# Toward a Sound Analysis of Guarded LTI Loops with Inputs by Abstract Acceleration (extended version)

Colas Le Guernic



# Toward a Sound Analysis of Guarded LTI Loops with Inputs by Abstract Acceleration (extended version)

Colas Le Guernic[1,2]

[1] DGA Maîtrise de l'Information, France
[2] Inria Rennes - Bretagne Atlantique, France

**Abstract.** In a POPL 2014 paper, Jeannet et al. showed that abstract acceleration is a relevant approach for general linear loops thanks to the Jordan decomposition of the linear transformer. Bounding the number of loop iterations involves interval-linear constraints. After identifying sources of over-approximation, we present some improvements over their method. First, we improve precision by using interval hulls in the Jordan parameters space instead of the state space, avoiding further interval arithmetic. Then, we show how to use conic hulls instead of interval hulls to further improve precision.
Furthermore, we extend their work to handle linear loops with bounded nondeterministic input. This was already attempted by Cattaruzza et al. in a SAS 2015 paper, unfortunately their method is unsound. After explaining why, we propose a sound approach to guarded LTI loops with bounded nondeterministic inputs by reduction to the autonomous case.

## 1 Introduction

Finding bounds on the values taken by variables is an essential step in program, and model, verification. Difficulties arise in the presence of loops. We are here specifically interested in loops whose body consists of a linear transformation on the program variables, with a linear exit condition. Such loops are pervasive in cyber-physical models as well as embedded codes.

If the number of steps is bounded, linear loops can be analyzed with iterative methods [12, 14] or bounded model checking [4]. When no bound is known or small enough, invariants may be derived through abstract interpretation [7], with abstract domains tailored to some of the emergent nonlinear relations [8, 17], or barrier certificates [16].

An alternative approach, abstract acceleration [9, 10], aims at replacing the loop by a single abstract transformer. Jeannet et al. [11] proved the approach tractable for general linear loops. Cattaruzza et al. [5, 6] tried to extend their result to general linear loops with bounded inputs, unfortunately their analysis is not clearly stated and based on unsound assumptions.

We describe our problem in more details and introduce some notations in section 2. We then express, in section 3, Jeannet et al.'s approach [11] in our

setting. This is more than just a reminder, in particular we clearly identify sources of over-approximation, and is necessary for the clarity of the rest of the paper. Our contributions to the abstract acceleration of loops without inputs are presented in section 4 and experimentally evaluated in section 5. Inputs are considered in section 6 in which we first demonstrate the unsoundness of Cattaruzza et al. [5] approach before presenting our solution. Finally we discuss the impact of floating point computations in section 7.

## 2 Preliminaries

We are interested in the approximation of invariants for linear time-invariant (LTI) loops over $\mathbb{R}$ of the form:

$$\begin{aligned} &\text{assume}(x \in X_0) \\ &\text{while}(Gx \leq h) \quad x := Ax + B \cdot \text{Get}(U) + c \end{aligned}$$

where $G$, $A$, and $B$ are matrices, $c$ and $h$ constant vectors, $X_0$ the initial set, and $\text{Get}(U)$ nondeterministically returns a fresh vector from the set $U$ at each loop iteration.

Without loss of generality we can restrict ourselves to (see Appendix 1):

$$\text{while}(Gx \leq 0) \quad x := Ax + \text{Get}(U) \qquad \text{with } 0 \in U \ . \tag{1}$$

Noting $\tau$ the effect of one loop iteration, abstract acceleration aims at finding a sound approximation of $\tau^*$: $X_0 \mapsto \bigcup_{i=0}^{\infty} \tau^i(X_0)$.

We call Equation (1) the loop representation of the problem. It can also be stated with a sequential representation:

$$X_{n+1} = A(X_n \cap \mathcal{G}) \oplus U \tag{2}$$

where $\mathcal{G}$ is the set $\{x \mid Gx \leq 0\}$, and $\oplus$ is the Minkowski sum: the sum of two sets is the set of the sums of elements of each set. Given an initial set $X_0$, we want to over-approximate $\mathcal{X} = \bigcup_{i=0}^{\infty} X_i$. Throughout this paper we will mainly use this last representation.

For the reader's sake, we now list the notations used throughout this paper:

- $A$ is the matrix of the linear transformation performed in the loop body;
- $\mathcal{A}$ is the set $\{A^k \mid k \in \mathbb{N}\}$, and $\mathcal{A}_n = \{A^k \mid k \in \mathbb{N},\, k < n\}$;
- $\mathcal{B}$ denotes a box, or a product of intervals;
- $d$ is the dimension of the system, $x \in \mathbb{R}^d$;
- $d' \leq d$ is the degree of the minimal polynomial of $A$;
- $\text{Vect}\,(\mathcal{A})$ is the vector space generated by $\mathcal{A}$;
- for a given basis $(M_0, \ldots, M_{d'-1})$ of $\text{Vect}\,(\mathcal{A})$, $m(n)$ is the vector describing $A^n$ in that basis, $\mathcal{M}$ is the set $\left\{m \mid \exists k \in \mathbb{N},\, \sum_{i=0}^{d'-1} m_i M_i = A^k\right\}$, and $\mathcal{M}_n$ is $\left\{m \mid \exists k \in \mathbb{N},\, k < n,\, \sum_{i=0}^{d'-1} m_i M_i = A^k\right\}$;

- $\mathcal{X} = \bigcup_{k=0}^{\infty} X_k$ and $\mathcal{X}_n = \bigcup_{k=0}^{n-1} X_k$;
- $N$ is the smallest index, if it exists, such that $\mathcal{X}_N = \mathcal{X}$;
- $\Pi_i$ and $\Pi_{i,j}$ are orthogonal projections over the line generated by component $i$ and the plane generated by components $i$ and $j$ respectively.
- for a given matrix $A$, $|A|$ is the matrix obtained by taking the absolute value of each component of $A$.
- for a given set $S$, $\overline{S}$ is an over-approximation of $S$, $\square(S)$ is its interval hull, $\boxdot(S) = \square(S \cup -S)$ is its centrally symmetric interval hull, and $\square_T(S)$ is the over-approximation of $S$ by a polyhedron with template $T$.

## 3 Abstract Acceleration of LTI Systems without Inputs

In this section we are interested in loops of the form: while$(Gx \leq 0)\, x := Ax$.

The next two subsections summarize the results of Jeannet et al. [11] using the sequential representation of the problem. For the sake of clarity we will not systematically cite their paper. In section 3.3 we identify independent sources of over-approximations inherent to the method.

### 3.1 Linear Systems without Guards

Without guards, the program becomes while$(true)\, x := Ax$, and generates the sequence $X_{n+1} = AX_n$. Then: $\mathcal{X} = \bigcup_{i=0}^{\infty} A^i X_0$.

Considering $\mathcal{A} = \{A^i \mid i \in [0..\infty]\}$, one can compute $\mathcal{X}$ by applying $\mathcal{A}$ on $X_0$ element-wise: $\mathcal{X} = \mathcal{A}X_0 = \{Mx \mid M \in \mathcal{A},\, x \in X_0\}$.

In order to render this representation effectively useful, Jeannet et al. [11] proceeds in three steps: express $A^n$ as a nonlinear function of $n$; use this symbolic expression to tightly over-approximate $\mathcal{A}$ with a logahedron (a certain type of polyhedron) $\overline{\mathcal{A}}$; tightly over-approximate $\overline{\mathcal{A}}X_0$ with a polyhedron.

**Symbolic Expression for $\mathcal{A}$** First let us remark that, following Cayley-Hamilton theorem, $\mathcal{A}$ lies in a subspace of $\mathbb{R}^{d\times d}$ of dimension $d' \leq d$. Thus, for a given basis $M_0, \ldots, M_{d'-1}$ of this subspace, and for any $n \in \mathbb{N}$, there exists a unique vector $m(n)$ such that:

$$A^n = \sum_{i=0}^{d'-1} m_i(n) M_i, \quad \text{and} \quad \mathcal{A} = \left\{ \sum_{i=0}^{d'-1} m_i M_i \mid m \in \mathcal{M} \right\}$$

where $\mathcal{M} = \{m(n) \mid n \in \mathbb{N}\}$.

In order to find a suitable basis, with an easy to represent and approximate $\mathcal{M}$, Jeannet et al. [11] suggest to use the Jordan decomposition of $A$: $PJP^{-1}$ (see Appendix 2 to get the intuition on a simple case or [11] for the full expression).

**Tight Over-Approximation of $\mathcal{M}$.** Since $\mathcal{A}$ is the image of $\mathcal{M}$ by a linear transformation, one can obtain a polyhedral over-approximation $\overline{\mathcal{A}}$ of $\mathcal{A}$ from a polyhedral over-approximation $\overline{\mathcal{M}}$ of $\mathcal{M}$:

$$\overline{\mathcal{A}} = \left\{ \sum_{i=0}^{d'-1} m_i M_i \mid m \in \overline{\mathcal{M}} \right\} \quad .$$

Since each of the components of $m$ are nonlinear, computing precisely supporting hyperplanes in arbitrary directions is hard. Jeannet et al. [11] restricts constraints to linear combinations of (almost) any two components and provides a way to compute the corresponding supporting hyperplane, leading to a logahedral approximation of $\mathcal{M}$.

