Set-Based Invariants over Polynomial Systems

Alberto Casagrande[1,*], Alessandro Cimatti[2], Luca Dorigo[2,3], Carla Piazza[3] and Stefano Tonetta[2]

[1]Dip. di Matematica e Geoscienze, Università di Trieste, via Alfonso Valerio 12/1, 34127, Trieste, Italy
[3]Dip. di Matematica, Informatica e Fisica, Università di Udine, via delle Scienze 205, 33100, Udine, Italy
[2]Fondazione Bruno Kessler, Via Sommarive 18, 38123 Povo, Trento, Italy

Abstract

Dynamical systems model the time evolution of both natural and engineered processes. The automatic analysis of such models relies on different techniques ranging from reachability analysis, model checking, theorem proving, and abstractions. In this context, invariants are subsets of the state space containing all the states reachable from themself. The verification and synthesis of invariants is still a challenging problem over many classes of dynamical systems, since it involves the analysis of an infinite time horizon. In this paper we propose a method for computing invariants through sets of trajectories propagation. The method has been implemented and tested in the tool Sapo which provides reachability methods over discrete time polynomial dynamical systems.

Keywords

Dynamical Systems, Infinite State Systems, Invariants, K-Induction

1. Introduction

Invariants are subsets of the state space of dynamical systems that contain all the states reachable from themself. This notion is crucial in may fields such as control theory [1], software verification [2, 3], and analysis of both continuous and hybrid systems [4, 5].

Many studies investigating the verification and synthesis of invariants have been published in the literature so far. Some of them deals with polynomial systems and investigated semi-algebraic invariants [4, 6, 7]. Some others focus on invariant sets representable by using polytopes [8, 5] to avoid the double exponential decision procedure of the semi-algebraic theory. The vast majorities of the works in the literature take advantage of the system continuity to either validate or identify an invariant set.

This work focuses on discrete-time transition systems. The applications of this kind of systems go from biology [9, 10] to engineering [11, 12]. We rely on a strengthening of a special case of induction, k-inductiveness [13], to validate a possible invariant P. We start from an initial set of states and we enlarge it to include all the states reachable in $k-1$ discrete transitions.

CILC'23: 38th Italian Conference on Computational Logic, June 21–23, 2023, Udine, Italy
*Corresponding author.
acasagrande@units.it (A. Casagrande); cimatti@fbk.eu (A. Cimatti); dorigo.luca001@spes.uniud.it (L. Dorigo); carla.piazza@uniud.it (C. Piazza); tonetta@fbk.eu (S. Tonetta)
0000-0002-8681-1482 (A. Casagrande); 0000-0002-1315-6990 (A. Cimatti); 0000-0002-1396-2167 (L. Dorigo); 0000-0002-2072-1628 (C. Piazza); 0000-0001-9091-7899 (S. Tonetta)

This is the candidate invariant and it has to be included in the invariant P. If this is the case, we have to prove that if p is a path of length k starting from P and having the first $k-1$ steps in P itself, then p does not exit from P.

Other approaches involving k-inductiveness have been proposed in the literature (see, e.g., [14, 15]). However, they assume the possibility of computing the reverse transition, which is not always possible when dealing with discrete polynomial systems. Indeed our goal was to develop a general method for (possibly) non-reversible discrete transition systems.

When all operations are computed without approximations the proposed method converges if and only if reachability converges within a finite number of steps. Surprisingly, when reachability, unions and intersections are over-approximated, convergence is achieved on a larger class of systems.

The paper is organized as follows. Section 2 introduces the problem, the notation, and the basic tools. Section 3 incrementally presents our proposal for a set-based algorithm deciding k-inductiveness. In Section 4 we detail the approach over polynomial systems as implemented in Sapo and discuss an example. Section 5 draws concluding remarks.

2. Preliminaries

In this section, we present some central notions for this work.

Definition 1 (Transition System). *A* transition system *is a pair $\langle S, R\rangle$, where S is the set of states and $R \subseteq S \times S$ is the transition relation. An* initial set *for a transition system TS $= \langle S, R\rangle$ is a set of states $I \subseteq S$.*

The transition relation R induces a *transition function* T that maps any set $A \subseteq S$ into the *forward image of* A itself, i.e. $T(A) \stackrel{\text{def}}{=} \{s \in S \mid \exists s' \in A \ \ (s', s) \in R\}$.

For any function $F : 2^S \to 2^S$ mapping a subset of the states into another subset of the states, we define the F *image of* A *restricted to* B as $F_B(A) \stackrel{\text{def}}{=} F(A) \cap B$. We write $F^k(A)$ to denote k consecutive applications of F to A, i.e.,

$$F^k(A) \stackrel{\text{def}}{=} \begin{cases} A & \text{if } k = 0 \\ F\big(F^{k-1}(A)\big) & \text{if } k > 0 \end{cases}.$$

So, $T_B(A)$, $T^k(A)$ and $T^k_B(A)$ stand for the forward image of A restricted to B, k consecutive applications of the forward image, and k consecutive applications of the forward image restricted to B, respectively.

A set C *under-approximates* a set C' when $C \subseteq C'$. A set C *over-approximates* a set C' when $C \supseteq C'$. A function $T' : \mathcal{D} \to \mathcal{D}$ *under-approximates* (*over-approximates*) a function T'' when $T'(A) \subseteq T''(A)$ ($T'(A) \supseteq T''(A)$, respectively) for all $A \in \mathcal{D}$. It is easy to see that, if $\overline{T}$ over/under-approximates T, then $\overline{T}_B$, $\overline{T}^k$, and $\overline{T}^k_B$ do the same for T_B, T^k, and T^k_B, respectively.

