Noname manuscript No.
(will be inserted by the editor)

Practical Policy Iterations

A practical use of policy iterations for static analysis – The quadratic case.

Pierre Roux · **Pierre-Loïc Garoche**

Received: date / Accepted: date

Abstract Policy iterations is a technique based on game theory that relies on a sequence of numerical optimization queries to compute the fixpoint of a set of equations. It has been proposed to support the static analysis of programs as an alternative to widening, when the latter is ineffective. This happens for instance with highly numerical codes, such as found at cores of control command applications. In this paper we present a complete, yet practical, description of the use of policy iteration in this context. We recall the rationale behind policy iteration and address required steps towards an automatic use of it: synthesis of numerical templates, floating point semantics of the analyzed program and issues with the accuracy of numerical solvers.

Keywords policy iterations · abstract interpretation · static analysis · quadratic templates · ellipsoids · Lyapunov functions · widening · controllers

1 Introduction

Since the first proposals in the 70s, static analysis, and more specifically abstract interpretation through Kleene-based fixpoint computation [10,11,12], has been widely developed and is now considered usable on realistic systems. Those techniques provide an over-approximation of the program semantics by computing *iteratively* a fixpoint in an *abstract domain*. To cope with convergence issues, the iterative sequence of increasing elements is itself approximated using the (in)famous widening operator.

When the target property, e.g., boundedness of the variable values, or unreachability of bad values, cannot be guaranteed, those two elements can be blamed: the choice of the abstract domain and the use of widening.

Since almost 40 years, the set of abstractions proposed by the static analysis community is mainly bound to linear abstractions: from interval arithmetic based analysis, to convex

Pierre Roux
ISAE, University of Toulouse, Toulouse, France

Pierre Roux · Pierre-Loïc Garoche
ONERA - The French Aerospace Lab, Toulouse, France
E-mail: pierre{.roux,-loic.garoche}@onera.fr

polyhedra [13], or less expensive yet precise abstractions such as zonotopes [22] or octagons [31].

Regarding the widening, its use is mandatory but hard to control. Threshold widening, or widening up to [27], for instance strongly depends on the set of thresholds chosen *a priori*, and can give radically different results depending on this choice.

Among the solutions offered for those issues, policy[1] iterations were introduced in the last decade [18,19]. It is inspired by classical approaches in game theory. In short, using policy iterations could replace widening and compute precise fixpoints using numerical solvers. It also enables the use of other than linear abstractions, such as quadratic polynomials [1,20,21], thanks to the recent availability of efficient numerical solvers for convex optimization.

While policy iteration could have a wide impact in static analysis, its current use is hampered by some practical issues:

1. It relies on an *a priori* known set of templates on which the computation is performed. This choice of templates can have a dramatic impact on the result.
   How to choose appropriate templates for a given problem/program?
2. It uses numerical solvers, relying on floating-point arithmetic, and analyzes programs, themselves using floating-point arithmetic.
   How to trust the validity of the resulting invariant?

The goal of the current paper is to address these two issues and therefore to assess and sustain a wider applicability of policy iteration in static analysis.

Targeted programs are the typical time-triggered linear controllers as found in a wide range of critical cyber-physical systems such as car engines, flight commands of an aircraft or even medical devices such as pacemakers or insulin pumps. Most of them are based on a linear update performed within an infinite loop.

For such systems, we offer

- an algorithm to compute appropriate quadratic templates;
- an *a posteriori* proof that the invariant, computed using numerical solvers, is actually valid, even in presence of floating point computations in the analyzed program.

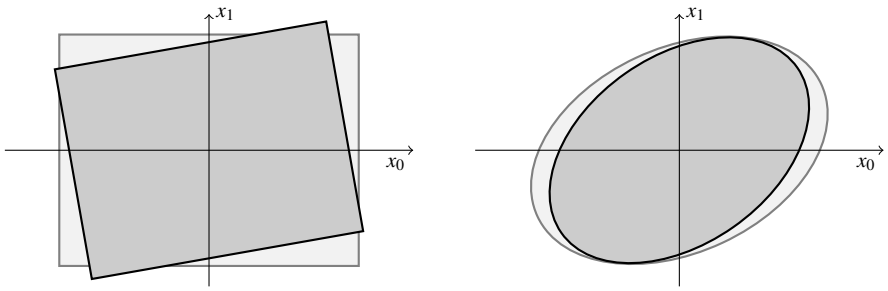
The first step is presented in Section 5 and relies on numerical optimization to identify appropriate templates. This mostly consists in a more comprehensive exposition of material already presented in previous papers [7,36,37,38]. The second step, detailed in Section 7, is based on a set of theorems about error bounds on floating point computations. Their proofs being particularly tedious and error prone, they are supported and mechanically checked by a proof assistant (Coq [8]). Before that, Sections 2, 3 and 4 introduce respectively our interest for quadratic invariants, compared to linear ones, the use of semi-definite programming and policy iterations. Finally, Section 8 gives experimental results.

2 Need for Quadratic Invariants

This section introduces quadratic invariants and compares them to the more usual linear invariants for static analysis of control-command systems.

```
x0 := 0; x1 := 0; x2 := 0;
while −1 ≤ 0 do
  in := ?(−1, 1);
  x0' := x0; x1' := x1; x2' := x2;
  x0 := 0.9379 x0' −0.0381 x1' −0.0414 x2' +0.0237 in;
  x1 := −0.0404 x0' +0.968 x1' −0.0179 x2' +0.0143 in;
  x2 := 0.0142 x0' −0.0197 x1' +0.9823 x2' +0.0077 in;
od
```

Fig. 1: Example of a control-command program.



(a) an interval property: not inductive

(b) a quadratic property: inductive

Fig. 2: Intervals vs quadratic properties for a linear transformation.

2.1 Linear Domains

Most control systems are based on a linear core. This is for instance the case of the Linear Quadratic Gaussian regulator given in Figure 1. Unfortunately, these are hard to analyze using simple linear abstract domains, such as the intervals domain (for instance, the previous regulator does not admit any non trivial invariant in this domain). Figure 2a gives an intuition of this point. An interval property on two variables undergoes a small rotation composed with an homothety of factor $0.92 < 1$ (i.e., a strictly contracting linear transformation), showing that the property is not inductive.

Nevertheless, as control theorists know for long, stable linear systems admit quadratic invariants (called Lyapunov functions [5,30]). Such invariants can be depicted as ellipsoids. On Figure 2b, an ellipsoid is depicted along with its image by the same linear transformation as previously. This time, the property appears to be inductive.

In practice, when a quadratic invariant exists, approximating it with enough faces can give an invariant in classic linear abstract domains such as the polyhedra [13] or the zonotope domains [22]. Thus, it could be thought that quadratic invariants are useless and that the same results can be obtained using solely linear invariants. However, this suffers from two, often prohibitive, drawbacks making it a mere theoretical approach:

Large Number of Faces. "enough faces" can be way too large to be actually tractable, particularly when the number of variables grows. This issue becomes even more stringent in presence of weakly contracting transformations, intuitively requiring the linear invariant to be "smooth" enough to be inductive, hence composed of a large number of faces. More than the memory space required to store these objects, the cost of their ma-

[1] The term *strategy* is also used in the literature, with equivalent meaning.

```
x0 := 0; x1 := 0; x2 := 0; x3 := 0;
while −1 ≤ 0 do
  in0 := ?(−1, 1);
  in1 := ?(−1, 1);
  x0' := x0; x1' := x1; x2' := x2; x3' := x3;
  x0 := 0.6227 x0'+0.03871 x1'−0.113 x2'+0.0102x3'+0.3064 in0+0.1826in1;
  x1 := −0.3407 x0'+0.9103 x1'−0.3388 x2'+0.0649x3'−0.0054 in0+0.6731in1;
  x2 := 0.0918 x0'−0.0265 x1'−0.7319 x2'+0.2669x3'+0.0494 in0+1.6138in1;
  x3 := 0.2643 x0'−0.1298 x1'−0.9903 x2'+0.3331x3'−0.0531 in0+0.4012in1;
  if x0 > 0.5 then x0 := 0.5; else if x0 < −0.5 then x0 := −0.5; fi fi
od
```

Fig. 3: Example of a control-command program with guards.

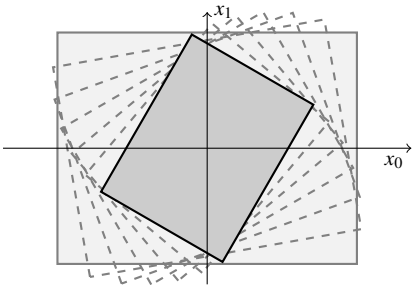


Fig. 4: The interval property of Figure 2 is 6-inductive.

nipulation may render the approach intractable. Compared to linear invariants, quadratic invariants have a space complexity quadratic in the number of variables and are intrinsically "smooth".

Ineffectiveness of Kleene Iterations. Existence of an invariant does not mean existence of a practical way to compute it. In particular, Kleene iterations with polyhedra are known to perform poorly when trying to generate such linear invariants [42]. Moreover none of the classic widening strategies allows to find such results without performing a large number of iterations.

Moreover, control systems can also contain guards. For instance, resets and saturations are two common kinds of guards. *Resets* can at any time reset the value of all program variables to some constant (possibly different from the initial value). They are usually rather easy to handle since they can just be considered as an additional initial value. *Saturations* force a variable – such as x0 on Figure 3 – to remain in some range – $[-0.5, 0.5]$ on Figure 3 – by keeping it constant when it reaches the boundaries of the range. They can be much harder to handle. Adding a saturation to a stable linear system can even make it unstable.

2.2 Unrolling

Unrolling constitutes a practical alternative to the search for linear inductive invariants with myriads of faces. For purely linear systems, unrolling to depths k ranging from a few hundreds to a few thousands allows to compute precise k-inductive invariants while keeping the number of faces reasonably small [15, 22]. Recent work [42] even demonstrates that precise bounds (i.e., the maximum reachable values) can be computed with simple support functions

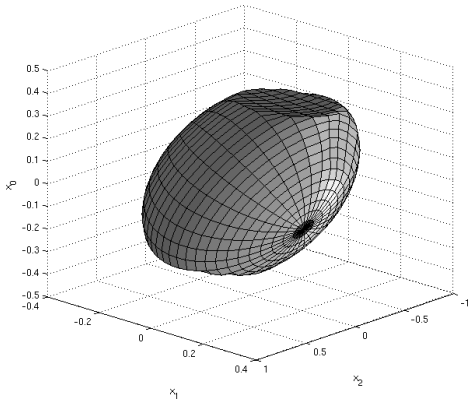


Fig. 5: Invariant for our running example.

by fully unrolling the system. Intuitively, unrolling turns a contracting transformation into a more contracting one. Thus, properties which are not inductive may appear k-inductive. This is illustrated on Figure 4.

These results are definitely interesting but only produce k-inductive invariants for large values of k which exhibits the following drawbacks:

Checking Results. When the user does not trust the analyzer and wants to check its results a posteriori, not having a simple inductive invariant can seriously complicate the task[2].

Difficulty of Unrolling in Presence of Guards. Unrolling purely linear systems[3] works well because the size of the unrolled system is linear in the unrolling depth k. However, when the system contains guards, as on Figure 3, things can become more intricate. Considering all paths through k iterations, can lead to a system of exponential size 2^k which rapidly becomes intractable for large values of k.

2.3 Quadratic Invariants

Although quadratic invariants are known for long, as quadratic *Lyapunov functions*, from control theorists [5,30], their use in static analysis dates back from just a decade.

The first famous use of quadratic invariants for static analysis was two dimensional ellipsoids to bound second order filters [15,16,33]. Bounds on filters of order n could then be computed by decomposing them in filters of order 1 (bounded with intervals) and 2 (bounded with ellipsis) and refining the obtained bounds by means of some kind of unrolling [15,16, 33]. The method then presents the same advantages (precision of the computed bounds) and drawbacks (no inductive invariant is produced) as other unrolling methods.

Other works offer to compute quadratic inductive invariants of higher dimensions on larger classes of linear systems [1,2,20,34]. Such invariants are computed thanks to the use of some numerical solvers, namely semi-definite programming solvers.

[2] Although the k-inductive invariants can be made (1)-inductive by adding extra variables, representing past values of program variables, in their expression.

[3] Like the one in Figure 1.

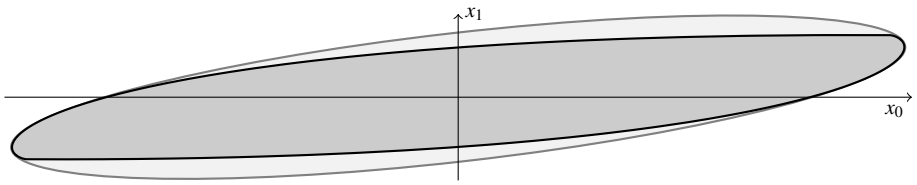


Fig. 6: Reachable state space for Ex. 3 (dark gray), compared to an ellipsoid (light gray).

	linear domains with unrolling [15,16,22,33,42]	quadratic domains [1,2,20,21,38]
pure linear systems	+ high precision	− less precise
simple guards (e.g., reset)	+ easily handled	+ easily handled
other guards (e.g., saturation)	− cannot be handled	+ often handled
size of generated invariants	− potentially huge	+ quadratic

Table 1: Pros and cons of linear domains with unrolling and quadratic domains.

Example 1 In the remainder of this paper, the following invariant[4] will be *fully automatically* computed on the code of Figure 1: $6.2547x_0^2 + 12.1868x_1^2 + 3.8775x_2^2 - 10.61x_0x_1 - 2.4306x_0x_2 + 2.4182x_1x_2 \leq 1.0029 \wedge |x_0| \leq 0.4236 \wedge |x_1| \leq 0.3371 \wedge |x_2| \leq 0.5251$. This invariant is a cropped ellipsoid as displayed on Figure 5.

Example 2 For the code of Figure 3, the following invariant is computed: $0.0483x_0^2 + 0.0591x_1^2 + 0.1351x_2^2 + 0.1388x_3^2 - 0.0415x_0x_1 + 0.0229x_0x_2 - 0.0313x_0x_3 - 0.0763x_1x_2 + 0.0718x_1x_3 - 0.2214x_2x_3 \leq 2.4641 \wedge |x_0| \leq 0.5 \wedge |x_1| \leq 7.0380 \wedge |x_2| \leq 5.5389 \wedge |x_3| \leq 6.1607$.

Remark 1 (Exact Reachable State Space is not an Ellipsoid) Despite the ability of quadratic invariants to bound any stable linear system, it should be noted that the reachable state space of such systems is usually not an ellipsoid. Thus, although ellipsoids are good invariants, they will not always yield the tightest possible bounds.

Example 3 Consider the sequence defined by $x_{(0)} := [0,0]^T$ and $x_{(k+1)} := Ax_{(k)} + Bu_{(k)}$ where

$$A := \begin{bmatrix} 0.92565 & -0.0935 \\ 0.00935 & 0.935 \end{bmatrix} \qquad B := \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

and for all $k \in \mathbb{N}$, $x_{(k)}, u_{(k)} \in \mathbb{R}^{2\times 1}$ are column vectors and $\|u_{(k)}\|_\infty \leq 1$. The reachable state space of this system, depicted in Figure 6, is not an ellipsoid.

