through experimental evaluation on a variety of benchmark programs. Our results show significant improvements in both analysis termination and precision.

1 INTRODUCTION

Static analysis by abstract interpretation aims to automatically proving properties of computer programs, by computing invariants that over-approximate the set of program behaviours (Cousot and Cousot, 1977; Cousot and Cousot, 1992). 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 (Cousot and Cousot, 1977). The main idea is to define a new (abstract) semantic function $F^\sharp$ which computes an abstract program invariant including the concrete program invariant. 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 (Cousot and Halbwachs, 1978). The polyhedron abstract domain (Cousot and Halbwachs, 1978) returns invariants that express a large set of properties. However, this expressiveness makes the analysis more complex and expensive: it has exponential space and time complexity in the worst case.

The polyhedra abstract domain also faces significant challenges and limitations. These include scalability concerns in handling large programs with high-dimensional state spaces, imprecision in handling non-linear expressions and non-convex constraints, and the trade-off between precision and efficiency in analysis algorithms. Various techniques have been proposed to alleviate this complexity burden, aiming to strike a balance between analysis efficiency and accuracy. However, many of these approaches inevitably sacrifice accuracy in order to achieve computational tractability. In (Castagna and Gordon, 2017), a new version of the polyhedra abstract domain is defined, to decrease the complexity of the analysis. For that, a decomposed version of the polyhedron is used. In (Yu and Monniaux, 2019), authors propose a new polyhedral projection operator based on parametric linear programming using floating-point arithmetic. This operator is more efficient than the classical Fourier-Motzkin elimination method. A new operator is proposed in (Arceri et al., 2023) to model the splitting of control flow paths. This operator is used to allow more efficient analysis when using abstract domains that are expensive to compute. Note that the proposed split operator has no impact on precision. To find a good trade-off between expressiveness and efficiency, new domains have been developed. These domains provide coarse-grained representations of numerical properties, sacrificing precision for computational efficiency. While suitable for certain types of analysis, these approximations may fail to capture intricate dependencies or non-linear re-

Seladji, Y.
Smart Widening for the Polyhedral Analysis.
DOI: 10.5220/0012860500003753

In *Proceedings of the 19th International Conference on Software Technologies (ICSOFT 2024)*, pages 79-87
ISBN: 978-989-758-706-1; ISSN: 2184-2833

lationships present in the program, leading to imprecise results. They are known as weakly relational abstract domains (Miné, 2006; Sankaranarayanan et al., 2005; Goubault et al., 2012; Seladji and Bouissou, 2013; Seladji, 2017).

The other tool to reduce the computation time of the analysis is to modify the Kleene algorithm to make it faster. By using operators like widening (Cousot and Cousot, 1992) and narrowing that accelerate convergence in iterative analysis algorithms by approximating fixpoints or refining abstractions. Widening introduces imprecision by over-approximating program states to ensure termination, while narrowing aims to restore precision by refining abstractions iteratively. However, striking the right balance between widening and narrowing parameters is crucial to avoid excessive imprecision or divergence in the analysis. The widening operator in the polyhedra domain is used to ensure termination in iterative analysis algorithms, particularly when dealing with loops or recursive structures. By removing constraints that are not stable from one iteration to the next. However, it is essential to recognize that widening introduces a large over-approximation, leading to imprecise analysis results.

Techniques have been developed to minimise this loss of accuracy. Threshold widening (Lakhdar-Chaouch et al., 2011) predefines a set of thresholds, the elements of which are used as candidate bounds at each iteration. However, the choice of widening thresholds significantly impacts the analysis quality, balancing termination and precision. Finding the optimal widening threshold is non-trivial and often requires manual tuning, limiting the scalability and effectiveness of the analysis. In our previous work (Bouissou et al., 2012), we presented a method that dynamically computes a relevant set of thresholds using relevant set of thresholds using numerical analysis tools. This technique was applied to the interval and octagon abstract domains. It should be noted that the solutions proposed to improve the computational time of a polyhedral analysis can be categorized into two main strategies:

- restricted data representation : One approach to improving computational time in polyhedral analysis involves simplifying the representation of program data to facilitate manipulation.
- improvement of fixed-point computation techniques : it focuses on enhancing the efficiency of fixed-point computation algorithms, typically by employing better widening operators or convergence strategies.

In this paper, we combines the advantages of both data representation simplification and fixed-point computation improvement techniques to improve the polyhedral analysis. By exploiting the compactness of support function-based representations, our method facilitates faster convergence in fixed-point computation algorithms while maintaining analysis precision. In addition, the use of support functions allows efficient manipulation of program constraints, further improving computational efficiency. The paper is organized as follows: in Section 2, we explain the proposed method using a simple example. In Section 3, we introduce some useful definitions. In Section 4, we present our main contribution. Some benchmarks are given in Section 5. Finally, we conclude with a conclusion and some perspectives.