**Applying a *Set* of Linear Transformations.** Jeannet et al. [11] suggest two approaches, but the one they recommend involves expressing $\mathcal{M}$ and $X_0$ with vertices (and rays) and multiplying them pairwise, leading to the best convex approximation of the result. They acknowledge that the exponential complexity in the dimension starts to show at dimension 8.

### 3.2 Linear Systems with Guards

We are now interested in the sequence: $X_{n+1} = A(X_n \cap \mathcal{G})$. Using the closed form of $X_n$ we can deduce that:

$$\mathcal{X} = \bigcup_{n=0}^{\infty} A^n \left( X_0 \cap \bigcap_{i=0}^{n-1} \left\{ x \mid GA^i x \leq 0 \right\} \right) \quad .$$

**Reduction to the Unguarded Case.** In order to avoid this alternation of unions and intersections Jeannet et al. [11] over-approximate $\mathcal{X}$ with:

$$X_0 \cup A\left(\mathcal{A}\left(X_0 \cap \mathcal{G}\right) \cap \mathcal{G}\right)$$

which is equivalent to applying the unguarded acceleration to $X_0 \cap \mathcal{G}$ before applying the loop transformer, $\tau$, once. In order to improve over this approximation Jeannet et al. [11] proposes to search for the first $N$ such that $X_N \cap \mathcal{G} = \emptyset$. Then:

$$\mathcal{X} \subseteq X_0 \cup A\left(\mathcal{A}_N\left(X_0 \cap \mathcal{G}\right) \cap \mathcal{G}\right) \quad .$$

**Bounding the Number of Steps.** We want to find the smallest $n$ such that $X_n \cap \mathcal{G}$ is empty, which is equivalent fo finding the smallest $n$ such that:

$$\emptyset = X_0 \cap \bigcap_{i=0}^{n} \left\{ x \mid GA^i x \leq 0 \right\} \quad .$$

Again, this might be hard to compute directly, instead Jeannet et al. [11] look for the smallest $n$ such that $A^n(X_0 \cap \mathcal{G}) \cap \mathcal{G}$ is empty.

*Moving to Vect*$(\mathcal{A})$. In order to do so, they express the problem in the vector space generated by $\mathcal{A}$:

$$\left(\sum_{i=0}^{d'-1} m_i(n) M_i\right) (X_0 \cap \mathcal{G}) \cap \mathcal{G} = \emptyset \iff m(n) \notin \mathcal{K}$$

where $\mathcal{K}$ is the set of $m \in \mathcal{M}$ such that the intersection is not empty:

$$\mathcal{K} = \mathcal{M} \cap \left\{ m \mid \exists x \in X_0 \cap \mathcal{G},\ \sum_{i=0}^{d'-1} m_i G M_i x \leq 0 \right\} .$$

The intersection with $\mathcal{M}$ is not necessary here but will be useful to constrain the over-approximations of $\mathcal{K}$. The other part of the definition of $\mathcal{K}$ is simply the set of parameters $m$ such that the image by the corresponding linear transformation, $\sum_{i=0}^{d'-1} m_i M_i$, of at least one point of $X_0 \cap \mathcal{G}$ lies in $\mathcal{G}$.

We are now looking for the smallest $n$ such that $m(n) \notin \mathcal{K}$. First they over-approximate $\mathcal{K}$ with a simpler convex set, then they look for a separating hyperplane.

*Approximating* $\mathcal{K}$. The first step consists in replacing the bilinear constraints with *interval-linear* constraints by substituting $X_0 \cap \mathcal{G}$ with its interval hull $\square (X_0 \cap \mathcal{G})$[3]:

$$\mathcal{K} \subseteq \mathcal{M} \cap \left\{ m \mid \sum_{i=0}^{d'-1} m_i G M_i \square (X_0 \cap \mathcal{G}) \leq 0 \right\} .$$

Then, linearization techniques exploiting the template approximation of $\mathcal{M}$ are applied to obtain a convex polyhedron:

$$\overline{\mathcal{K}} = \overline{\square_T (\mathcal{M}) \cap \left\{ m \mid \sum_{i=0}^{d'-1} m_i G M_i \square (X_0 \cap \mathcal{G}) \leq 0 \right\}} .$$

*Approxmating* $N$. Since $\overline{\mathcal{K}}$ is convex, $m(n)$ leaves $\overline{\mathcal{K}}$ as soon as one of its constraints is violated. Thus for each constraint $gx \leq h$ of $\overline{\mathcal{K}}$, we are looking for the smallest positive integer $n$ such that $g \cdot m(n) > h$. Unfortunately this expression is nonlinear, and finding the smallest $n$ for arbitrary $g$ might be costly. Instead, Jeannet et al. [11] restricts the set of constraints to linear combinations of two components as in section 3.1 by over-approximating $\overline{\mathcal{K}}$ with its template polyhedron $\square_T \left(\overline{\mathcal{K}}\right)$. See [11] for technical details on how each minimization is performed.

[3] To be more precise, Jeannet et al. [11] substitute $P^{-1}(X_0 \cap \mathcal{G})$ by its interval hull in $PJ_iP^{-1}(X_0 \cap \mathcal{G})$, where $P$ is the invertible matrix leading to the Jordan form of $A = PJP^{-1}$. Use of this transformation is not justified and its advantage is not clear.

### 3.3 Recap: Sources of Over-Approximation.

We are interested in a conservative approximation of $\bigcup_{i=0}^{\infty} X_i$. This is done in two steps: first a bound $N$ on the smallest $n$ such that $X_{n+1}$ is empty is computed, then $\bigcup_{i=0}^{N} X_i$ is over-approximated. None of these steps can be done exactly in a reasonable time, thus several approximations are performed to render the problem practical.

**Bounding the number of steps:**

**Ignoring the guard:** Instead of looking for the smallest $n$ such that $X_0$ and $\bigcap_{i=0}^{n} \left\{x \mid GA^i x \leq 0\right\}$ have an empty intersection, we look for the smallest $n$ such that $X_0 \cap \mathcal{G} \cap \{x \mid GA^n x \leq 0\}$ is empty, ignoring the influence of $\bigcap_{i=1}^{n-1} \left\{x \mid GA^i x \leq 0\right\}$.

**Bounding $X_0 \cap \mathcal{G}$:** The problem is then further simplified to transform bilinear constraints in $\mathcal{K}$ into interval-linear constraints. We look for the smallest $n$ such that $\square\left(X_0 \cap \mathcal{G}\right) \cap \{x \mid GA^n x \leq 0\}$ is empty. This approximation is propagated and amplified by the use of interval arithmetic to approximate $\mathcal{K}$ with a set of interval-linear constraints.

**Linearization:** $\mathcal{K}$ is then further approximated to replace interval-linear constraints with linear constraints.

**Bounding $\overline{\mathcal{K}}$:** Finally, $\overline{\mathcal{K}}$ is tightly over-approximated with a template polyhedron (logahedron) so that minimizing $n$ such that any of its constraint is violated becomes tractable.

We do not consider the over-approximation of $\mathcal{M}$ in the computation of $\mathcal{K}$ to be a source of error. Indeed, $\mathcal{M}$ here is only used to limit the error produced by the last two steps. If they did not produce errors, the quality of the approximation of $\mathcal{M}$ would have no incidence on the computed bound on the number of steps.

At last the minimization procedure itself may produce an over-approximation: for each constraint of $\square_T\left(\overline{\mathcal{K}}\right)$ it is not guaranteed to return a finite value, but if it does it is the minimum integer $n$ such that $m(n)$ violate that constraint.

**Approximating $\bigcup_{n=0}^{N} X_n$:**

**Ignoring the guard:** Again, the first step is to ignore some of the influence of the guard: $\bigcup_{n=0}^{N} X_n = \bigcup_{n=0}^{N} A^n \left(X_0 \cap \bigcap_{i=0}^{n-1} \left\{x \mid GA^i x \leq 0\right\}\right)$ is over-approximated with $X_0 \cup \bigcup_{n=1}^{N} A^n \left(X_0 \cap \mathcal{G} \cap \left\{x \mid GA^{n-1} x \leq 0\right\}\right)$ expressed as $X_0 \cup A\left(\mathcal{A}_N\left(X_0 \cap \mathcal{G}\right) \cap \mathcal{G}\right)$.