Table 1 summarizes the respective advantages and drawbacks of linear invariants with unrolling and quadratic invariants.

This section advocated how nice ellipsoids are to bound linear systems. Yet, they suffer a significant disadvantage compared to classic abstract domains such as polyhedra, octagons, zonotopes,...: the set of ellipsoids with the inclusion order can hardly be equipped with a sensible join operator[5], as illustrated on Figure 7. This constitutes a major obstacle to practical computations through Kleene iterations on the whole set of ellipsoids as usually done in the abstract interpretation framework. A common solution is to choose — prior to

[4] All figures are rounded to the fourth digit.

[5] Although the minimum volume (Löwner-Johns) ellipsoid [6, Section 8.4] could be a reasonable choice.

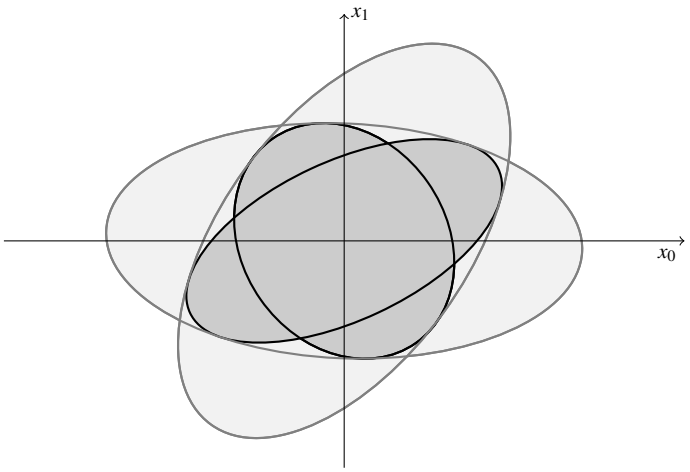


Fig. 7: There is usually no smallest ellipsoid containing two other given ellipsoids: the two light gray ellipsoids both contain the two darker ellipsoids but none of them is included in the other.

the analysis — ellipsoid *shapes* and then only compute their *radii*. Section 4 reviews policy iteration, an efficient technique to compute such radii, while Section 5 offers a technique to determine relevant ellipsoid shapes.

3 Instrumentation: Use of Semi-Definite Programming

Let us first recall some definitions regarding semi-definite programming [6,43].

3.1 Semi-Definite Matrices and Linear Matrix Inequalities

A matrix $M \in \mathbb{R}^{n\times n}$ is called *positive semi-definite*, which will be denoted by $M \succeq 0$, when

$$\forall x \in \mathbb{R}^n,\ x^T M x \geq 0.$$

A matrix $M \in \mathbb{R}^{n\times n}$ is called *positive definite*, which will be denoted by $M \succ 0$, when

$$\forall x \in \mathbb{R}^n,\ x \neq 0 \Rightarrow x^T M x > 0.$$

The notations $M \preceq 0$ and $M \prec 0$ will be used as syntactic sugar for $-M \succeq 0$ and $-M \succ 0$ respectively and $A \succeq B$ and $A \preceq B$ for $A - B \succeq 0$ and $A - B \preceq 0$ respectively.

A Linear Matrix Inequality (LMI) [5] is an inequality of the form

$$\sum_i y_i A_i \succeq 0$$

where the $A_i \in \mathbb{R}^{n\times n}$ are known matrices and the $y_i \in \mathbb{R}$ are scalar variables.

3.2 Semi-Definite Programming

Semi-Definite Programming (SDP) [6,43] aims at minimizing a linear objective function under LMI constraints. That is, given $c_i \in \mathbb{R}$ and $A_{i,j} \in \mathbb{R}^{n\times n}$, we seek values of variables y_i minimizing $\sum_i c_i y_i$ under the constraints $\sum_i y_i A_{i,j} \succeq 0$. We will use the following notation for semi-definite programs:

$$\begin{aligned} \text{minimize } & \textstyle\sum_i c_i y_i \\ \text{subject to } & \textstyle\sum_i y_i A_{i,1} \succeq 0 \\ & \vdots \\ & \textstyle\sum_i y_i A_{i,m} \succeq 0. \end{aligned}$$

Semi-definite programming solvers compute approximate solutions to these programs in polynomial time.

Although variables y_i are scalars, we can easily have matrix variables since a matrix $Y \in \mathbb{R}^{n\times n}$ can be expressed as $\sum_{i=0,j=0}^{n-1,n-1} Y_{i,j} E^{i,j}$, where $E^{i,j}$ is the matrix with zeros everywhere except a one at line i and column j.

3.3 Abstract Implications with Lagrangian Relaxation

We will often encounter implications "⇒". They will be handled thanks to a Lagrangian relaxation.

Theorem 1 (Lagrangian relaxation) *Assume f and $g_1,\dots,g_k$ functions $\mathbb{R}\to\mathbb{R}$, if there exist $\lambda_1,\dots,\lambda_k \in \mathbb{R}$ all non negative such that.*

$$\forall x, f(x) - \sum_i \lambda_i g_i(x) \geq 0 \tag{1}$$

then

$$\forall x, \left(\bigwedge_i g_i(x) \geq 0\right) \Rightarrow f(x) \geq 0. \tag{2}$$

Semi-definite programming solvers being unable to directly handle (2), they are fed with (1). Although the converse of the theorem does not generally hold, this relaxation is usually an efficient way to handle implications, it is even exact in some cases [6, Section B.2].

Example 4 For any $P, P_1,\dots,P_k \in \mathbb{R}^{n\times n}$ and $b, b_1,\dots,b_k \in \mathbb{R}$, the following

$$\exists \lambda_1,\dots,\lambda_k \in \mathbb{R}, \left(\bigwedge_{i=1}^k \lambda_i \geq 0\right) \wedge \begin{bmatrix} -P & 0 \\ 0 & b \end{bmatrix} - \sum_{i=1}^k \lambda_i \begin{bmatrix} -P_i & 0 \\ 0 & b_i \end{bmatrix} \succeq 0 \tag{3}$$

is a sufficient condition for

$$\forall x \in \mathbb{R}^n, \left(\bigwedge_{i=1}^k x^T P_i x \leq b_i\right) \Rightarrow x^T P x \leq b. \tag{4}$$

The previous relaxation is sometimes called S-Procedure by control theorists [17].

$M' := 0$;
for j **from** 1 **to** n **do**
  for i **from** 1 **to** $j-1$ **do**
    $M'_{i,j} := \left(M_{i,j} - \sum_{k=1}^{i-1} M'_{k,i} M'_{k,j}\right) / M'_{i,i}$;
  od
  $M'_{j,j} := \sqrt{M_{j,j} - \sum_{k=1}^{j-1} {M'_{k,j}}^2}$;
od

Fig. 8: Cholesky decomposition: from a matrix $M \succeq 0$, computes M' such that $M = M'^T M'$.

3.4 Checking Positive Definiteness

To prove that a scalar $r \in \mathbb{R}$ is non negative, one can exhibit some $r' \in \mathbb{R}$ such that $r = r'^2$ (typically $r' = \sqrt{r}$). Similarly, one can prove that a matrix $M \in \mathbb{R}^{n \times n}$ is positive semi-definite by exposing a matrix M' such that $M = M'^T M'$ (since, for all $x \in \mathbb{R}^n$, $x^T (M'^T M') x = (M'x)^T (M'x) = \|M'x\|_2^2 \geq 0$ for all $x \in \mathbb{R}^n$). The Cholesky decomposition is an algorithm that computes such a matrix M' (c.f., Figure 8). It is interesting to notice that the actual value of r' or M' doesn't matter. It is enough to prove it exists. Indeed, if the Cholesky decomposition of M runs to completion, without ever attempting to take the square root of a negative value or perform a division by zero, hence produces a M', this proves $M \succeq 0$. This property will later be used to check the results of SDP solvers.

4 Policy Iterations: State of the Art

Computation of precise invariants on numerical programs can be hard to achieve using classic Kleene iterations with widening. Policy iterations [1,9,19,20,21] is one of the alternatives to simple widening developed during the last decade [4,14,23,26,41, and references therein]. This technique allows computing precise postfixpoints, usually by relying on mathematical optimization solvers. Such techniques have been recently developed for the computation of quadratic invariants for linear systems [1,20,21].

Policy iterations basically perform iterations with two phases

- *Compute a policy*, that is locally simplify the fixpoint problem;
- *Solve the policy* with efficient tools specialized for this simpler problem.

These two phases are alternatively performed until a good result is reached.

4.1 Template Domains

Policy iteration is performed on template domains. Given a finite set $\{t_1, \ldots, t_n\}$ of expressions on program variables $\mathbb{V}$, the template domain $\mathscr{T}$ is defined as $\overline{\mathbb{R}}^n = (\mathbb{R} \cup \{-\infty, +\infty\})^n$ and the meaning of an abstract value $(b_1, \ldots, b_n) \in \mathscr{T}$ is the set of environments

$$\gamma_{\mathscr{T}}(b_1, \ldots, b_n) = \{\rho \in (\mathbb{V} \to \mathbb{R}) \mid [\![t_1]\!](\rho) \leq b_1, \ldots, [\![t_n]\!](\rho) \leq b_n\}$$

where $[\![t_i]\!](\rho)$ is the result of the evaluation of expression t_i in environment ρ. In other words, the abstract value $(b_1, \ldots, b_n)$ represents all the environments satisfying all the constraints $t_i \leq b_i$.

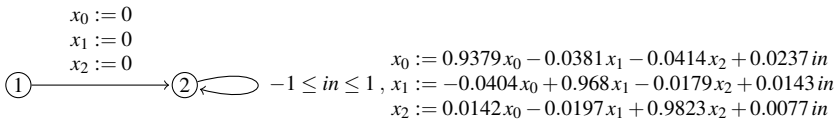


Fig. 9: Control flow graph for our running example.

Example 5 Given the quadratic templates $t_1 := 6.2547x_0^2 + 12.1868x_1^2 + 3.8775x_2^2 - 10.61x_0x_1 - 2.4306x_0x_2 + 2.4182x_1x_2$, $t_2 := x_0^2$, $t_3 := x_1^2$ and $t_4 := x_2^2$, the quadratic invariant from Example 1, page 5, can be written $(1.0029, 0.1795, 0.1136, 0.2757) \in \mathscr{T}$.

Indeed, many common abstract domains can be rephrased as template domains. For instance the intervals domain is obtained with templates $-x_i$ and x_i for all variables $x_i \in \mathbb{V}$ and the octagon domain [31] by adding all the $\pm x_i \pm x_j$. The shape of the templates to be considered for policy iteration depends on the optimization tools used. For instance, linear programming [18,19] allows any linear templates whereas quadratic templates can be handled thanks to semi-definite programming [1,20,21]. This paper focuses on the latter case.

4.2 System of Equations

While Kleene iterations iterate locally through each construct of the program, policy iterations require a global view on the analyzed program. For that purpose, the whole program is first translated into a system of equations which is later solved.

Starting from the control flow graph of the analyzed program, a system of equations is defined with a variable $b_{i,j}$ for each vertex i of the graph and each template t_j.

Example 6 Figure 9 displays the control flow graph for our running example (Figure 1, page 3). Here is its translation as a system of equations:

$$\begin{cases} b_{1,1} = +\infty \qquad b_{1,2} = +\infty \qquad b_{1,3} = +\infty \qquad b_{1,4} = +\infty \\ b_{2,1} = \max\{0 \mid \mathrm{be}(1)\} \sqcup \max\{r(t_1) \mid (-1 \leq in \leq 1) \wedge \mathrm{be}(2)\} \\ b_{2,2} = \max\{0 \mid \mathrm{be}(1)\} \sqcup \max\{r(t_2) \mid (-1 \leq in \leq 1) \wedge \mathrm{be}(2)\} \\ b_{2,3} = \max\{0 \mid \mathrm{be}(1)\} \sqcup \max\{r(t_3) \mid (-1 \leq in \leq 1) \wedge \mathrm{be}(2)\} \\ b_{2,4} = \max\{0 \mid \mathrm{be}(1)\} \sqcup \max\{r(t_4) \mid (-1 \leq in \leq 1) \wedge \mathrm{be}(2)\} \end{cases} \tag{5}$$

where $\mathrm{be}(i)$ denotes $(t_1 \leq b_{i,1}) \wedge (t_2 \leq b_{i,2}) \wedge (t_3 \leq b_{i,3}) \wedge (t_4 \leq b_{i,4})$ and $r(t)$ is the template t in which variable x_0 is replaced by $0.9379x_0 - 0.0381x_1 - 0.0414x_2 + 0.0237in$, variable x_1 is replaced by $-0.0404x_0 + 0.968x_1 - 0.0179x_2 + 0.0143in$ and variable x_2 is replaced by $0.0142x_0 - 0.0197x_1 + 0.9823x_2 + 0.0077in$. The usual maximum on $\overline{\mathbb{R}}$ is denoted $\sqcup$[6].

Each $b_{i,j}$ bounds the template t_j at program point i and is defined in one equation as a maximum over as many terms as incoming edges in i. More precisely, each edge between two vertices v and v' translates to a term in each equation $b_{v',j}$ on the pattern: $\max\{r(t_j) \mid c \wedge \bigwedge_j (t_j \leq b_{v,j})\}$ where c and r are respectively the constraints and the assignments associated to this edge. This expresses the maximum value the template t_j can reach

[6] $\vee$ is often used instead in the policy iteration literature.

in destination vertex v' when applying the assignments r on values satisfying both the constraints c of the edge and the constraints $t_j \leq b_{v,j}$ of the initial vertex v. Finally, the program starting point is initialized to $(+\infty,\ldots,+\infty)$, meaning all equations for $b_{i_0,j}$, where i_0 is the starting point, become $b_{i_0,j} = +\infty$.

Thus, for any solution $(b_{1,1},\ldots,b_{1,n},\ldots)$ of the equations, $\gamma_{\mathscr{T}}(b_{i,1},\ldots,b_{i,n})$ is an overapproximation of reachable states of the program at point i. According to the Knaster-Tarski theorem, this set of solutions has a least element which then gives the best overapproximation of the reachable state space of the program.

4.3 Policy Iterations

Policy iterations intend to compute the least solution of the previous system of equations. They are an iterative process with two phases. First, an abstraction of the problem is computed. This abstraction, called *policy*, can then be solved using techniques which were not applicable on the original problem. This gives an approximation of the final result enabling to find a better policy, itself giving a better approximation of the result and so on.

Two different techniques, min- and max-policies, can be found in the literature. They basically apply the previous scheme top down or bottom up respectively.

4.3.1 Min-Policy Iterations

To some extent, Min-Policy iterations [1] can be seen as a very efficient *narrowing*, since they perform descending iterations from a postfixpoint towards some fixpoint, working in a way similar to the Newton-Raphson numerical method. Iterations are not guaranteed to reach a fixpoint but can be stopped at any time leaving an overapproximation thereof. Moreover, convergence is usually fast.