2 AN ILLUSTRATED EXAMPLE

To illustrate the intuition behind our approach, let's consider a simple example involving a linear filter program. Figure 1 depicts the structure of the program. Analyzing the given program using the polyhedral abstract domain presents challenges due to its inherent complexity. The analysis couldn't terminate within a reasonable time (e.g., more than 10 minutes), it indicates the complexity of the program and the limitations of the polyhedral analysis approach for this particular case. In such scenarios, it's essential to consider alternative analysis techniques such as the use of the widening operator. Note that, the analysis result for the given program, using widening, is an unbounded polyhedron. Indeed, widening with thresholds (Lakhdar-Chaouch et al., 2011), offers an interesting alternative approach. By using a predefined set of thresholds as fixed-point candidates, this technique aims to control the trade-off between analysis precision and computational complexity. However, the static nature of threshold selection can be a bottleneck, as it does not take into account the dynamics of the analysed program. In this paper, we propose an approach to compute dynamically relevant thresholds for polyhedral analysis. This approach aims to adaptively adjust widening thresholds based on the evolving program state.

For that, we perform a pre-analysis using a sub-polyhedra abstract domain (Seladji and Bouissou, 2013). This domain is based on a given set of templates. In this example, we define these templates as a set of vectors uniformly distributed on a unit sphere. First, a pre-analysis is performed using an sub-polyhedron abstract domain (Seladji and Bouissou, 2013) based on a given set of templates, such as vectors uniformly distributed on a unit sphere. In Figure 2, the yellow polyhedron represents the result of

```
double input() {
        double u = 10.0, l = 0.0;
        return ( rand()/(double)RAND_MAX ) * (u-l) + l;
}
int main() {
        double xn, yn, ynm1,ynm2 ;
        xn=xnm1=xnm2=yn=ynm1=ynm2=0;
        int i=0;
        while (i<4000) {
          yn = xn + 0.5*ynm1 - 0.45*ynm2;
          printf("%f %f\n",ynm1,yn);
          ynm2=ynm1;
          ynm1=yn;
          xn = input();
        }
        return 0;
}
```

Figure 1: The body of the illustrated program called `filter2`.

the pre-analysis, denoted as P. The obtained result corresponds to a template representation of the fixed point obtained using the polyhedra abstract domain. Note that this pre-analysis has polynomial complexity in the number of iterations and linear complexity in the number of template vectors, which contributes to the scalability and efficiency of the analysis process.

The main idea is to use P to improve the polyhedral analysis. To do this, we consider P as a fixed point candidate and use it as a relevant threshold for the widening operator to accelerate the analysis.

In Figure 2, the accuracy and efficiency of the proposed approach is demonstrated by comparing the orange polyhedron obtained by using P as a relevant threshold with the brown polyhedron obtained from the 200th Kleene iteration using polyhedral analysis Note that, this analysis is done in less than 7 seconds, which we consider very fast.

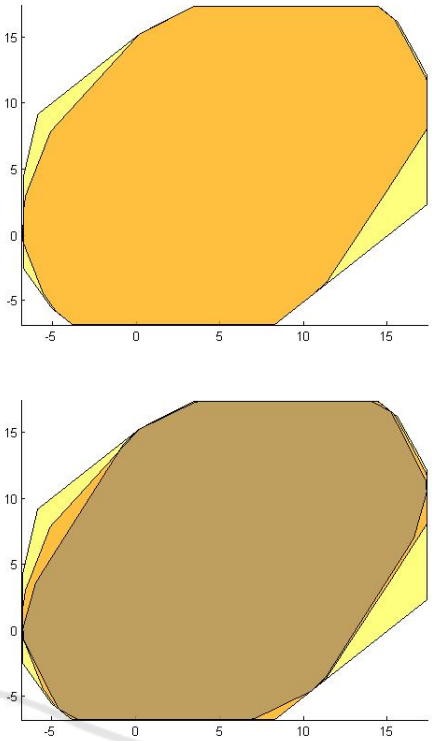

Figure 2: The yellow polyhedron represents the one obtained using the sub-polyhedra abstract domain and the orange polyhedron is the one obtained using our method. The polyhedron in brown represents the one obtained in the 200^{th} kleene iteration using the polyhedra abstract domain.

3 BACKGROUND

This section outlines some of the important notions used in this article. In the rest, we note by $\mathbb{R}$ the set of real numbers and $\forall x, y \in \mathbb{R}, < x, y >$ represents the scalar product of x by y.