**Bounding $\mathcal{A}_N$:** $\mathcal{A}_N$ is over-approximated with a logahedron: a template polyhedron whose constraints only involve two components at most. This over-approximation is tight, meaning that each face of $\square_T\left(\mathcal{A}_N\right)$ touches $\mathcal{A}_N$. The only room for improvement here lies in the choice of the directions of approximation and the ability to find tight bounds in arbitrary directions.

**Bounding $\square_T\left(\mathcal{A}_N\right)\left(X_0 \cap \mathcal{G}\right) \cap \mathcal{G}$:** This operation is already quite precise, indeed $\square_T\left(\mathcal{A}_N\right)\left(X_0 \cap \mathcal{G}\right)$ is over-approximated by its convex hull. Yet, the quality of the approximation with respect to $\mathcal{A}_N\left(X_0 \cap \mathcal{G}\right) \cap \mathcal{G}$ is hard to evaluate.

Concerning complexity, the most costly operations are:

- The symbolic computation of the Jordan form of $A$.
- The product $\square_T(\mathcal{A}_N)(X_0 \cap \mathcal{G})$ done by computing the pairwise products of the vertices and rays generating $\square_T(\mathcal{A}_N)$ and $(X_0 \cap \mathcal{G})$.

## 4 Contributions to Abstract Acceleration of Linear Systems without Inputs

The previous section offered an original presentation of the results of Jeannet et al. [11]. Moreover, we highlighted the different sources of over-approximation. In the current section we present our own contributions, focusing on the approximation of $\mathcal{K}$, the set used to bound the maximum number of iterations.

When the number of steps is finite, correctly bounding it can provide a great improvement on precision. This bound is computed by finding the smallest $n$ such that $m(n) \notin \mathcal{K}$. Let us recall the expression for $\mathcal{K}$:

$$\mathcal{K} = \mathcal{M} \cap \left\{ m \mid \exists x \in X_0 \cap \mathcal{G},\ \sum_{i=0}^{d'-1} m_i G M_i x \leq 0 \right\} \quad .$$

For the sake of presentation we will consider here that $G$ is composed of only one constraint $g$. When $G$ has multiple rows, each row can be treated independently, similarly to what is implicitly[4] done by Jeannet et al. [11].

*Remark 1.* Independent treatment of each constraint leads to a first over-approximation, indeed the set $\{m \mid \exists x \in X,\ f(g_0, m, x) \leq 0 \wedge f(g_1, m, x) \leq 0\}$ may be smaller than the intersection of $\{m \mid \exists x \in X,\ f(g_0, m, x) \leq 0\}$ and $\{m \mid \exists x \in X,\ f(g_1, m, x) \leq 0\}$. In the first case, both constraints must be verified for the same $x$, in the second case, one can choose two different $x$. It can be partially overcome by considering linear combinations of constraints.

By noting $L$ the matrix whose rows are the $gM_i$ we can express $\mathcal{K}$ as:

$$\mathcal{K} = \mathcal{M} \cap \{m \mid \exists x \in X \cap \mathcal{G},\ m \cdot Lx \leq 0\} \quad .$$

In the following subsections, we make the assumption that there is no $x$ in $X_0 \cap \mathcal{G}$ such that $Lx = 0$, if there was, $\mathcal{K}$ would be equal to $\mathcal{M}$, and the number of steps would be unbounded.

### 4.1 Avoiding Interval Arithmetic.

As presented earlier, the first step of the approximation of $\mathcal{K}$ is to replace bilinear with interval-linear constraints by first over-approximating $X \cap \mathcal{G}$ with its interval

[4] When approximating $\mathcal{K}$ with interval linear constraints the relation between the constraints parameters are lost.

hull and then propagate those intervals by interval arithmetic:

$$\begin{aligned}\mathcal{K} &\subseteq \mathcal{M} \cap \{m \mid \exists x \in \square\,(X \cap \mathcal{G})\,,\, m \cdot Lx \leq 0\} \\ &\subseteq \mathcal{M} \cap \{m \mid m \cdot L\square\,(X \cap \mathcal{G}) \leq 0\}\end{aligned}$$

leading to a superset[5] of:

$$\mathcal{M} \cap \{m \mid m \cdot \square\,(L\square\,(X \cap \mathcal{G})) \leq 0\} \quad .$$

Expressing $\sum_{i=0}^{d'-1} m_i G M_i x$ as $m \cdot Lx$ makes it clear that interval arithmetic can be avoided. Thus we suggest to use the interval hull of $L(X \cap \mathcal{G})$ directly:

$$\mathcal{K} \subseteq \mathcal{M} \cap \{m \mid \exists \ell \in \square\,(L(X \cap \mathcal{G}))\,,\, m \cdot \ell \leq 0\}$$

leading to a more precise approximation. $L(X \cap \mathcal{G})$ does not need to be computed explicitly, $\square\,(L(X \cap \mathcal{G}))$ can be computed directly by optimizing linear functions on $X \cap \mathcal{G}$ in the directions given by $L^\top$. The procedure is efficient even if $L$ is not invertible.

*Example 1.* Starting from any point in the convex hull of $\{(1,0);(0,1)\}$, consider the loop: while$(x + y \leq 10)\{x := 2x;\ y := 2y\}$. Jeannet et al. [11] can not find any bound on the number of steps while our approach finds the correct one (see Appendix 3.1 for details).

The improvement is not always that dramatic but the resulting bound is always at least as good as the one given by Jeannet et al. [11]. Indeed, their interval linear constraints are based on a superset of $\square\,(L\square\,(X \cap \mathcal{G}))$ while ours is based on $\square\,(L(X \cap \mathcal{G}))$ directly.

Furthermore, our method does not introduce any overhead in terms of time complexity. Indeed computing the interval hull of $L(X \cap \mathcal{G})$ or $X \cap \mathcal{G}$ both involve maximizing $2d'$ or $2d$ respectively linear functions over $X \cap \mathcal{G}$ as discussed earlier.

### 4.2 Avoiding Interval Hull.

We are here again interested in the first step of the approximation of $\mathcal{K}$, replacing bilinear with interval-linear constraints.

In the previous section we were interested in what to get a product of intervals from, we got: $\mathcal{K} \subseteq \mathcal{M} \cap \{m \mid \exists \ell \in \square\,(L(X \cap \mathcal{G}))\,,\, m \cdot \ell \leq 0\}$.

In this section we are interested in how to get a suitable product of intervals $\mathcal{B}$ such that: $\mathcal{K} \subseteq \mathcal{M} \cap \{m \mid \exists \ell \in \mathcal{B},\, m \cdot \ell \leq 0\}$.

Intuitively the best way to get such a product of intervals is to take the interval hull: $\mathcal{B} = \square\,(L(X \cap \mathcal{G}))$. We will show that it is possible to use another box leading to a better approximation of $\mathcal{K}$. Let us first remark that:

$$m \cdot \ell \leq 0 \iff \exists \alpha \in \mathbb{R}^+,\, \alpha \neq 0 \wedge m \cdot (\alpha \ell) \leq 0 \quad .$$

[5] Both sets would be equal if interval arithmetic did not amplify and propagate any approximation.

Thus, if $0 \notin L(X_0 \cap \mathcal{G})$:

$$\mathcal{K} = \mathcal{M} \cap \{m \mid \exists \ell \in \angle\left(L(X_0 \cap \mathcal{G})\right), \ell \neq 0 \wedge m \cdot \ell \leq 0\}$$

where $\angle(X) = \{\alpha\ell \mid \alpha \in \mathbb{R}^+, \ell \in X\}$ is the conic hull of $X$.

We will not use this conic representation directly, instead we will look for a product of intervals that generates a cone containing $\angle(X_0 \cap \mathcal{G})$, but not bigger (and hopefully smaller) than the cone generated by $\square(X_0 \cap \mathcal{G})$, leading to a better approximation of $\mathcal{K}$. Ideally, we would like to find a product of intervals generating the smallest possible cone. Unfortunately such a box does not necessarily exist. Instead, we will show how to compute a suitable subset of $\square(X_0 \cap \mathcal{G})$ and optimal degenerate boxes. First, let us characterize conic hulls of products of closed intervals.

**Theorem 1.** *For any product of closed intervals, its conic hull $\angle(\mathcal{B})$ is entirely determined by its projection on canonical planes.*

*Proof.* See Appendix 4.

Algorithm 1 exploits this characterization to compute a box $\mathcal{B}$ included in $\square(L(X_0 \cap \mathcal{G}))$ such that $\angle(L(X_0 \cap \mathcal{G}))$ is included in $\angle(\mathcal{B})$. The problem is projected on each of the $d(d-1)/2$ canonical planes, then solved by GET_PAIR which returns two points with minimal coordinates along the two half-lines (that may be colinear) delimiting the cone and on the frontier of the interval hull. The coordinates of those points are then used to update the full dimensional solution.