Writing a system of equations $b = F(b)$ with $b = (b_{i,j})_{i\in[\![1,n]\!],j\in[\![1,p]\!]}$ and $F : \overline{\mathbb{R}}^{np} \to \overline{\mathbb{R}}^{np}$ (n being the number of templates and p the number of vertices in the control flow graph), a min-policy is defined as follows: $\underline{F}$ is a min-policy for F if for every $b \in \overline{\mathbb{R}}^{np}$, $F(b) \leq \underline{F}(b)$ and there exist some $b_0 \in \overline{\mathbb{R}}^{np}$ such that $\underline{F}(b_0) = F(b_0)$. We will only consider *linear* min-policies in the remaining of this paper. For instance, for a smooth concave function, its min-policies are the tangents to its graph.

Example 7 Considering the system of one equation $b_{1,1} = 0 \sqcup \sqrt{b_{1,1}}$, where $\sqrt{x}$ is defined as $-\infty$ for negative numbers x, $\underline{F}$ defined as $\underline{F}(b) := 0 \sqcup \left(\frac{b_{1,1}}{8} + 2\right)$ is a min-policy. Indeed, for all $b_{1,1} \in \overline{\mathbb{R}}$, $F(b) = 0 \sqcup \sqrt{b_{1,1}} \leq 0 \sqcup \frac{b_{1,1}}{8} + 2 = \underline{F}(b)$, and for $b_0 = 16$, $F(b_0) = \sqrt{16} = \frac{16}{8} + 2 = \underline{F}(b_0)$. This is illustrated on Figure 10 on which $\underline{F}_1$ is the above $\underline{F}$.

The following theorem can then be used to compute the least fixpoint of F.

Theorem 2 *Given a (potentially infinite) set $\underline{\mathscr{F}}$ of min-policies for F. If for all $b \in \overline{\mathbb{R}}^{np}$ there exist a policy $\underline{F} \in \underline{\mathscr{F}}$ interpolating F at point b (i.e. $\underline{F}(b) = F(b)$) and if each $\underline{F} \in \underline{\mathscr{F}}$ has a least fixpoint* $\operatorname{lfp}\underline{F}$*, then the least fixpoint of F satisfies*

$$\operatorname{lfp} F = \bigwedge_{\underline{F}\in\underline{\mathscr{F}}} \operatorname{lfp}\underline{F}.$$

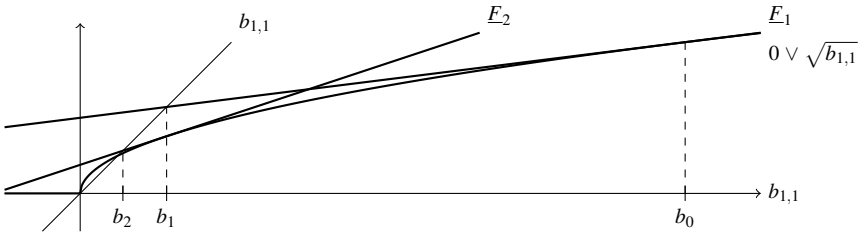


Fig. 10: Illustration of Example 8.

Remark 2 This enables to better understand the name *min*-policies since, in the hypotheses of the previous theorem, F is the pointwise minimum of the min-policies $\underline{F} \in \underline{\mathscr{F}}$:

$$F = \bigwedge_{\underline{F} \in \underline{\mathscr{F}}} \underline{F}.$$

Iterations are done with two main objects: a min-policy $\underline{F}$ and a tuple b of values for variables $b_{i,j}$ of the system of equations. The following policy iteration algorithm starts from some postfixpoint b_0 of F and aims at refining it to produce a better overapproximation of a fixpoint of F. Policy iteration algorithms always proceed by iterating two phases: first a policy $\underline{F}_i$ is selected, then it is solved giving some b_i. More precisely in our case:

- find a linear min-policy $\underline{F}_{i+1}$ being tangent to F at point b_i, this can be done thanks to SDP solvers [21, Section 5.4];
- compute the least fixpoint b_{i+1} of policy $\underline{F}_{i+1}$ thanks to linear programming.

Iterations can be stopped at any point (for instance after a fixed number of iterations or when progress between b_i and b_{i+1} is considered small enough) leaving an overapproximation b of a fixpoint of F.

Example 8 We perform min-policy iterations on the system of equation of Example 7.

- We start from the postfixpoint $b_0 = 16$. This postfixpoint could be obtained through Kleene iterations for instance[7].
- For each term of the unique equation, we look for a hyperplane tangent to the term at point b_0. 0 is tangent to 0 at point b_0 and $\frac{b_{1,1}}{8} + 2$ is tangent to $\sqrt{b_{1,1}}$ at point b_0 (c.f., Figure 10), this gives the following linear min-policy:
  $\underline{F}_1 = \left\{ b_{1,1} = 0 \sqcup \left(\frac{b_{1,1}}{8} + 2 \right) \right.$
- The least fixpoint of $\underline{F}_1$ is then: $b_1 = \frac{16}{7} \simeq 2.2857$.
- Looking for hyperplanes tangent at point b_1 gives the min-policy:
  $\underline{F}_2 = \left\{ b_{1,1} = 0 \sqcup \left(\frac{\sqrt{7}}{8} b_{1,1} + \frac{2}{\sqrt{7}} \right) \right.$
- Hence $b_2 = \frac{16}{8\sqrt{7}-7} \simeq 1.1295$.

These two first iterations are illustrated on Figure 10. The procedure then rapidly converges to the fixpoint $b_{1,1} = 1$ (the next iterates being $b_3 \simeq 1.0035$ and $b_4 \simeq 1.0000$) and can be stopped as soon as the accuracy is deemed satisfying.

[7] Or a large enough guess can be used. Thanks to the fast convergence of min-policy iterations, there is often no need for this postfixpoint to be close from the fixpoint eventually computed.

Remark 3 The Newton-Raphson method on a smooth concave function is a particular case of min-policy iterations.

Example 9 We perform min-policy iterations on the running example (Equation (5), page 10).

- We start from the postfixpoint $b_0 = (+\infty, +\infty, +\infty, +\infty, 1000000, +\infty, +\infty, +\infty)$, which could be obtained through Kleene iterations for instance.
- For each term of each equation, we look for an hyperplane tangent to the term at point b_0. This can be done thanks to SDP solvers and gives the following linear min-policy: $\underline{F}_1 =$
$$\begin{cases} b_{1,1} = +\infty & b_{1,2} = +\infty & b_{1,3} = +\infty & b_{1,4} = +\infty \\ b_{2,1} = 0 \sqcup 0.9857\, b_{2,1} + 0.0152 & b_{2,2} = 0 \sqcup 0.2195\, b_{2,1} + 11.0979 \\ b_{2,3} = 0 \sqcup 0.1143\, b_{2,1} + 4.8347 & b_{2,4} = 0 \sqcup 0.2669\, b_{2,1} + 3.9796 \end{cases}$$
For instance, looking at the second term of $b_{2,1}$, i.e., $\max\{r(t_1) \mid (-1 \leq in \leq 1) \wedge \mathrm{be}(2)\}$, that is $\max\{r(t_1) \mid (-1 \leq in \leq 1) \wedge (t_1 \leq 1000000)\}$ at point b_0. This can be rewritten
$$\max\left\{x^T R_r^T P_{t_1} R_r x \;\middle|\; x^T E_{in} x \leq 1 \wedge x^T P_{t_1} x \leq 1000000\right\} \tag{6}$$
where R_r is a matrix encoding assignment r (i.e., $R_r x$ is the result of assignment r on variables x) and P_{t_1} and E_{in} are matrix representation of respectively template t_1 and in^2 (i.e., $x^T P_{t_1} x$ is the template t_1 on variables x). In our case $x := [x_0\ x_1\ x_2\ in]^T$,
$$R_r := \begin{bmatrix} 0.9379 & -0.0381 & -0.041 & 0.0237 \\ -0.0404 & 0.968 & -0.0179 & 0.0143 \\ 0.0142 & -0.0197 & 0.9823 & 0.0077 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad P_{t_1} := \begin{bmatrix} 6.2547 & -5.305 & -1.2153 & 0 \\ -5.305 & 12.1868 & 1.2091 & 0 \\ -1.2153 & 1.2091 & 3.9775 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$
$$E_{in} := \begin{bmatrix} 0_{3\times 3} & 0_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix}$$
(6) can be rewritten
$$\max\left\{\mathrm{tr}((R_r^T P_{t_1} R_r)(x^T x)) \;\middle|\; \mathrm{tr}(E_{in}(x^T x)) \leq 1 \wedge \mathrm{tr}(P_{t_1}(x^T x)) \leq 1000000\right\}$$
and overapproximated by
$$\max_X \left\{\mathrm{tr}((R_r^T P_{t_1} R_r) X) \;\middle|\; \mathrm{tr}(E_{in} X) \leq 1 \wedge \mathrm{tr}(P_{t_1} X) \leq 1000000 \wedge X \succeq 0\right\}$$
which by duality [6, Chapter 5] is less or equal
$$\min_y \left\{1 y_0 + 1000000 y_1 \;\middle|\; y_0 E_{in} + y_1 P_{t_1} \succeq R_r^T P_{t_1} R_r \wedge y_0 \geq 0 \wedge y_1 \geq 0\right\}.$$
A SDP solver eventually answers $y_0 = 0.0152$ and $y_1 = 0.9857$ on the previous problem.
- A linear programming solver allows computing the least fixpoint of $\underline{F}_1$: $b_1 = (+\infty, +\infty, +\infty, +\infty, 1.0664, 11.3324, 4.9568, 4.2644)$.
- $\underline{F}_2 =$
$$\begin{cases} b_{1,1} = +\infty & b_{1,2} = +\infty & b_{1,3} = +\infty & b_{1,4} = +\infty \\ b_{2,1} = 0 \sqcup 0.9857\, b_{2,1} + 0.0143 & b_{2,2} = 0 \sqcup 0.2302\, b_{2,1} + 0.0120 \\ b_{2,3} = 0 \sqcup 0.1190\, b_{2,1} + 0.0052 & b_{2,4} = 0 \sqcup 0.2708\, b_{2,1} + 0.0042 \end{cases}$$
- $b_2 = (+\infty, +\infty, +\infty, +\infty, 1.0029, 0.2429, 0.1245, 0.2757)$.

- $\underline{F}_3 = \begin{cases} b_{1,1} = +\infty & b_{1,2} = +\infty & b_{1,3} = +\infty & b_{1,4} = +\infty \\ b_{2,1} = 0 \sqcup 0.9857\, b_{2,1} + 0.0143 \\ b_{2,2} = 0 \sqcup 0.0390\, b_{2,1} + 0.7426\, b_{2,2} + 0.0114 \\ b_{2,3} = 0 \sqcup 0.0340\, b_{2,1} + 0.6635\, b_{2,3} + 0.0050 \\ b_{2,4} = 0 \sqcup 0.2709\, b_{2,1} + 0.0040 \end{cases}$
- $b_3 = (+\infty, +\infty, +\infty, +\infty, 1.0029, 0.1962, 0.1160, 0.2757)$.
- $\underline{F}_4 = \begin{cases} b_{1,1} = +\infty & b_{1,2} = +\infty & b_{1,3} = +\infty & b_{1,4} = +\infty \\ b_{2,1} = 0 \sqcup 0.9857\, b_{2,1} + 0.0143 \\ b_{2,2} = 0 \sqcup 0.0194\, b_{2,1} + 0.8340\, b_{2,2} + 0.0104 \\ b_{2,3} = 0 \sqcup 0.0214\, b_{2,1} + 0.7688\, b_{2,3} + 0.0049 \\ b_{2,4} = 0 \sqcup 0.2709\, b_{2,1} + 0.0040 \end{cases}$
- $b_4 = (+\infty, +\infty, +\infty, +\infty, 1.0029, 0.1803, 0.1137, 0.2757)$.

Two additional iterations lead to $b_6 = (+\infty, +\infty, +\infty, +\infty, 1.0029, 0.1795, 0.1136, 0.2757)$ which is the invariant given in Example 1 and depicted on Figure 5.

Remark 4 (Number and size of semi-definite programs) At each iteration, one semi-definite program has to be solved for each term of each equation in order to compute a new policy. This leads to many semi-definite programs but each focusing on a single term, hence rather small. The computed policies being linear are then solved through linear programming. This way, at the scale of the whole system, only linear programs are solved, which scales better than semi-definite programming.

4.3.2 Max-Policy Iterations

Behaving somewhat as a super *widening*, Max-Policy iterations [20] work in the opposite direction compared to Min-Policy iterations. They start from bottom and iterate computations of greatest fixpoints on a set of max-policies until a global fixpoint is reached. Unlike the previous approach, this terminates with a *theoretically* precise fixpoint, but the user has to wait until the end since intermediate results are not overapproximations of a fixpoint.

Max-policies are the dual of min-policies: $\overline{F}$ is a max-policy for F if for every $b \in \overline{\mathbb{R}}^{np}$, $\overline{F}(b) \leq F(b)$ and there exist some $b_0 \in \overline{\mathbb{R}}^{np}$ such that $\overline{F}(b_0) = F(b_0)$. In particular, the choice of one term in each equation is a max-policy. From now on, only this last kind of max-policies will be considered.

Example 10 A max-policy of the system of equations from Example 6:

$$\begin{cases} b_{1,1} = +\infty & b_{1,2} = +\infty & b_{1,3} = +\infty & b_{1,4} = +\infty \\ b_{2,1} = \max\{r(t_1) \mid (-1 \leq in \leq 1) \wedge \mathrm{be}(2)\} \\ b_{2,2} = \max\{0 \mid \mathrm{be}(1)\} \\ b_{2,3} = \max\{0 \mid \mathrm{be}(1)\} \\ b_{2,4} = \max\{r(t_4) \mid (-1 \leq in \leq 1) \wedge \mathrm{be}(2)\} \end{cases}$$

Theorem 3 *Given the set $\overline{\mathscr{F}}$ of max-policies for F as defined above (choice of one term in each equation), any fixpoint of F is also a fixpoint of some $\overline{F}_0 \in \overline{\mathscr{F}}$.*

Iterations are again done with two main objects: a max-policy $\overline{F}$ and a tuple b of values for variables $b_{i,j}$ of the system of equations. The following policy iteration algorithm aims at finding a policy $\overline{F}_0$ as in the above theorem by solving optimization problems. The initial value $b_0 := (-\infty, \ldots, -\infty)$ is chosen, then policies are iterated:

- find an improving policy $\overline{F}_{i+1}$ at point b_i, i.e. that reaches (strictly) greater values evaluated at point b_i, this can be done by evaluating each term of the system of equations at point b_i [21, Section 6.2]; if none is found, exit;
- compute the greatest fixpoint[8] b_{i+1} of policy $\overline{F}_{i+1}$.

The last tuple b is then a fixpoint of the whole system of equations.

Remark 5 Although min and max policies are dual concepts, we are in both cases looking for *over*approximations of the least fixpoint of the system of equations, thus the algorithms are *not* dual.

Example 11 We perform max-policy iterations on the running example (Equation (5), page 10).

