Combining Zonotope Abstraction and Constraint Programming for Synthesizing Inductive Invariants

Bibek Kabi[1], Eric Goubault[1], Antoine Miné[2], and Sylvie Putot[1]

[1] LIX, Ecole Polytechnique, CNRS, Institut Polytechnique de Paris, France
[2] Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, France

Abstract. We propose to extend an existing framework combining abstract interpretation and continuous constraint programming for numerical invariant synthesis, by using more expressive underlying abstract domains, such as zonotopes. The original method, which relies on iterative refinement, splitting and tightening a collection of abstract elements until reaching an inductive set, was initially presented in combination with simple underlying abstract elements: boxes and octagons. The novelty of our work is to use zonotopes, a sub-polyhedric domain that shows a good compromise between cost and precision. As zonotopes are not closed under intersection, we had to extend the existing framework, in addition to designing new operations on zonotopes, such as a novel splitting algorithm based on paving zonotopes by sub-zonotopes and parallelotopes. We implemented this method on top of the APRON library, and tested it on programs with non-linear loops that present complex, possibly non-convex, invariants. We present some results demonstrating both the interest of this splitting-based algorithm to synthesize invariants on such programs, and the good compromise presented by its use in combination with zonotopes with respect to its use with both simpler domains such as boxes and octagons, and more expressive domains such as polyhedra.

1 Introduction

Proving loop invariants is a key ingredient in the verification of safety properties on programs. The classic method to prove that a set is indeed an invariant is to look for an *inductive invariant* which implies it, i.e., a state property that is stable by an iteration of the loop. Notably, the set of program states reachable from the initial states is the least (i.e. most precise) inductive invariant. Generally, this set is difficult to compute, so that we settle for an over-approximation, as any such over-approximation is also an invariant. Inductive invariants play a special role in program verification because they can be checked by running a single loop iteration and checking its stability, even for unbounded loops. A key issue is that not all invariants are inductive, and it is often necessary, given a target invariant property to prove, to first strengthen it into an inductive invariant. We illustrate the main results of this work on the program (taken from online

```
x=[0.9,1.1];y=[0.9,1.1];
while (True) {
    xnew=2x/(0.2 + x^2 + y^2 +
        1.53x^2y^2);
    ynew=2y/(0.2 + x^2 + y^2 +
        1.53x^2y^2);
    x=xnew;y=ynew;
}
```

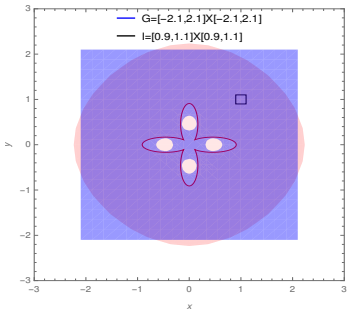
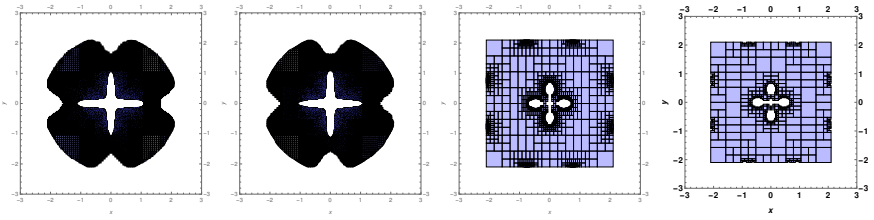


Fig. 1. (a) Example program; (b) Non-inductive invariants for the program.

additional material of [1]) in Fig. 1(a), with initial values of variables (x, y) in the box $I \stackrel{\text{def}}{=} [0.9, 1.1]^2$ and the effect of a loop iteration on a set X of possible variable values $(x, y) \in X$ given by the function $F : \mathcal{P}(\mathbb{R}^2) \to \mathcal{P}(\mathbb{R}^2)$ defined as the loop body of Fig. 1(a). The box $T = [-2.1, 2.1]^2$, shown in blue in Fig. 1(b), is a valid invariant: T includes all the states reachable at the loop head, i.e., $\bigcup_{n\in\mathbb{N}} F^n(I) \subseteq T$. However, T is not an inductive invariant as $F(T) \not\subseteq T$: indeed, F maps the box T to a circle, that goes a bit outside the box T. Consider the four-petal flower shape towards the center shown in Fig. 1(b): its interior is not reachable from the initial box I, and it contains the four small circles in white which are the inverse image by F of the four parts of the circle that go beyond box T. To prove that T is an invariant for executions beginning in I, we need to infer an inductive invariant G that is included in T, and precise enough to express that the small circles inside each petal of T are not reachable states. But all such invariants have a complex shape, that cannot be represented in classical abstract domains. The idea of [1], inspired from contraint programming approaches, is to synthetize an inductive invariant as a collection of abstract elements, that are iteratively split and refined. In set-based constraint programming, these elements are generally boxes. Previous work [1] was limited to abstract domains that are closed by intersection (such as polyhedra or octagons) and required a non-standard operation: split. In this article, we extend this work to other abstract domains such as zonotopes. We show that their use in this context provides an interesting trade-off between expressiveness and efficiency, by comparing their use with that of boxes, octagons, and polyhedra. For example, on the program of Fig. 1, Figs. 2(a)–2(d) show the inductive invariant G within T as found by our algorithm in combination with box, octagon, polyhedron and zonotope abstract domains. Inference with intervals (resp. octagons, polyhedra, and zonotopes) takes 646.8 (resp. 8850.8, 126.8, and 35.6) seconds and produces an inductive invariant composed of 129781 (resp. 129767, 2368, and 488) parts. Less expressive domains such as boxes and octagons rely heavily on splitting, hence output a large set of elements and are slower in total, in spite of a smaller cost of manipulating a single abstract element. In particular, in polyhedra and



(a) (b) (c) (d)

Fig. 2. Inductive invariant obtained by our algorithm using (a) boxes; (b) octagons; (c) polyhedra and (d) zonotopes.

zonotopes, the image by the loop body in the abstract domain, of a box on the left corner in Figure 2(c)-2(d) is within the collection of elements, thus proven invariant, whereas it will not be the case in the box or octagon abstract domains, ultimately leading by splitting to the refinement of Figures 2(a)-2(b).

Zonotopes are a cost-effective, versatile, and precise abstract domain [2] that can represent restricted forms of polyhedra as Minkowski sums of line segments. They feature more lightweight algorithms than general polyhedra, while being more expressive than other sub-polyhedra domains (e.g., octagons). They are particularly well suited to approximate non-linear functions. Yet, zonotopes do not form a lattice and do not enjoy an exact intersection, which is required in [1]. We thus need to adapt this approach, in addition to defining split and meet operators, intersection tests, and inclusion tests specifically for zonotopes.

