Abstract Acceleration of General Linear Loops
Bertrand Jeannet, Peter Schrammel, Sriram Sankaranarayanan

To cite this version:
Bertrand Jeannet, Peter Schrammel, Sriram Sankaranarayanan. Abstract Acceleration of General Linear Loops. Principles of Programming Languages, POPL, Jan 2014, San Diego, United States. pp.529-540, 10.1145/2535838.2535843 . hal-00932342

HAL Id: hal-00932342
https://inria.hal.science/hal-00932342
Submitted on 16 Jan 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract Acceleration of General Linear Loops

Bertrand Jeannet
INRIA
bertrand.jeannet@inria.fr

Peter Schrammel *
University of Oxford
peter.schrammel@cs.ox.ac.uk

Sriram Sankaranarayanan †
University of Colorado, Boulder
srirams@colorado.edu

Abstract

We present abstract acceleration techniques for computing loop invariants for numerical programs with linear assignments and conditions. Whereas abstract interpretation techniques typically over-approximate the set of reachable states iteratively, abstract acceleration captures the effect of the loop with a single, non-iterative transfer function applied to the initial states at the loop head. In contrast to previous acceleration techniques, our approach applies to any linear loop without restrictions. Its novelty lies in the use of the Jordan normal form decomposition of the loop body to derive symbolic expressions for the entries of the matrix modeling the effect of \( n \geq 0 \) iterations of the loop. The entries of such a matrix depend on \( n \) through complex polynomial, exponential and trigonometric functions. Therefore, we introduce an abstract domain for matrices that captures the linear inequality relations between these complex expressions. This results in an abstract matrix for describing the fixpoint semantics of the loop.

Our approach integrates smoothly into standard abstract interpreters and can handle programs with nested loops and loops containing conditional branches. We evaluate it over small but complex loops that are commonly found in control software, comparing it with other tools for computing linear loop invariants. The loops in our benchmarks typically exhibit polynomial, exponential and oscillatory behaviors that present challenges to existing approaches. Our approach finds non-trivial invariants to prove useful bounds on the values of variables for such loops, clearly outperforming the existing approaches in terms of precision while exhibiting good performance.

1. Introduction

We present a simple yet effective way of inferring accurate loop invariants of linear loops, i.e., loops containing linear assignments and guards, as exemplified by the programs shown in Figs. 1 and 2.

```
real x,y,z,t;
assume(-2<=x<=2 and -2<=y<=2 and -2<=z<=2);
loop t := 0;
  while (x+y <= 30) loop guard
    { x := x+y; y := y+z; z := z+1; }
  loop exit 
```

Figure 1. Linear loop having a cubic behavior.

functions using splines. Static analysis of such programs using standard abstract interpretation theory over polyhedral abstract domains often incurs a significant loss of precision due to the use of extrapolation techniques (widening) to force termination of the analysis. However, widening is well-known to be too imprecise for such loops. In fact, specialized domains such as ellipsoids and arithmetic-geometric progressions were proposed to deal with two frequently occurring patterns that are encountered in control loops [12, 13]. These domains enable static analyzers for control systems, e.g., ASTREE, to find the strong loop invariants that can establish bounds on the variables or the absence of run-time errors [10].

In this paper, we present a promising alternative approach to such loops by capturing the effect of a linear loop by means of a so-called meta-transition [5] that maps an initial set of states to an invariant at the loop head. This process is commonly termed acceleration. The idea of accelerations was first studied for communicating finite-state machines [5] and counter automata [14]. Such accelerations can be either exact (see [5] for a survey), or abstract [15]. Abstract acceleration seeks to devise a transformer that maps initial sets of states to the best correct over-approximation of the invariant at the loop head for a given abstract domain, typically the convex polyhedra domain. Abstract acceleration enables static analyzers to avoid widening for the innermost loops of the program by replacing them by meta-transitions. As discussed in [39] and observed experimentally in [38], abstract acceleration presents the following benefits w.r.t. widening:

(i) It is locally more precise because it takes into account the loop body in the extrapolation it performs, whereas widening considers only sequences of invariants.

(ii) It performs more predictable approximations because it is monotonic (unlike widening).

(iii) It makes the analysis more efficient by speeding up convergence to a fixed point. For programs without nested loops, our acceleration renders the program loop-free.

Aside from abstract interpretation, techniques such as symbolic execution and bounded model-checking, that are especially efficient over loop-free systems, can benefit from loop acceleration.

In this paper we present a novel approach to computing abstract accelerations. Our approach is non-iterative, avoiding widening. We focus on the linear transformation induced by a linear loop body...
modeled by a square matrix $A$. We seek to approximate the set of matrices \{ $I$, $A$, $A^2$, $\ldots$ \}, which represent the possible linear transformations that can be applied to the initial state of the program to obtain the current state. This set could be defined as fixed point equations on matrices and solved iteratively on a suitable abstract domain for matrices. However, such an approach does not avoid widening and suffers from efficiency issues, because a matrix for a program with $n$ variables has $n^2$ entries, thus requiring a matrix abstract domain with $n^2$ different dimensions.

Contributions. The overall contribution of this paper is an abstract acceleration technique for computing the precise effect of any linear loop on an input predicate. It relies on the computation of the Jordan normal form of the square matrix $A$ for the loop body. Being based on abstract acceleration, it integrates smoothly into an abstract interpretation-based analyzer and can be exploited for the analysis of general programs with nested loops and conditionals by transforming them into multiple loops around a program location.

The first technical contribution is an abstract acceleration method for computing, non-iteratively, an approximation of the set \{ $I$, $A$, $A^2$, $\ldots$ \} in an abstract domain for matrices. It enables the analysis of any infinite, non-guarded linear loop. The main idea is to consider the Jordan normal form $J$ of the transformation matrix $A$. Indeed, the particular structure of the Jordan normal form $J$ has two advantages:

(i) It results in closed-form expressions for the coefficients of $J^n$, on which asymptotic analysis techniques can be applied that remove the need for widening.

(ii) It reduces the number of different coefficients of $J^n$ to at most the dimension of the vector space (efficiency issue).

This first contribution involves a conceptually simple but technically involved derivation that we omit in this paper and which can be found in the extended version [23].

The second technical contribution addresses loops with guards that are conjunctions of linear inequalities. We present an original technique for bounding the number of loop iterations. Once again, we utilize the Jordan normal form. These two techniques together make our approach more powerful than ellipsoidal methods (e.g. [34]) that are restricted to stable loops, because the guard is only weakly taken into account.

We evaluate our approach by comparing efficiency and the precision of the invariants produced with other invariant synthesis approaches, including abstract interpreters and constraint-based approaches. The evaluation is carried out over a series of simple loops, alone or inside outer loops (such as in Fig. 2), exhibiting behaviors such as polynomial, stable and unstable exponentials, and inward spirals (damped oscillators). We show the ability of our approach to discover polyhedral invariants that are sound over-approximations of the reachable state space. For such systems, any inductive reasoning in a linear domain as performed by, e.g., standard abstract interpretation with Kleene iteration and widening is often unable to find a linear invariant other than true. In contrast, our approach is shown to find useful bounds for many of the program variables that appear in such loops. To our knowledge, our method is the first one able to bound the variables in the convexCar example of Sankaranarayanan et al. [36].

Outline. We introduce some basic notions in §2. §3 gives an overview of the ideas of this paper. §§4 to 6 explain the contributions in detail. §7 summarizes our experimental results and §8 discusses related work before §9 concludes.

2. Preliminaries

In this section, we recall the notions of linear assertions and convex polyhedra, and we define the model of linear loops for which we will propose acceleration methods.

```plaintext
real t,te,time;
assume(te=14 and 16<=t and t<=17);
while true {
    time := 0; -- timer measuring duration in each mode
    while (t<=22) { -- heating mode
        t := 15/16*t-1/16*te+1; time++;
    }
    time := 0;
    while (t>=18) { -- cooling mode
        t := 15/16*t+1/16*te; time++;
    }
}
```

Figure 2. A thermostat system, composed of two simple loops inside a outer loop.

2.1 Linear assertions and convex polyhedra

Let $x_1, \ldots, x_p$ be real-valued variables, collectively forming a \( p \times 1 \) column vector $\vec{x}$. A linear expression is written as an inner product $\vec{c} \cdot \vec{x}$, wherein $\vec{c} \in \mathbb{R}^p$. A linear inequality is of the form $\vec{c} \cdot \vec{x} \leq d$, where $\vec{c}$ is a $p$-dimensional vector, and $d \in \mathbb{R}$. A linear assertion is a conjunction of linear inequalities:

\[ \phi \ := \ \bigwedge \ \lambda_i, \sum_{j=1}^{q} \mu_j \vec{r}_j \leq 0, \quad \sum_{i=1}^{l} \lambda_i = 1 \]

2.2 Linear loops

We consider linear loops consisting of a while loop, the body of which is a set of assignments without tests and the condition is a linear assertion.