Definition 2 (Reachability). *Given a transition system TS and an initial set I, the set of states* reachable from I in k-steps *is the set $T^k(I)$ and the set of states* reachable from I *is $\bigcup_{k\in\mathbb{N}} T^k(I)$.*

A property for a system is a set of states of the system. An invariant for a system and an initial set of states, I, is a property that contains all the states reachable from I.

Definition 3 (Property and Invariant). *Let TS $= \langle S, R \rangle$ be a transition system and let $I \subseteq S$ be an initial set of states for TS. A* property *of TS is a set of states $P \subseteq S$. An* invariant *for TS and I is a set $P \subseteq S$ such that*

$$\bigcup_{k \in \mathbb{N}} T^k(I) \subseteq P. \tag{1}$$

A well-know problem in the context of verification is that of checking whether a given property P is an invariant for a system (see, e.g., [7, 8, 5]).

One could consider the problem of constructing the smallest invariant for a system. The problem is well defined since if both P_1 and P_2 are invariants for TS and I, then also $P_1 \cap P_2$ is an invariant. In other terms, a given property P is an invariant for the system if and only if it includes the smallest invariant. However, as one could imagine finding the smallest invariant could be far more difficult than checking whether a given property is an invariant.

In this paper, we will describe a method for checking whether a property is an invariant, while trying to construct the smallest invariant. The method has to work also on infinite state systems assuming that only a restricted number of operations are computable. For instance, we will be able to over-approximate the forward image computation, but not the backward one. This is a reasonable compromise in term of computability and efficiency in the case of non-linear systems [16].

K-Inductiveness

A standard approach for verifying invariants is the use of *k-induction* [13] in combination with *Bounded Model Checking (BMC)* [17, 18]. If TS reaches from I a state outside P in k steps, i.e., $T^k(I) \not\subseteq P$, then P is not an invariant and we have a counter-example of length k. If this is not the case, then we try to prove that P is k-inductive, which implies that it is an invariant.

We now briefly explain the meaning of k-inductiveness. By Equation 1 and by induction principle over the naturals, P is an invariant if and only if

$$\begin{array}{cc} T^0(I) \subseteq P & \textit{(basis)} \\ T^i(I) \subseteq P \rightarrow T^{i+1}(I) \subseteq P & \textit{(step)} \end{array}$$

Induction and k-induction are equivalent over $\mathbb{N}$. Thus, P is an invariant if and only if

$$\begin{array}{cc} \bigwedge_{j=0}^{k-1} T^j(I) \subseteq P & \textit{(k-basis)} \\ \left(\bigwedge_{j=0}^{k-1} T^{i+j}(I) \subseteq P\right) \rightarrow T^{i+k}(I) \subseteq P & \textit{(k-step)} \end{array}$$

Unfortunately, the inductive k-step has to be proven for a generic $i \in \mathbb{N}$, which makes the technique not algorithmic for infinite systems. A stronger approach, named k-inductiveness, can be used to automatize invariant verification. Intuitively, k-inductiveness strengthens k-steps by requiring that all the paths laying inside P for $k-1$ steps stay inside P also after k steps, no matter whether they started from I or not. Considering all the possible paths starting from P and not only those from I allows to drop the dependence on i. The following definition is a set-based version of the definition given in [19].

Definition 4 (k-Inductive Property). *Given a transition system TS and an initial set I, a* k-inductive property *for TS and I is a set of states $P \subseteq S$ such that*

$$\bigwedge_{j=0}^{k-1} T^j(I) \subseteq P \qquad (k\text{-init})$$
$$T(T_P^{k-1}(P)) \subseteq P \qquad (k\text{-ind})$$

It is well known that k-inductiveness implies invariance.

Lemma 2.1. *Let TS be a transition system and I be and initial set for TS. If P is k-inductive for TS and I, then P is an invariant for TS and I.*

The reader should be aware that this is a specialized use of induction: a property could be an invariant and not k-inductive. Moreover, a set could be $k+1$-inductive and not k-inductive.

Example 1. *Let TS be the transition system $\langle S, R\rangle$, with $S = \{0,1,2,3\}$ and $R = \{(0,0), (1,3), (3,2), (2,2)\}$, and let I be the set $\{0\}$. The set $\{0,1\}$ is an invariant for TS and I, but it is not k-inductive for any k. Moreover, $P = \{0,1,2\}$ is 2-inductive because $T_P^1(P) = T(P) \cap P = \{0,2,3\} \cap \{0,1,2\} = \{0,2\}$ and $T(\{0,2\}) = \{0,2\} \subseteq P$. However, P is not 1-inductive. As a matter of the fact, $T_P^0(P) = P$ and $T(P) = \{0,2,3\} \not\subseteq P$.*

If k-*init* does not hold, not only P is not k-inductive, but it is not an invariant. If both k-*init* and k-*ind* succeed, P is k-inductive and it is an invariant. If k-*init* succeeds, but k-*ind* does not, then P is not k-inductive, but it could be $k+1$-inductive, i.e., k has to be incremented. This forces Algorithm 1 to return `True` if and only if there exists k such that P is k-inductive. Using two auxiliary variables, we can implement Algorithm 1 to require only 2 forward images, 1 intersection, and 2 subset tests at each iteration. Unfortunately, Algorithm 1 is not complete and, if P is not k-inductive for any k, but it is an invariant, it does not terminate.