Related work. Garg et al. [3] developed a learning paradigm for loop invariant synthesis known as ICE, where the learner synthesizes invariants and the teacher verifies them using a constraint solver such as SMT. Thakur et al. [4] discussed a method based on abstract interpretation and SMT solving for searching inductive invariants. Essentially, the recent line of work dedicated to synthesizing invariants combines abstract interpretation and constraint-based techniques (SAT/SMT solvers). The idea of using SAT/SMT is similar to ours, except that we focus on other classes of algorithms, those used in continuous constraint programming, and we work on geometric entities, such as boxes and zonotopes, instead of logical formulas.

Contribution. Our contributions are as follows: firstly, we extend the method of [1] to domains not closed under intersection; secondly, we introduce new zonotope operators: split, inclusion and intersection tests; finally, we present a prototype, extending the Taylor1+ zonotope abstract domain [5] in the Apron library [6], and adapting the algorithm of [1]. Section 2 recalls the algorithm from [1]; Sect. 3 recalls the zonotope domain, introduces our new operators, and discusses how to handle domains that are not intersection-closed; Sect. 4 presents our experimental evaluation; Sect. 5 concludes and discusses future works.

2 Refinement-based inductive invariant inference

We now recall the algorithm from [1] to tighten an invariant into an inductive invariant, by splitting and tightening a collection of abstract elements.

We assume without loss of generality that variables are real-valued (which includes integers and non-special float values). A program environment is a subset of $\mathbb{R}^n$, where n is the number of variables. The concrete semantics of a program is the collecting semantics of [7]: it is given as the least fixpoint of a function $F : \mathcal{P}(\mathbb{R}^n) \to \mathcal{P}(\mathbb{R}^n)$ over an initial environment $I \subseteq \mathbb{R}^n$; typically, when a program consists of just one loop, F is the transfer function for one loop iteration.

We also assume here that we are given a target invariant $T \subseteq \mathbb{R}^n$. The goal of this article is to infer an inductive invariant $G \subseteq \mathbb{R}^n$ that proves that T is indeed an invariant. This requires finding G such that: $I \subseteq G$ (G includes the initial states), $F(G) \subseteq G$ (induction), and $G \subseteq T$ (invariant entailment).

As a first step, we replace computations in $\mathcal{P}(\mathbb{R}^n)$ with computations in an abstract domain $\mathcal{D}^\sharp \subseteq \mathcal{P}(\mathbb{R}^n)$ of tractable subsets of $\mathbb{R}^n$, such as boxes, octagons, polyhedra, or zonotopes. The abstract version of F, overapproximating the concrete semantics, is denoted by $F^\sharp : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$.

Instead of generating an inductive invariant in $\mathcal{D}^\sharp$, as classically done in abstract interpretation based analyzers, we are going to look for one in the much more expressive *disjunctive completion* of $\mathcal{D}^\sharp$, i.e. in $\mathcal{P}(\mathcal{D}^\sharp)$. More precisely, we are going to synthetize finite collections $G^\sharp \subseteq \mathcal{D}^\sharp$ of abstract elements, $G^\sharp = \{S_1, \ldots, S_n\}$ that satisfies: $I \subseteq \bigcup_i S_i$, $\forall k : F^\sharp(S_k) \subseteq \bigcup_i S_i$, and $\bigcup_i S_i \subseteq T$, which implies that $\bigcup_i S_i$ is an inductive invariant. To simplify, we assume that both the initial state I and the target invariant T are exactly expressible in $\mathcal{D}^\sharp$.

Search algorithm. The algorithm of [1] we are building upon in this article maintains a finite collection $G^\sharp \subseteq \mathcal{D}^\sharp$ that forms a candidate inductive invariant. It is initialized with the (non-inductive) target invariant: $G^\sharp = \{T\}$. We ensure at all times that $I \subseteq \bigcup_i S_i \subseteq T$, and iteratively refine $G^\sharp$ until it becomes inductive, i.e., until $\forall k : F^\sharp(S_k) \subseteq \bigcup_i S_i$. While $G^\sharp$ is not inductive, we iterate the following steps, adding, removing, and updating elements in $G^\sharp$:

- pick an element $S_k \in G^\sharp$ with $F^\sharp(S_k) \not\subseteq \bigcup_i S_i$, i.e., preventing inductiveness;
- either discard S_k from $G^\sharp$, split it into two elements that are added back to $G^\sharp$, or tighten (i.e., shrink) it.

To decide the action to perform, it is useful to classify an abstract element S_k in relation to the other elements in $G^\sharp$ and their image by $F^\sharp$. We say that S_k is:

- *doomed* if $F^\sharp(S_k) \cap (\bigcup_i S_i) = \emptyset$; such an element will always prevent inductiveness and must be discarded;
- *benign* if $F^\sharp(S_k) \subseteq \bigcup_i S_i$, i.e., it does not prevent inductiveness and does not need to be changed;
- *necessary* if $S_k \cap I \neq \emptyset$, i.e., it cannot be discarded, to keep ensuring that $I \subseteq \bigcup_i S_i$ always holds;

– *useful* if $S_k \cap (\bigcup_i F^\sharp(S_i)) \neq \emptyset$, i.e., at least an element from $G^\sharp$ relies on S_k to be benign.

Remark 1. Note that, to decide if an abstract element is *necessary* or *useful*, the algorithm requires a test for checking intersection. This test must be an exact one because discarding a necessary or useful element can result in a failure.

The algorithm first selects a non-benign element S_k. It is discarded if it is either doomed or not useful, unless it is necessary. Otherwise, it is generally split. By spliting S_k, we can hope to isolate the part that is doomed, and is ultimately discarded, from the part that is benign, and kept in the final inductive invariant. To guide our choice, a useful quantitative measure is that of coverage:

$$\text{coverage}(S_k) := \frac{\text{vol}(F^\sharp(S_k) \cap (\bigcup_i S_i))}{\text{vol}(F^\sharp(S_k))} \tag{1}$$

where $\text{vol}(X)$ is the volume of a set X. Intuitively, the coverage measure denotes how much the image of S_k lies in the candidate invariant, i.e., how much it is inductive. Note that, for this to make sense, it is important to rely on the fact that the S_i do not overlap (except maybe on common borders that have a null volume). The algorithm systematically picks the element $S_k \in G^\sharp$ with least coverage in priority, as it requires the most urgent action.

Remark 2. In [1], elements with a very low coverage are systematically discarded, as unlikely to become benign. However, it is possible that an abstract element may intersect every inductive invariant, in which case discarding this element will result in a failure. So, in the current work, we only discard an abstract element if it has a coverage of 0, i.e., it is doomed.

Tightening. Any part of an S_k that does not intersect any $F^\sharp(S_i)$ is not useful and can safely be discarded, improving the likelihood that S_k becomes benign, without making other benign elements non-benign. An additional tightening step can help the search algorithm by replacing S_k with: $\bigcup_i (S_k \cap F^\sharp(S_i)) \cup (S_k \cap I)$ suitably overapproximated to an element representable in the abstract domain.