Definition 1 (Linear loop). A linear loop \((G, \vec{h}, \vec{A}, \vec{b})\) is a program fragment of the form

\[
\text{while}(G\vec{x} \leq \vec{h}) \vec{x} := A\vec{x} + \vec{b};
\]

where $\phi : G\vec{x} \leq \vec{h}$ is a linear assertion over the state variables $\vec{x}$ representing the loop condition and $(A, \vec{b})$ is the linear transformation associated with the loop body.

Figure 1 shows an example of a linear loop with a guard that computes $y = x(x+1)/2$ by the successive difference method. We give another example below.

Example 1 (Thermostat). Figure 2 models the operation of a thermostat that switches between the heating and cooling modes over time. The variables $t$, $te$ model the room and outside temperatures, respectively. We wish to show that the value of $t$ remains within some bounds that are close to the switch points 18, 22 units.

Any linear loop $(G, \vec{h}, A, \vec{b})$ can be homogenized by introducing a new variable $\xi$ that is a place holder for the constant 1 to a loop
of the form
\[(G \cdot \bar{h})(\bar{x}, \xi) \leq 0; \quad \{ \bar{x} := \begin{pmatrix} A & b \\ 0 & 1 \end{pmatrix}(\bar{x}, \xi) \} \]

Henceforth, we will use the notation \((G \cdot A) \) to denote the homogenized linear loop while \((G \cdot \bar{h})(\bar{x}, \xi) \leq 0\}

Definition 2 (Semantics function). The semantics function of a linear loop \((G \cdot A) \) over sets of states is the functional
\[(G \cdot A)(X) \equiv A(X \cap [G \cdot \bar{h}(0)]), \quad X \subseteq \mathbb{R}^p \]
\nwhere \(A(Y)\) denotes the image of a set \(Y\) by the transformation \(A\).

2.3 Convex and template polyhedral abstract domains

The set of convex polyhedra \(CP(\mathbb{R}^p)\) ordered by inclusion is a lattice with the greatest lower bound being the set intersection and the least upper bound being the convex hull. The definition of the domain includes an abstraction function \(\alpha\) that maps sets of states to a polyhedral abstraction and a corresponding concretization function \(\gamma\). We refer the reader to the original work of Cousot and Halbwachs for a complete description [9].

It is well-known that the abstract domain operations such as join and transfer function across non-invertible assignments are computationally expensive. As a result, many weakly-relational domains such as octagons and templates have been proposed [39, 37]. Given a matrix \(T \in \mathbb{R}^{p \times p}\) of \(p\) linear expressions, \(CP_T(\mathbb{R}^p) \subseteq CP(\mathbb{R}^p)\) denotes the set of template polyhedra on \(T\):
\[CP_T(\mathbb{R}^p) = \{ P \in CP(\mathbb{R}^p) \mid \exists \bar{u} \in \mathbb{R}^q : P = \{\bar{x} \mid T \bar{x} \leq \bar{u}\} \}
\nwhere \(\mathbb{R}\) denotes \(\mathbb{R} \cup \{\infty\}\). A template polyhedron will be denoted by \((T, \bar{u})\). If \(T\) is fixed, it is uniquely defined by the vector \(\bar{u}\). \(CP_T(\mathbb{R}^p)\) ordered by inclusion is a complete lattice. The abstraction \(\alpha_T\) and concretization \(\gamma_T\) are defined elsewhere [37].

3. Overview

This section provides a general overview of the ideas in this paper, starting with abstract acceleration techniques.

Abstract Acceleration

Given a set of initial states \(X_0\) and a loop with the semantic function \(\tau\), the smallest loop invariant \(X\) containing \(X_0\) can be formally written as
\[X = \tau^*(X_0) = \bigcup_{n \geq 0} \tau^n(X_0)\]

Abstract acceleration seeks an “optimal” approximation of \(\tau^*\) in a given abstract domain with abstraction function \(\alpha\) [18]. Whereas the standard abstract interpretation approach to solve the fix point equation \(Y^* = \alpha(X_0) \cup (\tau(Y^*))\) by iteratively computing
\[Y = \alpha \circ \tau^*(\alpha(X_0))\]

the abstract acceleration approach uses \(\tau^*\) to compute
\[Z = \alpha \circ \tau^*(\alpha(X_0))\]

Classically, Eqn. (1) is known as the minimal fixed point (MFP) solution of the reachability problem whereas Eqn. (2) is called the Merge-Over-All-Paths (MOP) solution. The latter is known to yield more precise results [24].

The technical challenge of abstract acceleration is thus to obtain a closed-form approximation of \(\alpha \circ \tau^*\) that avoids both inductive reasoning in the abstract domain and the use of widening.

Abstract acceleration without guards using matrix abstract domains

We now present an overview for linear loop without guards, with semantic function \(\tau = (true \rightarrow A)\). For any set \(X\), we have
\[\tau^*(X) = \bigcup_{n \geq 0} \tau^n(X) = \bigcup_{n \geq 0} A^n X\]

Our approach computes a finitely representable approximation \(\mathcal{M}\) of the countably infinite set of matrices \(\bigcup_{n \geq 0} A^n\). Thereafter, abstract acceleration simply applies \(\mathcal{M}\) to \(X\).

The following example illustrates the first step.

Example 2 (Exponential 1/4). We consider the program
\[\text{while(true)\{ x=1.5x; y=y+1 \}}\]

of which Fig. 3 depicts some trajectories. After homogenization, the loop’s semantics function is
\[G = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix} \rightarrow A = \begin{pmatrix} 1.5 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}\]

Here, it is easy to obtain a closed-form symbolic expression of \(A^n\):
\[A^n = \begin{pmatrix} 1.5^n & 0 & 0 \\ 0 & 1 & n \\ 0 & 0 & 1 \end{pmatrix}\]

The idea for approximating \(\bigcup_{n \geq 0} A^n\) is to consider a set of matrices of the form
\[
\mathcal{M} = \left\{ \begin{pmatrix} m_1 & 0 & 0 \\ 0 & 1 & m_2 \\ 0 & 0 & 1 \end{pmatrix} \mid \varphi\mathcal{M}(m_1, m_2) \right\}
\]

with \(\varphi\mathcal{M}\) a linear assertion in a template domain such that \(\forall n \geq 0 : A^n \subseteq \mathcal{M}\). Using an octagonal template, for instance, the following assertion satisfies the condition above:
\[
\varphi\mathcal{M} : \begin{cases} m_1 & \in [1, +\infty) \implies \inf_{n \geq 0} 1.5^n, \sup_{n \geq 0} 1.5^n \\ m_2 & \in [0, +\infty) \implies \inf_{n \geq 0} n, \sup_{n \geq 0} n \\ m_1 + m_2 & \in [1, +\infty) \implies \inf_{n \geq 0} (1.5^n + n), \sup_{n \geq 0} (1.5^n + n) \\ m_1 - m_2 & \in [0.25, +\infty) \implies \inf_{n \geq 0} (1.5^n - n), \sup_{n \geq 0} (1.5^n - n) \end{cases}
\]

These constraints actually define the smallest octagon on entries \(m_1, m_2\) that makes \(\mathcal{M}\) an overapproximation of \(A^n = \{A^n \mid n \geq 0\}\). It is depicted in Fig. 3. The technique to evaluate the non-linear inf and sup expressions above is described in §5.2.

This is the first important idea of the paper. §4 formalizes the notion of abstract matrices, whereas §5 will exploit the Jordan normal form of \(A\) to effectively compute \(\alpha(A^n)\) for any matrix \(A\), i.e. to accelerate the loop body.

Applying the abstraction to acceleration

The next step is to apply the matrix abstraction \(\mathcal{M} = \alpha(A^n)\) to an abstract element \(X\). For illustration, assume that both \(\mathcal{M}\) and \(X\) are defined by linear assertions \(\varphi\mathcal{M}\) and \(\varphi_X\) from the polyhedral domain or some sub-polyhedral domains. Applying the set of matrices \(\mathcal{M}\) to \(X\) amounts to computing (an approximation of)
\[
\left\{ \begin{pmatrix} m_1 & 0 & 0 \\ 0 & 1 & m_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \mid \varphi_X(x, y) \wedge \varphi\mathcal{M}(m_1, m_2) \right\}
\]

This is not trivial, as the matrix multiplication generates bilinear expressions. §4 proposes a general approach for performing the abstract matrix multiplication. The result of the procedure is illustrated by the example that follows:

Example 3 (Exponential 2/4). Assume that in Ex. 2 and Eqn. (3), \(\varphi_X = (x \in [1, 3] \wedge y \in [1, 2])\). We compute the abstract matrix multiplication
\[
\mathcal{M}X = \left\{ \begin{pmatrix} m_1 & x \\ 1 + m_2 & y \\ 1 \end{pmatrix} \mid 1 \leq x \leq 3 \wedge 0 \leq y \leq 2 \wedge m_1 \geq 1 \wedge m_2 \geq 0 \wedge m_1 - m_2 \geq 0.25 \right\}
\]
by exploiting the number of iterations. Conclusion.