3.1 The Polyhedral Analysis

A convex polyhedron is a subset of $\mathbb{R}^n$ defined as the intersection of finitely many half-spaces, each half-space is 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 (Cousot and Halbwachs, 1978). 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 *The abstract domain of convex polyhedra, noted C^n, is the set of all pairs (A,b) where $A \in \mathbb{R}^{n \times k}$ is a real matrix and $b \in \mathbb{R}^n$ is a real vector. A polyhedron* P *given by the pair (A,b) represents the set of all points $(x_1,\ldots,x_n) \in \mathbb{R}^n$ such that*

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

Given a polyhedron $\mathsf{P} = (A,b)$, the j^{th} constraint of P is $\langle A_j, X \rangle \leq b_j$ where X is the vector of variables and A_j is the j^{th} row of A.

The static analysis of a program with the polyhedra domain consists in computing the least fixpoint P_∞ 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

Algorithm 1: Kleene iteration algorithm for the polyhedra abstract domain.

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

$\mathsf{P}_\infty = \bigsqcup_{k\in\mathbb{I}} F^k(\bot)$, where $\bot$ is the least element of C^n. Kleene iteration Algorithm for the polyhedra abstract domain is given in Algorithm 1. So analysing programs using the abstract interpretation based static analysis of a program consists of computing the least fixed point of the equation $\mathsf{P} = \mathsf{P} \sqcup F(\mathsf{P})$. As Kleene algorithm may not terminate and may not compute the least fixed point P_∞, a widening operator is used to compute an over-approximation $\mathsf{P}_w \sqsupseteq \mathsf{P}_\infty$.

3.2 The Sub-Polyhedral Analysis

The sub-polyhedra abstract domain presented in (Seladji and Bouissou, 2013) is based on support functions (Hiriart-Urrut and Lemarechal, 2004). This 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 this abstract domain by P_Δ, parametrized by a template Δ. It represents a finite set of directions $\Delta = \{\vec{d}_1, \ldots, \vec{d}_l\}$. Note that, the set Δ can be computed randomly as vectors uniformly distributed on the unit sphere. A novel approach to compute a relevant Δ is presented in (Seladji, 2017), which is based on statistical analysis. The definition of P_Δ is given in Definition 2.

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

For each $\Omega \in \mathsf{P}_\Delta$, $\Omega(\vec{d})$ is the value of Ω in direction $\vec{d} \in \Delta$. Intuitively, Ω is a support function with finite domain.
The abstraction and concretization functions of P_Δ are defined in Definition 3.

Definition 3 *Let $\Delta \subseteq \mathbb{R}^n$ be the set of vectors.*
We define the concretization function $\gamma_\Delta : \mathsf{P}_\Delta \to C^n$ by:

$$\forall \Omega \in \mathsf{P}_\Delta,\ \gamma_\Delta(\Omega) = \bigcap_{\vec{d}\in\Delta} \{x \in \mathbb{R}^n, < \vec{d}, x > \leq \Omega(\vec{d})\} .$$

The abstraction function $\alpha_\Delta : C^n \to \mathsf{P}_\Delta$ is defined by:

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

where, $\delta_\mathsf{P}(\vec{d})$ is the support function (Hiriart-Urrut and Lemarechal, 2004) of the polyhedron P using the vector $\vec{d}$.

Note that, the concretization of an abstract element of P_Δ is a polyhedron defined by the intersection of half-spaces, where each one is characterized by its normal vector $\vec{d} \in \Delta$ and the coefficient $\Omega(\vec{d})$. The abstraction function on the other side is the restriction of the support function of the polyhedron on the set of directions Δ. The order structure of P_Δ is defined using properties of support functions.
We combine Kleene iteration algorithm and the P_Δ abstract domain to perform the analysis. Let take in consideration loops of the form:

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

We suppose that `A` is a real matrix, `b` may be a set of real 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 belong initially to the polyhedron P_0. Using properties of support function in the case of loops without guard, we can define the abstract element obtained in the i^{th} Kleene iteration as follows:

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

Equation 1 is used to define a special version of Kleene algorithm. The obtained analysis has a polynomial time complexity in the number of iterations and linear in the number of directions in Δ. In addition, its result is as accurate as possible: at each Kleene iteration, we have that $\Omega_i = \alpha_\Delta(\mathsf{P}_i)$, such that Ω_i (resp. P_i) is the result of the ith Kleene iteration using the P_Δ domain (respectively the polyhedra abstract domain). So $\Omega_\infty = \alpha_\Delta(\mathsf{P}_\infty)$, with Ω_∞ is the fixed point obtained in the P_Δ analysis and P_∞ is the one obtained using polyhedra domain.