Data structure. To compute the coverage, and in particular $\text{vol}(F^\sharp(S_k) \cap (\bigcup_i S_i))$, it is useful to identify which elements $S_i \in G^\sharp$ each image $F^\sharp(S_k)$ may intersect. To do this efficiently, we maintain a partition $B^\sharp \subseteq \mathcal{D}^\sharp$ of the target space T, as well as a map $\text{post} : G^\sharp \to \mathcal{P}(B^\sharp)$ to denote which parts of $B^\sharp$ intersect the image of each abstract element $S_k \in G^\sharp$: $\text{post}(S_k) := \{P \in B^\sharp \mid F^\sharp(S_k) \cap P \neq \emptyset\}$. Moreover, we ensure that any element in $B^\sharp$ completely contains at most one element in $G^\sharp$. We maintain a *contents-of* function $\text{cnt} : B^\sharp \to (G^\sharp \cup \{\emptyset\})$ indicating which abstract element of $G^\sharp$, if any, is contained in each element of $B^\sharp$: $\forall S_i \in G^\sharp, \exists! B_j \in B^\sharp, \text{cnt}(B_j) = S_i$; moreover, $S_i \subseteq B_j$. Assuming that $B^\sharp$ is available, then the coverage function can be optimized by only considering the relevant parts from $B^\sharp$:

$$\text{coverage}(S_k) := \frac{\sum \{\text{vol}(F^\sharp(S_k) \cap \text{cnt}(P)) \mid P \in \text{post}(S_k)\}}{\text{vol}(F^\sharp(S_k))} \tag{2}$$

$B^\sharp$ is initialized to $\{T\}$ (as $G^\sharp$) and it is split the same way $G^\sharp$ is. However, elements in $B^\sharp$ are not tightened, so that $B^\sharp$ remains a partition of T. The maps cnt and post can be efficiently maintained when the elements from $B^\sharp$ and $G^\sharp$ are split, removed, or tightened, as detailed in [1].

Remark 3. The coverage information is first used to decide which S_i in $G^\sharp$ require urgent action. Then, it is used to decide whether to split or discard. As this is only a heuristic, an exact volume computation is not mandatory to ensure correctness. [1] relies on the volume of bounding boxes as it is easy to compute for the domains considered there. As bounding boxes and volume computations are expensive for zonotopes, we will introduce another heuristic that does not rely on computing volumes.

Testing whether an abstract element is benign, i.e. maintains inductiveness, requires inclusion checking: S_k is benign if, whenever $F^\sharp(S_k)$ intersects some partition $P \in B^\sharp$, the intersection is included in $cnt(P) \in G^\sharp$. We also need to check that $F^\sharp(S_k)$ is included within the candidate invariant T. Formally:

$$S_k \text{ is benign} \iff \forall P \in \text{post}(S_k) : P \cap F^\sharp(S_k) \subseteq \text{cnt}(P) \wedge F^\sharp(S_k) \subseteq T \quad (3)$$

Unlike coverage computation, this inclusion check must be exact to ensure that we indeed have an inductive invariant, i.e., that our method is sound.

The algorithm is parameterized by a choice of abstract domain $\mathcal{D}^\sharp$. In addition to an abstract version $F^\sharp$ of F, already provided by abstract interpretation, it requires a split operator, a meet operator for tightening, tests to check intersection and inclusion. Such operators have been proposed for boxes and octagons in [1]. In the following, we will provide these operators for zonotopes and also adapt the coverage operation from [1] by replacing the volume operator.

3 Zonotope abstraction and constraint solving

We now adapt the algorithm introduced in Section 2 to use in combination with the abstract domain of zonotopes. We first introduce zonotopes as a classical abstract domain, then describe the new operations that were designed for the specific use here.

3.1 Affine forms and zonotopes

Zonotopes are based on affine arithmetic [8], an extension of interval arithmetic expressing dependencies between variables. More precisely, the set of values of a program variable x is abstracted by an affine expression of the form $\hat{x}{:=}{\alpha_0}^x + \sum_{i=1}^{n} {\alpha_i}^x \varepsilon_i$, where ε_i are symbolic variables, called noise symbols, whose values are restricted to $[-1, 1]$. Linear dependencies between program variables can be expressed by sharing noise symbols. The set of possible values of all the variables x_k when the noise symbols vary in $[-1, 1]$ is a zonotope, i.e., a center-symmetric

polytope (a bounded polyhedron) with center-symmetric faces. Compared to other restrictions of polyhedra, such as octagons or templates, the directions of zonotopic faces are not fixed *a priori*, providing some flexibility in the analysis. Zonotopes have already proved to be a simple and tractable set representation for program analysis [2, 5].

Consider a tuple of affine forms $\hat{x}_i$ for p variables x_i over n noise symbols ε_j, $\hat{x}_i{=}{\alpha_0}^{x_i} + \sum_{k=1}^{n} {\alpha_k}^{x_i}\varepsilon_k$. This defines a matrix $A \in M(n+1,p)$ whose (j,i) entry, for $i{=}1,\ldots,p$, $j{=}0,\ldots,n$ is $A_{j,i} = \alpha_j^{x_i}$. The zonotope concretization is

$$\gamma(A) = \left\{ A^{\mathrm{T}} \begin{pmatrix} 1 \\ e \end{pmatrix} | e \in \mathbb{R}^n, ||e||_\infty \leq 1 \right\} \subseteq \mathbb{R}^p \tag{4}$$

For $n{=}3$ and $p{=}2$, the zonotope (the center symmetric polygon in blue) in Fig. 3(a) is the concretisation (joint range) of the affine forms $X{=}(\hat{x},\hat{y})$ with $\hat{x}{=}18 + 2\varepsilon_1 + 1\varepsilon_2 + 3\varepsilon_3$, $\hat{y}{=}12 + 1\varepsilon_2 + 5\varepsilon_3$, that is, $A^{\mathrm{T}}{=}\begin{pmatrix} 18 & 2 & 1 & 3 \\ 12 & 0 & 1 & 5 \end{pmatrix}$. In what follows, we note $A_+ \in \mathcal{M}(n,p)$ the submatrix of $A \in \mathcal{M}(n+1,p)$, without its first column, that corresponds to the center of the zonotope. Each column of A_+ defines a generator of the zonotope, and we also represent zonotope A as $(c, g_1, \ldots g_n)$, i.e., as its center c and its collection of generators $g_1, \ldots, g_n$. For the example above, we would write $A{=}((18,12),(2,0),(1,1),(3,5))$. This zonotope is spanned by the generators drawn in its center in Fig. 3(a). The Minkowski sum of the line segments described by those vectors is the zonotope itself.