For some competing techniques:

We consider loops of the form \( A \) and \( \phi \), which is depicted in Fig. 4.

Handling Guards

We consider loops of the form \( \tau = G \rightarrow A \) and illustrate how the loop condition (guard) \( G \) is handled. A simple approach takes the guard into account after the fixpoint of the loop without guard is computed:

\[
(G \rightarrow A)^n(X) \subseteq X \cup (G \rightarrow A) \circ (G \rightarrow A^*)(X)
\]

which is then abstracted with \( X \cup (G \rightarrow A) \circ (G \rightarrow \alpha(A^*)) \circ \alpha(X) \).

However, such an approach is often unsatisfactory.

Example 4 (Exponential 3/4). We add the guard \( y \leq 3 \) to our running Ex. 2. Using the approximation above with \( X \) as in Ex. 3, we obtain the invariant \( Y \) depicted in Fig. 5. In this result \( y \) is bounded but \( x \) remains unbounded.

Our idea is based on the observation that the bound on \( y \) induced by the guard implies a bound \( N \) on the maximum number of iterations for any initial state in \( X \). Once this bound is known, we can exploit the knowledge \( \tau^n(X) = \bigcup_{n=0}^{N} \tau^n(X) \) and consider the better approximation

\[
(G \rightarrow A)^n(X) \subseteq X \cup (G \rightarrow A) \circ (G \rightarrow A^*) \circ \alpha(X)
\]

The set of matrices \( \bigcup_{n=0}^{N} A^n \) is then approximated in the same way as \( A^* \) in §3. We could perform an iterative computation for small \( N \), however, a polynomial analysis without widening operator is intractably expensive for hundreds or thousands iterations, while our method is both precise and efficient.

Example 5 (Exponential 4/4). In our running example, it is easy to see that the initial condition \( y_0 \in [0, 2] \) together with the guard \( y \leq 3 \) implies that the maximum number of iteration is \( N = 4 \).

Thus, we can consider the set of matrices \( M' = \alpha\left( \bigcup_{n=0}^{N} A^n \right) \) defined by the following assertion satisfies the condition \( \varphi_{M'} \): above:

\[
\begin{aligned}
& m_1 \in [1, 3.375] = \left[ \inf_{0 \leq n \leq 3} 1.5^n, \sup_{0 \leq n \leq 3} 1.5^n \right] \\
& m_2 \in [0, 3] = \left[ \inf_{0 \leq n \leq 3} n, \sup_{0 \leq n \leq 3} n \right] \\
& m_1 + m_2 \in [1, 6.375] = \left[ \inf_{0 \leq n \leq 3} (1.5^n + n), \sup_{0 \leq n \leq 3} (1.5^n + n) \right] \\
& m_1 - m_2 \in [0.25, 1] = \left[ \inf_{0 \leq n \leq 3} (1.5^n - n), \sup_{0 \leq n \leq 3} (1.5^n - n) \right]
\end{aligned}
\]

which is depicted in Fig. 3. Using the formula above, we obtain the invariant \( Z \) depicted in Fig. 5, which is much more precise than the invariant \( Y \) discovered with the simple technique.

§6 presents the technique for over-approximating the number of iterations of a loop where the guard \( G \) is a general linear assertion, \( A \) is any matrix and \( X \) any polyhedron. This is the third main contribution of the paper.

An Illustrative Comparison

The capability of our method to compute over-approximations of the reachable state space goes beyond state-of-the-art invariant inference techniques. The following table lists the bounds obtained on the variables of the thermostat of Ex. 1, Fig. 2 for some competing techniques:

| INTERPROC [22] | ASTREE [4] | STING [7, 46] | this paper |
|----------------|------------|---------------|------------|
| heating:       |            |               |            |
| 16 \leq t      | 0 \leq t   | 16 \leq t     | 0 \leq t   |
| 0 \leq t       | 0 \leq time| 16 \leq t     | 0 \leq time|
| 0 \leq time    | 0 \leq t   | 16 \leq t     | 0 \leq time|
| 0 \leq time    | 0 \leq t   | 16 \leq t     | 0 \leq time|
| 0 \leq time    | 0 \leq t   | 16 \leq t     | 0 \leq time|
| 0 \leq time    | 0 \leq t   | 16 \leq t     | 0 \leq time|
| 0 \leq time    | 0 \leq t   | 16 \leq t     | 0 \leq time|
| 0 \leq time    | 0 \leq t   | 16 \leq t     | 0 \leq time|
| 0 \leq time    | 0 \leq t   | 16 \leq t     | 0 \leq time|
| 0 \leq time    | 0 \leq t   | 16 \leq t     | 0 \leq time|
| 0 \leq time    | 0 \leq t   | 16 \leq t     | 0 \leq time|

There are many other invariant generation techniques and tools for linear systems (see §8). Many approaches sacrifice precision for speed, and therefore are inaccurate on the type of linear loops considered here. Other, more specialized approaches require conditions such as Lyapunov-stability, diagonalizability of the matrix, polynomial behavior (nilpotency or monoidal property), or handle only integer loops.

Outline of the rest of the paper

The rest of the paper develops the ideas illustrated in this section. §4 formalizes the notion of matrix abstract domains and presents a technique for the abstract matrix multiplication operation. §5 shows how to approximate the set of matrices \( A^+ = \bigcup_{n \geq 0} A^n \) for any square matrix \( A \) in order to accelerate loops without guards. §6 presents a technique for taking the guard of loops into account by approximating \( N \), the maximum number of iterations possible from a given set of initial states. §7 presents the experimental evaluation on various kinds of linear loops, possibly embedded into outer loops. §8 discusses related work and §9 concludes.

4. Matrix abstract domains

In this section we present abstract domains for matrices. We will use abstract matrices to represent the accelerated abstract transformer of a linear loop. Hence, the main operation on abstract matrices we use in this paper is abstract matrix multiplication (§4.2).

4.1 Extending abstract domains from vectors to matrices

We abstract sets of square matrices in \( \mathbb{R}^{p \times p} \) by viewing them as vectors in \( \mathbb{R}^{p^2} \) and by reusing known abstract domains over vectors. However, since the concrete matrices we will be dealing with belong to subspaces of \( \mathbb{R}^{p \times p} \), we first introduce matrix shapes that allow us to reduce the number of entries in abstract matrices.
DEFINITION 3 (Matrix shape). A matrix shape $\Psi : \mathbb{R}^m \rightarrow \mathbb{R}^{p \times p}$ is a bijective, linear map from $m$-dimensional vectors to $p \times p$ square matrices. Intuitively, matrix shapes represent matrices whose entries are linear (or affine) combinations of $m > 0$ entries.

EXAMPLE 6. In Ex. 2, we implicitly considered the matrix shape $\Psi : \mathbb{R}^2 \rightarrow \mathbb{Mat}_\Psi \subseteq \mathbb{R}^{3 \times 3}$

\[
\begin{pmatrix}
m_1 \\
m_2
\end{pmatrix} \mapsto \begin{pmatrix}
m_1 & 0 & 0 \\
0 & 1 & m_2 \\
0 & 0 & 1
\end{pmatrix}
\]

The set $\mathbb{Mat}_\Psi = \{ \Psi(\vec{m}) \mid \vec{m} \in \mathbb{R}^m \}$ represents all possible matrices that can be formed by any vector $\vec{m}$. It represents a subspace of the vector space of all matrices.

A matrix shape $\Psi$ induces an isomorphism between $\mathbb{R}^m$ and $\mathbb{Mat}_\Psi : \Psi(\vec{a}_1\vec{m}_1 + \vec{a}_2\vec{m}_2) = a_1\Psi(\vec{m}_1) + a_2\Psi(\vec{m}_2)$.

Abstract domain for matrices are constructed by (a) choosing an abstract domain for vectors $\vec{m}$ and (b) specifying a matrix shape $\Psi$. Given an abstract domain $A$ for vectors $\vec{m}$ (e.g., the polyhedral domain) and a shape $\Psi$, the corresponding matrix abstract domain defines a domain over subsets of $\mathbb{Mat}_\Psi$.

EXAMPLE 7. Recall the matrix shape $\Psi : \mathbb{R}^2 \rightarrow \mathbb{Mat}_\Psi \subseteq \mathbb{R}^{3 \times 3}$ from Ex. 6. Consider the octagon $P = \{ m_1 \geq 1 \land m_2 \geq 0 \land m_1 + m_2 \geq 1 \land m_1 - m_2 \geq 0.25 \} \subseteq \mathbb{Oct}(\mathbb{R}^2)$, together they represent an abstract matrix $(P, \Psi)$ which represents the set of matrices:

\[
\begin{pmatrix}
m_1 & 0 & 0 \\
0 & 1 & m_2 \\
0 & 0 & 1
\end{pmatrix} \mid m_1 \geq 1, m_2 \geq 0, m_1 + m_2 \geq 1, m_1 - m_2 \geq 0.25
\]

DEFINITION 4 (Abstract domain for matrices induced by $\Psi$).