4 THE SMART WIDENING

The Kleene algorithm plays a crucial role in static analysis by abstract interpretation, particularly in computing the least fixed point for various abstract domains, including the polyhedral abstract domain. In polyhedral analysis, it is performed to compute the least fixed point of a set of constraints. Widening is used to accelerate convergence by over-

approximating the solution space, it is applied iteratively until a stable fixed point is reached, thus ensuring the termination of the analysis.

One of the main drawbacks of Widening is the potential for overly coarse results that quickly diverge towards infinity. To deal with the loss of precision, we define the so-called **Smart widening operator**. It adds an intermediate step to extrapolate the potential fixed point using a relevant threshold.

In polyhedron analysis, finding a relevant threshold consists of automatically computing a polyhedron that serves as a candidate fixed point in the Kleene iteration. The polyhedron threshold is obtained using sub-polyhedral analysis, which means that the dynamics of the analysed program are taken into account in the threshold computation process.

For the rest, we adopt the constraint representation of the polyhedron. In this section, we present our main contribution: the dynamic computation of a relevant threshold to improve the widening operator in polyhedral analysis, thereby accelerating the analysis process. Our approach consists of two main steps: first, we describe the technique for dynamically computing a relevant threshold, and then we show how this threshold improves the widening operator and speeds up polyhedral analysis.

4.1 The Threshold Computation

In this section, We present a technique to dynamically compute a relevant threshold during the analysis process. This technique is based on sub-polyhedral analysis as described by (Seladji and Bouissou, 2013) . A relevant threshold is defined as a polyhedron that allows the polyhedral fixed point computation to be accelerated without losing accuracy.

For that we use the P_Δ abstract domain given in Section 3.2. Let us consider the case of programs with affine loops. Such loops are very common in embedded systems, especially in the case of linear filters. Let the program with the affine loop be given as follows:

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

With `A` $\in \mathbb{R}^{n\times n}$ and `b` $\in \mathbb{R}^n$. Let `C` be a boolean expression. We assume that the program variables are initially inside the polyhedron P_0. For the rest, we put `C = true`.

Using the polyhedra abstract domain C^n, the loop invariant, $\mathsf{P} \in C^n$, is defined as the least fixed point of the equation $\mathsf{P} = \mathsf{P} \sqcup \left(A\mathsf{P} + \mathsf{P}_b\right)$. Usually, the fixed point is defined as the limit of the kleene iterates given by $\mathsf{P}_i = \mathsf{P}_{i-1} \sqcup (A\mathsf{P}_{i-1} + \mathsf{P}_b)$. Using the P_Δ analysis, we have that, the i^{th} kleene iterate computes the support function of P_i, noted δ_{P_i}, as follows :

$$\forall \vec{d} \in \Delta, \delta_{\mathsf{P}_i}(\vec{d}) =$$

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

Let denote by $\mathsf{P}_\sharp$ the fixed point of the equation $\mathsf{P} = \mathsf{P} \sqcup \left(A\mathsf{P} + \mathsf{P}_b\right)$. The P_Δ analysis computes the support function of $\mathsf{P}_\sharp$ in a given vector $\vec{d} \in \mathbb{R}^n$, noted $\delta_{\mathsf{P}_\sharp}(\vec{d})$. As shown in Equation 2, the computation of $\delta_{\mathsf{P}_\sharp}(\vec{d})$ used only P_0, P_b and A, which represent the input of the program to be analysed. Let Θ be the function that computes $\delta_{\mathsf{P}_\sharp}(\vec{d})$ using the P_Δ analysis, it is given in Definition 4.

Definition 4 *Let $\mathsf{P}_\sharp$ be the fixed point of the equation $\mathsf{P} = \mathsf{P} \sqcup \left(A\mathsf{P} + \mathsf{P}_b\right)$.*
Let $\Theta : \mathbb{R}^n \times C^n \times C^n \times \mathbb{R}^{n\times n} \longrightarrow \mathbb{R}$ be a function defined as follows :
$(\forall \vec{d} \in \mathbb{R}^n, \mathsf{P}_0, \mathsf{P}_b \in C^n, A \in \mathbb{R}^{n\times n})$,

$$\Theta(\vec{d}, \mathsf{P}_0, \mathsf{P}_b, A) = \delta_{\mathsf{P}_\sharp}(\vec{d}) \tag{3}$$

Where $\delta_{\mathsf{P}_\sharp}(\vec{d})$ is the support function of $\mathsf{P}_\sharp$ for the vector $\vec{d}$, obtained using the P_Δ analysis.