**Algorithm 1** Interval hull subset.

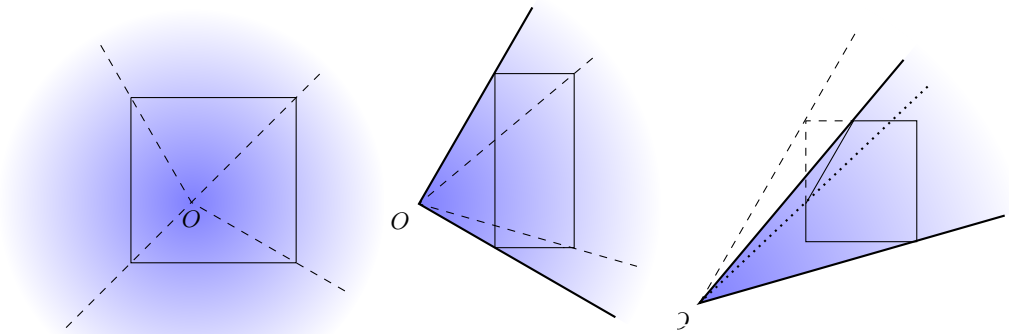
**Input:** A nonempty set $X$ in dimension $d$ such that $0 \notin \square(X)$.
**Output:** A box $B$ such that $\angle(X) \subseteq \angle(B)$ and $B \subseteq \square(X)$.

```
S ← ∅^d
for i = 0 to d − 2 do
    for j = i + 1 to d − 1 do
        (x, y) ← GET_PAIR(X, i, j)
        S_i ← □(S_i ∪ {x_0} ∪ {y_0})
        S_j ← □(S_j ∪ {x_1} ∪ {y_1})
    end for
end for
return S
```

There are some subtleties in GET_PAIR. Applied on fig. 1b it returns the coordinates of the bottom left and top left vertices of the box. But in situations similar to fig. 1a, there are no half-lines delimiting the cone. Any pair of point whose interval hull is $[-\epsilon;\epsilon] \times [-\epsilon;\epsilon]$ for $\epsilon$ sufficiently small is suitable, but $[0;0] \times [0;0]$ does not span the whole plane. Still, GET_PAIR can safely return a pair of points generating $[0;0] \times [0;0]$. It will conflict with the output requirement of algorithm 1, but not with the underlying goal of over-approximating $\mathcal{K}$. If

we denote by $\mathcal{B}_\epsilon$ the box returned using a conservative GET_PAIR, and $\mathcal{K}_\epsilon$ the set: $\mathcal{M} \cap \{m \mid \exists \ell \in \mathcal{B}_\epsilon,\ m \cdot \ell \leq 0\}$, then: $\mathcal{K} \subseteq \bigcap_{\epsilon>0} \mathcal{K}_\epsilon \subseteq \mathcal{K}_0$. Indeed, since $\mathcal{B}_0$ is closed, for any $m$ not in $\mathcal{K}$, there exists $\mu > 0$ such that for all $\ell \in \mathcal{B}_0$, $m \cdot \ell \geq \mu$. Thus there exists $\epsilon > 0$ such that $m$ does not belong to $\mathcal{K}_\epsilon$, which implies that $\bigcap_{\epsilon>0} \mathcal{K}_\epsilon \subseteq \mathcal{K}_0$. Similar arguments allow to return the same value when the origin is on the frontier of the interval hull.

Moreover, if a delimiting half-line does not intersect the interval hull, then GET_PAIR can safely return a point with one infinite coordinate.




(a) No faces. (b) Two faces. (c) Iterative refinement.

Fig. 1: Projection of a product of intervals and its conic hull on a plane.

Another subtlety, concerns the projection of $X$ on canonical planes. It can be costly and unnecessary. Instead, GET_PAIR takes as arguments $X$ and the indices of the canonical vectors to project on. Starting from the conic hull of the interval hull of $X$, an approximation of the exact two-dimensional conic hull is iteratively refined by optimizing linear functions on the full dimensional $X$ as illustrated in fig. 1c.

*Example 2.* Starting from any point in the convex hull of $\{(2,1);(6,3)\}$, consider the loop: while$(-2x + y \leq 0)\{x := 0.9x;\ y := y\}$. Using the interval hull leads to a bound of 23 steps, while algorithm 1 improves this bound to 13 (see Appendix 3.2 for details).

Unfortunately, algorithm 1 needs $\square\,(L(X_0 \cap \mathcal{G}))$ to be full dimensional in order to lead to any improvement and this is not always the case, especially for affine systems. Instead of finding a box included in $\square\,(L(X_0 \cap \mathcal{G}))$, algorithm 2 computes an optimal flat box whose conic hull contains $L(X_0 \cap \mathcal{G})$.

Similarly to algorithm 1, algorithm 2 makes use of GET_PAIR, and then scales the returned points to belong to the hyperplane $\{x \mid x_i = 1\}$, or $\{x \mid x_i = -1\}$ depending on the orientation of the cone. Of course this is only applicable if for all points in $X$, the sign of $x_i$ is the same. This also implies that $x_0$ and $y_0$ can not be 0 on line 5 of algorithm 2. Moreover, if the components of the points returned

**Algorithm 2** Optimal flat box.

**Input:** A nonempty set $X$ in dimension $d$ and an index $i$ such that $0 \notin \Pi_i \square(X)$.
**Output:** The smallest box $B$ such that $\angle(X) \subseteq \angle(B)$ and, $B \subseteq \{x \mid x_i = 1\}$ or $B \subseteq \{x \mid x_i = -1\}$.

```
S ← ∅^d
S_i ← Sign(Π_i □(X))
for all j ≠ i do
    (x, y) ← Get_Pair(X, i, j)
    S_j ← □(S_j ∪ {x_1/|x_0|, y_1/|y_0|})
end for
return □(S)
```

by Get_Pair can be infinite, then the other component is necessarily finite and different from 0. Thus standard arithmetic on $\mathbb{R} \cup \{\pm\infty\}$ can be applied.

*Example 3.* Starting from any point in the convex hull of $\{(2,1);(6,3)\}$, consider the loop: while$(-2x+y-0.1 \leq 0)\{x := 0.9x;\ y := y+1\}$. Using the interval hull leads to a bound of 5 steps, algorithm 1 cannot help us because $\square(L(X_0 \cap \mathcal{G}))$ is flat. Using algorithm 2 optimizing the first component bounds the number of steps by 4. (see Appendix 3.3 for details).

One might want to apply algorithm 2 even if zero belongs to $\Pi_i \square(L(X_0 \cap \mathcal{G})$, as long as this set does not contain points of opposing signs. But then, the resulting $\overline{\mathcal{K}}$ is not guaranteed to be an over-approximation, and the resulting bound $n$ has to be checked by optimizing the linear function $x \mapsto -L^\top m(n) \cdot x$ on $X_0 \cap \mathcal{G}$ in order to verify that there is no $\ell$ in $L(X_0 \cap \mathcal{G})$ such that $m(n) \cdot \ell \leq 0$.

As an example, consider the convex hull of $(0,1)$ and $(2,-2)$. Its interval hull is $[0;2] \times [-2;1]$ and algorithm 1 would return $[0;0] \times [0;0]$, then $\overline{\mathcal{K}} = \mathcal{M}$ and no bound on the number of iterations can be found. If instead one applies an adapted version of algorithm 2 on the first component, the resulting box would be $[1;1] \times [-1;\infty]$, only the half-line $[0;0] \times ]0;\infty]$ is missing from the resulting conic hull.

Algorithms 1 and 2 can be factorized to share the calls to Get_Pair, resulting in a set of interval linear constraints. If one wants to perform only $d'-1$ calls to Get_Pair instead of $d'(d'-1)/2$, then we propose the following heuristic to choose the dimension $i$ on which to apply algorithm 2: first we filter out all indices such that 0 belongs to $\Pi_i \square(X)$, then we only keep all indices corresponding to parameters associated with the eigenvalue $\lambda$ of maximal norm, then choose one on the highest diagonal. The rational behind this heuristic is that at some point the associated parameter will grow faster than any other suitable one.[6]

*Remark 2.* Unless the system has been augmented with one nonnull dimension, it may happen that $0 \in \square(L(X_0 \cap \mathcal{G}))$ even if $0 \notin L(X_0 \cap \mathcal{G})$. Then the only

[6] We use a similar heuristic in the experimental section to decide if an interval linear constraints can lead to a bound.

suitable cone generated by a product of interval spans the whole state space and a change of variable is necessary. Since $0 \notin L(X_0 \cap \mathcal{G})$, there exist a separating hyperplane between 0 and $L(X_0 \cap \mathcal{G})$, and a change of variable $R$ such that $0 \notin \square\,(RL(X_0 \cap \mathcal{G}))$ and we can use:

$$\mathcal{K} \subseteq \mathcal{M} \cap R^{\top} \{m \mid \exists \ell \in \square\,(RL(X \cap \mathcal{G}))\,,\, m \cdot \ell \leq 0\} \quad .$$

Such a change of variable might also be useful to improve precision, in particular if $RL(X \cap \mathcal{G})$ is already a product of intervals.