Let $A \subseteq \wp(\mathbb{R}^m)$ be an abstract domain for $m$-dimensional vectors ordered by set inclusion and with the abstraction function $\alpha_A : \wp(\mathbb{R}^m) \rightarrow A$. Then, $\Psi(A)$ ordered by set inclusion is an abstract domain for $\wp(\mathbb{Mat}_\Psi)$ with the abstraction function $\alpha_{\Psi(A)}(M) = \Psi \circ \alpha_A \circ \Psi^{-1}(M)$.

Note that since $\Psi$ is an isomorphism, the lattices $A$ and $\Psi(A)$ can be shown to be isomorphic. For generality, the base domain $A$ can be an arbitrary abstract domain for the data type of the matrix entries. In our examples, we specifically discuss common numerical domains such as convex polyhedra, intervals, octagons and templates.

4.2 Abstract matrix multiplication

We investigate now the problem of convex polyhedra matrix multiplication, motivated by the need for applying an acceleration $\alpha(A^*)$ to an abstract property $X$ as shown in \S3.

The problem. We consider two convex polyhedra matrices $M_1 = \Psi_1(P_1)$ and $M_2 = \Psi_2(P_2)$. We aim at computing an approximation of

\[
M = M_1M_2 = \{ M_1M_2 \mid M_1 \in M_1 \land M_2 \in M_2 \}
\]

under the form of a convex polyhedron on the coefficients of the resulting matrix. Observe that $M$ may be non-convex as shown by the following example.

EXAMPLE 8. Consider the two abstract matrices

$M_1 = \left\{ \begin{pmatrix} 1 - m & 0 \\ 0 & m \end{pmatrix} \mid m \in [0, 1] \right\}$

$M_2 = \left\{ \begin{pmatrix} 1 - n \\ n \end{pmatrix} \mid n \in [0, 1] \right\}$

We have $M_1M_2 = \left\{ \begin{pmatrix} (1 - m)(1 - n) \\ mn \end{pmatrix} \mid m \in [0, 1] \land n \in [0, 1] \right\}$

This corresponds to the well-known non-convex set of points $x \in [0, 1] \land (x - y)^2 + 1 \geq 2(x + y)$, depicted to the right.

We may follow at least two approaches for approximating $M_1M_2$:

- Either we consider the constraint representations of $M_1$ and $M_2$, and we resort to optimization techniques to obtain a template polyhedra approximation of the product;
- Or we consider their generator representations to obtain a convex polyhedron approximating the product.

We opted in this paper for the second, i.e. the generator approach, which leads to more accurate results:

- It delivers general convex polyhedra, more expressive than template polyhedra obtained by optimization;
- It computes the best correct approximation in the convex polyhedra domain for bounded matrices (Thm. 1 below), whereas in the constraint approach the exact optimization problem involves bilinear expressions (see Eqn. (3) or Ex. 8) and must be relaxed in practice.

Multiplying abstract matrices using generators. Given two finite sets of matrices $X = \{ X_1, \ldots, X_m \}$ and $Y = \{ Y_1, \ldots, Y_n \}$ we write $X \otimes Y$ to denote the set

\[
X \otimes Y = \{ X_iY_j \mid X_i \in X, Y_j \in Y \}
\]

If $M_n$ is expressed as a system of matrix vertices $V_n = (V_{i,s})$ and matrix rays $R_n = (R_{i,s})$, $s = 1, 2$, then

\[
M_n = \left\{ \sum_{i,s} \lambda_{i,s}V_{i,s} + \sum_{j,s} \mu_{j,s}R_{j,s} \mid \sum_{i,s} \lambda_{i,s} = 1 \right\}
\]

and Eqn. (4) can be rewritten

$M = M_1M_2 = \left\{ \sum_{i,s} \lambda_{i,s}V_{i,s} + \sum_{j,s} \mu_{j,s}R_{j,s} \mid \lambda_{i,s}\mu_{j,s} \geq 0 \right\}$. (5)

We obtain the following result:

THEOREM 1. Let $M_1$ and $M_2$ be two abstract matrices expressed as a system of vertices and rays $V_1, R_1$ for $M_1$ and $V_2, R_2$ for $M_2$. The matrix polyhedron $M$ defined by the set of vertices $V = V_1 \otimes V_2$ and the set of rays

\[
R = (V_1 \otimes R_2) \cup (R_1 \otimes V_2) \cup (R_1 \otimes R_2)
\]

is an overapproximation of $M = M_1M_2$.

Moreover, if $M_1$ and $M_2$ are bounded, i.e. if $R_1 = R_2 = \emptyset$, then $M$ is the smallest polyhedron matrix containing $M$.

PROOF. For the first part of the theorem, we observe that in Eqn. (5), $\sum_{i,s} \lambda_{i,s}\mu_{i,s} = 1$ and the other similar sums are positive and unbounded. Hence

$M \subseteq M = \left\{ \sum_{i,s} \lambda_{i,s}V_{i,s} + \sum_{j,s} \mu_{j,s}R_{j,s} \mid \lambda_{i,s}\mu_{i,s} \geq 0 \right\}$

which proves the first statement. Now assume that $R_1 = R_2 = \emptyset$, which means that both $M_1$ and $M_2$ are bounded and that $R = \emptyset$. We will show that all the generator vertices of $M$ belong to $M$, hence any of their convex combination (i.e. any element of $M$) belongs to the convex closure of $M$: Consider the generator vertex
and de

the number of constraints. In practice, we did not face complexity

voyCar3

the space

In Fig. 4 and generated

by 4 vertices, which resulted in the convex polyhedra Y depicted in Fig. 4 which is generated by 3 vertices and 2 rays (we omit redundant generators).

Regarding complexity, this operation is quadratic w.r.t. the number of generators, which is itself exponential in the worst-case w.r.t. the number of constraints. In practice, we did not face complexity problems in our experiments, apart from the high-dimensional convoyCar3 example described in §7.

Observe that by using generators and applying Thm. 1, we lose information about matrix shapes. In our case, we will perform only abstract matrix-vector multiplication, hence the number of entries of the product matrix (actually a vector) will be the dimension of the space \( \mathbb{R}^p \). The multiplication of an abstract matrix \( M \) and a concrete matrix \( R(\mathbb{M} R) \) or \( R(M) \) can be computed exactly by considering the generators of \( M \).

5. Abstract acceleration of loops without guards

In this section, we consider loops of the form

while(true) \{\( x := A \times z \)\}

Given an initial set of states \( X \) at the loop head, the least inductive invariant at loop head is

\[ A^* X = \{ A^n X \mid n \geq 0, n \in \mathbb{Z} \} \]

Our goal is to compute a template polyhedral matrix \( M \) such that

\[ \alpha(M(A^*)) \subseteq M \]

given a template \( T \) on the coefficients of the matrices \( M \in A^* \).

The key observation underlying our approach uses a well-known result from matrix algebra. Any square matrix \( A \) can be written in a special form known as the Jordan normal form using a change of basis transformation \( R \):

\[ A = R^{-1} J R \]

such that for any

\[ A^n = R^{-1} J^n R \]

As a result, instead of computing an abstraction of the set

\( A^* = \{ I, A, A^2, A^3, \ldots \} \)

we will abstract the set

\[ J^* = \{ I, J, J^2, J^3, \ldots \} \]

1. The block diagonal structure of \( J \) allows us to symbolically compute the coefficients of \( J^n \) as a function of \( n \). §5.1 presents details on the Jordan form and the symbolic representation of \( J^n \).

2. The form of \( J \) immediately dictates the matrix shape \( \Psi(\bar{m}) \) and the matrix subspace \( \text{Mat}_F \) containing \( J^* \).

3. We then consider a fixed set \( T \) of linear template expressions over \( \bar{m} \). We use asymptotic analysis to compute bounds on each expression in the template. §5.2 explains how this is computed.

4. Once we have computed an abstraction \( M \supseteq \alpha(J^*) \), we will return into the original basis by computing \( R^{-1} M R \) to obtain an abstraction for \( A^* \), which is the desired loop acceleration.

In this section, we assume arbitrary precision numerical computations. The use of finite precision computations is addressed in §7.

5.1 The real Jordan normal form of a matrix

A classical linear algebra result is that any matrix \( A \in \mathbb{R}^{p \times p} \) can be transformed in a real Jordan normal form by considering an appropriate basis [28]:

\[ J = \text{Diag}(J_1 \ldots J_k) \]

\[ A = R^{-1} \begin{pmatrix} J_1 & \cdots & \cdots & \cdots \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots \end{pmatrix} \]

\[ J_s = \begin{pmatrix} \Lambda_s I & \cdots & \cdots & \cdots \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots \end{pmatrix} \]

with \( \Lambda_s = \lambda_s I \) and \( I = 1 \)

if \( \lambda_s \) is a real eigenvalue of \( M \),

or \( \Lambda_s = \begin{pmatrix} \lambda_s \cos \theta_s & -\lambda_s \sin \theta_s \\ \lambda_s \sin \theta_s & \lambda_s \cos \theta_s \end{pmatrix} \) and \( I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \)

if \( \lambda_s e^{i \theta_s} \) and \( \lambda_s e^{-i \theta_s} \) are complex conjugate eigenvalues of \( M \), with \( \lambda_s > 0 \) and \( \theta_s \in [0, \pi] \).