Algorithm 1 K-Inductiveness

```
1:  function k-INDUCTIVE(TS = ⟨S, R⟩, I, P)
2:      if I ⊈ P then
3:          return False
4:      k ← 1
5:      while True do
6:          if T^k(I) ⊈ P then
7:              return False
8:          if T(T_P^{k-1}(P)) ⊆ P then
9:              return True
10:         k ← k + 1
```

Set-Based Reachability

In formal verification of infinite state systems and more in general in the context of symbolical representations of transition systems, traces representing punctual evolutions are not always

feasible because of the infinite domain. Processing sets of traces, potentially infinite in number, even though involving a finite set of states, is certainly computationally more expensive than computing the evolution of sets consisting in an infinite number of states. Moreover, it is usually not reasonable to work on single points/traces when we do not have knowledge of the system at infinite precision level [20]. In such a context, *set-based analysis* aims at defining techniques for dealing with possible infinite sets of states, processing each set as a single object.

In particular, in the case of infinite states systems and reachability computation a generic set-based technique depends on the choice of:

- A domain $\mathcal{D} \subseteq 2^S$ providing finite representations for the set of points to be manipulated. The most used domains are ellipsoids, polytopes, zonotopes.
- Methods for computing/approximating basic set operations and tests over $\mathcal{D}$, e.g., union, intersection, and inclusion.
- A family $\mathcal{T}$ of admissible transition relations. For instance, linear or polynomial systems could be considered.
- For all the transition functions $T \in \mathcal{T}$, a method $\overline{T} : \mathcal{D} \to \mathcal{D}$ that computes/approximates the image of any set $S \in \mathcal{D}$ through T.

Such tools are sufficient to obtain an implementation of Algorithm 1 instantiated on a specific domain. In particular, as far as basic set operations are concerned, it only requires the computation/approximation of intersections and the tests of inclusion. If both forward images and intersections are over-approximated and inclusion tests never provide false positive answers, then the procedure returns `True` if and only if P is k-inductive for some k.

It is worth to notice that computing backward images, which could be problematic in the case of non-linear transition relations, is not required. This is a crucial difference with respect to algorithms such as IC3/PDR [14, 15] which use transition reversibility to compute counter-examples and refine over-approximations of the reachability sets.

Algorithm 1 has two main flaws: 1) if P is an invariant, but it is not k-inductive for any k, Algorithm 1 does not terminate; 2) the algorithm is not able to refine P and discover possible k-inductive invariant included in P. Section 3 tries to overcome both issues by presenting algorithms that, along the computation, search for the "smallest" invariant included in P that is also k-inductive for some k.

3. Set-Based K-Inductiveness

We propose a set-based algorithm that takes in input a system TS, a set of initial states I and a property P and aims at constructing a k-inductive property CI, *candidate inductive*, included in P. Intuitively, if TS reaches a state outside P in k transition steps from I, then P is not an invariant and the algorithm returns `False`. If this is not the case, then the algorithm considers the candidate invariant CI built so far. CI has two main properties: it is included in P and it includes all the states reachable from I in $k-1$ steps. If CI is k-inductive, then P is an invariant and the algorithm returns `True`. Otherwise, CI is augmented with the states reachable in k-steps and k is incremented. If P is an invariant, but it does not includes any k-inductive set, then the algorithm does not terminate. These ideas are formalized in Algorithm 2 which describes

sets by using a finite representation domain $\mathcal{D}$ and the method $\overline{T}$ to compute/approximate the images of T as discussed in Section 2. In particular, while T maps any subsets of S in another subset of S according to R, $\overline{T} : \mathcal{D} \to \mathcal{D}$ with $\mathcal{D} \subseteq 2^S$.

Algorithm 2 Set–based K–inductiveness

```
1:  function CANDIDATE-INDUCTIVE(TS = ⟨S, R⟩, I, P)
2:      if I ⊈ P then
3:          return False
4:      k ← 1
5:      CI ← I
6:      while True do
7:          if T̄^k(I) ⊈ P then
8:              return False
9:          if T̄(T̄_CI^{k−1}(CI)) ⊆ CI then
10:             return True
11:         CI ← CI ∪ T̄^k(I)
12:         k ← k + 1
```

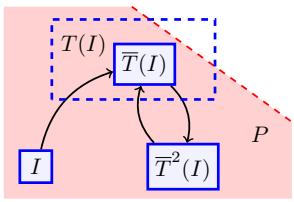
It is easy to see that the variable CI in Algorithm 2 accumulates the reachability set of TS and I as the computation proceeds. In particular, if $\overline{T}$ under/over-approximates T, then CI under/over-approximates $\bigcup_{j=0}^{k-1} T^j(I)$. This observation allows us to prove some properties about the results of Algorithm 2.

Theorem 3.1 (Correctness of Algorithm 2). *If $\overline{T}$ under-approximates T and Algorithm 2 returns* `False`*, then P is not an invariant for TS and I. If $\overline{T}$ over-approximates T and Algorithm 2 returns* `True` *after k while-loop iterations, then CI is k-inductive for TS and I and P is an invariant for them. Furthermore, if $\overline{T}(A) = T(A)$ for all $A \in \mathcal{D}$, Algorithm 2 returns* `False` *if and only if P is not an invariant for TS and I and, when it returns* `True`*, any $C \subset$ CI is not an invariant for TS and I.*

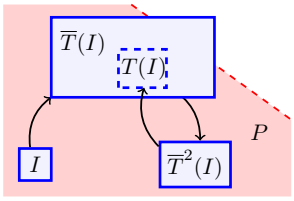
Let us underline that both the termination of Algorithm 2 and the meaning of its outcomes depend on the approximation of T done by $\overline{T}$. Figure 1 depicts two examples in which Algorithm 2 terminates, but does not return the aimed answer.