3.2 Zonotope operators

Inclusion. The best known methods for inclusion tests are known to have exponential time (in terms of the number of generators) for zonotopes [2]. Lemma 1 below is an extension of Lemma 4 from [2], which transforms the inclusion test into an infinite number of simple inequalities, that in turn translate into an exponential number of linear programs to be solved. We show below how to further decrease the number of linear programs to solve, so as to go from an exponential to a polynomial number.

Lemma 1. *For two zonotopes given by matrices $X \in \mathcal{M}(n_X+1,p)$ and $Y \in \mathcal{M}(n_Y+1,p)$, let $u{=}\{u_1,\ldots,u_k\}$ be vectors in $\mathbb{R}^p$ such that each face in $\gamma(Y)$ has a vector in u that is normal to it. Then $\gamma(X) \subseteq \gamma(Y)$ if and only if*

$$\left| \langle u_i, c_x - c_y \rangle \right| \leq ||Y_+ u_i||_1 - ||X_+ u_i||_1, \forall i = 1,\ldots,k \tag{5}$$

where c_x, c_y are the centers of the zonotopes $\gamma(X)$, $\gamma(Y)$ respectively

Remark 4. For a zonotope of dimension p with n generators, the upper bound on the number of faces, and thus on vectors u_i, is $2\binom{n}{p-1}$. Thus, the complexity of the inclusion test is $2\binom{n}{p-1} \times \mathcal{O}(np)$ which improves on the exponential bound

of [2]. Indeed, the authors proved in [2] that $\gamma(X) \subseteq \gamma(Y)$, if and only if (5) is satisfied for all $u \in \mathbb{R}^p$. Lemma 1 shows that it is sufficient to check the inequality (5) for only a finite (polynomial in p) number of u. We can use singular value decomposition to compute the vectors u.

Meet. As observed in [2], zonotopes are closed under linear transformation and under the Minkowski sum, but the set-theoretic intersection of zonotopes is not always a zonotope [2]. Inspired from previous works in [9, 10] on zonotope and polyhedron intersection, a zonotope-zonotope intersection can be over-approximated by sequential computation of intersection of the zonotope and the half-spaces.

Several methods have been proposed to compute an over-approximation (e.g., in [11, 12, 10]) of the intersection of a zonotope and a linear space geometrically. Ghorbal et al. [13] proposed a method based on functional interpretation of the intersection of a zonotope with a guard. It computes a simple yet sufficiently precise over-approximation by using constrained affine sets: the constraints produced by the tests in a program are interpreted over the noise symbols of the affine forms.

Computing the meet of a zonotope with another one as the sequence of meet of the zonotope with the faces of the other can be imprecise, as the meet with linear space is an over-approximation, and imprecision will accumulate quickly. Hence, the need for a zonotope meet that can take into account all faces of the second argument at once. There are methods which directly focus on meets between zonotopes by set representations based on collections of sets. For instance, Althoff et al. [14] and Tommaso et al. [15] introduced zonotope bundles, defined as the intersection of a set of zonotopes. Note that the intersection is not computed explicitely. Rather the zonotopes are stored in a list and all operations are performed on individual zonotopes. These methods can be accurate, but the related cost increases with the number of sets required, which can be large.

Here we introduce a new geometric meet operation on zonotopes based on the following observation. Let $\mathfrak{Z}_1$ (resp. $\mathfrak{Z}_2$) be a zonotope represented by matrix M_1 (resp. M_2) and let x be a point in the intersection of $\mathfrak{Z}_1$ and $\mathfrak{Z}_2$. Then, there exists $e \in [-1,1]^p$ (resp. $e' \in [-1,1]^p$) such that $x = M_1^{\mathrm{T}} \begin{pmatrix} 1 \\ e \end{pmatrix}$ and $x = M_2^{\mathrm{T}} \begin{pmatrix} 1 \\ e' \end{pmatrix}$. For any $\alpha \in [0,1]$, trivially, $x = \alpha x + (1-\alpha)x$, therefore:

$$\mathfrak{Z}_1 \cap \mathfrak{Z}_2 \subseteq \left\{ \alpha M_1^{\mathrm{T}} \begin{pmatrix} 1 \\ e \end{pmatrix} + (1-\alpha) M_2^{\mathrm{T}} \begin{pmatrix} 1 \\ e' \end{pmatrix}, ||e||_\infty \leq 1, ||e'||_\infty \leq 1 \right\}.$$

The right hand side of the inclusion above is the zonotope obtained as the Minkowski sum of zonotope $\mathfrak{Z}_1$ (scaled by coefficient α) with zonotope $\mathfrak{Z}_2$ (scaled by coefficient $1-\alpha$), up to some translation.

Note that the meet operator introduces new directions of faces. As we will see, this gives more flexibility to our splitting operator, that uses a tiling to pave the zonotope obtained after the meet into sub-zonotopes. This approach

only provides an over-approximation of the intersection. Therefore, when using the algorithm of [1], we will not apply the tightening at each step, but only at the initial step. While this does not prevent the convergence of the algorithm, the resulting invariant set may be larger than with tightening at each step. The aim of the method being to prove efficiently that an initial target region is actually invariant by computing an inner inductive invariant, we did not consider tightening at each step a key feature.

Intersection test. In the generic algorithm of Sect. 2, we need to know when two abstract elements intersect. Consider a zonotope $\mathfrak{Z}_1$ given by its center c_1 and generators $g_1, \ldots, g_k$, and $\mathfrak{Z}_2$ given by c_2 and $h_1, \ldots, h_m$. As observed in [16], $\mathfrak{Z}_1 \cap \mathfrak{Z}_2 \neq \emptyset$ if and only if the point $c_1 - c_2$ is included in the zonotope centered at the origin, with generators $g_1, \ldots, g_k, h_1, \ldots, h_m$. This is a simple linear satisfiability problem.

Splitting. A zonotope can be viewed as the affine projection of a n-dimensional unit cube, n being the number of noise symbols, onto a p-dimensional space, p being the number of variables. Our first idea was thus to use a split operation on the unit cube in order to define a splitting operation on the resulting zonotope. However, this splitting method does produce overlapping zonotopes. The resulting algorithm was cumbersome and inefficient.

Another natural way to split zonotopes, without overlapping this time, is to use the property that zonotopes can be tiled, using generally more than 2 sub-zonotopes. These tiles are more precisely parallelotopes, as we describe below.