The Jordan form is useful because we can write a closed-form expression for its \( n \)th power \( J^n = \text{Diag}(J_1^n \ldots J_k^n) \). Each block \( J_s^n \) is given by

\[ J_s^n = \begin{pmatrix} \lambda_s \gamma_0 \cdots \cdots \cdots \gamma_{n-1} & \cdots & \cdots & \cdots \\ \cdots & \ddots & \ddots & \cdots \\ \cdots & \ddots & \ddots & \ddots \\ \cdots & \ddots & \ddots & \ddots \end{pmatrix} \]

with the convention that \( \gamma_{n-k} \lambda_s^{-k} = 0 \) for \( k > n \).

Hence, coefficients of \( J_s^n \) have the general form

\[ \varphi(\lambda, \theta, r, k)(n) = \begin{pmatrix} \gamma_0 & \cdots & \cdots & \cdots \\ \cdots & \ddots & \ddots & \cdots \\ \cdots & \ddots & \ddots & \ddots \\ \cdots & \ddots & \ddots & \ddots \end{pmatrix} \]

(7)

with \( \lambda \geq 0, \theta \in [0, \pi], r \in \{0, 1\} \) and \( k \geq 0 \), in which \( r = 1 \) enables converting the cosine into a sine. The precise expressions for \( \lambda, \theta, r, k \) as functions of the position \( i, j \) in the matrix \( J \) are omitted here to preserve the clarity of presentation.

Next, we observe that the closed form \( J_s^n \) specifies the required shape \( \Psi_s(\bar{m}_s) \) for abstracting \( J_s^n \) for all \( n \geq 0 \). For instance, if \( \lambda_s \) is a real eigenvalue, we have

\[ \Psi_s : \begin{pmatrix} m_0 \\ \vdots \\ m_{p_s-1} \end{pmatrix} \mapsto \begin{pmatrix} m_0 & m_1 & \cdots & m_{p_s-1} \\ m_0 & m_1 & \cdots & m_{p_s-1} \\ \vdots & \ddots & \ddots & \vdots \\ m_0 & m_1 & \cdots & m_{p_s-1} \end{pmatrix} \]

Likewise, \( \Psi_s(\bar{m}) \) for the entire matrix \( J^n \) is obtained by the union of the parameters for each individual \( J_s^n \).

5.2 Abstracting \( J^* \) within template polyhedron matrices

The principle. Let us fix a template expression matrix \( T \in \mathbb{R}^{q \times m} \) composed of linear expressions \( \{ T_1, \ldots, T_q \} \) on parameters \( \bar{m} \).

Knowing the symbolic form of each \( J_s^n \), we obtain a symbolic form \( \Psi_s(\bar{m})(n) = \Psi_s^{-1}(J_s^n) \) for parameters \( \bar{m} \), hence a symbolic form for linear expressions \( c_j(n) = T_j : \bar{m} \) \( n \). By deriving an upper bound
The template polyhedron matrix $\mathcal{M}$ is at the heart of our technique. However, the actual derivations are version $\left[0 \leq n \leq 4\right]$. Concerning the choice of template expressions in our implementa-

### Example 6.1

The simple technique

The expression $(G \rightarrow A)^*$ unfolded in Eqn. (10) is too complex to be accelerated precisely. A simple technique to approximate it safely is to exploit the following inclusion:

**Proposition 2.** For any set $X$ and linear transformation $G \rightarrow A$,

$$\text{while} (G \rightarrow A)^* \subseteq \text{id} \cup (G \rightarrow A^t) \circ (G \rightarrow A^t)$$

finally, take into account the guard

acceleration without guard

$$\left(\left(\begin{array}{cc} 0.8 \cos \theta & -0.8 \sin \theta \\ 0 & 0 \end{array}\right) \right)^{\ast}$$

with $A$ already in real Jordan normal form. The matrix subspace containing $A^\ast$ is of the form $M = \left[\begin{array}{cc} m_1 & -m_2 \\ m_2 & 0 \end{array}\right]$. We have $m_1(n) = 0.8^n \cos n\theta$ and $m_2(n) = 0.8^n \sin n\theta$; applying our bounding technique on octagonal template constraints on $\vec{m}$, we obtain an approximation $M_A$ of $A^\ast$ defined by the constraints

$$m_1 \in [-0.29, 1.00], \quad m_1 + m_2 \in [-0.29, 1.12], \quad \{ \begin{array}{ll} m_1 \in [-0.15, 0.56] \\ m_2 \in [-0.57, 1.00] \end{array}$$

Consider, for example, the expression $m_1 + m_2 = 0.8^n \cos n\theta + \sin n\theta$ in Example 10 below, which falls into the second case above ($\theta_1 = \theta_2$ and $k_1 = k_2 = 0 \in \mathbb{R}$). We first rewrite it as $f(x) = 0.8^n \sqrt{2} \sin(x + \frac{\pi}{4})$. The term $0.8^n$ being decreasing, the least upper bound of $f$ in reals is in the range $x \in [0, \pi/4]$. Hence we can consider the upper bound max $f(n) \leq \pi/4 [0, 1]$.

The possible values $(m_1, m_2)$ are plotted in Fig. 6 (right).

Assuming an initial set $X : x \in [1, 3] \land y \in [0, 2]$, we compute $\alpha(\mathcal{M}X)$ to be the polyhedron depicted in Fig. 6 (right).

In this section, we have described the computation of a correct approximation $\mathcal{M}_A$ of $G^\ast$ in the template polyhedron domain, from which we can deduce a correct approximation $R^{-1} \mathcal{M}_A R$ of $A^\ast = R^{-1}J^\ast R$. Applying Thm. 1, we are thus able to approximate the set $A^\ast X$ of reachable states at the head of a linear loop while(true) $\{ \vec{x} := A\vec{x} \}$ with the expression $(R^{-1} \mathcal{M}_A R) \otimes X$, where $X$ is a convex polyhedron describing the initial states.

### 6. Abstract acceleration of loops with guards

---

**Example 10.** Take $a = 0.8 \cos \theta$ and $b = 0.8 \sin \theta$ with $\theta = \pi/6$, and consider the loop while(true) $\{ x' = ax - by; \ y = ax + by; \ x = x' \}$. The trajectories (see Fig. 6 (right)) of this loop follow an inward spiral. The loop body transformation is $A = \left[\begin{array}{cc} 0.8 \cos \theta & -0.8 \sin \theta \\ 0 & 0 \end{array}\right]$.
Fig. 7 illustrates graphically Prop. 2: the invariant attached to the accepting location of Fig. 7(a) is included in the invariant attached to the accepting location of Fig. 7(b). It is interesting to point out the fact that the abstract acceleration techniques described in [18, 39] make assumptions on the matrix $A$ and exploit convexity arguments so that the inclusion (11) becomes an equality. The idea behind Prop. 2 is applied to matrix abstract domains to yield Prop. 3:

**Proposition 3.** Let $A = R^{-1}JR$ with $J$ a real Jordan normal form, $T$ a template expression matrix, $M = R^{-1}\alpha_T(J^*)R$ and $X$ a convex polyhedron, then $(G \to A)^n(X)$ can be approximated by the convex polyhedron

$$X \cup (G \to A)(M \cap (X \cap G))$$

(12)

This approach essentially consists of partially unfolding the loop, as illustrated by Fig. 7(b), and reusing abstract acceleration without guard. However, since the guard is only taken into account after the actual acceleration, precision is lost regarding the variables that are not constrained by the guard. Ex. 4 and Figures 4 and 5 in §3 illustrate this weakness: $y$ is constrained by the guard, whereas $x$ remains unbounded.

### 6.2 Computing and exploiting bounds on the number of iterations.

To overcome the above issue, we propose a solution based on finding the maximal number of iterations $N$ of the loop for any initial state in $X$, and then to abstract the set of matrices $\{A^n, \ldots, A^N\}$ instead of the set $A^*$. The basic idea is that if there exists $N > 0$ such that $A^N X \cap G = \emptyset$, then $N$ is an upper bound on the number of iterations of the loop for any initial state in $X$. Bounding the number of iterations is a classical problem in termination analysis, to which our general approach provides a new, original solution.

The following theorem formalizes this idea: We assume now a guarded linear transformation $G \to J$ where $J$ is already a Jordan normal form.

**Theorem 3.** Given a set of states $X$, a template expression matrix $T$ and the set of matrices $\mathcal{M} = \alpha_T(J^*)$, we define

$$\mathcal{G} = \mathcal{M} \cap \{M \mid \exists \vec{x} \in (X \cap G) : GM\vec{x} \leq 0\}$$

$$N = \min \{n \mid J^n \notin \mathcal{G}\}$$

with the convention $\min \emptyset = \infty$.