When instead $\overline{T}$ exactly represents T, Algorithm 2 does not terminate only when P is an invariant of the system by Theorem 3.1. Moreover, when the algorithm loops forever, the set of points reachable from I does not converge after a finite number of while-loop iterations. Indeed, if $\bigcup_{j\in\mathbb{N}} T^j(I) = \bigcup_{j\leq h} T^j(I)$ for some h, then CI $= \bigcup_{j\in\mathbb{N}} T^j(I)$ after h iterations, CI is h-inductive, and the algorithm returns `True`. Hence, if we are assuming that all set-operations are implemented without approximations and $\overline{T}$ exactly represents T, Algorithm 2 terminates if and only if either P is not an invariant or the set of reachable states can be computed with a finite number of forward transitions.

Corollary 3.1.1 (Termination). *If $\overline{T}(A)$ exactly represents $T(A)$ for any $A \in \mathcal{D}$, then Algorithm 2 terminates if and only if either P is not an invariant or there exists h such that $\bigcup_{j\in\mathbb{N}} T^j(I) = \bigcup_{j\leq h} T^j(I)$.*



(a) Algorithm 2 returns `True` even though P is not an invariant for TS and I when $\overline{T}$ under-approximates T. However, $T(I)$ is not a subset of P and, as a consequence, P is not an invariant for TS and I.



(b) Algorithm 2 returns `False` even though P is an invariant for TS and I when $\overline{T}$ over-approximates T. However, $T(I)$ is a subset of P and $\overline{T}^3(I) = T(I)$. Thus, P is an invariant for TS and I.

Figure 1: Two scenarios in which, when the forward image function is approximated, Algorithm 2 terminates, but does not return the aimed answer. The dashed blue line rectangle depicts $T(I)$.

While the number of while-loop iterations of Algorithm 2 cannot be upper-bounded in the general case, we can provide a cost depending on the number of iterations for it. Each repetition of the while-loop in Algorithm 2 requires $\Theta(k)$ forward image computations, 2 inclusion tests, $\Theta(k)$ intersection and 1 union. Such operations are denoted as set-operations, since they transform sets of states.

Theorem 3.2 (Set Complexity). *If Algorithm 2 terminates exactly after k iterations of the while-loop, then it requires $\Theta(k^2)$ set operations.*

Without increasing the asymptotic complexity of the algorithm we could modify line 9 by considering the test

$$\overline{T}(\overline{T}_{\text{CI}}^{k-1}(\text{CI})) \subseteq \text{CI} \ \vee \ \overline{T}(\overline{T}_{P}^{k-1}(P)) \subseteq P,$$

i.e., at each step we also check whether P is k-inductive. This would guarantee termination also in the case in which P is k-inductive. However, we are interested in classes of systems where it is not reasonable to assume that P can be represented/approximated in the domain $\mathcal{D}$ of sets, for instance, P and $\mathcal{D}$ can be a half-space and the set of all the ellipsoids in the same space, respectively. This was one of the reasons for which we moved from Algorithm 1 to Algorithm 2.

Theorem 3.2 provides a complexity bound for Algorithm 2 in terms of set operations. However, the cost of each operation actually depends on the complexity of the involved sets. For instance, if U and W are the union of k_1 and k_2 basic sets, respectively, e.g., $U = \bigcup_{i=1}^{k_1} U_i$ and $W = \bigcup_{j=1}^{k_2} W_j$, the evaluation of $U \cap W$ may consists in up to $k_1 * k_2$ basic sets: the non-empty intersections between each U_i and each W_j. In the context of Algorithm 2, if we assume that both I and its forward image are basic sets, after k iterations of the algorithm while-loop, CI consists in the list $[\overline{T}^0(I), \ldots, \overline{T}^k(I)]$. Thus, the evaluation of $\overline{T}_{\text{CI}}^{k-1}(\text{CI})$ at line 9 may produce a set consisting of up to k^k basic sets and, as a consequence, requires $O(k^k)$ basic set operations.

The complexity of Algorithm 2 in terms of basic set operations pushes us to either avoid the test of line 9 or replace the exact union performed at line 11 with an approximate version of it that reduces the number of basic sets in CI.

Over-Approximation of the Union

In order to overcome the limits pointed out by both Corollary 3.1.1 and other complexity issues discussed at the end of the previous section, we consider a different approach that aims to reduce the number of basic sets in CI. This is done by replacing the line 11 in Algorithm 2 with the assignment:

$$\text{CI} \leftarrow \texttt{Join}(\text{CI}, \overline{T}^k(J)),$$

where the operator `Join` over-approximates the set union and returns one single basic set and J is a new variable initially set to I. Please notice that we are not forcing `Join` to be a specific function, but instead we are requiring it to return one single basic set over-approximating the union of its parameters. For instance, depending on the adopted set representation, `Join` may be implemented by the convex-hull function, the bounding box function, or the Halbwachs widening operator [21].

Due to the proposed change, CI may be not included in P even when $\overline{T}^j(J) \subseteq P$ for all $j \in \mathbb{N}$ (e.g., see Figure 2). When this happens we can either return `Unknown`, since we do not have a counter-example, but our k-inductive set is growing outside P, or re-initialize J, CI, and k. Algorithm 3 implements this approach.