Consider a zonotope $\mathfrak{Z}(V)$ on a set of generators $V = (v_1, \ldots, v_n) \in \mathbb{R}^{p.n}$. A zonotopal tiling of $\mathfrak{Z}(V)$ is a set of tiles $\{Z_1, Z_2, \ldots, Z_M\}$ constructed from the vectors in V such that $\bigcup_{i=1}^{M} Z_i = \mathfrak{Z}(V)$. Provided with a sign vector $\sigma \in \{+, -, 0\}^n$ we can define a zonotope:

$$\mathfrak{Z}(\sigma) := \sum_{\{i|\ \sigma_i = 0\}} [-v_i, +v_i] + \sum_{\{i|\ \sigma_i = +\}} v_i - \sum_{\{i|\ \sigma_i = -\}} v_i \tag{6}$$

where $\mathfrak{Z}((0, 0, \ldots, 0))$ is the largest zonotope obtainable, i.e., $\mathfrak{Z}(V)$ and for all other sign vectors we obtain zonotopes which are contained in $\mathfrak{Z}(V)$. The zero entries of σ characterize the shape of $\mathfrak{Z}(\sigma)$ and the non-zero entries describe how $\mathfrak{Z}(\sigma)$ will be translated with respect to the origin. Thus, given a set of vectors $V \in \mathbb{R}^{p.n}$, which generates $\mathfrak{Z} := \mathfrak{Z}(V)$, we can associate a zonotopal tiling $\mathfrak{Z}(\sigma) \subseteq \mathfrak{Z}(V)$ with every sign vector $\sigma \in \{+, -, 0\}^n$. One such special kind of tiling is known as a parallelotope tiling, i.e., a tiling formed from all linearly independent subsets of $\{v_1, v_2, \ldots, v_n\}$.

In total, there are 2^n vertices in an n-dimensional hypercube, which can be projected with the generator matrix into the zonotope. All the extremal points of the zonotopes are projected hypercube vertices, but not all projected vertices are extremal in the zonotope. The parallelotopes in our tiling also have,

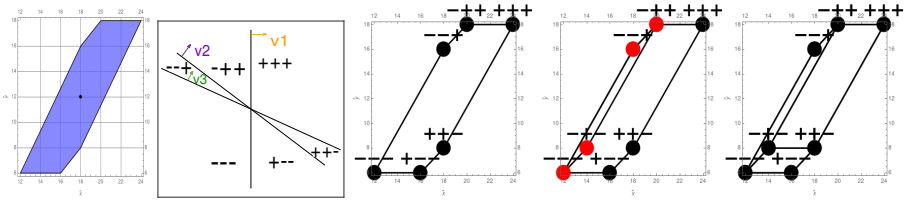


Fig. 3. (a) Zonotope concretization $\gamma(A)$; (b) Arrangement of hyperplanes; (c) Sign vectors; (d) Sign vectors after fixing the first generator to '-'; (e) All parallelotopic tiles.

as extremal points, projected vertices from the hypercube, some of which are internal points in the zonotope. Enumerating the parallelotopes tiles reduces to an enumeration of a set of hyper-cube vertices. We develop below an algorithm for tiling, which enumerate only the vertices characterizing the sub-zonotopes tiling a given zonotope. We use ideas issued from matroid theory established by Bohne-Dress theorem [17, 18], i.e., there is a close connection between a zonotope and the signs of its vectors since they abstract combinatorial facts about the structure of the zonotope. Thus, the key objective of the algorithm is to enumerate the vertices of the tiles as sign vectors of the so-called hyperplane arrangement [19] corresponding to a zonotope, that we are going to define now.

Hyperplane arrangements. A finite family $\mathcal{A}=\{h_j : j=1,\ldots,m\}$ of hyperplanes in $\mathbb{R}^p$ is called an arrangement of hyperplanes. Any hyperplane partitions the space $\mathbb{R}^p$ into three sets: $h_j^+=\{x \mid v_j^{\mathrm{T}}x>b_j\}$, $h_j^0=\{x \mid v_j^{\mathrm{T}}x=b_j\}$ and $h_j^-=\{x \mid v_j^{\mathrm{T}}x<b_j\}$. For each point x in $\mathbb{R}^p$, there is a sign vector $\sigma(x)\in\{+,-,0\}^n$ giving its relative location with respect to the hyperplane arrangement, defined as follows:

$$\sigma(x)_j = +, \text{if } x \in h_j^+ \quad \text{or} \quad -, \text{if } x \in h_j^- \quad \text{or} \quad 0, \text{if } x \in h_j^0 \tag{7}$$

The set of points with a given sign vector is an open polyhedron, whose faces of every dimension (including full dimensional p-dimensional cells partitioning the polyhedron) are determined by the intersection of some sets of the form h_j^0, h_j^- and h_j^+, hence are in one to one correspondence with sign vectors. Such a set is a cell if the corresponding sign vectors do not have zero entries. For a zonotope $\mathfrak{Z}=\mathfrak{Z}(V)$ generated by the columns of V, we define the associated central arrangement $\mathcal{A}=\mathcal{A}(V)$ of n hyperplanes in $\mathbb{R}^p$, each having v_j as its normal vector : $\mathcal{A}(V)=\{h_j^0 \mid j=1,2,\ldots,n\}$ where $h_j^0=\{x\in\mathbb{R}^p \mid v_j^{\mathrm{T}}x=0\}$ for $j=1,2,\ldots,n$. There is a duality relation between a zonotope and its corresponding hyperplane arrangement. Every d-dimensional open polyhedron in $\mathcal{A}$ determined by its sign vectors corresponds to a $p-d$-dimensional region of a zonotope where $d \leq p$; e.g., a full-dimensional open polyhedron corresponds to the vertices of the zonotope.

We denote by $\Sigma=\Sigma(V)$ a set of sign vectors of cells of the arrangement where each vector corresponds to a cell. For example, Fig. 3(b) illustrates an

arrangement of 3 hyperplanes in $\mathbb{R}^2$ for the vectors that define the generators of the zonotope in Fig. 3(a). Each cell (region between two adjacent hyperplanes) is represented by a sign vector of dimension 3 and corresponds to an extremal point of the zonotope as shown in Fig. 3(c). Furthermore, two extreme points in $\mathfrak{Z}$ are adjacent in $\mathfrak{Z}$ if and only if the associated cells are adjacent, i.e., if and only if their sign vectors are different in exactly one component. Finding the sign vectors is a cell enumeration problem. We use here a reverse search algorithm [19] that has a time complexity of $\mathcal{O}(n\ p\ LP(n,p)\ |\Sigma|)$ to compute $\Sigma{=}\Sigma(V)$ for any given rational $p \times n$ matrix V, where $LP(n,p)$ is the time to solve a linear programming problem with n inequalities in p variables.