$G$ is the set of matrices $M \subseteq M$ of which the image of at least one input state $\vec{x} \in X$ satisfies $G$. If $N$ is bounded, then

$$(G \to J)^n(X) = \bigcup_{n=0}^{N} (G \to J)^n(X)$$

**Proof.** As $J^N \notin \mathcal{M} \setminus G$, we have $\exists \vec{x} \in (X \cap G) : GJ^N\vec{x} > 0$, or in other words $(J^N X) \cap G = \emptyset$. This implies that $((G \to J)^N X) \cap G = \emptyset$ and $(G \to J)^{N+1} X = \emptyset$. The definition of $G$ and $N$ in Thm. 3 can be transposed in the space of vectors using the matrix shape $Ψ$.

**Theorem 4.** Under the assumption of Thm. 3, and considering $\vec{m}(n) = Ψ^{-1}(J^n)$, we have

$$\Psi^{-1}(G) = \left\{ \vec{m} \mid \exists \vec{x} \in X \cap G \mid GΨ(\vec{m})\vec{x} \leq 0 \right\}$$

$$N = \min \{n \mid \vec{m}(n) \notin \Psi^{-1}(G)\}$$

(13)

(14)

Our approach to take into the guard is thus to compute a finite bound $N$ with Eqsns. (13) and (14) and to replace in Eqn. (12)

$$\mathcal{M} = R^{-1} \cdot \alpha_T(J^*) \cdot R$$

with $\mathcal{M}' = R^{-1} \cdot \alpha_T(\{J^n \mid 0 \leq n \leq N-1\}) \cdot R$ and $R$ the basis transformation matrix (see Prop. 3).

**Example 11.** In our Examples 2-5, we had $\vec{m}(n) = Ψ^{-1}(J^n) = (1.5^n)$. $Ψ^{-1}(G) = (m_1 \geq 1 \& m_2 \geq 0 \& m_1 + m_2 \geq 1 \& m_1 - m_2 \geq 0.25)$, see Fig. 3 (light gray), $X = (x \in [1, 3] \& y \in [0, 2])$, see Fig. 5 (dark gray), and $y \leq 3$. The second term indicates the intersection in Eqn. (13) evaluates to $m_2 \leq 3$, thus $Ψ^{-1}(G) = (m_1 \geq 1 \& 0 \leq m_2 \leq 3 \& m_1 + m_2 \geq 1 \& m_1 - m_2 \geq 0.25)$. This gives us through Eqn. (14) the bound $N = 4$ on the number of loop iterations. The abstraction $\mathcal{M}' = \alpha_T(\{J^n \mid 0 \leq n \leq 3\})$, depicted on Fig. 3 (dark gray), removes those matrices from $\mathcal{M}$ that do not contribute to the acceleration result due to the guard. Finally, we accelerate using $\mathcal{M}'$ and obtain the result shown in Fig. 5 (medium gray).

More details about these computations are given in the next section.

### 6.3 Technical issues

Applying Thms. 3 and 4 requires many steps. First, we approximate $Ψ^{-1}(G)$ (see Eqn. (13)), and then as a second step we can approximate the maximum number of iterations $N$ according to Eqn. (14). Finally, we have to compute $\alpha_T(J^n \mid 0 \leq n \leq N-1)$.}

**Approximating $Ψ^{-1}(G)$.** Let us denote $Q = Ψ^{-1}(G)$ and $P = Ψ^{-1}(\mathcal{M})$. $Q$ is defined in Eqn. (13) by the conjunction of quadratic constraints on $\vec{x}$ and $\vec{m}$ followed by an elimination of $\vec{x}$. Exact solutions exist for this problem, but they are costly. The alternative adopted in this paper is to approximate $Q$ by quantifying $\vec{x}$ on the bounding box of $X \cap G$ instead of quantifying it on $X \cap G$. Let us denote the bounding box of a polyhedron $Z$ with the vector of intervals $[Z, Z]$. We have

$$Q \subseteq P \cap \{\vec{m} \mid \exists \vec{x} : \vec{x} \in (X \cap G) \cap GΨ(\vec{m})\vec{x} \leq 0\}$$

$$= P \cap \{\vec{m} \mid GΨ(\vec{m}) (X \cap G) \cap GΨ(\vec{m})\vec{x} \leq 0\} \triangleq Q'$$

(15)

$Q'$ is defined by intersecting $P$ with interval-linear constraints on $\vec{m}$ and it is much easier to compute than $Q$: one can use

- algorithms for interval linear constraints [6], in particular interval linear programming [33], or
- the linearization techniques of [30] that are effective if the vectors $\vec{m}$ are “well-constrained” by $P$.

In this paper, we exploit the last method which is implemented in the APRON library [21].

**Example 12.** Coming back to our running Examples 2-5 and 11, we have

$$GΨ(\vec{m}) [X \cap G, X \cap G] = (0 1 -3) \begin{pmatrix} m_1 & 0 & 0 \\ 0 & 1 & m_2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} [1, 3] \\ [0, 2] \end{pmatrix}$$

$$= (0 1 -3) \begin{pmatrix} [1, 3]m_1 \\ [0, 2] + m_2 \\ 1 \end{pmatrix} = m_2 + [-3, -1]$$

**Hence** $Q' = P \cap [m_2 + [-3, -1] \leq 0] = P \cap [m_2 \leq 3]$.

536
Approximating the maximum number of iterations $N$. Computing $N$ as defined in Eqn. (14) is not easy either, because the components of vector $\vec{m}(n)$ are functions of defined by Eqn. (7). Our approach is to exploit a matrix of template expressions.

**Proposition 4.** Under the assumption of Thm. 4, for any polyhedron $Q^* \supseteq Q$ and template expression matrix $T^*$,

$$N \leq \min \left\{ n \mid \exists j : T^*_j \cdot \vec{m}(n) > \sup T^*_j \cdot \vec{m} \right\}$$

**(16)**

**Proof.** We have $Q \subseteq Q^* \subseteq \alpha_T^{-1}(Q^*)$. From $Q \subseteq \alpha_T^{-1}(Q^*)$, it follows $\vec{m}(n) / \alpha_T^{-1}(Q^*)$ and $\vec{m}(n) / Q$, and hence $\alpha_T^{-1}(Q^*)$ is equivalent to $\exists j : T^*_j \cdot \vec{m}(n) > \sup T^*_j \cdot \vec{m}$ and $\sup T^*_j \cdot \vec{m} = \sup T^*_j \cdot \vec{m}$, hence we get the result. \hfill \Box

In our implementation we compute such a $Q^*$ as described in the previous paragraph and we choose for $T^*$ the template matrix $T$ considered in §5.2, to which we may add constraints of $Q^*$. The computation of such minima ultimately relies on the Newton-Raphson method for solving equations. Our implementation deals with the same cases as those mentioned in §5.2 and in other cases safely returns $N_j = \infty$. We refer to the extended version [23] for details.

**Example 13.** Coming back to Examples 2-5 and 11-12, and considering $T_j = (0, 1)$, we have

$T_j \cdot \vec{m}(n) = (0, 1) \cdot \left( \frac{1.5}{n} \right) = n, e_j = \sup T_j \cdot \vec{m} = \sup m_2 = 3$

and $N_j = \min \{ n \mid T_j \cdot \vec{m}(n) > e_j \} = 4$, which proves that is an upper bound on the maximum number of iterations of the loop, as claimed in Ex. 5.

**Computing $\alpha_T(J^n \mid 0 \leq n \leq N - 1)$**. If no finite upper bound $N$ is obtained with Prop. 4 (given an input polyhedron $X$), then we apply the method of §6.1. Otherwise, we replace in Eqn. (12) the set $\mathcal{M} = \alpha_T(A^n)$ with $\mathcal{M}' = \alpha_T(\{J^n \mid 0 \leq n \leq N - 1\})$. This set $\mathcal{M}'$ can be computed using the same technique as those mentioned in §5.2 and detailed in [23], or even by enumeration if $N$ is small.

**Example 14 (Running example with more complex guard).** Coming back to Examples 2-5, we consider the same loop but with the guard $x + 2y \leq 100$ represented by the matrix (1 2 100). Compared to Ex. 12 we have now $G \Psi(\vec{m})]X \land G, X \land G] = [1, 3]m_1 + [0, 4] + 2m_2 - 100$, hence, if $P$ is defined by $\varphi_M$ as in Ex. 2,

$Q = P \cap [1, 3]m_1 + [0, 4] + 2m_2 - 100$, linearization based on $P \Rightarrow m_1 \geq 1 \Rightarrow m_1 + 2 \leq [1, 3]m_1$

Using $Q'$ to bound the number of iterations according to Eqn. (16) leads to $N = 11$ (obtained with the template expression $T_j = (1 2)$ taken from the guard). At last we obtain the invariant $Z$ depicted in Fig. 8. If instead of octagonal template expressions with $e = 1$ in Eqn. (9), we choose $e = 3$, we do not improve the bound $N = 11$ but we still obtain the better invariant $Z'$ on Fig. 8.