### 4.3 Avoiding Interval-Linear Constraints.

These last two improvements, *avoiding interval arithmetic* and *avoiding interval hull*, tackle the approximation errors introduced in the step designated as *Bounding* $X_0 \cap \mathcal{G}$ in section 3.3 leading to interval linear constraints. The only requirement is that one can optimize linear functions over $X_0 \cap G$.

If $X_0$ is convex and $X_0 \cap G$ can be expressed as the Minkowski sum of a compact convex set $\mathcal{V}$ and the conic hull of a compact convex set $\mathcal{R}$, then it is relatively easy to show that for any $\alpha > 0$:

$$\angle\,(X_0 \cap G) = \angle\,(\textsc{ConvexHull}(\mathcal{V} \cup \alpha\mathcal{R})) \quad .$$

This can be used before applying algorithms 1 or 2. If additionally, $\mathcal{V}$ and $\mathcal{R}$ are, or can be over-approximated by, compact polyhedra with a low number of vertices, then one can avoid interval linear constraints altogether. In fact $\angle\,(L(X_0 \cap \mathcal{G}))$ can be directly represented as a sum of rays:

$$\angle\,(L(X_0 \cap \mathcal{G})) = \left\{ \sum_i \alpha_i r_i | \forall i,\, \alpha_i \geq 0 \right\} \quad .$$

Then for any $\ell = \sum_i \alpha_i r_i$ in $\angle\,(L(X_0 \cap \mathcal{G}))$:

$$m \cdot \ell \leq 0 \Leftrightarrow \sum_i \alpha_i m \cdot r_i \leq 0 \Longrightarrow \exists i,\, m \cdot r_i \leq 0 \quad .$$

Indeed, if all elements in the sum were positive, the sum would be positive. This only works because we are considering one constraint at a time, if the comparison were multidimensional component-wise comparison, not being smaller than 0 would not imply being bigger than 0.

Thus:

$$\mathcal{K} \subseteq \bigcup_i \mathcal{M} \cap \{m \mid m \cdot r_i \leq 0\} \quad .$$

The other direction is trivial since all $r_i$ belong to $\angle\,(L(X_0 \cap \mathcal{G}))$.

$\mathcal{K}$ can be exactly[7] expressed as a union of half-spaces (intersected with $\mathcal{M}$). In other words, $\mathcal{K}$ is the complement in $\mathcal{M}$ of a polyhedron[8].

[7] If there are several constraints, treating each constraint independently leads to an over-approximation.

[8] Almost the dual cone of $L(X_0 \cap \mathcal{G})$.

### 4.4 Iterative Improvement

As explained before, the number of iterations of the loop under consideration is the smallest $n$ such that $m(n)$ does not belong to $\mathcal{K}$. Since $m(n)$ belongs to $\mathcal{M}$ by definition, the presence of $\mathcal{M}$ in the expression of $\mathcal{K}$ is superfluous. Nevertheless it helps bound the successive approximations needed to get a manageable expression for $\overline{\mathcal{K}}$. Once a bound $N$ is found, one can restart the process (or continue with another constraints) expressing $\mathcal{K}$ as:

$$\mathcal{K} \subseteq \mathcal{M}_N \cap \{m \mid \exists \ell \in L(X \cap \mathcal{G}),\, m \cdot \ell \leq 0\} \quad .$$

Taking $\mathcal{M}_N$ instead of $\mathcal{M}$ will improve subsequent approximations, and may lead to a smaller bound on the number of iterations.

## 5 Experimental Evaluation

Before considering loops with bounded inputs, let us evaluate the improvements presented so far. We only here focus on the techniques introduced in sections 4.1 and 4.2 because they have minimal requirements on $X_0$ and focus on one specific step that can be easily compared with Jeannet et al. [11] approach: deriving an interval linear constraint in order to bound the number of loop iterations.

We randomly generated 100 linear loops in several dimensions (2, 5, 10, 15, 20). To avoid diagonal systems we generated directly the Jordan form from a random partition of $d$. Then the guard and four different initial sets (balls for the 1, 2, and $\infty$ norms as well as a half-line, leading to an unbounded input set) were chosen such that all trajectories leave the guard after at most 32 steps, resulting in a total of 2000 instances.

We compared the performances of our own implementation of Jeannet et al. [11] algorithm with the improvement introduced in section 4.1, a version of algorithm 1 that returns a subset of the interval hull of $L(X_0 \cap \mathcal{G})$ and all possible flat boxes by factorizing calls to Get_Pair at no visible additional cost as suggested at the end of section 4.2, and a version of Algorithm 2 that returns the interval hull of $L(X_0 \cap \mathcal{G})$ and a box flat in the direction associated with the asymptotically bigger component as described at the end of section 4.2.

The quality of the computed interval linear constraint is assessed by computing the first $n$ such that $m(n)$ violates it, iteratively using exact arithmetic[9], and taking the difference with the exact bound. At some point, some components of $m(n)$ grows faster than all others and one may prove that the constraint will never be violated.

Results are shown in fig. 2: algorithms 1 and 2 can bound exactly more than one fifth of our instances, twice as many as Jeannet et al., and one third with an error smaller than 8 steps. Moreover, for instances that are bounded but not exactly, the error is a few orders of magnitude smaller, notice the log-scale for

[9] This is not how a bound $N$ should be computed, and is only useful to asses the quality of the interval linear constraint.

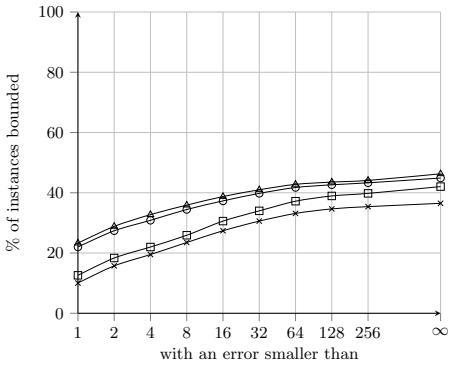



Fig. 2: Cumulative frequency of instances bounded by interval linear constraint(s) generated by: (×) Jeannet et al. [11], (□) section 4.1, (○) algorithm 2, and (△) algorithm 1.

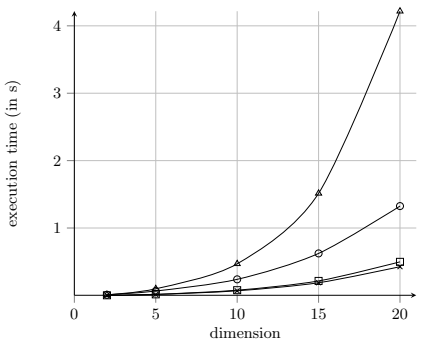



Fig. 3: Median execution time to generate an interval linear constraint using: (×) Jeannet et al. [11], (□) section 4.1, (○) algorithm 2, and (△) algorithm 1.

the x-coordinates. We also found bounds for instances that Jeannet et al. was unable to bound, representing almost 10% of our instances.

Computations were performed with a 2.5GHz CPU and 6Gb of memory using Python 3.5. As expected, fig. 3 shows that algorithm 2 has a complexity similar to Jeannet et al. but with a much higher constant, while algorithm 1 suffers from an additional factor $d$.

Figure 4 illustrates the effect of dimension on precision. Note that algorithm 2 in dimension 15 has a precision similar to Jeannet et al. approach in dimension 5.

## 6 Abstract Acceleration of LTI Systems with Inputs

In the previous section we introduced a few improvements over Jeannet et al. [11] techniques for the analysis of LTI loops with no inputs. We are now interested in loops of the form:

$$\text{while}(Gx \leq 0) \quad x := Ax + \text{Get}(U)$$

with $\text{Get}(U)$ nondeterministically returning a fresh point in $U$ at each loop iteration, and, without loss of generality, $0 \in U$.

Extending the work of Jeannet et al. [11] to systems with inputs has already been attempted in Cattaruzza et al. [5]. Unfortunately the method described is unsound. We will explain why in the next subsection before presenting a sound extension to systems with inputs, by reducing the problem to systems without inputs.

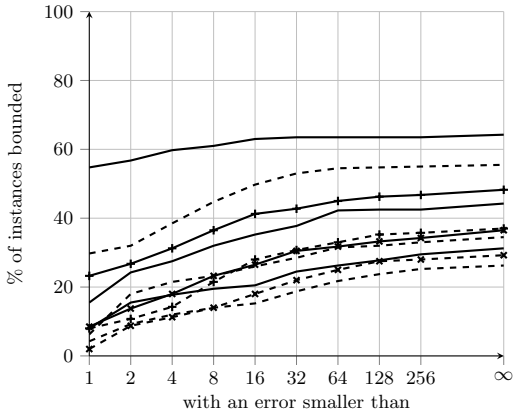



Fig. 4: Cumulative frequency of instances bounded by interval linear constraint(s) generated by: Jeannet et al. [11] (dashed), and algorithm 2 (solid), in dimension 2, 5 (+), 10, 15 (×), and 20 (top to bottom).