- We start with the initial value
  $b_0 = (-\infty, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty, -\infty)$.
- We now look for an improving policy $\overline{F}_1$ at point b_0. For the first four equations, there is no choice and the term $+\infty$ is chosen. For the first remaining equations, replacing the $b_{i,j}$ with values $-\infty$ from b_0 in $\mathrm{be}(1)$ and $\mathrm{be}(2)$ gives formula equivalent to $false$, hence both terms of these equations are maximum of the empty set and evaluate to $-\infty$. We can then choose any of them and the following policy is then a suitable choice:
  $\overline{F}_1 =$
  $$\begin{cases} b_{1,1} = +\infty & b_{1,2} = +\infty & b_{1,3} = +\infty & b_{1,4} = +\infty \\ b_{2,1} = \max\{0 \mid \mathrm{be}(1)\} & & b_{2,2} = \max\{0 \mid \mathrm{be}(1)\} & \\ b_{2,3} = \max\{0 \mid \mathrm{be}(1)\} & & b_{2,4} = \max\{0 \mid \mathrm{be}(1)\}. & \end{cases}$$
- Hence
  $b_1 = (+\infty, +\infty, +\infty, +\infty, -\infty, -\infty, -\infty, -\infty)$.
- We again look for an improving policy $\overline{F}_2$, but this time at point b_1. There is still no choice for the first four equations. In the four remaining equations, replacing the $b_{i,j}$ with values from b_1 in $\mathrm{be}(1)$ and $\mathrm{be}(2)$ respectively gives formula equivalent to $true$ and $false$. This way, for these four equations, the first term reduces to 0 whereas the second term evaluates to $-\infty$. 0 being greater than the $-\infty$ from b_1, we get an improving policy
  $\overline{F}_2 =$
  $$\begin{cases} b_{1,1} = +\infty & b_{1,2} = +\infty & b_{1,3} = +\infty & b_{1,4} = +\infty \\ b_{2,1} = \max\{0 \mid \mathrm{be}(1)\} & & b_{2,2} = \max\{0 \mid \mathrm{be}(1)\} & \\ b_{2,3} = \max\{0 \mid \mathrm{be}(1)\} & & b_{2,4} = \max\{0 \mid \mathrm{be}(1)\}. & \end{cases}$$
- $b_2 = (+\infty, +\infty, +\infty, +\infty, 0, 0, 0, 0)$.
- Now that the $b_{2,j}$ in b_2 are no longer $-\infty$, $\mathrm{be}(2)$ is no longer $false$ and it becomes interesting to select the second terms in the last four equations, hence
  $\overline{F}_3 =$
  $$\begin{cases} b_{1,1} = +\infty & b_{1,2} = +\infty & b_{1,3} = +\infty & b_{1,4} = +\infty \\ b_{2,1} = \max\{r(t_1) \mid -1 \leq in \leq 1 \wedge \mathrm{be}(2)\} & & & \\ b_{2,2} = \max\{r(t_2) \mid -1 \leq in \leq 1 \wedge \mathrm{be}(2)\} & & & \\ b_{2,3} = \max\{r(t_3) \mid -1 \leq in \leq 1 \wedge \mathrm{be}(2)\} & & & \\ b_{2,4} = \max\{r(t_4) \mid -1 \leq in \leq 1 \wedge \mathrm{be}(2)\}. & & & \end{cases}$$

[8] More precisely, first determine which $b_{i,j}$ are $\pm\infty$ in the least fixpoint in $\overline{\mathbb{R}}^{np}$ greater than b_i, then compute a greatest fixpoint for the remaining values in $\mathbb{R}$.

– The greatest fixpoint of $\overline{F}_3$ can be computed thanks to SDP solvers [21, Section 3.7]: $b_3 = (+\infty, +\infty, +\infty, +\infty, 1.0077, 0.1801, 0.1141, 0.2771)$. More precisely, the greatest fixpoint of $\overline{F}_3$ is, according to the Knaster-Tarski theorem, its largest prefixpoint. We are then looking for maximal $b_{2,1}$, $b_{2,2}$, $b_{2,3}$ and $b_{2,4}$ satisfying

$$\begin{cases} b_{2,1} \geq \max\{r(t_1) \mid -1 \leq in \leq 1 \wedge \mathrm{be}(2)\} \\ b_{2,2} \geq \max\{r(t_2) \mid -1 \leq in \leq 1 \wedge \mathrm{be}(2)\} \\ b_{2,3} \geq \max\{r(t_3) \mid -1 \leq in \leq 1 \wedge \mathrm{be}(2)\} \\ b_{2,4} \geq \max\{r(t_4) \mid -1 \leq in \leq 1 \wedge \mathrm{be}(2)\}. \end{cases}$$

or equivalently for

$$\max_{b_{2,i}} \left\{ b_{2,1} + b_{2,2} + b_{2,3} + b_{2,4} \;\middle|\; -1 \leq in \leq 1 \wedge \mathrm{be}(2) \wedge \bigwedge_i r(t_i) \leq b_{2,i} \right\}$$

which can be encoded as a SDP program, similarly to what was done in Example 9.

– No more improving policy.

After four iterations, the algorithm has found the same least fixpoint than min policies in Example 9.

Remark 6 (Number and size of semi-definite programs) Contrary to min-policies (c.f., Remark 4), max-policies are not linear. Solving them then requires semi-definite programs whereas min-policies only solves linear programs at the scale of the whole system.

The Max-Policy iteration builds an ascending chain of abstract elements similarly to Kleene iterations elements. However it is guaranteed to be finite, while Kleene iterations require the use of widening to ensure termination. Indeed, since there are exponentially many max-policies, in the number of templates and points of the control flow graph, and since each policy can give only an exponential number of solutions[9], we have a bound on the number of iterations. And despite these exponential bounds, in practice, only a small number of policies are usually considered and the number of iterations remains reasonable.

5 Template Generation

Template domains used by policy iteration require templates to be given prior to the analyses. This greatly limits the automaticity of the method. However, heuristics can be designed for linear systems of the form $x_{(k+1)} = Ax_{(k)} + Bu_{(k)}$, like our running example. Those are ubiquitous in control applications where the vector x represents the internal state of the controller and u a bounded input.

After a brief introduction to Lyapunov stability theory, this section first focuses on generating templates for pure linear systems then for guarded linear systems given as a control flow graph.

[9] More precisely, for a given policy $\overline{F}_{i+1}$, once determined which $b_{i,j}$ are $\pm\infty$ there is a unique greatest fixpoint for the remaining $b_{i,j} \in \mathbb{R}$, hence finitely many possible b_{i+1}.

5.1 Introduction to Lyapunov Stability Theory

One common way to establish stability of a discrete, time-invariant closed (i.e., with no inputs) system described in state space form, (i.e., $x_{(k+1)} = f(x_{(k)})$) is to use what is called a Lyapunov function. It is a function $V : \mathbb{R}^n \to \mathbb{R}$ which must satisfy the following properties

$$V(0) = 0 \wedge \forall x \in \mathbb{R}^n \backslash \{0\}, V(x) > 0 \wedge \lim_{\|x\| \to \infty} V(x) = \infty \tag{7}$$

$$\forall x \in \mathbb{R}^n, V(f(x)) - V(x) \leq 0. \tag{8}$$

It is shown, for instance in [25], that exhibiting such a function proves the Lyapunov stability of the system, meaning that its state variables will remain bounded through time. Equation (8) expresses the fact that the function $k \mapsto V(x_{(k)})$ decreases, which, combined with (7), shows that the state variables remain in the bounded sublevel-set $\{x \in \mathbb{R}^n | V(x) \leq V(x_{(0)})\}$ at all instants $k \in \mathbb{N}$.

In the case of Linear Time Invariant systems [5] (of the form $x_{(k+1)} = Ax_{(k)}$, with $A \in \mathbb{R}^{n \times n}$), one can always look for V as a quadratic form in the state variables of the system: $V(x) = x^T P x$ with $P \in \mathbb{R}^{n \times n}$ a symmetric matrix such that

$$P \succ 0 \tag{9}$$

$$A^T P A - P \preceq 0. \tag{10}$$

Now, to account for the presence of an external input to the system (which is usually the case with controllers: they use data collected from sensors to generate their output), the model is usually extended into the form

$$x_{(k+1)} = Ax_{(k)} + Bu_{(k)}, \|u_{(k)}\|_\infty \leq 1. \tag{11}$$

The condition $\|u_{(k)}\|_\infty \leq 1$ reflects the fact that values coming from input sensors usually lie in a given range. The bound 1 is chosen without loss of generality since one can always alter the matrix B to account for different bounds. Then, through a slight reinforcement of Equation (10) into

$$A^T P A - P \prec 0 \tag{12}$$

we can still guarantee that the state variables x of (11) will remain in the sublevel set (for some $\lambda > 0$) $\left\{x \in \mathbb{R}^n \;\middle|\; x^T P x \leq \lambda\right\}$, which is an ellipsoid in this case, as illustrated on Figure 11. This approach only enables us to study control laws that are inherently stable, i.e., stable when taken separately from the plant they control. Nevertheless a wide range of controllers remains that can be analyzed. In addition, inherent stability is required in a context of critical applications.

These stability proofs have the very nice side effect that they provide a quadratic invariant on the state variables, which can be used at the code level to find bounds on the program variables. Furthermore, there are many P matrices that fulfill the equations described above. This gives some flexibility as to the choice of such a matrix: by adding relevant constraints on P, one can obtain increasingly better bounds.

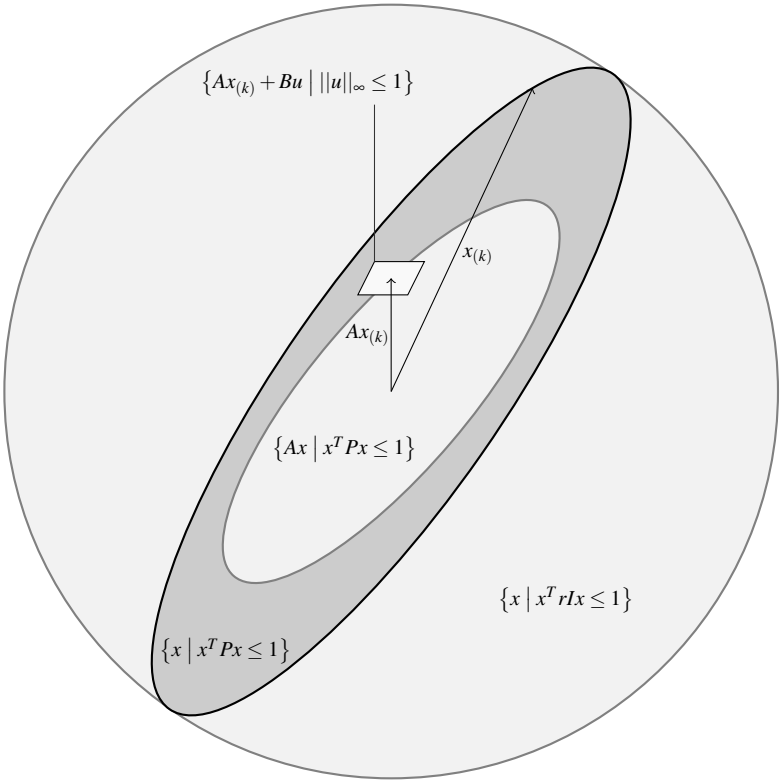


Fig. 11: Illustration of the stability concepts: if $x_{(k)}$ is in the dark gray ellipse, then, after a time step, $Ax_{(k)}$ is in the light gray one, which is exactly what is expressed by Equation (10). The white box represents the potential values of $x_{(k+1)}$ after adding the effect of the bounded input $u_{(k)}$. We see here the necessity that the light gray ellipse be strictly included in the dark gray one, which is the stronger condition expressed by Equation (12). We will then look for an invariant ellipsoid included in the smallest possible sphere by maximizing r.

5.2 Generating Templates

Given a pure linear system ($x_{(k+1)} = Ax_{(k)} + Bu_{(k)}$ with $\|u_{(k)}\|_\infty \leq 1$), we want to generate a quadratic template enabling policy iterations to bound the system. According to Section 5.1, any positive definite matrix P solution of the Lyapunov equation (12) gives a quadratic template $t := x^T Px$ enabling policy iterations to bound the system. Section 3 introduced semi-definite programming which constitutes an efficient way to solve this equation. However, taking any random solution may lead to very grossly overapproximated invariants. It would be interesting to constrain more the set of solutions. Multiple approaches exist [38], the one we detail here gives good results at a reasonable cost.

The basic idea is to force the invariant to lie in a sphere as small as possible. More precisely, we will look for an ellipsoid included in the smallest possible sphere and which is stable, i.e., a symmetric positive definite matrix P with r maximal in $P \succeq rI$ such that

$$\forall x, \forall u, \left(||u||_\infty \leq 1 \wedge x^T Px \leq 1\right) \Rightarrow (Ax+Bu)^T P (Ax+Bu) \leq 1. \tag{13}$$

This is illustrated on Figure 11.

Remark 7 If $P \succeq Q$, then the ellipsoid $\left\{x \mid x^T P x \leq 1\right\}$ is included in $\left\{x \mid x^T Q x \leq 1\right\}$. In particular, if $P \succeq rI$, the ellipsoid defined by P is included in the sphere of radius $\frac{1}{\sqrt{r}}$. Indeed, for all $x \in \left\{x \mid x^T P x \leq 1\right\}$, we have $x^T Q x \leq x^T P x \leq 1$, hence $x \in \left\{x \mid x^T Q x \leq 1\right\}$.

The rationale behind this heuristic[10] is to minimize the largest bound given by the invariant ellipsoid. Other heuristics are possible, for instance if the user is interested in a particular variable, the sphere can be replaced by an ellipsoid that is 'thinner' in this dimension [35, Section 5.3.4].

The previous condition (13) can be rewritten

$$\forall x, \forall u, \left(\left(\bigwedge_{i=0}^{p-1} \left(e_i^T u\right)^2 \leq 1 \right) \wedge x^T P x \leq 1 \right) \Rightarrow (Ax+Bu)^T P (Ax+Bu) \leq 1.$$

where e_i is the i-th vector of the canonical basis (i.e., with all coefficients equal to 0 except the i-th one which is 1). This amounts to: $\forall x, \forall u$,

$$\left(\bigwedge_{i=0}^{p-1} \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} 0 & 0 \\ 0 & E^{i,i} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \leq 1 \right) \wedge \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} P & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \leq 1 \Rightarrow \begin{bmatrix} x \\ u \end{bmatrix}^T \begin{bmatrix} A^T P A & A^T P B \\ B^T P A & B^T P B \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \leq 1$$

where $E^{i,j}$ is the matrix with 0 everywhere except the coefficient at line i, column j which is 1. Using a Lagrangian Relaxation (c.f., Example 4, page 8), a sufficient condition for this to hold is the existence of τ and $\lambda_0, \ldots, \lambda_{p-1}$ all non negative such that

$$\begin{bmatrix} -A^T P A & -A^T P B & 0 \\ -B^T P A & -B^T P B & 0 \\ 0 & 0 & 1 \end{bmatrix} - \tau \begin{bmatrix} -P & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \sum_{i=0}^{p-1} \lambda_i \begin{bmatrix} 0 & 0 & 0 \\ 0 & -E^{i,i} & 0 \\ 0 & 0 & 1 \end{bmatrix} \succeq 0 \tag{14}$$

This is not a LMI since τ and P are both variables which means it cannot be directly solved 'as is'. The idea will then be to fix the value of τ and solve the resulting LMI to obtain P. The first thing to determine is then the set of 'interesting' values for τ. The following lemmas will prove that there is a $\tau_{min} \in [0,1]$ such that all solutions satisfy $\tau \in [\tau_{min}, 1]$ and, conversely, there is a solution for all $\tau \in (\tau_{min}, 1]$.