As presented in (Seladji and Bouissou, 2013), the computation of $\delta_{\mathsf{P}_\sharp}(\vec{d})$ has a polynomial complexity in the number of iterations and linear in the number of $\vec{d}$ in Δ. So, this computation is efficiency.

We know that for a given convex polyhedron P, the support function allows us to define P in the following way :

$$\forall \vec{d} \in \mathbb{R}^n, \mathsf{P} = \bigcap_{\vec{d}\in\mathbb{R}^n} \left\{x \in \mathbb{R}^n \,|\, \langle x, \vec{d}\rangle \leq \delta_{\mathsf{P}}(\vec{d})\right\} \ . \tag{4}$$

Note that computing $\delta_{\mathsf{P}}(\vec{d})$ for all $\vec{d} \in \mathbb{R}^n$ is a difficult task. We can only compute it for a subset Δ of $\mathbb{R}^n$, which allows us to get an over-approximation of P.

$$\forall \Delta \subseteq \mathbb{R}^n, P \subseteq \bigcap_{\vec{d}\in\Delta} \left\{x \in \mathbb{R}^n \,|\, \langle x, \vec{d}\rangle \leq \delta_{\mathsf{P}}(\vec{d})\right\} \ . \tag{5}$$

So, we can compute an over-approximation of $\mathsf{P}_\sharp$ using the Equation 3 of Definition 4. This computation is given in Definition 5.

Definition 5 *Let the function*
$\Lambda : (\mathbb{R}^n \times \mathbb{R}^n) \times C^n \times C^n \times \mathbb{R}^{n\times n} \longmapsto C^n$ *defined as follows :* $(\forall \Delta \subseteq \mathbb{R}^n, \mathsf{P}_0, \mathsf{P}_b \in C^n, A \in \mathbb{R}^{n\times n}), \Lambda(\Delta, \mathsf{P}_0, \mathsf{P}_b, A) =$

$$\bigcap_{\vec{d}\in\Delta} \left\{x \in \mathbb{R}^n \,|\, \langle x, \vec{d}\rangle \leq \Theta(\vec{d}, \mathsf{P}_0, \mathsf{P}_b, A)\right\} \ . \tag{6}$$

So, from Definition 5, we can deduce that $\mathsf{P}_\sharp \subseteq \Lambda(\Delta, \mathsf{P}_0, \mathsf{P}_b, A)$.

In the context of polyhedral analysis, P_Δ analysis refers to an abstract domain that captures properties of convex polyhedra based on a set Δ of templates. The function Λ is used to define the best template abstraction of $\mathsf{P}^\sharp$ using the set Δ. The polyhedron given in definition 5 is considered as a relevant threshold. By dynamically computing this relevant threshold, the analysis can adapt to the specific characteristics and dynamics of the program being analysed. Once the threshold polyhedron is computed, it is integrated into the polyhedral analysis framework as a relevant threshold to improve the analysis process. This threshold guides the widening and refinement operations, ensuring that the analysis efficiently converges to a stable fixed point solution.

The P_Δ analysis can be applied to a wider range of programs than just those with affine loops. While affine loops are a common scenario, especially in embedded systems and signal processing applications. The P_Δ analysis provides a powerful framework that can consider programs with complex loop structures involving non-linear loops and affine transformations.

4.2 The Widening Computation

In this section, we integrate the threshold polyhedron obtained from Definition 5 into the widening operator used in the Kleene iteration. For that, new operator are defined called the smart widening operator, denoted ∇_Λ. This operator considers the polyhedron given in Definition 6 as a Kleene fixed point candidate and uses it as a threshold in the Kleene iteration algorithm. Use the modified widening operator within the Kleene iteration algorithm to iteratively refine the analysis results towards a fixed point solution. At each iteration, the polyhedron is used as a threshold to guide the widening process, ensuring that the analysis converges efficiently and accurately. The definition of ∇_Λ is given in Definition 6.

Definition 6 *For a given* $\mathsf{P}, \mathsf{P}_0, \mathsf{P}_b \in C^n$, $A \in \mathbb{R}^{n\times n}$ *and* $\Delta \subseteq \mathbb{R}^n$, *the smart widening operator* ∇_Λ *is defined as follows :*

$$\mathsf{P}\nabla_\Lambda\ (A\mathsf{P}+\mathsf{P}_b) = \mathsf{P}\cup\Lambda(\Delta,\mathsf{P}_0,\mathsf{P}_b,A).$$

In Definition 6, the smart widening operator substitutes the expression $(A\mathsf{P}+\mathsf{P}_b)$ with the threshold returned by the function $\Lambda(\Delta,\mathsf{P}_0,\mathsf{P}_b,A)$. Afterwards, the union operator is performed between P and the polyhedron resulting from $\Lambda(\Delta,\mathsf{P}_0,\mathsf{P}_b,A)$.