Algorithm 3 Set-based Join K-inductiveness

```
1:  function JOIN-CANDIDATE-INDUCTIVE(TS = ⟨S, R⟩, I, P)
2:      if I ⊈ P then
3:          return False
4:      J ← I
5:      k ← 1
6:      CI ← J
7:      while True do
8:          if T̄^k(J) ⊈ P then
9:              return False
10:         if T̄(T̄_CI^{k−1}(CI)) ⊆ CI then
11:             return True
12:         CI ← Join(CI, T̄^k(J))
13:         if CI ⊈ P then
14:             J ← T̄^k(J)
15:             k ← 0
16:             CI ← J
17:         k ← k + 1
```

Figure 2 depicts a situation in which J and CI have to be re-initialized, i.e., CI $\not\subseteq P$ at line 13 of Algorithm 3. In this example the domain $\mathcal{D}$ consists in rectangles in $\mathbb{R}^2$ and the `Join` over-approximates the union of two rectangles with the smallest rectangle including both of them, i.e., their bounding box. The violation of P is clearly an artifact of the over-approximation. Hence, the algorithm can safely re-initialize J and CI.

As done for Algorithm 2, we aim to investigate the correctness of Algorithm 3 when $\overline{T}$ is an

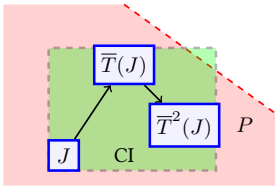


Figure 2: A scenario in which CI $\not\subseteq P$ despite $\bigcup_{i=0}^{k} \overline{T}^i(J) \subseteq P$ when the `Join` operator of Algorithm 3 approximates the union as the bounding box.

under-approximation, an over-approximation, and the exact representation of the forward image function T. In order to achieve this goal, we must notice two main properties of the algorithm. First of all, CI is a subset of P at the beginning of each while-loop iteration. Moreover, if $\overline{T}$ under-approximates (over-approximates) the forward image function T, then, after h iterations of the while-loop, $\overline{T}^k(J)$ under-approximates (over-approximates) $T^h(I)$. These properties allow us to prove the following correctness result.

Theorem 3.3 (Correctness of Algorithm 3). *If $\overline{T}$ under-approximates T and Algorithm 3 returns* `False`*, then P is not an invariant for TS and I. If $\overline{T}$ over-approximates T and Algorithm 3 returns* `True`*, then P is an invariant for TS and I. Let $\overline{T}(A)$ equal $T(A)$ for any $A \in \mathcal{D}$. If Algorithm 3 returns* `True`*, then P is an invariant for TS and I. Moreover, Algorithm 3 returns* `False` *if and only if P is not an invariant for TS and I.*

The set complexity of Algorithm 3 equals that of Algorithm 2 where the cost of exact unions is replaced by that of the `Join` operations. Nevertheless, the number of basic sets represented in CI and the costs of line 10 depend on `Join`. If `Join` always returns one single basic set, then the evaluation of the condition at line 10 requires to compute k forward images of a single basic set and $k-1$ intersections and 1 inclusion test between two basic sets. Thus, in this case, the asymptotic complexity of the test at line 10, whose costs in Algorithm 2 was $O(k^k)$, becomes $O(k)$. In Section 4, we present two possible semantics for the `Join` operator and one of them achieves this complexity.

As far as termination is concerned, a lot depend once more on the choice of `Join`. Algorithm 3 can loop forever only when P is an invariant as Algorithm 2 did too. Intuitively, if there exists a k-inductive set which includes the forward images of $T^h(I)$ for a given $h \geq 0$ and such set can be obtained through successive `Join` operations, then the algorithm terminates. However, it is no more necessary that the forward images converge within a finite number of steps.

Approximating Intersections

The analysis of Algorithm 3 in the previous section relies on the hypothesis that intersections, needed to compute $\overline{T}_{\mathrm{CI}}^{k-1}(\mathrm{CI})$ at line 10, are computed without approximations. However, this is not always feasible and depends on the choice of the domain $\mathcal{D}$. For instance, let us consider the domain of ellipsoids; the intersection of two incomparable ellipsoids is not an ellipsoid. So, we now discuss the properties of Algorithm 3 under the hypothesis that the intersections between sets in $\mathcal{D}$ are approximated.

As already observed, the intersections are exclusively used to evaluate the condition at line 10 and do not affect the correctness of the result when `False` is returned. Because of this, we focus on the cases in which Algorithm 3 returns `True` which, by Theorem 3.3, implies P is an invariant for TS and I only when $\overline{T}$ is an over-approximation of T. It is easy to see that if we over-approximate $\cap$, the algorithm still has no false positive outcomes. Intuitively, if CI contains the over-approximation of the reachable sets it also contains the exact ones, and if the over-approximation of the states reachable from CI after $k-1$ steps inside it are still inside CI, then this also holds for the exact reachable sets.

Corollary 3.3.1 (Over-approximation of T and $\cap$). *Let $\overline{T}(A)$ and $\overline{T}_B(A)$ over-approximate $T(A)$ and $\overline{T}(A) \cap B$, respectively, for any $A, B \in \mathcal{D}$. If Algorithm 3 returns* `True`*, then P is an invariant for TS and I.*

4. Implementation and Tests over Polynomial Systems

We now consider transition systems defined through discrete time polynomial dynamical systems, i.e., dynamical systems described by equations of the form $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k)$, where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a vector of n-variate polynomial functions. The transition system TS $= \langle S, R \rangle$ associated to a dynamical system of this form has $S = \mathbb{R}^n$ and $R = \{(\mathbf{x}_k, \mathbf{x}_{k+1}) \mid \mathbf{x}_k, \mathbf{x}_{k+1} \in \mathbb{R}^n,\ \mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k)\}$.

Sapo [10, 22, 16] implements bounded reachability, verification of Signal Temporal Logic specifications, and parameter synthesis for such systems. The framework supports state sets specified as polytopes, i.e., closed and bounded subsets of $\mathbb{R}^n$ enclosed by a finite number of hyperplanes. A direction of the polytope is a unit vector normal to one of the enclosing hyperplane. Forward images of polytopes are over-approximated exploiting Bernstein's coefficients. The forward image of a polytope is a new polytope specified by the same directions of the initial one, but with new thresholds. A detailed description of both algorithms and data structures implemented by `Sapo` is given in [16].