6.4 Summary

We summarize now our method in the general case:

Given a guarded linear transformation $G \rightarrow A$, with $J = RAR^{-1}$ the (real) Jordan normal form of $A$, its associated matrix shape $\Psi$, a template expression matrix $T$ and a convex polyhedron $\vec{x}$.

**Figure 8.** Initial set of states $X$ (dark gray), loop invariants $Z$ (light gray) and $Z'$ (medium gray) obtained in Ex. 14.

**7. Implementation and Experiments**

We implemented the presented approach in a prototype tool, evaluated over a series of benchmark examples with various kinds of linear loops, possibly embedded into outer loops, and compared it to state-of-the-art invariant generators.

7.1 Implementation

We integrated our method in an abstract interpreter based on the APRON library [21]. We detail below some of the issues involved.

**Computation of the Jordan normal form**. To ensure soundness, we have taken a symbolic approach using the computer algebra software SAGE\(^1\) for computing the Jordan normal forms and transformation matrices over the field of algebraic numbers. The matrices are then approximated by safe enclosing interval matrices.

**Loops with conditionals**. Loops of which the body contains conditionals like while($G$) { if ($C$) $\vec{x} = A_1 \vec{x}$ else $\vec{x} = A_2 \vec{x}$} can be transformed into two self-loops $\tau_1, \tau_2$ around a “head” location that are executed in non-deterministic order: We iteratively acceler-\(\tau_1 \cup \tau_2\)erate each self-loop separately $(\tau_1 \circ \tau_2)$\(^*\). Since convergence of the outer loop $(\tau_1 \circ \tau_2)$\(^*\) is not guaranteed in general, we might have to use widening. Yet, practical experience shows that in many cases a fix point is reached after a few iterations.

**Nest loops**. We could use a similar trick to transform nested loops while($G_1$) { while($G_2$) $\{ \vec{x} = A_3 \vec{x} \}$} into multiple linear self-loops $\tau_1, \tau_2, \tau_3$ by adding a variable $y$ (initialized to 0) to encode the control flow:

$\tau_1 : (G_1 \vec{x} \leq 0 \land y = 0) \rightarrow (\vec{x} = \vec{x} \land y' = 1)$

$\tau_2 : (G_2 \vec{x} \leq 0 \land y = 1) \rightarrow (\vec{x} = A_2 \vec{x} \land y' = 1)$

$\tau_3 : (G_3 \vec{x} > 0 \land y = 0) \rightarrow (\vec{x} = A_3 \vec{x} \land y' = 0)$

However, the encoding of the control flow in an integer variable is ineffective because of the convex approximation of the polyhedral

\[ \text{www.sagemath.org} \]
abstract domain, this transformation causes an unacceptable loss of precision. For this reason, we accelerate only inner loops in nested loops situations. Our experimental comparison shows that computing precise over-approximations of inner loops greatly improves the analysis of nested loops, even if widening is applied to outer loops.

7.2 Evaluation

Benchmarks

Our benchmarks listed in Table 1 include various examples of linear loops as commonly found in control software. They contain filters and integrators that correspond to various cases of linear transformations (real or complex eigenvalues, size of Jordan blocks). parabola and cubic are loops with polynomial behavior (similar to Fig. 1), exp_div is Ex. 5 and thermostat is Ex. 1. inv_pendulum is the classical model of a pendulum balanced in upright position by movements of the cart it is mounted on. oscillator is a damped oscillator, and oscillator2 models a pendulum that is only damped in a range around its lowest position, as if it was grazing the ground, for example. This is modeled using several modes. In ConvoyCar [36], a leading car is followed by one or more cars, trying to maintain their position at 50m from each other:

The equations for \( N - 1 \) following cars are:

\[
\begin{align*}
\dot{x}_2 &= c(x_{i-1} - \dot{x}_i) + d(x_{i-1} - x_i - 50) \\
\dot{x}_i &= c(x_{i-1} - \dot{x}_i) + d(x_{i-1} - x_i - 50)
\end{align*}
\]

We analyzed a discretized version of this example to show that there is no collision, and to compute bounds on the relative positions and speeds of the cars. This example is particularly interesting because the real Jordan form of the loop body has blocks associated to complex eigenvalues of size \( N - 1 \).

All benchmarks have non-deterministic initial states (typically bounding boxes). Some benchmarks where analyzed for different sets of initial states (indicated by suffix \( \pm \)).

Comparison

Existing tools can only handle subsets of these examples with reasonable precision. We compared our method with

- INTERPROC [22] that implements standard polyhedral analysis with widening [8];
- STING that implements the method of [7, 36].

A comparison with the ASTRÉE tool on the thermostat example has been given in Section 3. A detailed qualitative comparison between various methods described in §8 is shown in Table 2.

Results

Table 1 lists our experimental results. We compared the tools based on the number of finite bounds inferred for the program variables in each control point. Where applicable, we report the number of bounds more/less precise finite bounds inferred by our tool in comparison to the other tools.

We note that our analysis dramatically improves the accuracy over the two competing techniques. On all the benchmarks considered, it generally provides strictly stronger invariants and is practically able to infer finite variable bounds whenever they exist. For instance, for the thermostat example (Fig. 2), we infer that at least 6.28 and at most 9.76 seconds are continuously spent in heating mode and 10.72 to 12.79 seconds in cooling mode. INTERPROC just reports that time is non-negative and STING is able to find the much weaker bounds [0.88, 13.0] and [0.94, 19.0]. On the convoyCar examples, our method is the only one that is able to obtain non-trivial bounds on distances, speeds and accelerations.

Yet, this comes at the price of increased computation times in general. INTERPROC is significantly faster on all examples; STING is faster on half of the benchmarks and significantly slower on the other half: for two examples, STING does not terminate within the given timeout of one hour, whereas our tool gives precise bounds after a few seconds. It must be noted that part of the higher computation time of our tool is a “one-time” investment for computing loop accelerations that can pay off for multiple applications in the course of a deployment of our approach in a tool such as ASTRÉE. In all but two of the examples (the exceptions being convoyCar3_2 [123]), the one-time cost dominates the overall

\(^2\)A detailed account of the benchmarks and the obtained invariants can be found on http://www.cs.ox.ac.uk/people/peter.schrammel/acceleration/jordan/.
cost of our analysis. All in all, computation times remain reason-
able in the view of the tremendous gain in precision.

8. Related work

Invariants of linear loops. The original abstract acceleration technique of Gonnord et al. [11, 18] precisely abstracts linear loops performing translations, translations/resets and some other special cases. The affine derivative closure method [2] approximates any loop by a translation; hence it can handle any linear transformation, but it is only precise for translations. The tool INVGEN [19] uses template-based constraint solving techniques for property-directed invariant generation, but it is restricted to integer programs.

Methods from linear filter analysis target stable linear systems in the sense of Lyapunov stability. These techniques consider ellip-
soidal domains. Unlike our method, they are able to handle inputs, but they do not deal with guards. Feret [12, 13] designs specialized domains for first and second order filters. These methods are imple-
mented in the ASTRÉE tool [4]. Monniaux [31] computes interval bounds for composed filters. Roux et al. [34] present a method based on semidefinite programming that infers both shape and ratio of an ellipsoid that is an invariant of the system. In contrast, our method does not impose any stability requirement.

Colon et al. [7, 36] describe a method, implemented in the tool STING, for computing single inductive, polyhedral invariants for loops based on non-linear constraint solvers. It is a computationally expensive method and in contrast, our approach is able to infer sound polyhedral over-approximations for loops where the only inductive polyhedral invariant is true.

Relational abstraction methods [27, 35, 41] aim at finding a relation between the initial state $\vec{x}$ and any future state $\vec{x}$ in continu-
ous linear systems $d\vec{x}(t)/dt = A\vec{x}(t)$, which is a problem similar to the acceleration of discrete linear loops. The invariants are com-
puted based on off-the-shelf quantifier elimination methods over real arithmetic. In contrast to our method, they handle only diag-
ogonalizable matrices $A$. [32] proposes a similar approach for discrete

| Type of Linear Transformation | Illustrating Examples | Linear Abstr. Accel. | Ellipsoids | Relational Abstraction | ALIGATOR | STING |
|------------------------------|-----------------------|---------------------|-----------|-----------------------|----------|-------|
| Translations/Resets          | $\lambda_i = 1/x_k \leq 2/\lambda_i = 0$ | yes                 | yes       | no                    | yes      | no    |
| Polynomials                  | $\lambda_i \in (0, 1)$ | $\text{parabola, cubic}$ | yes       | no                    | no       | yes   |
| Exponentials                 | $\lambda_i \in R^+ \land s_k = 1$ | $\text{exp, div}$ | yes       | no                    | no       | yes   |
| Rotations                    | $\lambda_i \in C \land s_k = 1$ | $\text{oscillator, oscillator2}$ | yes       | no                    | if $|\lambda_i| < 1$ | no |
| Non-Diagonalizable & Non-Polynomial | $s_k \geq 2 \land \lambda_i \not\in \{0, 1\}$ | $\text{thermostat, convoyCar}$ | yes       | no                    | no       | no    |
| Unstable & Non-Polynomial    | $|\lambda_i| > 1$ | $\text{exp, div, oscillator2}$ | yes       | no                    | no       | no    |
| Loop Guard Handling          | $\text{parabola, exp, div, thermostat}$ | yes               | yes       | partially             | yes      | no    |
| Inputs/Noise                 | $\text{no}$ | yes | yes | yes | yes | yes |
| Abstract Domain              | $\text{polyhedra, polyhedra, ellipsoids, template polyhedra, polynomial equals}$ | $\text{polyhedra}$ | $\text{polyhedra}$ | $\text{ellipsoids}$ | $\text{template polyhedra}$ | $\text{polynomial equals}$ | $\text{polyhedra}$ |

$s_k$ is the size of a Jordan block and $\lambda_i$ its associated eigenvalue.