So, we have that the function $\Lambda(\Delta,\mathsf{P}_0,\mathsf{P}_b,A)$ computes a threshold polyhedron based on the input parameters : the set of templates Δ, the initial polyhedron represented by P_0 and P_b, and the affine transformation matrix A. This threshold polyhedron represents a relevant approximation of the program state space. The smart widening operator replaces the expression $(A\mathsf{P}+\mathsf{P}_b)$ by the obtained threshold. This ensures that the widening operation is guided by the threshold polyhedron, focusing the analysis on relevant ranges of the program state space where significant changes are occurring. After substituting the expression with the threshold polyhedron, the union operator is performed between the current polyhedron P and the threshold one. This union operation combines the information collected by p with the refined approximation provided by the threshold polyhedron, improving the accuracy and convergence of the analysis. In Algorithm 2 we consider the adapted version of the Kleene iteration algorithm using the smart widening operator. This adapted algorithm incorporates the smart widening operator to guide the iterative refinement process towards a stable fixed-point solution.

Algorithm 2: Kleene iteration algorithm based on smart widening operator.

Require: Δ, A, P_0,P_b
1: $i := 0$
2: **repeat**
3: **if** $i \geq 10$ **then**
4: ΔGeneration();
5: $\mathsf{P}_i := \mathsf{P}_{i-1}\ \nabla_\Lambda\ (A\mathsf{P}_{i-1}+\mathsf{P}_b)$
6: $i := 0$
7: **else**
8: $\mathsf{P}_i := \mathsf{P}_{i-1}\ \cup (A\mathsf{P}_{i-1}+\mathsf{P}_b)$
9: $i := i+1$
10: **end if**
11: **until** $\mathsf{P}_i\ \sqsubseteq \mathsf{P}_{i-1}$

The smart widening operator is used to extrapolate the sequence of polyhedra obtained by Kleene iteration up to the polyhedron computed by the Λ function. In the algorithm 2 we introduce a deliberate delay in the application of the smart widening operator, choosing to apply it after every 10 Kleene iterations. This delay is designed to allow the Kleene algorithm to stabilise and attempt to converge to the fixed point before introducing the refinement provided by the smart widening operator. Note that practical experiments show that only one application of smart widening is needed to speed up fixed-point computation.

In fact, the quality of the polyhedron calculated with the function Λ is closely related to the quality of the chosen set Δ. The set Δ serves as the basis for constructing templates that approximate the behaviour of the program, and the effectiveness of these templates directly affects the accuracy and precision of the analysis results. One intuitive method for selecting templates is to take vectors that are uniformly distributed on the unit sphere. Another method is to consider the current kleene iteration polyhedron to compute the

set Δ. The polyhedron P_{i-1} obtained at the $(i^{th}-1)$ Kleene iteration contains constraints that define the boundaries of the program's state space at that iteration. The normal vectors of the constraints of P_{i-1} represent directions in the program's state space that are relevant to the analysis. These normal vectors capture the local geometry and behavior of the program at the previous iteration. They can be used as templates for constructing the set Δ for the current iteration. By basing the selection of templates on the previous iteration's polyhedron, this method adapts dynamically to the program's behavior and changes in the state space. It allows the analysis to focus on directions that are relevant to the current state of the program, enhancing the accuracy and effectiveness of the analysis process. This method is represented in the Algorithm by the function ΔGeneration(). To ensure efficiency and effectiveness, it's advantageous to apply this method after computing a sufficient number of Kleene iterations to obtain a polyhedron P_{i-1} with a substantial number of constraints. Constructing Δ based on a polyhedron with a large number of constraints enables the selection of a diverse and representative set of vectors, enhancing the quality and coverage of the template set.
Utilizing Principal Components Analysis (PCA), as presented in (Seladji, 2017), to compute an interesting set Δ offers another alternative for constructing templates in polyhedral analysis. PCA is a statistical technique that can be leveraged to identify the most significant directions or principal components in a high-dimensional dataset.