### 6.1 Unsoundness of Cattaruzza et al. SAS Paper

There are several issues with Cattaruzza et al. [5] paper and its extended version [6]. Among them, two clearly make the method unsound and another one exhibits this unsoundness.

The first one concerns the use of numerical algorithms to compute the Jordan form and is discussed in more details in section 7.1. The second one is central to the method and can not be corrected easily. Almost all approximations are based on the following (wrong) assumption [5, p. 322]:

> Let $g_i = \sum_{j=1}^{p} a_{ij} v_j$, where $v_j$ are generalised eigenvectors of $A$. [...] Then $A^n g_i = \sum_{j=1}^{p} \lambda_j^n a_{ij} v_j$ where $\lambda_j$ is the corresponding eigenvalue of $v_j$.

Which amounts to forgetting about the upper-diagonal in the Jordan decomposition. It would only be true if the $v_j$ were eigenvectors instead of generalized eigenvectors, but then $g_i$ would not necessarily admit a decomposition over them.[10]

Last, the experimental results section presents a comparison between their method and LGG algorithm [15] on an example with no guards. In this context, LGG algorithm is known to perform a tight over-approximation [13]: the exact set touches all the faces of the computed approximation. Since they used octahedral abstractions, the projections of the sets computed by LGG on state variables coincide with the projection of the exact reachable set. Yet, their algorithm computes a set with a smaller range in one dimension, effectively missing some reachable states and exhibiting the unsoundness of the method.

[10] The authors attempted a correction [6], unfortunately equation (16) is only true if all the $k_{ij}$ are positive and the method is still unsound.

### 6.2 Reduction to Systems without Inputs

In order to tackle LTI loops with inputs, we will reduce the problem to LTI loops without inputs. Some over-approximations are necessary, and we try to limit them to over-approximation that are already present in the analysis of systems without inputs, starting with the influence of the guard, which is only considered at the first and last step. For the transient behavior, we ignore the guards, then:

$$X_n = AX_{n-1} \oplus U = A^n X_0 \oplus \bigoplus_{i=0}^{n-1} A^i U \ .$$

**Lemma 1.** *For any real matrix $A$ and any set $U$, if we denote by $\square(.)$ the interval hull, $\boxdot(.)$ the centrally symmetric interval hull, and $|A|$ the matrix whose entries are the absolute value of the corresponding entries of $A$, we have:*

$$\bigoplus_{i=0}^{n-1} A^i U \subseteq \square\left(\left(\sum_{i=0}^{n-1} |A|^i\right) \boxdot (U)\right) \ .$$

*Proof.* For any point $u \in U$ and any vector $\ell$, it is relatively easy to show that: $A^i u \cdot \ell \leq |A|^i |u| \cdot |\ell|$. By denoting $u_{\max}$ the supremum, component-wise, of all $|u|$ for $u \in U$, we can further deduce that:

$$\forall \ell,\ \forall v \in \bigoplus_{i=0}^{n-1} A^i U,\ v \cdot \ell \leq \left(\sum_{i=0}^{n-1} |A|^i\right) u_{\max} \cdot |\ell| \ .$$

This set of constraints uniquely defines the set

$$\square\left(\left(\sum_{i=0}^{n-1} |A|^i\right) \boxdot (U)\right) \ . \qquad \square$$

Let us know consider the following LTI system with no inputs:

$$Y_{k+1} = \begin{pmatrix} 0 & I & I & 0 \\ 0 & A & 0 & 0 \\ 0 & 0 & I & I \\ 0 & 0 & 0 & |A| \end{pmatrix} Y_k, \text{ and } Y_0 = \begin{pmatrix} X_0 \\ AX_0 \\ \boxdot(U) \\ A \boxdot (U) \end{pmatrix} \ .$$

Then for any $k$, $X_k \subseteq \square\left(\begin{pmatrix} I & 0 & 0 & 0 \end{pmatrix} Y_k\right)$, as a direct consequence of lemma 1.

The maximum number of iterations remains to be bounded. We can not directly use $G$, because it will only guarantee that $Y_N$ is fully outside the guards while we are interested in $\square(Y_N)$. If one of the components of $\mathcal{X}$ is bounded and always has the same sign (which will necessarily be the case if the linear system was obtained from an affine system by adding one dimension) we denote by $\mathcal{B}$ the resulting band, then we define a new set of guards $\mathcal{G}'$ as the smallest cone containing the interval hull of the intersection between $\mathcal{G}$ and $\mathcal{B}$. This is still not enough, but if we consider each of the guards defined by $\mathcal{G}'$ independently, we

can guarantee that if $Y_N$ is outside of those guards, then $\square(Y_N)$ is outside of $\mathcal{G}$. Indeed, consider fig. 1b, if the projection of $\mathcal{G}$ on the corresponding plane is delimited by the doted lines, and the bounds on the first component defined by the width of the rectangle, then for any set within the bounds and outside of the solid lines, its interval hull is outside of the dotted lines.

Thus a bound on the number of iteration before $Y_n$ leaves $\mathcal{G}'$ also bounds the number of iteration before $X_n$ leaves $\mathcal{G}$ and we can now use the method described in the first section for systems without inputs to analyse systems with inputs.

In order to limit the over-approximation, one should first perform a change of variables that isolate contracting stable subspaces of $A$, using a real jordan form, and/or limit the over-approximation induced by $\mathcal{G}'$.

It is to be noted that the resulting system's dimension is four times the initial system's. Nevertheless the high sparsity and redundancy of the resulting system, if correctly exploited, should lead to a moderate increase in computation time.

## 7 About Floating Point Numbers

Floating point numbers are an approximation of real numbers with a compact computer representation. When doing verification, one must take into account the errors introduced by their use in the verification process but also in the process to verify.

### 7.1 Mixing Floats and Symbolic Relations

One important relation for the method to work is that for any $n$, $A^n = PJ^nP^{-1}$. Jeannet et al. [11] ensure soundness by computing the Jordan decomposition symbolically, then enclosing $P$, $J$, and $P^{-1}$ with interval matrices. Thus $PJ^nP^{-1}$ represents a set of matrices containing the exact $A^n$.

Cattaruzza et al. [5] use a numerical algorithm to compute the Jordan form, which is much faster. Note that the Jordan decomposition is known to be numerically unstable. The authors still claim soundness by bloating the diagonal element of $J$ with some constant $\delta_{\max} = |A - PJP^{-1}|$; it is not clear from the paper which norm is considered and the proof is left to the reader. Unfortunately, this bloating is not sufficient. Consider floating point numbers with a mantissa of size $2m$ such that $2^{2m} - 1$ and $2^{2m} + 2$ can be represented exactly but $2^{2m} + 1$ cannot. Then consider:

$$\begin{pmatrix} 1 + 2^{1-2m} & 1 \\ -2^{-4m} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ -2^{-2m} & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \\ 2^{-2m} & 1 \end{pmatrix}$$

All of these matrices can be represented exactly. With the notation $A = PJP'$ and interval arithmetic over floating point numbers one can verify that $A-(PJ)P' = 0$, thus no bloating is necessary according to Cattaruzza et al. [5]. Yet, for any $n > 1$, $A^n \neq PJ^nP'$. The reason why is that $PP' = P'P = (1 + 2^{-2m})I$ (which might be rounded to $I$ using floating point numbers). The bloating they introduce is unsound, in particular because the authors do not seem to consider approximations in the matrices $P$ and $P^{-1}$.

### 7.2 Loops with floating point arithmetic

The use of floating point arithmetic in the process to verify is not an issue for Jeannet et al. [11], Cattaruzza et al. [5, 6], or us, since we all consider loops involving real arithmetic only. A more realistic scenario in the context of program verification would be to consider floating point arithmetic which can have a dramatic effect especially when the floating point errors make the system stay longer in the body of the loop.

Again consider floating point numbers with a mantissa of size $2m$ such that $2^{2m}-1$ and $2^{2m}+2$ can be represented exactly but $2^{2m}+1$ cannot. Then consider:

$$\text{while}(x - 2^m y \leq 0) \quad \begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} 2^m & 1 \\ 0 & 2^m \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

with $x_0 = 2^m$ and $y_0 = 1$. Depending on the rounding mode, this might loop until $x = +\infty$. Yet, Jeannet et al. [11] method returns $\big([2^m; 2^{2m}+2], [1; 2^m]\big)$, which is correct with respect to an implementation of the considered loop using *real* numbers.

## 8 Conclusion

We proposed a novel sound approach to the unbounded reachability analysis of guarded LTI systems with inputs. A solution has already been presented [5, 6], unfortunately several major issues makes it unsound as we have demonstrated in this paper.