Table 2. Classification of linear loops and the capabilities of various analysis methods to infer precise invariants
in case of exponential and oscillating behavior. Our experiments show that we are able to infer invariants that are out of the reach of existing methods. The precise analysis of linear loops is an essential feature of static analyzers for control programs. Precise loop invariants are equally important for alternative verification methods based on model checking, for example. Further possible applications include termination proofs and deriving complexity bounds of algorithms.

**Ongoing work.** In this paper, we considered only closed systems, *i.e.* without inputs. However, important classes of programs that we want to analyze, *e.g.* digital filters, have inputs. Hence, we are extending our methods to loops of the form $G\vec{x} + H\vec{\xi} \leq 0 \rightarrow \vec{x}' = A\vec{x} + B\vec{\xi}$ with inputs $\xi$ (similar to [39]). Moreover, our method easily generalizes to linear continuous systems $d\vec{x}(t)/dt = A\vec{x}(t)$, *e.g.* representing the modes of a hybrid automaton, by considering their analytical solution $\vec{x}(t) = e^{At}\vec{x}(0)$.

**References**

[1] C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *SAS*, volume 6337 of *LNCS*, pages 117–133, 2010.

[2] C. Ancourt, F. Coelho, and F. Iriogoin. A modular static analysis approach to affine loop invariants detection. In *NSAD*, volume 267 of *ENTCS*, pages 3–16. Elsevier, 2010.

[3] S. Bardin, A. Finkel, J. Leroux, and L. Petrucci. Fast: acceleration from theory to practice. *STTT*, 10(5):401–424, 2008.

[4] B. Blanchet, P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, D. Monniaux, and X. Rival. A static analyzer for large safety-critical software. In *PLDI*, pages 196–207. ACM, 2003.

[5] B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *CAV*, volume 1102 of *LNCS*, July 1996.

[6] L. Chen, A. Miné, J. Wang, and P. Cousot. An abstract domain to discover interval linear equalities. In *VMCAI*, volume 5944 of *LNCS*, 2010.

[7] M. Colón, S. Sankaranarayanan, and H. Sipma. Linear invariant generation using non-linear constraint solving. In *Computer Aided Verification, CAV’03*, volume 2725, 2003.

[8] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Symposium on Principles of programming languages, POPL’78*, pages 84–96, 1978.

[9] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among the variables of a program. In *POPL*, pages 84–97, 1978.

[10] P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Miné, and X. Rival. Why does Astére scale up? *FMSD*, 35(3), 2009.

[11] P. Feautrier and L. Gonnord. Accelerated invariant generation for C programs with Aspic and C2ism. *ENTCS*, 267(2):3–13, 2010.

[12] J. Feret. Static analysis of digital filters. In *ESOP*, volume 2986 of *LNCS*, pages 33–48, 2004.

[13] J. Feret. Numerical abstract domains for digital filters. In *Numerical and Symbolic Abstract Domains*, 2005.

[14] A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. In *FSTTCS*, volume 2556 of *LNCS*, pages 145–156, 2002.

[15] S. Gaubert, E. Goubault, A. Taly, and S. Zennou. Static analysis by policy iteration on relational domains. In *ESOP*, volume 4421 of *LNCS*, 2007.

[16] T. M. Gawlitza and H. Seidl. Precise relational invariants through strategy iteration. In *Computer Science Logic*, volume 4646 of *LNCS*, pages 23–40. Springer, 2007.

[17] T. M. Gawlitza, H. Seidl, A. Adjé, S. Gaubert, and E. Goubault. Abstract interpretation meets convex optimization. *Journal of Symbolic Computation*, 47(12):1512–1532, 2012.

[18] L. Gonnord and N. Halbwachs. Combining widening and acceleration in linear relation analysis. In *SAS*, volume 4218 of *LNCS*, 2006.

[19] A. Gupta and A. Rybalchenko. InvGen: an efficient invariant generator. In *CAV*, volume 5643 of *LNCS*, pages 634–640, 2009.

[20] J. M. Howe and A. King. Logahedra: A new weakly relational domain. In *Automated Technology for Verification and Analysis, ATVA’09*, volume 5790 of *LNCS*, pages 306–320. Springer, 2009.

[21] B. Jeannet and A. Miné. APRON: A library of numerical abstract domains for static analysis. In *CAV*, volume 5643 of *LNCS*, pages 661–667, 2009, [http://apron.cri.ensmp.fr/library/](http://apron.cri.ensmp.fr/library/).

[22] B. Jeannet, M. Argoud, and G. Laliré. The Interproc interprocedural analyzer. [http://pop-art.inrialpes.fr/interproc/interprocew.cgi](http://pop-art.inrialpes.fr/interproc/interprocew.cgi).

[23] B. Jeannet, P. Schrammel, and S. Sankaranarayanan. Abstract acceleration of general linear loops. *CoRR*, abs/1311.768, 2013.

[24] J. B. Kam and J. D. Ullman. Monotone data flow analysis frameworks. *Acta Informatica*, 7:305–317, 1977.

[25] J. Knoop, L. Kovács, and J. Zwicknagel. Symbolic loop bound computation for wct analysis. In *Perspectives of Systems Informatics*, volume 7162 of *LNCS*, pages 227–242. Springer, 2011.

[26] L. Kovács. Invariant generation for p-solvable loops with assignments. In *CSR*, volume 5010 of *LNCS*, pages 349–359, 2008.

[27] G. Lafferriere, G. J. Pappas, and S. Yovine. Symbolic reachability computation for families of linear vector fields. *JSC*, 32(3):231–253, 2001.

[28] P. Lancaster and M. Tismenetsky. *The Theory of Matrices (2nd edition)*. Academic Press, 1984.

[29] A. Miné. The octagon abstract domain. In *AST 2001 in WCRE 2001*, IEEE, pages 310–319. IEEE CS Press, October 2001.

[30] A. Miné. Symbolic methods to enhance the precision of numerical abstract domains. In *VMCAI*, volume 3855 of *LNCS*, pages 348–363, 2006.

[31] D. Monniaux. Compositional analysis of floating-point linear numerical filters. In *CAV*, volume 3576 of *LNCS*, pages 199–212, 2005.

[32] D. Monniaux. Automatic modular abstractions for linear constraints. In *POPL*, ACM, 2009.

[33] J. Rohn. Solvability of systems of interval linear equations and inequalities. In *Linear Optimization Problems with Inexact Data*, pages 35–77, 2006.

[34] P. Roux, R. Jobredeaux, P.-L. Garoche, and E. Feron. A generic ellipsoid abstract domain for linear time invariant systems. In *HSCC*, pages 105–114. ACM, 2012.

[35] S. Sankaranarayanan and A. Tiwari. Relational abstractions for continuous and hybrid systems. In *CAV*, volume 6806 of *LNCS*, pages 686–702. Springer, 2011.

[36] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constraint-based linear-relations analysis. In *SAS*, volume 3148 of *LNCS*, pages 53–68, 2004.

[37] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *VMCAI*, volume 3385 of *LNCS*, 2005.

[38] P. Schrammel and B. Jeannet. Logico-numerical abstract acceleration and application to the verification of data-flow programs. In *SAS*, volume 6887 of *LNCS*, pages 233–248, 2011.

[39] P. Schrammel and B. Jeannet. Applying abstract acceleration to (co)-reachability analysis of reactive programs. *Journal of Symbolic Computation*, 47(12):1512–1532, 2012.

[40] A. Simon and A. King. Widening polyhedra with landmarks. In *Prog. Languages and Systems, APLAS’06*, volume 4279 of *LNCS*, pages 427–442. Springer, 2006.

[41] A. Tiwari. Approximate reachability for linear systems. In *HSCC*, volume 2623 of *LNCS*, pages 514–525. Springer, 2003.

[42] H. Yazarel and G. J. Pappas. Geometric programming relaxations for linear system reachability. In *American Control Conference*, pages 553–559, 2004.