Definition 1. *Let $\sigma \in \{+,-,0\}^n$ a sign vector. Fixing a single-element $j \in 1,\ldots,n$ means setting $\sigma_j \in \{+,-\}$ and defines a sub-zonotope*

$$\mathfrak{Z}(V \setminus j^{\{+,-\}}) := \sum_{\{i\neq j|\ \sigma_i=0\}} [-v_i,+v_i] + \sum_{\{i\neq j|\ \sigma_i=+\}} v_i - \sum_{\{i\neq j|\ \sigma_i=-\}} v_i + \sigma_j v_j \quad (8)$$

Freeing a single-element $j \in 1,\ldots,n$ means setting $\sigma_j = 0$ and defines a larger zonotope

$$\mathfrak{Z}(\hat{V}/j^{\{+,-\}}) := \sum_{\{i\neq j|\ \sigma_i=0\}} [-v_i,+v_i] + \sum_{\{i\neq j|\ \sigma_i=+\}} v_i - \sum_{\{i\neq j|\ \sigma_i=-\}} v_i + [-v_j,+v_j] \quad (9)$$

By Definition 1, we know that a sub-zonotope is obtained by fixing the sign of any one of the generators of a zonotope. Thus a parallelotopic tile is a zonotope with p free generators and $n-p$ fixed generators (p is the number of variables and n is the number of generators). Note that a sub-zonotope has some of its vertices in common with the vertices of the original zonotope and some inner vertices. Incrementally, if we keep fixing the sign of the generators until we enumerate the first parallelotopic tile then we have enumerated a sufficient number of inner vertices corresponding to the extremal points of each sub-zonotope to construct the remaining parallelotopic tiles. This is true by induction.

Algorithm. Fig. 4 illustrates a recursive algorithm for computing all the parallelotopic tiles characterizing a given p dimensional zonotope $\mathfrak{Z}{=}\mathfrak{Z}(c,v_1,v_2,\ldots,v_n)$. First, it checks if the input is already a tile i.e., $n = p$, then it returns the singleton containing the zonotope itself, otherwise it arbitrarily chooses a sign to fix the first generator. Fixing v_1 will produce a sub-zonotope defined by: $\mathfrak{Z}_{sub}{=}\mathfrak{Z}((\sigma_1,0,\ldots,0))$ computed according to (6) where σ_1 is either '+' or '-'. Then we make a recursive call of the tiling function on $\mathfrak{Z}_{sub}$ which computes its tiling and stores the result in T. The remaining step consist in finding all the adjacent parallelotopic tiles of $\mathfrak{Z}_{sub}$. First, we compute the sign vectors (Σ') of $\mathfrak{Z}_{sub}$ and prepend to them the sign of the first generator (σ_1) and add all the non-existent sign vectors to Σ. This corresponds to the extremal points of $\mathfrak{Z}_{sub}$ which are not extremal points of $\mathfrak{Z}$. We know that the first generator was fixed by computing $\mathfrak{Z}_{sub}$. Now, we free it and for each subset of generators $v_2 \ldots v_n$ of length $p{-}1$ we compute the parallelotopes for the p free generators $\{1\} \cup S$.

```
procedure TILINGS(ʒ)
    if n = p then
        return {ʒ}
    else
        Compute Σ = Σ(V)
        Compute ʒ_sub = ʒ((σ_1, 0, ..., 0))
        T = TILINGS(ʒ_sub)
        Σ' = Σ(V') where V' are the generators of ʒ_sub
        Prepend σ_1 to Σ', and add to Σ
        // Find all tiles adjacent to ʒ_sub
        for S in 2^{2,...,n} of length p − 1 do
            Find parallelotopic tiles for S in Σ
            Add to T
    return T
```

Fig. 4. Algorithm to compute the tilings of the zonotope $\mathfrak{Z}{=}\mathfrak{Z}(c, v_1, v_2, \ldots, v_n)$ defined by center c and the generators $v_1, v_2, \ldots, v_n{=}V$.

Example 1. Fixing to '-' the first generator of the zonotope $\mathfrak{Z}(V)$ of Figure 3(c), defined by $V{=}((2,0),(1,1),(3,5))$ and center $(18,12)$, we obtain the parallelotopic tile $\mathfrak{Z}(V \setminus 1^{-})$ shown in Fig. 3(d), where the extremal points are marked in red. The center of this tile is computed as $\big(18,12\big)+(-1)\big(2,0\big)+0\big(1,1\big)+0\big(3,5\big)$. Now, we shall be using this tile to generate the others.

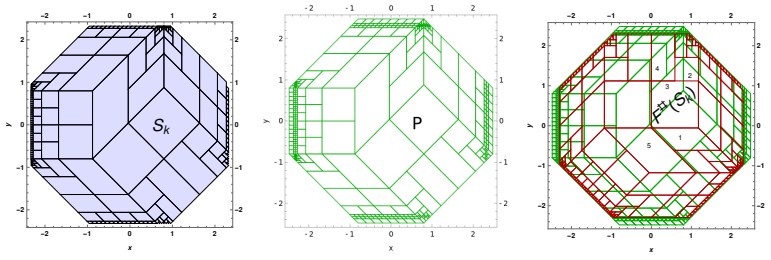
The generator $\big(2,0\big)$ was fixed in order to generate the parallelotopic tile. Now, we free this generator (v_1) and for each subset of generators v_2, v_3 of length $p{-}1$ we compute the parallelotopes. Consider the free generator combination (v_1, v_2) with the vector v_3 fixed. We search for the sign vectors of $\mathfrak{Z}(V)$ in which each individual fixed generator has the same sign. For instance the sign vectors: $(+,-,-),(-.-,-),(-,+,-),(+,+,-)$ characterize a parallelotope as shown in Fig. 3(e). Similarly, we can find the tile with vectors (v_1, v_3) free and v_2 fixed.

3.3 Coverage operation

Consider a set of zonotopes $G^\sharp$ in Fig. 5(a) partitioning an inductive invariant. Fig. 5(b) illustrates the set of partitions $B^\sharp$ which covers the whole invariant space. Consider the zonotope partition noted P in Fig. 5(b). Recall from Section 2 that the *contents-of* function corresponding to P (i.e., $\mathrm{cnt}(P)$) will return the parallelotope S_k marked in Fig. 5(a). The image of the zonotopes in Fig. 5(a) by a loop iteration is shown in edgeform color red in Fig. 5(c). We superimposed the images and the partitions to show which parts of $B^\sharp$ intersect the image of a zonotope $S_k \in G^\sharp$. That is the map post.

As discussed in Section 2 in the current work we only apply tightening during the first iteration. So, the benign test in Equation (3) can be reduced to

$$S_k \text{is benign} \iff \forall P \in \mathrm{post}(S_k) : \mathrm{cnt}(P) \neq \emptyset \wedge F^\sharp(S_k) \subseteq T \tag{10}$$



(a) (b) (c)

Fig. 5. (a) Set of zonotopes $G^\sharp$ partitioning an inductive invariant; (b) Partitions $B^\sharp$; (c) Map post

i.e., a zonotope S_k is benign if $F^\sharp(S_k)$ is included within the candidate invariant, and if, when $F^\sharp(S_k)$ intersects some zonotope $P \in B^\sharp$, we have that $\text{cnt}(P)$ is not the emptyset. For instance, consider the zonotope S_k shown in Fig. 5(a) (the parallelotope labeled as S_k) and its image under $F^\sharp$, the zonotope labeled as $F^\sharp(S_k)$ shown in Fig. 5(c). The sub-parallelotopes numbered 1, 2, up to 5 in Fig. 5(c) are thus in $\text{post}(S_k)$. Thus, the benign test for S_k consists in checking whether these partitions contain zonotopes from $G^\sharp$, and whether the image of S_k is entailed within the initial invariant, which are true here.