Lemma 1 *For all solutions of (14),* $\tau \in [0,1]$.

Proof τ is non negative and according to the bottom right coefficient of (14), $1 - \tau - \sum \lambda_i \geq 0$ hence $\tau \leq 1$ since all λ_i are non negative.

Lemma 2 *If (14) has a solution for some* τ*, it has a solution for any* $\tau' \in (\tau, 1]$.

Proof Assume the non negative scalars $\tau, \lambda_0, \ldots, \lambda_{p-1}$ and the symmetric positive definite matrix P are solutions of (14) and $\tau' \in (\tau, 1]$. Then by multiplying the first diagonal block of (14) by $\frac{1-\tau'}{1-\tau} \geq 0$ we get

$$\begin{bmatrix} -A^T P' A & -A^T P' B \\ -B^T P' A & -B^T P' B \end{bmatrix} - \tau \begin{bmatrix} -P' & 0 \\ 0 & 0 \end{bmatrix} - \sum_{i=0}^{p-1} \lambda_i' \begin{bmatrix} 0 & 0 \\ 0 & -E^{i,i} \end{bmatrix} \succeq 0$$

[10] There is usually no best ellipsoidal invariant, so we have to resort on a heuristic.

with $P' := \frac{1-\tau'}{1-\tau}P$ and $\lambda_i' := \frac{1-\tau'}{1-\tau}\lambda_i$. Since $P \succeq 0$ and $\tau' \geq \tau \geq 0$ we have $\tau' P' \succeq \tau P'$, hence

$$\begin{bmatrix} -A^T P' A & -A^T P' B \\ -B^T P' A & -B^T P' B \end{bmatrix} - \tau' \begin{bmatrix} -P' & 0 \\ 0 & 0 \end{bmatrix} - \sum_{i=0}^{p-1} \lambda_i' \begin{bmatrix} 0 & 0 \\ 0 & -E^{i,i} \end{bmatrix} \succeq 0$$

Moreover

$$1 - \tau' - \sum \lambda_i' = 1 - \tau' - \frac{1-\tau'}{1-\tau} \sum \lambda_i = \frac{1-\tau'}{1-\tau} \left(1 - \tau - \sum \lambda_i\right) \geq 0$$

These last two inequalities being equivalent to (14) prove that it has a solution for τ'.

Lemma 3 *There is a $\tau_{min} \in [0,1]$ such that all solutions of (14) satisfy $\tau \in [\tau_{min}, 1]$ and, conversely, there is a solution for all $\tau \in (\tau_{min}, 1]$.*

Proof Defining τ_{min} as the least upper bound of the set of τ solution of (14) (or 1 if this set is empty), the result follows by the two previous lemmas.

The value of τ_{min} can then be approximated by a binary search on $\tau \in [0,1]$ in (14), which is a LMI for any fixed value of τ. A last lemma will enable a more efficient computation by performing the binary search in the, simpler but equivalent, inequality $A^T P A - \tau P \preceq 0$.

Lemma 4 *For $\tau \in [0,1]$, $\tau \neq \tau_{min}$, (14) has a solution if and only if $A^T P A - \tau P \preceq 0$ does.*

Proof If τ is solution of (14), then according to the upper left block of (14), $-A^T P A - \tau(-P) \succeq 0$, that is $A^T P A - \tau P \preceq 0$. Conversely, if $A^T P A - \tau P \preceq 0$ has a solution for some τ, then for all $\tau' \in (\tau, 1)$, we have $A^T P A - \tau' P \prec 0$. This implies the existence of a solution of (14) for τ' (the formal proof being a bit painful is omitted, intuitively the strict inequality leaves enough room to fit the additional bounded input Bu by scaling P by a small enough factor $\alpha \in (0,1)$ (hence a large enough ellipsoid defined by αP)). Thus, $\tau \geq \tau_{min}$ and if $\tau \neq \tau_{min}$, (14) has a solution for τ.

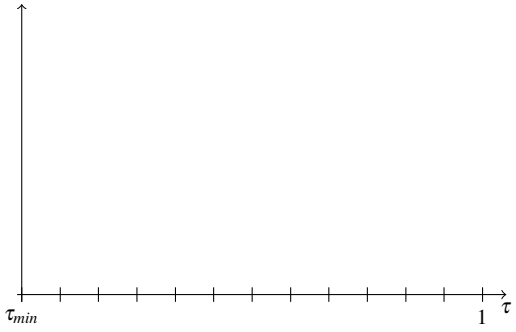
Example 12 With the following matrix A of the running example (c.f., Figure 1, page 3):

$$A := \begin{bmatrix} 0.9379 & -0.0381 & -0.0414 \\ -0.0404 & 0.968 & -0.0179 \\ 0.0142 & -0.0197 & 0.9823 \end{bmatrix},$$

looking by binary search for $\tau_{min} \in [0,1]$, the first tested value is $\tau = 0.5$, i.e., a solution to the following semi-definite program is looked for

$$\begin{aligned} &\text{minimize } 0 \\ &\text{subject to } A^T P A - 0.5P \preceq 0 \\ &\qquad\qquad P \succ 0 \\ &\qquad\qquad P^T = P. \end{aligned}$$

Since there is no solution, τ_{min} is now looked for in interval $[0.5, 1]$. $\tau = 0.75$ is tested, without more success, then $tau = 0.875$, $\tau = 0.9375$ and $\tau = 0.96875$ are still unsuccessful. $\tau = 0.984375$ is eventually successful meaning $\tau_{min} \in [0.96875, 0.984375]$. Then $\tau = 0.9765625$ fails and $\tau = 0.98046875$ succeeds. Stopping here leaves the overapproximated value $\tau_{min} = 0.98046875$.



τ	$\frac{1}{\sqrt{f(\tau)}}$	bounds
0.9805	0.78	0.50, 0.49, 0.71
0.9821	0.64	0.46, 0.42, 0.61
0.9837	0.56	0.44, 0.37, 0.56
0.9854	0.58	0.43, 0.35, 0.53
0.9870	0.61	0.43, 0.33, 0.52
0.9886	0.65	0.43, 0.32, 0.52
0.9902	0.70	0.43, 0.31, 0.54
0.9918	0.77	0.44, 0.31, 0.56
0.9935	0.85	0.46, 0.31, 0.59
0.9951	0.98	0.49, 0.32, 0.63
0.9967	1.19	0.53, 0.35, 0.70
0.9984	1.68	0.65, 0.42, 0.83

Fig. 12: Example 13: radius of the smallest sphere ($1/\sqrt{f(\tau)}$) containing the invariant with respect to τ. The table on the right displays the values actually computed and the bounds on the three program variables that would be obtained if the solution P for the given value of τ is kept as template (for comparison, the maximal reachable values are 0.38, 0.26, 0.48).

It now remains to choose the 'best' τ in this interval $[\tau_{min}, 1]$, that is the one leading to a solution P defining an ellipsoid included in the smallest sphere. We denote f the function mapping $\tau \in (\tau_{min}, 1]$ to the optimal value of the following semi-definite program:

$$\begin{array}{ll} \text{maximize} & r \\ \text{subject to} & (14), P \succeq rI, P^T = P, \bigwedge_{i=0}^{p-1} (\lambda_i > 0) \end{array}$$

Thus, this function can be evaluated for a given input τ simply by solving the above semi-definite program. f is then sampled for some equally spaced values in the interval $(\tau_{min}, 1]$ and the resulting matrix P linked to the maximum r is kept.

Example 13 With the following matrices A and B of the running example:

$$A := \begin{bmatrix} 0.9379 & -0.0381 & -0.0414 \\ -0.0404 & 0.968 & -0.0179 \\ 0.0142 & -0.0197 & 0.9823 \end{bmatrix} \qquad B := \begin{bmatrix} 0.0237 \\ 0.0143 \\ 0.0077 \end{bmatrix},$$

eight steps of binary search gave $\tau_{min} = 0.98046875$ in Example 12. Then the function f is computed for a dozen of values between τ_{min} and 1 as displayed on Figure 12 and the following matrix P is selected, corresponding to $\tau = 0.9837$:

$$P = \begin{bmatrix} 5.6309 & -3.9553 & -0.9322 \\ -3.9553 & 9.9229 & 1.5347 \\ -0.9322 & 1.5347 & 3.5184 \end{bmatrix}.$$

The quadratic template $x^T P x$ can then be used by policy iterations as seen in examples of Section 4.

From a control flow graph, matrices A and B are extracted by looking, along each edge, at the strongly connected components of the relation "variable x linearly depends on variable y". Templates are then generated as above for these matrices. This is a pure heuristic since existence of templates for such subsystems does not mean that they will allow to bound the

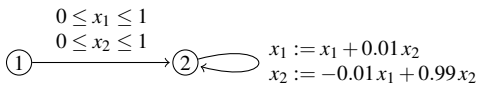


Fig. 13: Control flow graph for an harmonic oscillator [1,20,21].

whole system, not even that it is stable. However, this is a reasonable choice since actual systems are usually designed around a pure linear core.

Finally, as seen in the running example (c.f., Example 1, page 5), we add templates x^2 for each variable modified by the program. In the literature [1,20,21], templates x and $-x$ are often used. Since results are usually symmetrical in our context (i.e. the same bound b is obtained for both templates: $x \leq b$ and $-x \leq b$), templates x^2 yield the same result (i.e. $x^2 \leq b^2$) making use of two times less templates for policy iteration[11], hence saving on computation costs.

6 Initializing Policy Iterations

This section deals with policy iterations and initial values of the analyzed program, an important point in order to get a fully automatic implementation of policy iterations. This rather technical issue is not essential, though, for the understanding of the remainder of this article.

According to the seminal papers on policy iteration with quadratic templates [1,20,21], the control flow graph of Figure 13 should give the following system of equations for the templates $t_1 := -x_1$, $t_2 := x_1$, $t_3 := -x_2$, $t_4 := x_2$ and $t_5 := 2x_1^2 + 3x_2^2 + 2x_1x_2$:

$$\begin{cases} b_{2,1} = 0 \sqcup \max\{r(t_1) \mid \mathrm{be}(2)\} \\ b_{2,2} = 1 \sqcup \max\{r(t_2) \mid \mathrm{be}(2)\} \\ b_{2,3} = 0 \sqcup \max\{r(t_3) \mid \mathrm{be}(2)\} \\ b_{2,4} = 1 \sqcup \max\{r(t_4) \mid \mathrm{be}(2)\} \\ b_{2,5} = 7 \sqcup \max\{r(t_5) \mid \mathrm{be}(2)\} \end{cases}$$

where $\mathrm{be}(i)$ denotes $(t_1 \leq b_{i,1}) \wedge (t_2 \leq b_{i,2}) \wedge (t_3 \leq b_{i,3}) \wedge (t_4 \leq b_{i,4}) \wedge (t_5 \leq b_{i,5})$ and $r(t)$ is the template t in which variable x_1 is replaced by $x_1 + 0.01x_2$ and variable x_2 is replaced by $-0.01x_1 + 0.99x_2$. This translation is not straightforward. In particular, it is unclear how the numerical values 0, 1 and 7 should be computed[12]. It would be possible to compute them using a SDP solver but an easy and not much more expensive solution is to use the process described in Section 4.2 which automatically gives the system of equations

$$\begin{cases} b_{1,1} = +\infty \quad b_{1,2} = +\infty \quad b_{1,3} = +\infty \quad b_{1,4} = +\infty \quad b_{1,5} = +\infty \\ b_{2,1} = \max\{t_1 \mid 0 \leq x_1 \leq 1 \wedge 0 \leq x_2 \leq 1 \wedge \mathrm{be}(1)\} \sqcup \max\{r(t_1) \mid \mathrm{be}(2)\} \\ b_{2,2} = \max\{t_2 \mid 0 \leq x_1 \leq 1 \wedge 0 \leq x_2 \leq 1 \wedge \mathrm{be}(1)\} \sqcup \max\{r(t_2) \mid \mathrm{be}(2)\} \\ b_{2,3} = \max\{t_3 \mid 0 \leq x_1 \leq 1 \wedge 0 \leq x_2 \leq 1 \wedge \mathrm{be}(1)\} \sqcup \max\{r(t_3) \mid \mathrm{be}(2)\} \\ b_{2,4} = \max\{t_4 \mid 0 \leq x_1 \leq 1 \wedge 0 \leq x_2 \leq 1 \wedge \mathrm{be}(1)\} \sqcup \max\{r(t_4) \mid \mathrm{be}(2)\} \\ b_{2,5} = \max\{t_5 \mid 0 \leq x_1 \leq 1 \wedge 0 \leq x_2 \leq 1 \wedge \mathrm{be}(1)\} \sqcup \max\{r(t_5) \mid \mathrm{be}(2)\}. \end{cases}$$

This system of equations is essentially the same than the previous one, except that the computation of the values 0, 1 and 7 is now delegated to policy iteration itself.

[11] Moreover, x^2 being an homogeneous degree two polynomial is easier to express in semi-definite programs than linear constraints which would require an extra dimension to encode linear terms.

[12] Although they are clearly the maximal values of each template under constraint $0 \leq x_1 \leq 1 \wedge 0 \leq x_2 \leq 1$.

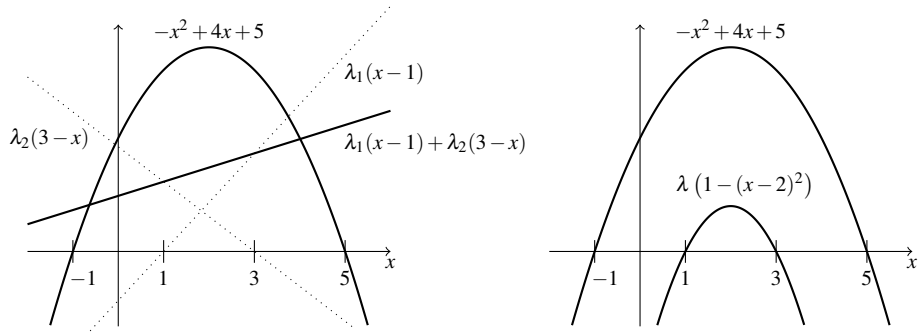


Fig. 14: Relaxation of interval constraints.

The above transformation introduced computation of optimal values under linear constraints ($0 \leq x_1 \leq 1$, for instance). Unfortunately, the relaxation offered in the policy iteration literature [1,20,21] does not enable to compute such values. The remainder of this section exemplifies the problem and offers an efficient workaround.

We have seen that implications are handled thanks to a Lagrangian relaxation (Theorem 1, page 8). This relaxation is not always exact, in particular with a quadratic objective f and two linear constraints g_1 and g_2.