Since `Sapo` can exclusively over-approximate forward images, we can only aim to validate candidate invariants as proved in Theorem 3.3. Indeed, nothing can be deduced when Algorithm 3 returns `False` as highlighted by Figure 1b which depicts a scenario in which the algorithm returns `False` and P is an invariant.

`Sapo` natively represents exact unions of polytopes as list of polytopes. Moreover, it supports exact intersections and inclusion test of exact union polytopes. Thus, in order to implement Algorithm 3 we need to:

1. choose a specification for candidate invariants to be tested. `Sapo` already supports polytopes, but we aimed to deal with a broader class of properties;
2. implement the test $A \subseteq P$ where A is a generic polytope and P is the candidate invariant;
3. define a function `Join` to over-approximate the union of polytopes.

As far as the candidate invariant specification is concerned, we decided to admit convex linear sets, i.e., sets corresponding to the solutions of a linear system. This specification includes polytopes, but it also covers an infinite number of non-bounded sets.

Example 2. *A possible candidate invariant P for the* Sapo *implementation of Algorithm 3 is the set of unbounded solutions of the linear system*

$$P : \begin{cases} x + 2 * y \leq 5 \\ 3 * x \geq y + 1 \end{cases} .$$

We implemented the test $A \not\subseteq P$, where A is a generic polytope and P is the candidate invariant, by using linear programming. Any polytope A is specified as a system of linear inequalities, $\mathcal{L}_A$. Testing $A \not\subseteq P$ is equivalent to test whether there exists an inequality $p(\mathbf{x}) \leq c$ in the specification of P such that $\mathcal{L}_A$ enriched with $p(\mathbf{x}) > c$ has a feasible solution. As a matter of the fact, such a solution does exist if and only if there exists a state, i.e., the solution itself, that belongs to A, because it satisfies $\mathcal{L}_A$, and does not belong to P, because it satisfies $p(\mathbf{x}) > c$ that is the negation of one of the constraints of P. Once $A \not\subseteq P$ has been implemented, $A \subseteq P$ can be trivially computed as $\neg(A \not\subseteq P)$.

The implementation of the test $A \not\subseteq P$ relies on the solution of up to m linear programming problems where m is the number of linear inequalities in $\mathcal{L}_P$. If the problems are solved by using Karmarkar's algorithm [23], the asymptotic complexity of the algorithm is polynomial with respect to both the space dimension and the number of constraints in $\mathcal{L}_A$ and $\mathcal{L}_P$.

We consider two alternative semantics for the function `Join`: `Listing` and `Packaging`. The most naïve semantics for `Join` is the exact union, i.e., $\texttt{Join}(A, B) = A \cup B$. Since exact unions are usually represented as lists of basic sets, we refer to this approach as `Listing`. An alternative semantics for the function $\texttt{Join}(A, B)$ consists in finding the smallest polytope, having the same directions of A, that includes both A and B. We call this approach `Packaging` because it packs many polytopes into a new one. Figures 2 and 3 depicts CI during an evaluation of Algorithm 3 when the `Join` implements the `Packaging` semantic.

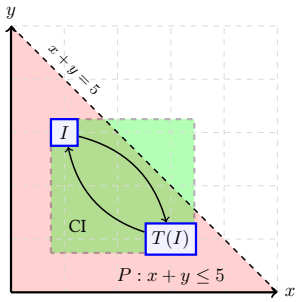
`Packaging` usually produces coarser union approximations than `Listing` or, for instance, convex hull. However, it preserves the number of directions needed to represent sets and, as a consequence, it guarantees that both the memory required to represent a single set and the cost of set functions do not change during the computation. Neither `Listing` nor convex hull ensure these features: the former increases the number of polytopes per set as the computation proceeds, the latter produces only one polytopes per set, but tends to increase the number of directions required to represent it.

While `Packaging` is usually more effective than `Listing`, there are cases in which the former leads to an infinite computation, while the latter produces an answer.

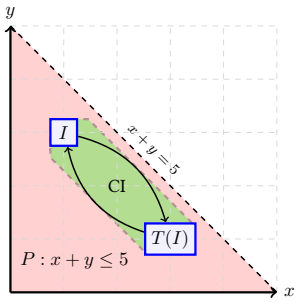
Example 3. *Consider the following system*

$$\begin{cases} x_{k+1} &= y_k \\ y_{k+1} &= x_k \end{cases} .$$

If the system initial state is (x, y), this system keeps flipping x and y values at each evolution step. For instance, if the initial set is $I = [2.9, 3.1] \times [0.9, 1.1]$, after one step the system reaches $T(I) = [0.9, 1.1] \times [2.9, 3.1]$, and one further step brings back the system to its initial condition since $T^2(I) = I$. So, if we select as candidate invariant $x + y \leq 5$, the packaging of any pair of consecutive forward images violates it and Algorithm 3 resets CI and J to the last one of the two.



(a) Algorithm 3 with the `Packaging` semantics cannot validate the invariant $x + y \leq 5$. The bounding box of I and $T(I)$ violates $x + y \leq 5$ and the algorithm loops forever.



(b) Algorithm 3 with the `enhanced Packaging` semantics that adds the candidate invariant directions to the packaging can validate the invariant $x + y \leq 5$ in 2 while-loop iterations.

Figure 3: A graphical representation of Example 3. The system evolution starts from set labelled I. One step brings the system in the state set $T(I)$ and one further step let it go back to its initial condition.