Recall that, to compute coverage, we actually only need some approximation (with bounded ratio), so we use an heuristic measure instead of computing volume: we count the number of zonotopes $P \in B^\sharp$ which intersect $F^\sharp(S_k)$, i.e., $\#\{P | P \in \text{post}(S_k)\}$ and then, among these zonotopes, we count the ones for which $\text{cnt}(P) \neq \emptyset$, i.e., we compute $\#\{P | \text{cnt}(P) \neq \emptyset, P \in \text{post}(S_k)\}$. Our heuristic measure is thus:

$$\text{coverage}(S_k) := \frac{\#\{P | \text{cnt}(P) \neq \emptyset, P \in \text{post}(S_k)\}}{\#\{P | P \in \text{post}(S_k)\}} \tag{11}$$

4 Experiments

A prototype analyzer was written in OCaml for the algorithm of [1] using the box and octagon abstract domains available in the Apron library [6]. We adapted this prototype analyzer, as described in Section 3, in order to use the zonotopic abstract domain implementation Taylor1+ [5] and polyhedra abstract domain New Polka from Apron. We added the operations split, intersection, coverage measure and the new inclusion test of Section 3 in Taylor1+[3]. This adapted version is available online[4].

We have chosen hard non-linear problems[5] from various articles on non-linear numeric invariant inference. Most of these examples belong to SV-COMP

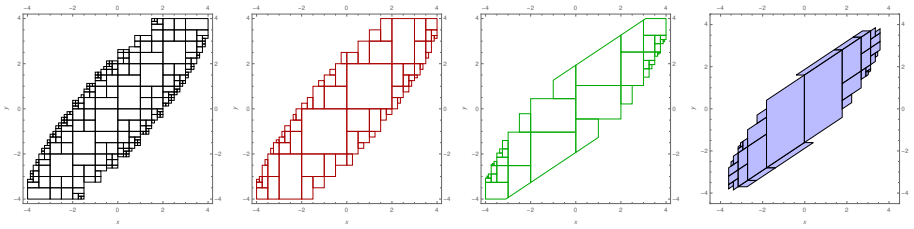
[3] `https://github.com/bibekkabi/taylor1plus`

[4] `https://github.com/bibekkabi/Prototype_analyzerwithApron`

[5] `https://github.com/bibekkabi/Prototype_analyzerwithApron/tree/master/NSV_files`

Table 1. Experimental results with tightening applied only during first iteration.

Program	Boxes			Octagons			Zonotopes			Polyhedra		
	#elems.	#iters.	time(s)	#elems.	#iters.	time(s)	#elems.	#iters.	time(s)	#elems.	#iters.	time(s)
Octagon	752	2621	0.1042	752	2756	0.6115	**1**	**1**	**0.0001**	**1**	**1**	**0.0001**
Filter	238	1310	**0.1029**	74	736	0.2105	**38**	**222**	0.5020	42	312	0.2554
Filter2	14	58	0.0034	7	13	0.0013	8	16	0.0045	**1**	**1**	**0.0009**
Arrow-Hurwicz	1784	1643	0.4033	369	931	0.5147	**15**	**38**	**0.0235**	134	484	1.0059
Harm	87	438	**0.0112**	88	448	0.0647	60	254	0.5143	**53**	**243**	0.2442
Harm-reset	87	438	**0.0204**	88	446	0.1478	60	268	0.9717	**53**	**253**	0.3867
Harm-saturated	23	15	**0.0011**	24	16	0.0112	9	14	0.0157	**5**	**9**	0.0124
Sine	240	1448	0.4395	154	348	0.1102	**21**	**33**	**0.0547**	136	286	1.1145
Square root	7	10	0.0005	4	4	0.0016	**1**	**1**	**0.0001**	4	4	0.0066
Newton	200	102	0.1097	158	76	0.1785	**11**	**17**	**0.0197**	64	26	2.0660
Newton2	1806	499	6.6861	709	430	2.2207	**8**	**6**	**0.0193**	12	12	2.7498
Corner	129781	1847	646.8494	129767	1847	8850.8766	**488**	**999**	**35.6245**	2368	4248	126.7980



(a) (b) (c) (d)

Fig. 6. Inductive invariant for *Filter* example (a) 238 boxes 1310 iterations, 0.1029 s; (b) 74 octagons, 736 iterations, 0.2105 s; (c) 42 polyhedra, 312 iterations, 0.2554 s; (d) 38 zonotopes, 222 iterations, 0.5020 s.

benchmarks[6], e.g., Sine, Square root, Newton, Newton2, Filter. In Table 1, we compare on some small but challenging loops the results of the algorithm applied to boxes, octagons, polyhedra and zonotopes. For each abstract domain, we give the number of iterations and time until a first inductive invariant is found, and the number of elements that compose this invariant. For each example, we highlight in bold in Table 1 the entries corresponding to the smallest number of elements, iterations, or execution time.

Example *Octagon* in Table 1 corresponds to the motivational example from [1]. Its loop body performs a 45-degree rotation around the origin, with a slight inward scaling. The initial element obtained after the first tightening step is already inductive with zonotopes. The classical abstract semantics for addition and subtraction on octagons is too coarse to prove this is an inductive invariant, explaining why the analyzer had to iterate a lot, contrarily to the zonotopic case.

Filter is a second-order digital filter taken from [1]. The candidate invariant provided to the algorithm is $[-4, 4]$ which is not inductive. Figs. 6(a), 6(b), 6(c), and 6(d) compare the result of the algorithm on the *Filter* program using intervals, octagons, polyhedra and zonotopes. The natural inductive invariants for such filters are ellipsoids. The inductive invariant found within each abstraction is indeed an approximation of an ellipsoid. It is composed of fewer zonotopes

[6] `https://github.com/sosy-lab/sv-benchmarks/tree/master/c/floats-cdfpl`

and polyhedra than boxes and, to a lesser extent, octagons, and requires fewer iterations to be synthesized. For *Filter2* from [20], our inductive invariant shows that x and y remain within $[-0.2, 1]^2$.

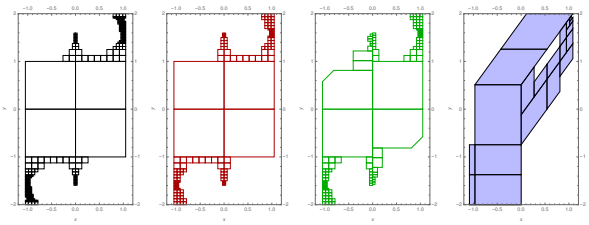
We analyzed the *Arrow-Hurwicz* loop taken from [20] as a two variable program. The algorithm with boxes, octagons, polyhedra and zonotopes was able to verify that the variables remain within the bound $[-1.73, 1.73]^2$. The analysis with zonotopes was faster and generated far fewer abstract elements compared to other domains.