Example 14 We want to apply a relaxation on $x \in [1,3] \Rightarrow -x^2+4x+5 \geq 0$, that is (2), page 8, with $f := x \mapsto -x^2+4x+5$, $g_1 := x \mapsto x-1$ and $g_2 := 3-x$. Formula (1), page 8, then boils down to: $\forall x, -x^2+(4-\lambda_1+\lambda_2)x+(5+\lambda_1-3\lambda_2) \geq 0$. Unfortunately, not any non negative $\lambda_1, \lambda_2 \in \mathbb{R}$ satisfy this[13]. This is depicted on left of Figure 14.

This case is commonly encountered in practice, for instance with initial values of a program living in some range (such as $0 \leq x_1 \leq 1$ on Figure 13) or with inputs bounded by an interval (c.f., $-1 \leq in \leq 1$ on Figure 9). Replacing the two linear constraints by an equivalent quadratic one constitutes an efficient workaround, i.e., any constraint $b \leq a^T x \leq c$ is translated into $\left(a^T x - \frac{b+c}{2}\right)^2 \leq \left(\frac{c-b}{2}\right)^2$.

Example 15 When constraints $x-1 \geq 0$ and $3-x \geq 0$ are replaced by the equivalent $1-(x-2)^2 \geq 0$, relaxation works just fine (with relaxation coefficient $\lambda = 2$ for instance). This is depicted on right of Figure 14.

7 Floating-Point Issues

Two fundamentally different issues arise with floating-point arithmetic:

The analysis itself is carried out with floating-point computations for the sake of efficiency, this usually works well in practice but might give erroneous results, hence the need for some a-posteriori validation, see Section 7.1 for further details;

The analyzed system uses floating-point arithmetic with rounding errors, making it behave differently from the way it would using real arithmetic, this is discussed in Section 7.2.

[13] For, denoting p the previous degree two polynomial, $\lim_{x\to\infty} p(x) = -\infty$, whatever the values of λ_1 and λ_2.

7.1 Floating-Point Arithmetic in the Analyzer

For the sake of efficiency, the semi-definite programming solvers we use perform all their computations on floating-point numbers and do not offer any strict soundness guarantee on their results. To address this issue, we adopt the following strategy:

- first perform policy iterations with unsound solvers, just padding the equations to hopefully get a correct result;
- then check the soundness of previous result.

Padding the Equations means for min-policies multiplying each temporary result b_i by $(1+\varepsilon)$ for some small ε. For max-policies, all equations $\max\{p \mid q \leq c\}$ are basically replaced by $\max\{(1+\varepsilon)p \mid q \leq (1+\varepsilon)c\}$. In practice, a value of 10^{-4} for ε appears to be a good choice. The induced loss of accuracy on the final result is considered acceptable since bounds finally computed by our analysis are usually found to be at least a few percent larger than the actual maximal values reachable by the program. In case the following check fails, each bound can be multiplied by $(1+\varepsilon')$ with $\varepsilon' > \varepsilon$ (for instance $\varepsilon' = 10^{-3}$ or $\varepsilon' = 10^{-2}$) and the check can be run again. This often enables to retrieve a correct result at the cost of a, relatively cheap, additional check.

Checking Soundness of the Result Policy iterations return a vector of values $(b_{i,j}) \in \overline{\mathbb{R}}^{np}$ (where n is the number of templates and p the number of vertices in the control flow graph) expected to be an overapproximation for a solution of the system of equations introduced in Section 4.2. Since this result was computed using floating-point arithmetic, we have to check it.

This amounts to checking that for each equation (i.e., for each vertex $v' \in [\![1,p]\!]$ and each template t_j), the following inequality holds

$$b_{v',j} \geq \bigsqcup_{v \in [\![1,p]\!]} \max \left\{ r(t_j) \;\middle|\; e \leq c \wedge \bigwedge_{j'} (t_{j'} \leq b_{v,j'}) \right\} \tag{15}$$

where $e \leq c$ and r are respectively the constraint[14] and the assignments associated to the edge between v and v'. This can be rewritten

$$b_{v',j} \geq \bigsqcup_{v \in [\![1,p]\!]} \max \left\{ \begin{bmatrix} x \\ 1 \end{bmatrix}^T R_r^T P_{t_j} R_r \begin{bmatrix} x \\ 1 \end{bmatrix} \;\middle|\; \begin{array}{l} \begin{bmatrix} x \\ 1 \end{bmatrix}^T P_e \begin{bmatrix} x \\ 1 \end{bmatrix} \leq c \wedge \\ \bigwedge\limits_{j' \in [\![1,n]\!]} \begin{bmatrix} x \\ 1 \end{bmatrix}^T P_{t_{j'}} \begin{bmatrix} x \\ 1 \end{bmatrix} \leq b_{v,j'} \end{array} \right\} \tag{16}$$

by denoting R_r the matrix implementing the linear assignments r (i.e., for all x, $R_r \begin{bmatrix} x^T, 1 \end{bmatrix}^T = \begin{bmatrix} x'^T, 1 \end{bmatrix}^T$ where x' is the result of assignments r on x) and P_t the one expressing the quadratic template t (i.e., unfolding $\begin{bmatrix} x^T, 1 \end{bmatrix} P_t \begin{bmatrix} x^T, 1 \end{bmatrix}^T$ gives t). Thus, we have to check that for all $v, v' \in [\![1,p]\!]$ and all t_j:

$$\forall x, \left(\begin{bmatrix} x \\ 1 \end{bmatrix}^T P_e \begin{bmatrix} x \\ 1 \end{bmatrix} \leq c \wedge \bigwedge_{j' \in [\![1,n]\!]} \begin{bmatrix} x \\ 1 \end{bmatrix}^T P_{t_{j'}} \begin{bmatrix} x \\ 1 \end{bmatrix} \leq b_{v,j'} \right) \Rightarrow \begin{bmatrix} x \\ 1 \end{bmatrix}^T R_r^T P_{t_j} R_r \begin{bmatrix} x \\ 1 \end{bmatrix} \leq b_{v',j}.$$

[14] Only one constraint here for ease of exposition. Everything works the same with multiple constraints.

We can restrict ourselves to the case where all c and b lie in $\mathbb{R}$ since constraints of the form $. \leq +\infty$ can just be ignored and constraints $. \leq -\infty$ make the implication trivially true.

According to Theorem 1, it is enough to prove the existence of τ and $\lambda_{j'}$ all non negative such that

$$\left(\begin{bmatrix}0 & 0\\ 0 & b_{v',j}\end{bmatrix} - R_r^T P_{t_j} R_r\right) - \tau\left(\begin{bmatrix}0 & 0\\ 0 & c\end{bmatrix} - P_e\right) - \sum_{j'\in[\![1,n]\!]} \lambda_{j'}\left(\begin{bmatrix}0 & 0\\ 0 & b_{v,j'}\end{bmatrix} - P_{t_{j'}}\right) \succeq 0. \tag{17}$$

Such τ and $\lambda_{j'}$ can be computed (as floating-point values) using a SDP solver. Then, the matrix in previous inequality can be computed exactly using rational numbers (or it could be approximated using cheaper floating-point interval arithmetic) and it remains to check its positive definiteness.

This can be done by carefully bounding the rounding error on a floating point Cholesky decomposition [39][15] as detailed in the following paragraph. Proof of positive definiteness of an $n \times n$ matrix can then be achieved with $\mathrm{O}(n^3)$ floating point operations, which in practice induces only a very small overhead to the whole analysis.

Positive Definiteness Check According to Section 3.4, a Cholesky decomposition can be used to prove that a matrix M is positive definite. However, performing it with floating-point arithmetic, it could run to completion while $M \not\succeq 0$, due to rounding errors. But rounding errors remain bounded, so that there exists an $e \in \mathbb{R}$ such that, if the floating-point Cholesky decomposition of M succeeds, then $M + e\,Id \succeq 0$. The successful floating-point Cholesky decomposition of $M - e\,Id$ is then a sufficient condition for $M \succeq 0$. Moreover, such an e can be easily computed from simple characteristics of M and the floating-point arithmetic format used, as stated by the following theorem.

Remark 8 (Alternative to the floating-point Cholesky decomposition) When coefficients of M are rational, a slightly modified Cholesky decomposition can compute matrices M' and D with rational coefficients such that $M = M'^T D M'$ and D is diagonal with non negative coefficients. This would be much simpler, hence easier to trust, than the following non trivial theorem. However, rational arithmetic is more expensive than floating-point arithmetic.[16]

Theorem 4 ([39, Corollary 2.7]) *Given* `eps` *the precision of the floating point format* $\mathbb{F}$ *used,* `eta` *its precision in case of underflows and* $M \in \mathbb{R}^{n\times n}$*, if* $4(n+1)\texttt{eps} < 1$*, if* $M^T = M$*, if there exist* $m_d, \varepsilon \in \mathbb{R}$ *and* $\widetilde{M} \in \mathbb{F}^{n,n}$ *such that for all* i*,* $0 \leq M_{i,i} \leq m_d$*, for all* $i \neq j$*,* $\left|\widetilde{M}_{i,j} - M_{i,j}\right| \leq \varepsilon$ *and for all* i*,* $\widetilde{M}_{i,i} \leq M_{i,i} - \frac{(n+1)\texttt{eps}}{1-2(n+1)\texttt{eps}}\mathrm{tr}(M) - 4n\left(2(n+1)+m_d\right)\texttt{eta} - n\varepsilon$*, if the floating-point Cholesky decomposition (c.f., Section 3.4) of* $\widetilde{M}$ *succeeds then* $M \succ 0$*.*

`eps` and `eta` are very small constants defined by the floating-point format in use. For instance, $\texttt{eps} = 2^{-53} (\simeq 10^{-16})$ and $\texttt{eta} = 2^{-1075} (\simeq 10^{-323})$ for the IEE754 binary64 format [29][17]. Pen and paper proofs of this kind of results being particularly tedious hence error prone, we mechanically checked it with a proof assistant (Coq [8]). Our development (4.3 kloc of Coq) is available at `http://cavale.enseeiht.fr/practicalpolicy2014/` and based on the Flocq library [3] for the formal definition of floating-point arithmetic.

[15] Thanks to Timothy WANG for pointing this to us.

[16] Although, in our case, this positive definiteness check only accounts for a very small part of the total analysis time. Thus, the eventual overhead would remain limited.

[17] Usual implementation of type `double` in C.

Remark 9 (Matrices interval) The matrix M has coefficients in $\mathbb{R}$ whereas $\widetilde{M}$ has coefficients in $\mathbb{F}$, since we actually compute its Cholesky decomposition, which prevents $M = \widetilde{M}$. That's why Theorem 4 handles intervals of matrices $\left\{ M \;\middle|\; M^T = M \wedge \forall i,j, \left|\widetilde{M} - M\right|_{i,j} \leq \varepsilon \right\}$. This is particularly convenient, whether (17) is computed with rationals or approximated with interval arithmetic.

Remark 10 Theorem 4 differs slightly from [39, Corollary 2.7] for two reasons. We only use real and not complex numbers and minor issues (all pertaining with denormalized numbers) had to be fixed when performing the Coq proof.

Remark 11 (Positive semi-*definiteness)* The criterion given in Theorem 4 is not complete. In particular, it is only able to prove positive definiteness ($M \succ 0$) and not positive *semi*-definiteness ($M \succeq 0$) as in (17). This is not an actual issue thanks to the padding previously performed.

Remark 12 Theorem 4 should be valid for any reasonable implementation of the Cholesky decomposition. However, the Coq proof has been performed for the algorithm shown in Figure 8, page 9 with sums performed from left to right. Furthermore, the proof considers the binary64 format with normal and denormalized numbers, ignoring special values `NaN` and $\pm\infty$ (although handling them should be relatively easy through a corollary of the current theorem). Finally, only the algorithm, not its implementation, is proved. However, this implementation is particularly straightforward compared to the non trivial result proved.

7.2 Floating-Point Arithmetic in the Analyzed Program

Until now, the analyzed program was considered as if it were executed with arithmetic operations in the real field $\mathbb{R}$. Actual implementations will usually use floating-point arithmetic instead. This induces rounding errors which have to be taken into account in our analysis.

Taking floating-point arithmetic into account, (15) becomes

$$b_{v',j} \geq \bigsqcup_{v \in [\![1,p]\!]} \max \left\{ \mathrm{fl}(r)\,(t_j) \;\middle|\; \mathrm{fl}(e) \leq \mathrm{fl}(c) \wedge \bigwedge_{j'} (t_{j'} \leq b_{v,j'}) \right\} \tag{18}$$

since guards $e \leq c$ and assignments r are now performed in $\mathbb{F}$. All the remaining of (15) is kept unchanged since it only corresponds to mathematical expressions (in $\mathbb{R}$) and not to parts of the analyzed program (in $\mathbb{F}$). Our goal is to derive slightly modified bounds c' and $b'_{v',j}$ such that

$$b'_{v',j} \geq \bigsqcup_{v \in [\![1,p]\!]} \max \left\{ r(t_j) \;\middle|\; e \leq c' \wedge \bigwedge_{j'} (t_{j'} \leq b_{v,j'}) \right\}$$

is a sufficient condition for the previous (18) to hold. These inequalities can then be checked just as (15) in the previous Section 7.1.

Definition 1 $\mathbb{F} \subset \mathbb{R}$ denotes the set of floating point values and $\mathrm{fl}(e) \in \mathbb{F}$ represents the floating point evaluation of expression e with any rounding mode and any order of evaluation[18].

[18] Order of evaluation matters since floating point addition is not associative.

Example 16 The value $\mathrm{fl}(1+2+3)$ can be either $\mathrm{round}(1+\mathrm{round}(2+3))$ or $\mathrm{round}(\mathrm{round}(1+2)+3)$ with round any valid rounding mode (toward $+\infty$ or to nearest for instance).

We will first see how to handle the guards $\mathrm{fl}(e) \leq \mathrm{fl}(c)$ then the assignments $\mathrm{fl}(r)(t_j)$. That is, we will first derive c' such that $\mathrm{fl}(e) \leq \mathrm{fl}(c)$ implies $e \leq c'$ which guarantees that $\max\{\mathrm{fl}(r)(t_j) \mid e \leq c' \wedge \bigwedge_{j'}(t_{j'} \leq b_{v,j'})\} \geq \max\{\mathrm{fl}(r)(t_j) \mid \mathrm{fl}(e) \leq \mathrm{fl}(c) \wedge \bigwedge_{j'}(t_{j'} \leq b_{v,j'})\}$. Then $b'_{v',j}$ will be derived such that $r(t_j) \leq b'_{v',j}$ implies $\mathrm{fl}(r)(t_j) \leq b_{v',j}$.

Guards For all guards $e \leq c$, the actually implemented guard is $\mathrm{fl}(e) \leq \mathrm{fl}(c)$ and there can be values of program variables such that the later holds but not the former. Our goal is to define a $c' \geq c$ such that $\mathrm{fl}(e) \leq \mathrm{fl}(c)$ implies $e \leq c'$. To that end, we will bound $\mathrm{fl}(e) - e$, the rounding error on e, and choose $c' \geq \mathrm{fl}(c) - (\mathrm{fl}(e) - e)$. We will only consider the case of linear guards $a^T x \leq c$ with $a \in \mathbb{R}^n$, $c \in \mathbb{R}$, $x \in \mathbb{F}^n$. That is, we have to bound the rounding error $\mathrm{fl}(a^T x) - a^T x$ on the dotproduct $a^T x$.