Despite this, the packaging of the new pair of consecutive forward images still does not satisfy $x + y \leq 5$ *and the algorithm loops forever (see Figure 3a).*

Intriguingly, Algorithm 3 with `Listing` *can easily prove that* $x + y \leq 5$ *is an actual invariant for the system and* $I = [2.9, 3.1] \times [0.9, 1.1]$. *The reachability computation converges after* 1 *step and all reachable points satisfies the candidate invariant, hence, in this scenario,* `Listing` *is preferable to* `Packaging`.

In Example 3, `Listing` and `Packaging` produce different outcomes because the packaging of two polytopes A and B may not satisfy a property even when A and B individually do it (see Figure 3a). We can mitigate this issue by adding the property directions to the packaging and minimizing and maximizing them on A and B. This approach is called `enhanced Packaging`. Since we focus on convex linear properties, A and B satisfy a property P if and only if the enhanced packaging of A and B satisfies P too. Indeed, the packaging does not satisfy P if and only if there exists an inequality $p(\mathbf{x}) \leq c$ in the specification of P that is not satisfied. However, the enhanced packaging of A and B is also defined by a set of inequalities among which there is $p(\mathbf{x}) \leq \max(c_A, c_B)$ where c_A and c_B are the maxima of $p(\mathbf{x})$ in A and B, respectively. Thus, if the packaging does not satisfy $p(\mathbf{x}) \leq c$ either A or B do not satisfy it too. The `enhanced Packaging` may increase the complexity of the packaging representation with respect to A and B. However, this only happens when the property directions are not yet present in the representation of the two sets. Figure 3b shows `enhanced Packaging` on Example 3.

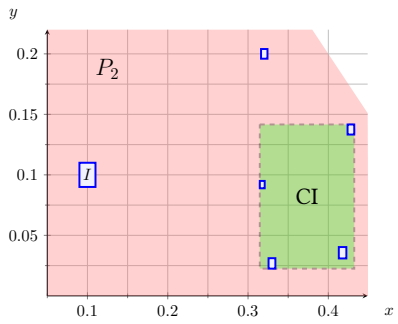
The algorithms presented in Sections 3 and 4 have been implemented in `Sapo` [16]. We now briefly illustrate their effectiveness on the following transition system over $\mathbb{R}^2$

$$\begin{cases} x_{k+1} &= 2x_k y_k + b \\ y_{k+1} &= y_k^2 - x_k^2 + a \end{cases}$$

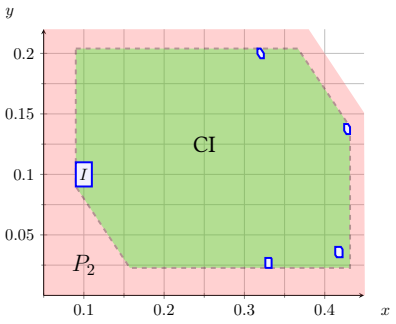
where $a, b \in \mathbb{R}$. This system represents the complex quadratic generator function $g_c(z) = z^2 + c$ with $z, c \in \mathbb{C}$ which is used to define Mandelbrot set [24, 25]. The variables x and y are the real

and imaginary components of z, respectively, and similarly a and b are the components of c. The dynamical system iterates the computation of g_c.

Let $[0.09, 0.11] \times [0.09, 0.11]$ be the system initial set I. Different choices of a and b can produce very different behaviours. We focus on the behaviour obtained with $a \in [0.19, 0.2]$ and $b \in [0.29, 0.3]$ and we consider the candidate invariant $P_1 : y \leq 0.3$. `Sapo` proved that P_1 is an invariant for the system by using `Listing` in less than 1 second on a MacBook Pro 2020 with 16GB of RAM. The proof ended after 12 iterations of the Algorithm 3 while-loop. `Packaging` can achieve the same result after only 4 iterations. `Packaging` can also be used to prove that $P_2 : x + y \leq 0.6$ is an invariant for the investigated system in 5 while-loop iterations; `enhanced Packaging` requires only 4 while-loop iterations for the same goal (see Figure 4b).



(a) `Packaging`: CI is not included in P_2 at the beginning the second iteration of Algorithm 3 while-loop, thus, CI is assigned to $\overline{T}^2(I)$. After 3 further while-loop iterations, CI becomes the bounding box of the forward images which is 3-inductive.



(b) `enhanced Packaging`: the candidate invariant directions are added to I specification. The set CI and all the $\overline{T}^k(I)$ with $k > 0$ are no more rectangles and P_2 always includes them. Thus, CI is never re-initialized and it eventually becomes 4-inductive.

Figure 4: The Mandelbrot system with $a \in [0.19, 0.2]$ and $b \in [0.29, 0.3]$, initial set $I = [0.09, 0.11] \times [0.09, 0.11]$, and $P_2 : x + y \leq 0.6$ as candidate invariant. The red area represents the candidate invariant, the green region is CI during the n-th and final iteration of the while-loop, the blue regions are the over-approximation of the set reachable in up to n steps.

5. Conclusions and Future Work

We presented a set-based method for checking invariant properties over discrete time dynamical systems. The method exploits the notion of k-inductiveness together with over-approximation procedures for reachability, unions, and intersections. When it converges to a positive answer, it computes the "smallest" invariant that is a subset of the given one. Our approach achieves convergence on a larger class of systems when set-based operations are over-approximated rather then when exact computations are performed.

The approach has been implemented and tested in `Sapo` providing two possible semantics for the over-approximation of unions. The implementation was able to prove two invariants over a well-known and deeply investigated fractal example [24, 25].

As future work we plan to investigate other examples and eventually include heuristics for allowing convergence in limit cases.