5 EXPERIMENTATIONS

The smart widening operator proposed a good trade off between expressiveness and efficiency. For the experimentation, we apply Algorithm 2 with the polyhedra abstract domain to analyze several programs, this analysis is called the smart polyhedral analysis. The programs used contain a number of stable linear systems and digital filters, which are well-known to be difficult to analyse using polyhedral analysis. To show the efficiency and the scalability of the *the smart polyhedral analysis*, it is tested on programs with many variables. The smart polyhedral analysis is implemented using PPL (Bag,). **PPL** (the **P**arma **P**olyhedra **L**ibrary) is a library that provides an efficient implementation of the polyhedra abstract domain. Note that its implementation is based on rational numbers. We use also the MPFR Library[1] to handle the floating point arithmetics. The experimentations are done on a $2.4GH_Z$ Intel Core2 Duo laptop, with $8Gb$ of RAM.

First of all, the efficiency aspect of the smart polyhedral analysis strongly depends on its execution time. So, we evaluate the efficiency of this analysis by computing its execution time. Table 1 shows the execution time obtained using the smart polyhedral analysis. In this experimentations, the function ΔGeneration() in the Algorithm 2 defined Δ as a set of vectors uniformly distributed on a surface of the n-dimensional sphere plus the orthogonal vectors, its cardinality is noted $|\Delta|$. We have also that the column labeled t is the execution time (in seconds). The column labeled by $|V|$ represents the number of programs variables. Note that, $|V|$ also represents the dimension of the computed polyhedron. In general, when a program uses a large number of variables, it means that the corresponding analysis is calculating high-dimensional polyhedra. High-dimensional polyhedra present several challenges in terms of computation and analysis. The polyhedra generated depend on the analysis tool and the program under analysis. Their dimensions can range from a few variables in C programs to a thousand in Lustre programs. In general, the analysis tools try to limit their use of polyhedral analysis to a small number of variables and constraints. In the case where those limits are exceeded, the analysis switches to less expressive abstract domain, as for example the interval domain. So, the parameter $|V|$ is used in our experimentation to show the scalability of our method.

We emphasize that for most of these programs the polyhedral analysis don't terminate before the timeout (more than 10 minutes). If we use the widening operator the results go to top, which represents a bad accuracy. In Table 1 thus shows the efficiency of our method with several programs.
To further illustrate the expressiveness aspect of the smart widening operator, we display on Figure 3 and Figure 4 the results of the smart polyhedral analysis, given by the red and orange polyhedra, respectively, the used threshold obtained using the P_Δ analysis is given by the blue and yellow polyhedra, respectively. Note that, on Figure 3 the blue polyhedra are contained within the red polyhedra , this shows the quality of the invariant we compute.

6 CONCLUSION

In this article we have defined a new method to improve polyhedral analysis. Introducing a new version of the Kleene iteration algorithm that utilizes a

[1] https://www.mpfr.org/

Table 1: Table of results obtained using our analysis and the polyhedral one.

Program		Polyhedral analysis with ∇_Λ		Polyhedral analysis
Name	$\|V\|$	$\|\Delta\|$	t(s)	t(s)
prog	2	308	2.202	TO
filter1	4	332	2.202	TO
filter2	4	332	6.264	TO
filter3	4	332	59.764	TO
filter4	5	350	2m10.953	TO
lead_leg_controller	5	350	5m58.748	TO
Dampened_oscillator	6	332	15.291	TO
Harmonic_oscillator	6	332	4.882	TO
lp_iir_9600_2	6	372	1m13.537	TO

```
double input1() {
  double u = 10.0;
  double l = -10.0;
  return ( rand()/(double)RAND_MAX )
        * (u-l) + l;
}
double input2() {
  double u = 3.0;
  double l =  0.0;
  return ( rand()/(double)RAND_MAX )
     * (u-l) + l;
}
int main() {
double x, xn, y,yn;
srand(time(NULL));
x=input1();
y=input2();
xn=yn=0;
int i=0;
while (i<10000){
xn = 0.5 *x - y - 2.5;
yn = 0.9 *y + 10;
x = xn;
y = yn;
}
return 0;
}
```

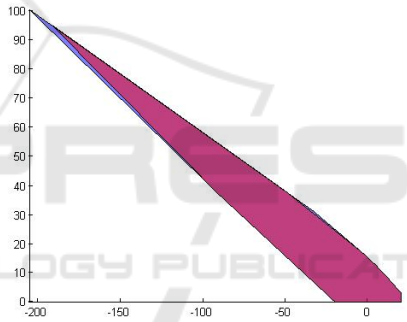

Figure 3: (Left) The body of the program called prog. (Right) The red polyhedron is the post fixed point obtained using the smart widening operator. The blue polyhedron is the threshold obtained using P_Δ analysis.