Property 1 ([40]) `eps` is the precision of the floating point format $\mathbb{F}$ and `eta` its precision in case of underflows. In particular, we have for all $x, y \in \mathbb{F}$

$$\exists \delta \in \mathbb{R}, |\delta| \leq \mathtt{eps} \wedge \mathrm{fl}(x+y) = (1+\delta)(x+y)$$

and

$$\exists \delta, \eta \in \mathbb{R}, |\delta| \leq \mathtt{eps} \wedge |\eta| \leq \mathtt{eta} \wedge \mathrm{fl}(x \times y) = (1+\delta)(x \times y) + \eta.$$

Remark 13 `eps` is the relative error pertaining with regular, called normal, numbers whereas `eta` is the absolute error pertaining with very small number, close to 0, called denormalized numbers. The addition and subtraction are exact for denormalized numbers.

Property 2 ([28, Lemma 3.3]) The values $\gamma_n := \frac{n\mathtt{eps}}{1-n\mathtt{eps}}$ with $n \in \mathbb{N}$ have the following property, particularly useful to accumulate relative error bounds: for all $n \in \mathbb{N}$, assuming $(n+1)\,\mathtt{eps} < 1$

$$\forall \theta_n, \forall \delta, (|\theta_n| \leq \gamma_n \wedge |\delta| \leq \mathtt{eps}) \Rightarrow \exists \theta_{n+1}, |\theta_{n+1}| \leq \gamma_{n+1} \wedge (1+\theta_n)(1+\delta) = 1+\theta_{n+1}.$$

The following corollary will also be used: for all $n \in \mathbb{N}$, if $n\,\mathtt{eps} < 1$ then

$$\forall \delta_1, \ldots, \delta_n, (\forall i, |\delta_i| \leq \mathtt{eps}) \Rightarrow \exists \theta_n, |\theta_n| \leq \gamma_n \wedge \prod_{i=1}^{n}(1+\delta_i) = 1+\theta_n$$

as well as the monotonicity of γ (for all n, n', if $n \leq n'$ and $n'\,\mathtt{eps} < 1$ then $\gamma_n \leq \gamma_{n'}$).

Theorem 5 *Assuming* $2(n+1)\mathtt{eps} < 1$*, we have for all* $a \in \mathbb{R}^n$ *and* $x \in \mathbb{F}^n$

$$\left| \mathrm{fl}\left(\sum_{i=1}^{n} a_i x_i\right) - \sum_{i=1}^{n} a_i x_i \right| \leq \gamma_{n+1} \sum_{i=1}^{n} |a_i x_i| + 2\left(n + \sum_{i=1}^{n} |x_i|\right) \mathtt{eta}.$$

To prove the previous theorem, we first need the following lemma.

Lemma 5 *Assuming* $(n-1)\mathtt{eps} < 1$*, for all* $x \in \mathbb{F}^n$*, there exists* $\theta \in \mathbb{R}^n$ *such that for all i,* $|\theta_i| \leq \gamma_{n-1}$ *and*

$$\mathrm{fl}\left(\sum_{i=1}^{n} x_i\right) = \sum_{i=1}^{n}(1+\theta_i)x_i.$$

Remark 14 Those are fairly classic notations and results [28,40].

Proof (Lemma 5) According to Definition 1, if the sum is computed from left to right, there exists $\delta_{n-1} \in \mathbb{R}$ such that $|\delta_{n-1}| \leq \texttt{eps}$ and

$$\mathrm{fl}\left(\sum_{i=1}^{n} x_i\right) = \mathrm{fl}\left(\mathrm{fl}\left(\sum_{i=1}^{n-1} x_i\right) + x_n\right) = (1+\delta_{n-1})\left(\mathrm{fl}\left(\sum_{i=1}^{n-1} x_i\right) + x_n\right).$$

Then, by an immediate induction, there exists $\delta \in \mathbb{R}^{n-1}$ such that for all i, $|\delta_i| \leq \texttt{eps}$ and

$$\mathrm{fl}\left(\sum_{i=1}^{n} x_i\right) = \left(\prod_{j=1}^{n-1}(1+\delta_j)\right)x_1 + \sum_{i=2}^{n}\left(\left(\prod_{j=i-1}^{n-1}(1+\delta_j)\right)x_i\right).$$

According to Property 2, for all i, there exists $\theta_i \in \mathbb{R}$ such that $|\theta_i| \leq \gamma_{n-i+1}$ and $\prod_{j=i-1}^{n-1}(1+\delta_j) = 1+\theta_i$, hence the result[19].

Proof (Theorem 5) According to Lemma 5, there exists $\theta \in \mathbb{R}^n$ such that for all i, $|\theta_i| \leq \gamma_{n-1}$ and

$$\mathrm{fl}\left(\sum_{i=1}^{n} a_i x_i\right) = \sum_{i=1}^{n}(1+\theta_i)\mathrm{fl}(a_i x_i).$$

Then, according to Definition 1, there exist $\delta, \eta \in \mathbb{R}^n$ such that for all i, $|\delta_i| \leq \texttt{eps}$, $|\eta_i| \leq \texttt{eta}$ and

$$\mathrm{fl}\left(\sum_{i=1}^{n} a_i x_i\right) = \sum_{i=1}^{n}(1+\theta_i)\left((1+\delta_i)\mathrm{fl}(a_i)\,\mathrm{fl}(x_i) + \eta_i\right).$$

Since $x_i \in \mathbb{F}$, $\mathrm{fl}(x_i) = x_i$ but $a_i \in \mathbb{R}$ hence $\mathrm{fl}(a_i) = (1+\delta_i')a_i + \eta_i'$ for some $\delta_i', \eta_i' \in \mathbb{R}$, $|\delta_i'| \leq \texttt{eps}$ and $|\eta_i'| \leq \texttt{eta}$. Hence

$$\mathrm{fl}\left(\sum_{i=1}^{n} a_i x_i\right) = \sum_{i=1}^{n}(1+\theta_i)(1+\delta_i)(1+\delta_i')a_i x_i + (1+\theta_i)(1+\delta_i)\eta_i' x_i + (1+\theta_i)\eta_i.$$

According to Property 2, for all i, there exists $\theta_i' \in \mathbb{R}$ such that $|\theta_i'| \leq \gamma_{n+1}$ and $(1+\theta_i)(1+\delta_i)(1+\delta_i') = 1+\theta_i'$. Similarly, there exists $\theta_i'' \in \mathbb{R}$ such that $|\theta_i''| \leq \gamma_n$ and $(1+\theta_i)(1+\delta_i) = (1+\theta_i'')$, which gives

$$\mathrm{fl}\left(\sum_{i=1}^{n} a_i x_i\right) = \sum_{i=1}^{n}\left((1+\theta_i')a_i x_i + (1+\theta_i'')\eta_i' x_i + (1+\theta_i)\eta_i\right).$$

Then

$$\mathrm{fl}\left(\sum_{i=1}^{n} a_i x_i\right) - \sum_{i=1}^{n} a_i x_i = \sum_{i=1}^{n}\theta_i' a_i x_i + \sum_{i=1}^{n}\left((1+\theta_i'')\eta_i' x_i + (1+\theta_i)\eta_i\right).$$

We can notice that

$$\left|\sum_{i=1}^{n}\theta_i' a_i x_i\right| \leq \sum_{i=1}^{n}\left|\theta_i'\right||a_i x_i| \leq \sum_{i=1}^{n}\gamma_{n+1}|a_i x_i| = \gamma_{n+1}\sum_{i=1}^{n}|a_i x_i|$$

[19] A similar proof can be performed if the sum is not computed in this left-right order.

and similarly

$$\left|\sum_{i=1}^{n}\left((1+\theta_i'')\eta_i' x_i+(1+\theta_i)\eta_i\right)\right| \leq 2\left(n+\sum_{i=1}^{n}|x_i|\right)\texttt{eta}$$

since $|\theta_i''| \leq \gamma_n \leq 1$ and $|\theta_i| \leq \gamma_{n-1} \leq 1$, which finally gives the result.

This theorem gives the desired property, for all $c' \geq \mathrm{fl}(c)+\gamma_{n+1}\,|a|^T|x|+2(n+\|x\|_1)\texttt{eta}$, the inequality $\mathrm{fl}\left(a^T x\right) \leq \mathrm{fl}(c)$ implies $a^T x \leq c'$. Since we used templates x_i^2 for each variable x_i of the analyzed program, we actually have a bound on $\|x\|_1$ and it is easy to compute such an appropriate c' (for instance with floating-point arithmetic and rounding toward $+\infty$).

Assignments Things are a bit more involved than in the case of guards. We are now looking for a $b'_{v',j} \leq b_{v',j}$ such that $r(t_j) \leq b'_{v',j}$ implies $\mathrm{fl}(r)\,(t_j) \leq b_{v',j}$, that is $[x\ 1]R_r^T P_{t_j} R_r [x\ 1]^T \leq b'_{v',j}$ implies $\mathrm{fl}\left([x\ 1]R_r^T\right) P_{t_j} \mathrm{fl}\left(R_r[x\ 1]^T\right) \leq b_{v',j}$ (where R_r and P_t are matrix encoding of the assignment r and the template t as already used in (16)). The next theorem will guarantee us that this property holds for any $b'_{v',j} \leq \left(\sqrt{b_{v',j}}-\sqrt{s}\|e\|_2\right)^2$ with $s \in \mathbb{R}$ such that $P_{t_j} \preceq sI$ and $e_i := \gamma_{n+2}\left|R_{ri,.}\right|\left[|x|^T\ \ 1\right]^T + 2\,(n+2+\|x\|_1)\,\texttt{eta}$. Again, such a $b'_{v,j}$ is easy to compute (with a SDP solver for s and rounding toward $+\infty$ for e).

Theorem 6 *Given matrices $P,R \in \mathbb{R}^{(n+1)\times(n+1)}$, with $2(n+2)\texttt{eps} < 1$, and scalars $s,b \in \mathbb{R}$ such that P is symmetric positive semi-definite (i.e., $P^T = P$ and $P \succeq 0$) and $P \preceq sI$, for any $x \in \mathbb{F}^n$, denoting $e_i := \gamma_{n+2}\,|R_{i,.}|\,[\,|x|^T\ \ 1]^T + 2\,(n+2+\|x\|_1)\,\texttt{eta}$, if $s\|e\|_2^2 \leq b$ and*

$$\begin{bmatrix}x\\1\end{bmatrix}^T R^T P R \begin{bmatrix}x\\1\end{bmatrix} \leq \left(\sqrt{b}-\sqrt{s}\|e\|_2\right)^2$$

then

$$\mathrm{fl}\left(\begin{bmatrix}x\\1\end{bmatrix}^T R^T\right) P\,\mathrm{fl}\left(R\begin{bmatrix}x\\1\end{bmatrix}\right) \leq b$$

where $R_{i,.}$ denotes the i-th line of the matrix R.

Proof Denoting $y := R[x\ 1]^T$ we have, thanks to Theorem 5, $|\mathrm{fl}(y_i)-y_i| \leq e_i$, hence $\mathrm{fl}(y)_i = y_i+\delta_i e_i$ for some $\delta_i \in \mathbb{R}$ such that $|\delta_i| \leq 1$. Thus, denoting D the diagonal matrix such that for all i, $D_{i,i} = \delta_i$, we have

$$\mathrm{fl}(y)^T P\,\mathrm{fl}(y) = (y+De)^T P\,(y+De) = y^T P y + e^T D^T P D e + 2y^T P D e.$$

Then, by the Cauchy-Schwarz inequality

$$\mathrm{fl}(y)^T P\,\mathrm{fl}(y) \leq y^T P y + e^T D^T P D e + 2\sqrt{y^T P y}\sqrt{e^T D^T P D e}$$

and since $P \preceq sI$

$$\mathrm{fl}(y)^T P\,\mathrm{fl}(y) \leq y^T P y + s\|e\|_2^2 + 2\sqrt{y^T P y}\sqrt{s}\|e\|_2.$$

Hence the result, since $y^T P y \leq \left(\sqrt{b}-\sqrt{s}\|e\|_2\right)^2$.

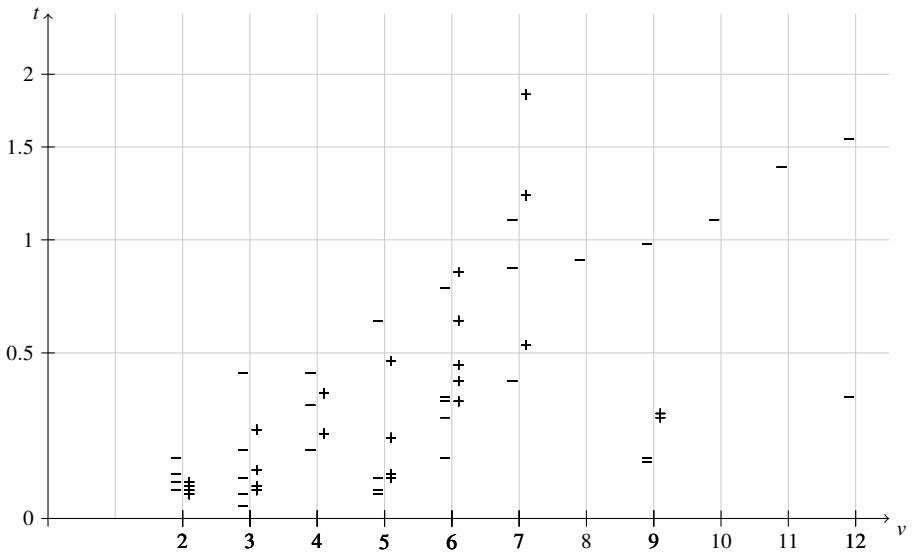


Fig. 15: Time (t in seconds) spent performing min ($-$ signs) and max ($+$ signs) policy iterations depending on the number v of variables in the analyzed program. Less $+$ than $-$ in a column indicates a failure of max-policies on a benchmark. All computations were performed on an Intel Core2 @ 2.66GHz.

Finally, if the following holds

$$b'_{v',j} \geq \bigsqcup_{v \in [\![1,p]\!]} \max \left\{ \begin{bmatrix} x \\ 1 \end{bmatrix}^T R_r^T P_{t_j} R_r \begin{bmatrix} x \\ 1 \end{bmatrix} \;\middle|\; \begin{bmatrix} x \\ 1 \end{bmatrix}^T P_e \begin{bmatrix} x \\ 1 \end{bmatrix} \leq c' \wedge \bigwedge_{j' \in [\![1,n]\!]} \begin{bmatrix} x \\ 1 \end{bmatrix}^T P_{t_{j'}} \begin{bmatrix} x \\ 1 \end{bmatrix} \leq b_{v,j'} \right\} \tag{19}$$

then (18) holds and we can now proceed as in Section 7.1 by just replacing (16) with the above (19). The check should still succeed since the differences between the values in (16) and (19) are orders of magnitude smaller than the accuracy the $b_{v,j}$ were initially computed with using SDP solvers[20].

Again, Theorems 5 and 6 are mechanically checked with a proof assistant (3.8 kloc of Coq, among which 2.8 kloc are common with the aforementioned positive-definiteness check proof).

Such use of abstract domains in the real field to soundly analyze floating-point computations is not new [32] and some techniques even allow to finely track rounding errors and their origin in the analyzed program [24].