Acknowledgments

This work is partially supported by PRIN project NiRvAna CUP G23C22000400005 and National Recovery and Resilience Plan (NRRP), Mission 4 Component 2 Investment 1.4 - Call for tender No. 3138 of 16 December 202, rectified by Decree n.3175 of 18 December 2021 of Italian Ministry of University and Research funded by the European Union - NextGenerationEU; Project code CN_00000033, Concession Decree No. 1034 of 17 June 2022 adopted by the Italian Ministry of University and Research, CUP G23C22001110007, Project title "National Biodiversity Future Center - NBFC" .

References

[1] F. Blanchini, Set invariance in control, Automatica 35 (1999) 1747–1767.
[2] X. Deng, M. B. Dwyer, J. Hatcliff, M. Mizuno, Invariant-based specification, synthesis, and verification of synchronization in concurrent programs, in: Proceedings of the 24th international conference on software engineering, 2002, pp. 442–452.
[3] B. Becker, D. Beyer, H. Giese, F. Klein, D. Schilling, Symbolic invariant verification for systems with dynamic structural adaptation, in: Proceedings of the 28th international conference on Software engineering, 2006, pp. 72–81.
[4] A. Platzer, E. M. Clarke, Computing differential invariants of hybrid systems as fixedpoints, Formal Methods in System Design 35 (2009) 98–120.
[5] M. A. B. Sassi, A. Girard, Computation of polytopic invariants for polynomial dynamical systems using linear programming, Automatica 48 (2012) 3114–3121.
[6] S. Sankaranarayanan, Automatic invariant generation for hybrid systems using ideal fixed points, in: Proceedings of the 13th ACM international conference on Hybrid systems: computation and control, 2010, pp. 221–230.
[7] A. Sogokon, K. Ghorbal, P. B. Jackson, A. Platzer, A method for invariant generation for polynomial continuous systems, in: International Conference on Verification, Model Checking, and Abstract Interpretation, Springer, 2016, pp. 268–288.
[8] S. Sankaranarayanan, T. Dang, F. Ivančić, Symbolic model checking of hybrid systems using template polyhedra, in: TACAS '08, Springer, 2008, pp. 188–202.
[9] A. Veliz-Cuba, A. S. Jarrah, R. Laubenbacher, Polynomial algebra of discrete models in systems biology, Bioinformatics 26 (2010) 1637–1643.
[10] T. Dang, T. Dreossi, C. Piazza, Parameter synthesis using parallelotopic enclosure and applications to epidemic models, in: Hybrid Systems and Biology, HSB, 2014, pp. 67–82.
[11] H. Ichihara, S. Tanabe, Y. Ebihara, D. Peaucelle, Analysis and synthesis of discrete-time interconnected positive systems, SICE Journal of Control, Measurement, and System Integration 11 (2018) 91–99.
[12] T. Dreossi, T. Dang, C. Piazza, Reachability computation for polynomial dynamical systems, Formal Methods in System Design 50 (2017) 1–38.

[13] M. Sheeran, S. Singh, G. Stålmarck, Checking safety properties using induction and a sat-solver, in: W. A. Hunt, S. D. Johnson (Eds.), Formal Methods in Computer-Aided Design, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 127–144.

[14] A. R. Bradley, Sat-based model checking without unrolling, in: International Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI 2011), volume 6538 of *Lecture Notes in Computer Science*, Springer, 2011, pp. 70–87.

[15] N. Eén, A. Mishchenko, R. Brayton, Efficient implementation of property directed reachability, in: International Conference on Formal Methods in Computer-Aided Design (FMCAD 2011), IEEE, 2011, pp. 125–134.

[16] A. Casagrande, T. Dang, L. Dorigo, T. Dreossi, C. Piazza, E. Pippia, Parameter synthesis of polynomial dynamical systems, Information and Computation 289 (2022) 104941.

[17] A. Biere, A. Cimatti, E. Clarke, Y. Zhu, Symbolic model checking without BDDs, in: TACAS '99, volume 1579 of *Lecture Notes in Computer Science*, Springer, 1999, pp. 193–207.

[18] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, Y. Zhu, Bounded model checking, Handbook of satisfiability 185 (2009) 457–481.

[19] A. F. Donaldson, L. Haller, D. Kroening, P. Rümmer, Software Verification Using k-Induction, in: E. Yahav (Ed.), Static Analysis, Springer, Berlin, Heidelberg, 2011, pp. 351–368.

[20] A. Casagrande, T. Dreossi, C. Piazza, Hybrid automata and ϵ-analysis on a neural oscillator, Electronic Proceedings in Theoretical Computer Science, EPTCS 92 (2012) 58 – 72.

[21] P. Cousot, N. Halbwachs, Automatic discovery of linear restraints among variables of a program, in: Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages, POPL '78, Association for Computing Machinery, New York, NY, USA, 1978, pp. 84–96.

[22] T. Dang, T. Dreossi, C. Piazza, Parameter synthesis through temporal logic specifications, in: Formal Methods, FM, 2015, pp. 213–230.

[23] N. Karmarkar, A new polynomial-time algorithm for linear programming, in: Symposium on Theory of computing, STOC, ACM, 1984, pp. 302–311.

[24] R. Brooks, J. P. Matelski, The dynamics of 2-generator subgroups of $\mathrm{PSL}(2, c)$, in: Riemann surfaces and related topics: Proceedings of the 1978 Stony Brook Conference, Ann. of Math. Stud, volume 97, 1981, pp. 65–71.

[25] B. B. Mandelbrot, Fractal aspects of the iteration of $z \leftarrow \lambda\, z(1 - z)$ for complex λ and z, Annals of the New York Academy of Sciences 357 (1980) 249–259.