Our approach is based on a reduction from the system with inputs to a system without inputs. This reduction is independent from the method used to solve the reduced system but we suggest to use abstract acceleration as described by Jeannet et al. [11] for which we have introduced several improvements, mainly involving the avoidance of interval arithmetic. Our improvements strictly increase the precision, potentially dramatically as illustrated by a few simple examples and an extensive experimentation. Moreover, our description of the work of Jeannet et al. [11] highlights various independent sources of over-approximation and paves the way for further improvements.

Nevertheless our reduction is accompanied by a fourfold increase of dimension and can not be efficiently exploited yet. One solution would be to take advantage of the sparsity of the resulting system. There are also opportunities to increase the efficiency of abstract acceleration in particular concerning the application of the set of linear transformations $\mathcal{A}$ to $X_0$, one could start with techniques developed for parametrized systems [2, 3] and implemented in CORA [1], the cost in terms of precision remains to be evaluated.

If the approach is sound for machine integers, as long as no overflow occurs[11], future work should also focus on the use of floating point arithmetic in the loop body, considering only diagonalizable systems might be enough since the set of real matrices diagonalizable in $M_n(\mathbb{C})$ is dense in $M_n(\mathbb{R})$.

[11] The corresponding constraint can be added to the loop guard.

## Appendix 1 Loop Homogenization

The loop

$$\text{while}(Gx \leq h) \quad x := Ax + B\text{Get}(U) + c$$

can be expressed as

$$\text{while}(G'y \leq 0) \quad y := A'y + \text{Get}(U')$$

with $0 \in U'$ by choosing any $u_0 \in U$ and using the following substitutions:

$$\begin{aligned} y &= \begin{pmatrix} x \\ 1 \end{pmatrix} \\ G' &= \begin{pmatrix} G & (-h) \end{pmatrix} \\ A' &= \begin{pmatrix} A & c' \\ 0 & 1 \end{pmatrix} && c' = c + Bu_0 \\ U' &= \left\{ \begin{pmatrix} B(u - u_0) \\ 0 \end{pmatrix} \mid u \in U \right\} \ . \end{aligned}$$

## Appendix 2 $A^n$ coordinates

Consider $A = PJP^{-1}$ where $J$ is a single Jordan block with a real eigenvalue $\lambda$, we have $J = \lambda I + N$ where $N$ is a nilpotent matrix with 1 on the upper diagonal and 0 everywhere else. Then:

$$J^n = (\lambda I + N)^n = \sum_{i=0}^{d-1} \binom{n}{i} \lambda^{n-i} N^i \ ,$$

and we can express $\mathcal{J}$, $\mathcal{A}$, and $\mathcal{M}$ as:

$$\begin{aligned} \mathcal{J} &= \left\{ \sum_{i=0}^{d-1} m_i N^i \mid m \in \mathcal{M} \right\} \ , \\ \mathcal{A} &= \left\{ \sum_{i=0}^{d-1} m_i P N^i P^{-1} \mid m \in \mathcal{M} \right\} \ , \\ \mathcal{M} &= \left\{ \left( \lambda^n, n\lambda^{n-1}, \ldots, \binom{n}{d-1} \lambda^{n-d+1} \right) \mid n \in \mathbb{N} \right\} \ . \end{aligned}$$

In the general case, the coefficients of $m$ have the form:

$$m_i(n) = \binom{n}{k_i} \lambda_i^{n-k_i} \cos\left( (n - k_i)\theta_i - r_i \frac{\pi}{2} \right) \tag{3}$$

with $\lambda_i \geq 0$, $\theta_i \in [0; \pi]$, $r_i \in \{0; 1\}$ and $k_i \in [0..d-1]$.

## Appendix 3 Examples

### Appendix 3.1 Avoiding Interval Arithmetic

Starting from any point in the convex hull of $\{(1,0);(0,1)\}$, consider the loop:

$$\text{while}(x+y \leq 10)\{x := 2x;\ y := 2y\}\ ,$$

or expressed as matrices:

$$g = \begin{pmatrix}1 & 1 & -10\end{pmatrix} \quad A = \begin{pmatrix}2 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 1\end{pmatrix}$$

where the extra dimension is used to account for the constant 10 in the guard. $A$ is already in Jordan form and the $A^n$ lie in the vector space generated by:

$$M_0 = \begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 0\end{pmatrix} \quad M_1 = \begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 1\end{pmatrix}\ .$$

For any $n$, we have $A^n = 2^n M_0 + 1M_1$. In order to bound the number of steps, Jeannet et al. [11] would first bound $X_0 \cap \mathcal{G}$ by its interval hull and compute:

$$\begin{aligned}\begin{pmatrix}1 & 1 & -10\end{pmatrix}\begin{pmatrix}m_0 & 0 & 0\\ 0 & m_0 & 0\\ 0 & 0 & 1\end{pmatrix}\begin{pmatrix}[0;1]\\ [0;1]\\ [1;1]\end{pmatrix} &\leq 0\\ \begin{pmatrix}1 & 1 & -10\end{pmatrix}\begin{pmatrix}[0;1]m_0\\ [0;1]m_0\\ 1\end{pmatrix} &\leq 0\\ [0;2]m_0 - 10 &\leq 0\ .\end{aligned}$$

The semantics of interval linear constraints is existential thus:

$$[0;2]m_0 - 10 \leq 0 \Leftrightarrow \exists \alpha \in [0;2],\ \alpha m_0 - 10 \leq 0\ .$$

Taking $\alpha = 0$ always makes the right hand side true, thus this interval linear constraints is a tautology and can not lead to any bound on the number of steps.

If instead of using their method we use ours, as described in **??**, we get:

$$L = \begin{pmatrix}gM_0\\ gM_1\end{pmatrix} = \begin{pmatrix}1 & 1 & 0\\ 0 & 0 & -10\end{pmatrix}$$

and

$$\square\left(L(X_0 \cap \mathcal{G})\right) = \begin{pmatrix}[1;1]\\ [-10;-10]\end{pmatrix}$$

leading to the constraint $m_0 - 10 \leq 0$, or $2^n - 10 \leq 0$, correctly bounding the number of steps by 3.

### Appendix 3.2 Avoiding Interval Hull: algorithm 1

Starting from any point in the convex hull of $\{(2,1);(6,3)\}$, consider the loop:

$$\text{while}(-2x+y\leq 0)\{x:=0.9x;\ y:=y\}$$

or expressed as matrices:

$$g=\begin{pmatrix}-2 & 1\end{pmatrix}\quad A=\begin{pmatrix}0.9 & 0\\ 0 & 1\end{pmatrix}\ .$$

$A$ is already in Jordan form and the $A^n$ lie in the vector space generated by:

$$M_0=\begin{pmatrix}1 & 0\\ 0 & 0\end{pmatrix}\quad M_1=\begin{pmatrix}0 & 0\\ 0 & 1\end{pmatrix}\ .$$

For any $n$, we have $A^n=0.9^nM_0+1M_1$. In order to bound the number of steps we first need[12] to compute $L(X_0\cap\mathcal{G})$:

$$L=\begin{pmatrix}gM_0\\ gM_1\end{pmatrix}=\begin{pmatrix}-2 & 0\\ 0 & 1\end{pmatrix}\ .$$

and $L(X_0\cap\mathcal{G})$ is the convex hull of $(-4,1)$ and $(-12,3)$. Using the interval hull of this set we get:

$$[-12;-4]m_0+[1;3]\leq 0$$

which can be linearized as $-12m_0+1\leq 0$, bounding the number of steps by 23.

If instead we apply algorithm 1 to $L(X_0\cap\mathcal{G})$, we get as a box $\mathcal{B}$ the singleton $(-4,1)$ leading to the constraint $-4m_0+1\leq 0$, correctly bounding the number of steps by 13.

### Appendix 3.3 Avoiding Interval Hull: algorithm 2

Starting from any point in the convex hull of $\{(2,1);(6,3)\}$, consider the loop:

$$\text{while}(-2x+y-0.1\leq 0)\{x:=0.9x;\ y:=y+1\}\ ,$$

or expressed as matrices:

$$g=\begin{pmatrix}-2 & 1 & -0.1\end{pmatrix}\quad A=\begin{pmatrix}0.9 & 0 & 0\\ 0 & 1 & 1\\ 0 & 0 & 1\end{pmatrix}\ .$$

$A$ is already in Jordan form and the $A^n$ lie in the vector space generated by $M_0$, $M_1$, and $M_2$:

$$\begin{pmatrix}1 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix}\quad\begin{pmatrix}0 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{pmatrix}\quad\begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 1\\ 0 & 0 & 0\end{pmatrix}\ .$$

[12] As explained before, $L(X\cap\mathcal{G})$ does not need to be computed explicitly, its projection on any subspace can be approximated with arbitrary precision by optimizing linear functions on $X\cap\mathcal{G}$. The procedure is efficient when the subspaces are of low dimension as is the case here (one to get the interval hull, two to compute GET_PAIR).