8 Experimental Results

All the elements presented in this paper have been implemented as a new abstract domain in our static analyzer. Experiments were conducted on a set of stable linear systems. These systems were extracted from the literature [1,16,38,42]. The analyzer is released under GPL and available with all examples at `http://cavale.enseeiht.fr/practicalpolicy2014/`.

Comparing Min- and Max-Policies As seen in Section 4, two methods exist to compute invariants by policy iterations, namely min- and max-policies. Figure 15 compares analysis times with min and max-policy iterations. In both cases, the number of iterations always remained reasonable. For min-policies, the number of iterations performed lies between 3 and 7 when the stopping criterion is a relative progress below 10^{-4} between two consecutive iterates. For max-policies, the number of iterations was between 4 and 7. As shown on Figure 15, computation times for min and max-policies are comparable but the actual difference appear on the largest benchmarks for which max-policies where unable to produce sound results while min-policies did[21]. Finally, it can be noticed that, when both methods work, results obtained with min and max-policies are the same. However, due to numerical issues, min-policies often yield slightly more precise results. For all these reasons, min policies were made the default in our tool.

Benchmarks Figure 15 only gave times for policy iterations. Total analysis times also include building the control flow graph and the equation system, computing appropriate templates and eventually checking the soundness of the result. Time needed for control flow graph construction and soundness checking is very small compared to the time spent in policy iterations, whereas computing templates takes the same amount of magnitude in time than min-policies iteration. All this is detailed in Table 2.

Finally, Table 3 details the bounds obtained for each benchmark and compares them with the maximum reachable values of the programs. The padding $\varepsilon = 10^{-4}$ was enough in most cases. Only 4 of the 22 successful cases required a larger padding ($\varepsilon' = 10^{-}2$ for Ex. 3, Ex. 3 with reset and Ex. 6 with saturation and $\varepsilon' = 10^{-1}$ for Ex. 3 with saturation).

9 Conclusion

In this paper we attempted to provided a complete, yet practical, use of policy iterations to perform static analysis of programs. Policy iteration is shown to be a strong candidate to support the computation of precise post-fixpoints when over-approximating the collecting semantics of a program.

We presented the background of policy iterations and the rationals of its use, either using min-policy decreasing iterations – a kind of smart narrowing, starting from a post-fixpoint – or max-policies, performing increasing iterations. In both cases, bounds over template domains are obtained relying on numerical solvers, at each step of the computation.

[20] The relative difference between the $b_{v',j}$ and the $b'_{v',j}$ or the c and c' never exceeded 10^{-10} in our experiments (to be compared to the 10^{-4} padding previously applied).

[21] This is explained by the fact that max-policies have to solve larger SDP problems, incurring more numerical difficulties [21, Conclusion] (c.f., Remarks 4, page 14 and 6, page 16).

	n	Total (s)	Templates (s)	Iterations (s)	Check (s)
Ex. 1 From [16, slides]	3	0.12	0.05	0.03	0.01
	3	0.16	0.05	0.06	0.02
	4	0.50	0.15	0.22	$\perp$ (0.01)
Ex. 2 From [16, slides]	5	0.33	0.16	0.06	0.02
	5	0.44	0.15	0.10	0.04
	6	0.77	0.15	0.28	0.12
Ex. 3 Discretized lead-lag controller	3	0.20	0.07	0.10	0.02
	3	0.32	0.07	0.18	0.03
	4	0.68	0.07	0.43	0.08
Ex. 4 Linear quadratic gaussian regulator	4	0.47	0.16	0.19	0.03
	4	0.67	0.16	0.32	0.06
	5	1.13	0.20	0.64	0.13
Ex. 5 Observer based controller for a coupled mass system	6	0.96	0.46	0.16	0.06
	6	1.25	0.47	0.33	0.11
	7	2.28	0.45	0.85	0.26
Ex. 6 Butterworth low-pass filter	6	1.18	0.47	0.35	0.08
	6	1.76	0.45	0.77	0.15
	7	2.67	0.45	1.10	0.26
Ex. 7 Dampened oscillator from [1]	2	0.14	0.01	0.09	0.01
	2	0.23	0.01	0.16	0.02
	3	0.36	0.01	0.20	$\perp$ (0.01)
Ex. 8 Harmonic oscillator from [1]	2	0.11	0.01	0.07	0.01
	2	0.19	0.01	0.12	0.03
	3	0.65	0.01	0.43	0.10

Table 2: Result of the experiments: quadratic invariants inference. For each of the eight examples, the first line is for the bare linear system, the second for the same system with an added reset and the third with a saturation. Column n gives the number of program variables considered for policy iteration while column 'Total' gives the time spent for the whole analysis. The remaining columns detail the computation time: 'Templates' corresponds to the quadratic template computation, 'Iterations' to the actual policy iterations and 'Check' to the soundness checking. $\perp$ indicates failure of the soundness checking (in both cases because the template generation heuristic failed to generate an appropriate template).

We supported the use of policy iteration, as a way to replace the widening operator when it is ineffective, by addressing key points required by the technique and often left unaddressed by former works:

- the automatic computation of templates;
- handling the floating point semantics of the analyzed program;
- guaranteeing the soundness of the computation despite the possible errors of the numerical solvers used.

We believe our contribution could support a wider use of policy iterations within the abstract interpretation framework and more generally the static analysis of programs. The setting in which the current work is performed is specific: the analysis of control software, focusing on quadratic templates and using SDP solvers as optimization solvers; but it is a first step towards more extensions and a wider applicability:

- more complex templates, e.g., disjunction of quadratic forms or polynomials (for instance thanks to sum-of-squares (SOS) relaxations);
- wider class of programs analyzable precisely, e.g., complex discrete versions of controlled systems including the system (also known as plant) behavior.

	$\max\|\lambda_i\|$	Bounds	Reachable
Ex. 1	0.837	15.98, 15.98	14.84, 14.84
		15.98, 15.98	14.84, 14.84
		$+\infty, +\infty$	12.31, 12.31
Ex. 2	0.837	1.65, 1.65, 1.00, 1.00	1.42, 1.42, 1.00, 1.00
		1.65, 1.65, 1.00, 1.00	1.42, 1.42, 1.00, 1.00
		2.20, 0.50, 1.00, 1.00	1.04, 0.50, 1.00, 1.00
Ex. 3	0.999	4.03, 20.41	3.97, 20.00
		4.03, 20.41	3.97, 20.00
		4.14, 21.41	2.04, 1.68
Ex. 4	0.989	0.43, 0.35, 0.54	0.38, 0.26, 0.48
		1.03, 1.01, 1.47	1.00, 1.00, 1.00
		0.45, 0.37, 0.56	0.19, 0.11, 0.17
Ex. 5	0.840	4.60, 4.74, 4.34, 4.38	2.79, 2.73, 3.50, 3.30
		4.60, 4.74, 4.34, 4.38	2.79, 2.73, 3.50, 3.30
		3.58, 7.04, 5.54, 6.17	1.28, 1.69, 3.31, 2.87
Ex. 6	0.804	1.42, 1.10, 1.75, 1.82, 2.57	1.42, 0.91, 1.44, 1.52, 2.14
		1.42, 1.76, 2.63, 3.14, 4.45	1.42, 0.91, 1.44, 1.52, 2.14
		1.03, 1.37, 1.99, 2.95, 4.02	1.03, 0.65, 0.77, 0.88, 1.16
Ex. 7	0.995	1.74, 1.74	1.29, 1.00
		1.74, 1.74	1.29, 1.00
		$+\infty, +\infty$	1.00, 1.00
Ex. 8	0.955	1.27, 1.27	1.10, 1.00
		1.27, 1.27	1.10, 1.00
		1.00, 1.01	1.00, 0.99

Table 3: Result of the experiments: quadratic invariants inference. The examples are the same than in Table 2. Column 'Bounds' gives the bounds on absolute values of each variables inferred and proved by the tool whereas column 'Reachable' gives underapproximations of the maximum reachable values (obtained by random simulation) for comparison purpose. $\max|\lambda_i|$ is the maximum of modules of eigenvalues of the linear application considered, which gives an idea of 'how contractive' the linear application is.

Another important aspect of the approach is the capability to express and analyze more complex behaviors than just the boundedness of the considered system. The synthesized templates, even if constrained by a bound, could express high level behavior of the program. In our setting of controllers, these templates can be used to encode stability, robustness or performance properties, leading to a broader impact of static analysis when applied to critical software and systems.

Acknowledgements We would like to deeply thank the anonymous reviewers for their highly relevant comments to improve this paper. This work has been partially supported by the ANR-INSE-2012-007 grant CAFEIN and the Aerospace Valley competitivity cluster.

References

1. Assalé Adjé, Stéphane Gaubert, and Éric Goubault. Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. In *ESOP*, pages 23–42, 2010.
2. Fernando Alegre, Éric Féron, and Santosh Pande. Using ellipsoidal domains to analyze control systems software. 2009. http://arxiv.org/abs/0909.1977.
3. Sylvie Boldo and Guillaume Melquiond. Flocq: A Unified Library for Proving Floating-point Algorithms in Coq. In *Proceedings of the 20th IEEE Symposium on Computer Arithmetic*, pages 243–252, Tübingen, Germany, July 2011.

4. Olivier Bouissou, Yassamine Seladji, and Alexandre Chapoutot. Acceleration of the abstract fixpoint computation in numerical program analysis. *J. Symb. Comput.*, 47(12):1479–1511, 2012.
5. Stephen Boyd, Laurent El Ghaoui, Éric Féron, and Venkataramanan Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, volume 15 of *SIAM*. SIAM, Philadelphia, PA, June 1994.
6. Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
7. Adrien Champion, Rémi Delmas, Michael Dierkes, Pierre-Loïc Garoche, Romain Jobredeaux, and Pierre Roux. Formal methods for the analysis of critical control systems models: Combining non-linear and linear analyses. In Charles Pecheur and Michael Dierkes, editors, *Formal Methods for Industrial Critical Systems - 18th International Workshop, FMICS 2013, Madrid, Spain, September 23-24, 2013. Proceedings*, volume 8187 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2013.
8. The Coq development team. *The Coq proof assistant reference manual*, 2012. Version 8.4.
9. Alexandru Costan, Stephane Gaubert, Eric Goubault, Matthieu Martel, and Sylvie Putot. A policy iteration algorithm for computing fixed points in static analysis of programs. In *CAV*, pages 462–475, 2005.
10. Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.
11. Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *POPL*, pages 269–282, 1979.
12. Patrick Cousot and Radhia Cousot. Abstract interpretation frameworks. *J. Log. Comput.*, 2(4):511–547, 1992.
13. Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL*, pages 84–96, 1978.
14. Paul Feautrier and Laure Gonnord. Accelerated invariant generation for c programs with aspic and c2fsm. *Electr. Notes Theor. Comput. Sci.*, 267(2):3–13, 2010.
15. Jérôme Feret. Static analysis of digital filters. In *ESOP*, number 2986 in LNCS. Springer, 2004.
16. Jérôme Feret. Numerical abstract domains for digital filters. In *International workshop on Numerical and Symbolic Abstract Domains (NSAD)*, 2005.
17. Éric Féron. From control systems to control software. *Control Systems, IEEE*, 30(6):50 –71, December 2010.
18. Stephane Gaubert, Eric Goubault, Ankur Taly, and Sarah Zennou. Static analysis by policy iteration on relational domains. In *ESOP*, pages 237–252, 2007.
19. Thomas Gawlitza and Helmut Seidl. Precise fixpoint computation through strategy iteration. In *ESOP*, pages 300–315, 2007.
20. Thomas Martin Gawlitza and Helmut Seidl. Computing relaxed abstract semantics w.r.t. quadratic zones precisely. In *SAS*, pages 271–286, 2010.
21. Thomas Martin Gawlitza, Helmut Seidl, Assalé Adjé, Stéphane Gaubert, and Eric Goubault. Abstract interpretation meets convex optimization. *J. Symb. Comput.*, 47(12):1416–1446, 2012.
22. Khalil Ghorbal, Eric Goubault, and Sylvie Putot. The zonotope abstract domain taylor1+. In *CAV*, pages 627–633, 2009.
23. Denis Gopan and Thomas W. Reps. Lookahead widening. In *CAV*, pages 452–466, 2006.
24. Éric Goubault and Sylvie Putot. Static analysis of finite precision computations. In *VMCAI*, pages 232–247, 2011.
25. Wassim M. Haddad and Vijay S. Chellaboina. *Nonlinear Dynamical Systems and Control: A Lyapunov-Based Approach*. Princeton University Press, 2008.
26. Nicolas Halbwachs and Julien Henry. When the decreasing sequence fails. In *SAS*, pages 198–213, 2012.
27. Nicolas Halbwachs, Yann-Erick Proy, and Patrick Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.
28. Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996.
29. IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic. *IEEE Standard 754-2008*, 2008.
30. Aleksandr Mikhailovich Lyapunov. Problème général de la stabilité du mouvement. *Annals of Mathematics Studies*, 17, 1947.
31. Antoine Miné. The octagon abstract domain. In *AST 2001 in WCRE 2001*, IEEE, pages 310–319. IEEE CS Press, October 2001.
32. Antoine Miné. Relational abstract domains for the detection of floating-point run-time errors. In *ESOP*, volume 2986 of *LNCS*, pages 3–17. Springer, 2004. http://www.di.ens.fr/~mine/publi/article-mine-esop04.pdf.
33. David Monniaux. Compositional analysis of floating-point linear numerical filters. In *CAV*, pages 199–212, 2005.

34. Mardavij Roozbehani, Éric Féron, and Alexandre Megretski. Modeling, optimization and computation for software verification. In *HSCC*, pages 606–622, 2005.
35. Pierre Roux. *Static Analysis of Control Command Systems: Synthetizing Non Linear Invariants*. PhD thesis, Institut Supérieur de l'Aéronautique et de l'Espace, 2013.
36. Pierre Roux and Pierre-Loïc Garoche. Integrating policy iterations in abstract interpreters. In Dang Van Hung and Mizuhito Ogawa, editors, *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, volume 8172 of *Lecture Notes in Computer Science*, pages 240–254. Springer, 2013.
37. Pierre Roux and Pierre-Loïc Garoche. Computing quadratic invariants with min- and max-policy iterations: A practical comparison. In Cliff B. Jones, Pekka Pihlajasaari, and Jun Sun, editors, *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, volume 8442 of *Lecture Notes in Computer Science*, pages 563–578. Springer, 2014.
38. Pierre Roux, Romain Jobredeaux, Pierre-Loïc Garoche, and Éric Féron. A generic ellipsoid abstract domain for linear time invariant systems. In *HSCC*, pages 105–114, 2012.
39. Siegfried M. Rump. Verification of positive definiteness. *BIT Numerical Mathematics*, 46:433–452, 2006.
40. Siegfried M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, May 2010.
41. Peter Schrammel and Bertrand Jeannet. Logico-numerical abstract acceleration and application to the verification of data-flow programs. In *SAS*, pages 233–248, 2011.
42. Yassamine Seladji and Olivier Bouissou. Numerical abstract domain using support functions. In *NFM*, 2013.
43. Lieven Vandenberghe and Stephen Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.