Harm is an harmonic oscillator program from [20]. Its loop body is close to the identity. The programs *Harm-reset* and *Harm-saturated* add some nondeterminism in the loop body. The polyhedra require fewer elements and iterations, but more time, compared to boxes and octagons. On simple cases, the use of complicated abstractions is not competitive, as expected.

For the lead-lag controllers (programs *Lead-lag*, *Lead-lag-reset* and *Lead-lag-saturated* from [21]), none of the abstract domains could find an inductive invariant before timeout. The case of the dampened oscillators [21] fails similarly.

Sine, *Square root*, *Newton*, *Newton2* (taken from [22]) and *Corner* (Fig. 1(a)) are programs with non-linear loop bodies. *Sine* and *Square root* compute the corresponding mathematical functions through Taylor expansions, while *Newton* and *Newton2* perform one step and two steps of Newton solving respectively. Zonotopes are the fastest on all these examples: they require much fewer iterations and elements compared to intervals, octagons and polyhedra.

For example, Figure 7(a), 7(b), 7(c) and 7(d) compare the result of the algorithm on the *Sine* program using intervals, octagons, polyhedras and zonotopes. Our zonotope-based method is better when aiming to prove as fast as possible



(a) (b) (c) (d)

Fig. 7. Inductive invariant for *Sine* example (a) 240 boxes 1448 iterations, 0.4395 s; (b) 154 octagons, 348 iterations, 0.1102 s; (c) 136 polyhedras, 286 iterations, 1.1145 s; (d) 21 zonotopes, 33 iterations, 0.0547 s.

that the initial invariant holds, strengthening it into an inductive invariant. Indeed, with a better interpretation of non affine operations, less splitting steps are needed before getting an inductive invariant. For example, zonotopes enabled us to prove the initial invariant as inductive for the *Square root* program.

These experiments confirm that zonotopes provide a very interesting trade-off between a general purpose abstraction, that stand the comparison with simpler

abstractions on basic linear examples, but are also faster and more flexible for the abstraction of more complex, non affine behaviors. In particular, the fact that they allow representing inductive invariants with fewer elements will be even more crucial for the scalability of the approach to higher dimension programs.

5 Conclusion and future work

In this paper, we investigated the use of constraint-solving inspired algorithms for inferring inductive invariants. The algorithm works by iteratively splitting and tightening a set of abstract elements until an inductive invariant is found. Our main contribution was to extend the type of abstract elements this algorithm can rely on, in particular when there is no natural bisection method. We instantiated it to the case of zonotopes, and demonstrated that they provide a good trade off, in particular on non-linear programs, and scale up much better than the same algorithm based on simpler domains, such as boxes. Future work includes the use of such techniques to infer invariants of continuous systems and higher dimensional programs.

Acknowledgement This work is being supported by project ANR-15-CE25-0002-01 COVERIF

References

1. Miné, A., Breck, J., Reps, T.: An algorithm inspired by constraint solvers to infer inductive invariants in numeric programs. In: Proceedings of ESOP. (2016) 560–588
2. Goubault, E., Putot, S.: A zonotopic framework for functional abstractions. Formal Methods in System Design **47**(3) (2015) 302–360
3. Garg, P., Löding, C., Madhusudan, P., Neider, D.: Ice: A robust framework for learning invariants. In: Proceedings of CAV, Springer (2014) 69–87
4. Thakur, A., Lal, A., Lim, J., Reps, T.: Posthat and all that: Automating abstract interpretation. Electronic Notes in Theoretical Computer Science **311** (2015) 15–32
5. Ghorbal, K., Goubault, E., Putot, S.: The zonotope abstract domain taylor1+. In: International Conference on Computer Aided Verification, Springer (2009) 627–633
6. Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: Proceedings of CAV, Springer (2009) 661–667
7. Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings of POPL, ACM (1977) 238–252
8. Stolfi, J., De Figueiredo, L.H.: Self-validated numerical methods and applications. In: Monograph for 21st Brazilian Mathematics Colloquium, IMPA. (1997)
9. Le, V.T.H., Stoica, C., Alamo, T., Camacho, E.F., Dumur, D.: Zonotope-based set-membership estimation for multi-output uncertain systems. In: 2013 IEEE International Symposium on Intelligent Control (ISIC), IEEE (2013) 212–217
10. Tabatabaeipour, S.M., Stoustrup, J.: Set-membership state estimation for discrete time piecewise affine systems using zonotopes. In: 2013 European Control Conference (ECC), IEEE (2013) 3143–3148

11. Girard, A., Le Guernic, C.: Zonotope/hyperplane intersection for hybrid systems reachability analysis. In: International Workshop on Hybrid Systems: Computation and Control, Springer (2008) 215–228
12. Combastel, C., Zhang, Q., Lalami, A.: Fault diagnosis based on the enclosure of parameters estimated with an adaptive observer. IFAC Proceedings Volumes **41**(2) (2008) 7314–7319
13. Ghorbal, K., Goubault, E., Putot, S.: A logical product approach to zonotope intersection. In: Proceedings of CAV. (2010) 212–226
14. Althoff, M., Krogh, B.H.: Zonotope bundles for the efficient computation of reachable sets. In: 2011 50th IEEE conference on decision and control and European control conference, IEEE (2011) 6814–6821
15. Dreossi, T., Dang, T., Piazza, C.: Parallelotope bundles for polynomial reachability. In: Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, ACM (2016) 297–306
16. Guibas, L.J., Nguyen, A., Zhang, L.: Zonotopes as bounding volumes. In: Proceedings of the ACM-SIAM symposium on Discrete algorithms. (2003) 803–812
17. Bailey, G.D.: Tilings of Zonotopes: Discriminantal Arrangements, Oriented Matroids and Enumeration. University of Minnesota (1997)
18. Richter-Gebert, J., Ziegler, G.M.: Zonotopal tilings and the bohne-dress theorem. Contemporary Mathematics **178** (1994) 211–211
19. Ferrez, J.A., Fukuda, K., Liebling, T.M.: Solving the fixed rank convex quadratic maximization in binary variables by a parallel zonotope construction algorithm. European Journal of Operational Research **166**(1) (2005) 35–50
20. Adjé, A., Gaubert, S., Goubault, E.: Coupling policy iteration with semi-definite relaxation to compute accurate numerical invariants in static analysis. In: European Symposium on Programming, Springer (2010) 23–42
21. Roux, P., Garoche, P.L.: Practical policy iterations. Formal Methods in System Design **46**(2) (2015) 163–196
22. D'Silva, V., Haller, L., Kroening, D., Tautschnig, M.: Numeric bounds analysis with conflict-driven learning. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer (2012) 48–63