For any $n$, we have $A^n = 0.9^n M_0 + 1M_1 + nM_2$. In order to bound the number of steps we first need[13] to compute $L(X_0 \cap \mathcal{G})$:

$$L = \begin{pmatrix} gM_0 \\ gM_1 \\ gM_2 \end{pmatrix} = \begin{pmatrix} -2 & 0 & 0 \\ 0 & 1 & -0.1 \\ 0 & 0 & 1 \end{pmatrix}$$

and $L(X_0 \cap \mathcal{G})$ is the convex hull of $(-4, 0.9, 1)$ and $(-12, 2.9, 1)$. Using the interval hull of this set we get:

$$[-12; -4]m_0 + [0.9; 2.9] + m_2 \leq 0 \ ,$$

which can be linearized as $-12m_0 + 0.9 + m_2 \leq 0$, bounding the number of steps by 5.

Here algorithm 1 cannot help us because $\square\,(L(X_0 \cap \mathcal{G}))$ is flat. Using algorithm 2 optimizing the first component we get a box $\mathcal{B} = [-1; -1] \times [\frac{9}{40}; \frac{29}{120}] \times [\frac{1}{12}; \frac{1}{4}]$ leading to the constraint $-12m_0 + 2.7 + m_2 \leq 0$, bounding the number of steps by 4.

## Appendix 4 Proof of theorem 1

**Theorem 1.** *For any product of closed intervals, its conic hull $\angle(\mathcal{B})$ is entirely determined by its projection on canonical planes.*

First, let us characterize conic hulls of products of compact intervals with nonempty interior.

**Lemma 2.** *For any product of compact intervals with nonempty interior $\mathcal{B}$, its conic hull $\angle(\mathcal{B})$ is entirely determined by its projection on canonical planes.*

*Proof.* In dimension $d$, for any compact box with non empty interior $\mathcal{B}$, the $(d-1)$-faces of $\angle(\mathcal{B})$ contains 0 and at least one $(d-2)$-face of $\mathcal{B}$. Thus each $(d-1)$-face of $\angle(\mathcal{B})$ has a normal in the orthogonal complement of a $(d-2)$-face of $\mathcal{B}$. These orthogonal complements are planes generated by two canonical vectors. Thus each $(d-1)$-face of $\angle(\mathcal{B})$ has a normal vector lying in a canonical plane. All $(d-1)$-face of $\angle(\mathcal{B})$ can be enumerated by projecting $\mathcal{B}$ on each of the $d(d-1)/2$ canonical planes as illustrated in fig. 1. On each plane, we can identify either zero, one, or two faces, for a maximum of $d(d-1)$ faces, or constraints, for the full dimensional cone. □

**Lemma 3.** *For any product of compact intervals, its conic hull $\angle(\mathcal{B})$ is entirely determined by its projection on canonical planes.*

*Proof.* Let $\mathcal{B}$ be a compact product of intervals with empty interior, and $y$ a point such that its projection on any canonical plane is included in the conic

[13] Again, this is not really necessary.

hull of the projection of $\mathcal{B}$. We need to prove that $y$ is in the conic hull of $\mathcal{B}$.[14] For any $\epsilon > 0$ let us define $\mathcal{B}_\epsilon$ as $\mathcal{B} \oplus [0;\epsilon]^d$. It is clear that the projection of $y$ on any canonical plane lies in the conic hull of the projection of $\mathcal{B}_\epsilon$. Since $\mathcal{B}_\epsilon$ is compact with a nonempty interior, there exists $x_\epsilon$ in $\mathcal{B}_\epsilon$ and $\alpha > 0$ such that $y = \alpha x_\epsilon$ according to lemma 2. From any strictly decreasing sequence of positive $\epsilon_i$ one can construct a sequence of $x_{\epsilon_i}$ from which one can extract a converging sequence (since $\mathcal{B}_{\epsilon_0}$ is compact). Let us denote by $x$ the limit of this sequence. Since $x$ is in the intersection of all $\mathcal{B}_{\epsilon_i}$, $x$ is in $\mathcal{B}$ and $y$ in its conic hull. □

We can now state the proof in the general case. Let $\mathcal{B}$ be a closed product of intervals, and $y$ a point such that its projection on any canonical plane is included in the conic hull of the projection of $\mathcal{B}$. Let us define $\mathcal{B}_\alpha$ as $\mathcal{B} \cap [-\alpha;\alpha]^d$. By taking $\alpha$ big enough and applying lemma 3, one can show that $y$ is in the conic hull of $\mathcal{B}$. □

## References


[1] M. Althoff. An introduction to CORA 2015. In G. Frehse and M. Althoff, editors, *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, volume 34 of *EPiC Series in Computing*, pages 120–151. EasyChair, 2015.

[2] M. Althoff, B. H. Krogh, and O. Stursberg. *Modeling, Design, and Simulation of Systems with Uncertainties*, chapter Analyzing Reachability of Linear Dynamic Systems with Parametric Uncertainties, pages 69–94. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-15956-5. doi: 10.1007/978-3-642-15956-5_4.

[3] M. Althoff, C. Le Guernic, and B. H. Krogh. Reachable set computation for uncertain time-varying linear systems. In *Proceedings of the 14th International Conference on Hybrid Systems: Computation and Control*, HSCC '11, pages 93–102, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0629-4. doi: 10.1145/1967701.1967717.

[4] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer, 1999. ISBN 3-540-65703-7. doi: 10.1007/3-540-49059-0_14.

[5] D. Cattaruzza, A. Abate, P. Schrammel, and D. Kroening. Unbounded-time analysis of guarded LTI systems with inputs by abstract acceleration. In S. Blazy and T. Jensen, editors, *Static Analysis - 22nd International Symposium, SAS 2015, Saint-Malo, France, September 9-11, 2015, Proceedings*, volume 9291 of *Lecture Notes in Computer Science*, pages 312–331. Springer, 2015. ISBN 978-3-662-48288-9. doi: 10.1007/978-3-662-48288-9_18.


[14] The other direction is obvious.

[6] D. Cattaruzza, A. Abate, P. Schrammel, and D. Kroening. Unbounded-time analysis of guarded LTI systems with inputs by abstract acceleration (extended version). *CoRR*, abs/1506.05607, 2015. URL http://arxiv.org/abs/1506.05607.

[7] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth ACM Symposium on Principles of Programming Languages, Los Angeles, California, USA, January 1977*, pages 238–252. ACM, 1977. doi: 10.1145/512950.512973.

[8] J. Feret. Static analysis of digital filters. In *Programming Languages and Systems, 13th European Symposium on Programming, ESOP 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2986 of *Lecture Notes in Computer Science*, pages 33–48. Springer, 2004. ISBN 3-540-21313-9. doi: 10.1007/978-3-540-24725-8_4.

[9] L. Gonnord and N. Halbwachs. Combining widening and acceleration in linear relation analysis. In *Static Analysis, 13th International Symposium, SAS 2006, Seoul, Korea, August 29-31, 2006, Proceedings*, volume 4134 of *Lecture Notes in Computer Science*, pages 144–160. Springer, 2006. doi: 10.1007/11823230_10.

[10] L. Gonnord and P. Schrammel. Abstract acceleration in linear relation analysis. *Sci. Comput. Program.*, 93:125–153, 2014. doi: 10.1016/j.scico.2013.09.016.

[11] B. Jeannet, P. Schrammel, and S. Sankaranarayanan. Abstract acceleration of general linear loops. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '14, pages 529–540, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2544-8. doi: 10.1145/2535838.2535843.

[12] A. A. Kurzhanskiy and P. Varaiya. Ellipsoidal techniques for reachability analysis of discrete-time linear systems. *IEEE Trans. Automat. Contr.*, 52 (1):26–38, 2007. doi: 10.1109/TAC.2006.887900.

[13] C. Le Guernic. *Reachability Analysis of Hybrid Systems with Linear Continuous Dynamics*. PhD thesis, Université Joseph Fourier - Grenoble I, 2009. URL https://tel.archives-ouvertes.fr/tel-00422569.

[14] C. Le Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*, pages 540–554. Springer, 2009. doi: 10.1007/978-3-642-02658-4_40.

[15] C. Le Guernic and A. Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250–262, 2010. ISSN 1751-570X. doi: 10.1016/j.nahs.2009.03.002. IFAC World Congress 2008.

[16] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control, 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25-27, 2004, Pro-*

*ceedings*, volume 2993 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2004. ISBN 3-540-21259-0. doi: 10.1007/978-3-540-24743-2_32.

[17] P. Roux, R. Jobredeaux, P. Garoche, and E. Feron. A generic ellipsoid abstract domain for linear time invariant systems. In *Hybrid Systems: Computation and Control (part of CPS Week 2012), HSCC'12, Beijing, China, April 17-19, 2012*, pages 105–114. ACM, 2012. ISBN 978-1-4503-1220-2. doi: 10.1145/2185632.2185651.