novel widening operator, the ”smart widening operator,” represents a significant advancement in polyhedral analysis techniques. The smart widening operator applies the P_Δ analysis on the program under analysis. P_Δ The result of the P_Δ analysis serves as the fixed-point candidate for the polyhedral analysis. The result obtained from the P_Δ represents a template representation of the fixed point achieved through polyhedral analysis. These templates capture important directions or patterns in the program's state space, providing a compact and informative representation of the fixed point. The smart widening operator is integrated into the Kleene iteration algorithm to guide the iterative refinement process. Instead of applying traditional widening techniques, the algorithm applies the P_Δ analysis to obtain the fixed-point candidate, which serves as the threshold for widening. By using the template representation obtained from P_Δ analysis, the smart widening operator aims to improve both the efficiency and precision of the polyhedral analysis. The experimental results show that the smart widening operator allows to speed up the poly-

```
double input() {
double u = 10.0, l = 0.0;
return ( rand()/(double)RAND_MAX )
       * (u-l) + l;
}
int main() {
double xn, yn, ynm1,ynm2 ;
xn=xnm1=xnm2=yn=ynm1=ynm2=0;
int i=0;
while (i<4000) {
  yn = xn + 0.5*ynm1 - 0.45*ynm2;
  printf("%f %f\n",ynm1,yn);
  ynm2=ynm1;
  ynm1=yn;
  xn = input();
}
return 0;
}
```

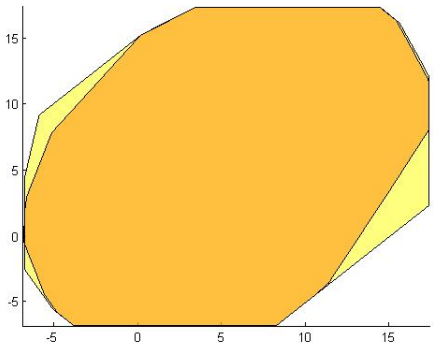

Figure 4: (Left)The body of the program called *Filter_2*.(right) The orange polyhedron is the post fixed point obtained using the smart widening operator. The yellow polyhedron is the threshold obtained using P_Δ analysis.

hedral analysis when the standard polyhedral analysis fails. The smart widening operator can easily be adapted to other abstract domains, for which the set Δ should be adapted to the shape of the corresponding domain. The choice of this set plays an important role in the accuracy and relevance of the result of the smart widening operator. In our future work, we will develop a method to dynamically define a relevant set Δ using the work presented in (Seladji, 2017).

REFERENCES

Arceri, V., Dolcetti, G., and Zaffanella, E. (2023). Speeding up static analysis with the split operator. In Ferrara, P. and Hadarean, L., editors, *Proceedings of the 12th International Workshop on the State Of the Art in Program Analysis,Orlando, FL, USA*.

Bouissou, O., Seladji, Y., and Chapoutot, A. (2012). Acceleration of the abstract fixpoint computation in numerical program analysis. *J. Symb. Comput.*, 47(12):1479–1511.

Castagna, G. and Gordon, A. D. (2017). Fast polyhedra abstract domain. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, Paris, France*.

Cousot, P. and Cousot, R. (1977). Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252.

Cousot, P. and Cousot, R. (1992). Comparing the Galois connection and widening/narrowing approaches to abstract interpretation, invited paper. In *PLILP*, volume 631 of *LNCS*, pages 269–295. Springer-Verlag.

Cousot, P. and Halbwachs, N. (1978). Automatic discovery of linear restraints among variables of a program. In *POPL*.

Goubault, E., Putot, S., and Védrine, F. (2012). Modular static analysis with zonotopes. In *Static Analysis-19th International Symposium, 2012, Deauville, France. Proceedings*.

Hiriart-Urrut, J.-B. and Lemarechal, C. (2004). *Fundamentals of Convex Analysis*. Springer.

Lakhdar-Chaouch, L., Jeannet, B., and Girault, A. (2011). Widening with thresholds for programs with complex control graphs. In *ATVA*, pages 492–502.

Miné, A. (2006). The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100.

Sankaranarayanan, S., Sipma, H. B., and Manna, Z. (2005). Scalable analysis of linear systems using mathematical programming. In *VMCAI*, volume 3385 of *LNCS*, pages 25–41.

Seladji, Y. (2017). Finding relevant templates via the principal component analysis. In *Verification, Model Checking, and Abstract Interpretation - 18th International Conference, VMCAI 2017, Paris, France, January 15-17, 2017, Proceedings*.

Seladji, Y. and Bouissou, O. (2013). Numerical abstract domain using support functions. In *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA. Proceedings*.

Yu, H. and Monniaux, D. (2019). An efficient parametric linear programming solver and application to polyhedral projection. In Chang, B.-Y. E., editor, *Static Analysis*.