Reachability computation for polynomial dynamical systems

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

Reachability Computation for Polynomial Dynamical Systems

Tommaso Dreossi · Thao Dang · Carla Piazza

Received: date / Accepted: date

Abstract This paper is concerned with the problem of computing the bounded time reachable set of a polynomial discrete-time dynamical system. The problem is well-known for being difficult when nonlinear systems are considered. In this regard, we propose three reachability methods that differ in the set representation. The proposed algorithms adopt boxes, parallelotopes, and parallelotope bundles to construct flowpipes that contain the actual reachable sets. The latter is a new data structure for the symbolic representation of polytopes. Our methods exploit the Bernstein expansion of polynomials to bound the images of sets. The scalability and precision of the presented methods are analyzed on a number of dynamical systems, in comparison with other existing approaches.

Keywords Reachability · Polynomial dynamical systems · Bernstein coefficients

This work is partially supported by Toyota under the CHESS center, DARPA BRASS project, ANR MALTHY project (ANR-12-INSE-003), and INdAM GNCS.

T. Dreossi
University of California, Berkeley
253 Cory Hall #1770
Berkeley, CA 94720-1770, USA
E-mail: tommasodreossi@berkeley.edu

T. Dang
Verimag, Centre Equation
2, avenue de Vignate
38610 Gières, France
E-mail: thao.dang@imag.fr

C. Piazza
University of Udine
Via Delle Scienze, 206
33100 Udine, Italy
E-mail: carla.piazza@uniud.it

1 Introduction

1.1 Dynamical Systems

Dynamical systems are important mathematical models used to describe the evolution of a system in time. They can be seen as a relationship between elements in a sequence (the states of the system) that captures the changes of the terms from one period to another another (time evolution). A dynamical system is said to be either *discrete-time* or *continuous-time* depending on whether the changes take place in discrete time instants or in continuous time, respectively. The common formalisms used to describe discrete- and continuous-time dynamical systems are *difference* and *differential* equations. In this work we will focus on discrete-time dynamical systems described by difference equations:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k) \tag{1}$$

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 reasons why we focus on this class of systems are mainly two: 1) many real phenomena can be observed only at discrete time events, aspect that makes discrete-time models, such as difference equations, good candidates for the formalizations and the study of a vast class of systems arising from the real world (see, e.g.,[52, 51, 36, 44, 4]); 2) discrete-time dynamical systems can be the result of the discretization techniques applied on continuous-time systems for which analytic solutions are intractable (or are not known) and thus discrete numerical integration methods (such as Euler or Runge-Kutta) are necessary (for surveys on numerical integration see, e.g., [43, 53, 28]).

1.2 The Reachability Problem

Formal verification of dynamical systems involves the rigorous and exhaustive study based on mathematical techniques. The development of tools for the formal verification is motivated by the fact that formal guarantees on a system imply the reliability and robustness of the modeled phenomenon. Formal verification is a quite general expression used to gather several problems and methods all related to the same goal of formally proving some properties of a system. Some examples of verification problems related to dynamical systems are the determination of equilibrium points (i.e., states that stabilize the system), invariant sets (i.e., regions from which the system does not escape), or periodic behaviors (i.e., evolutions that return to the same points after a certain amount of time). The *reachability problem*, that often plays an important role in questions as the above-mentioned ones, asks to determine all the states reachable by the evolutions of a dynamical system starting from a set of initial conditions, typically represented as compact infinite sets. Note that considering individual trajectories would require an infinite number of simulations. Hence, some techniques to handle flows of trajectories at the same time are required.

A common approach to the reachability computation problem is to represent the reachable set as a sequence of sets $X_0, X_1, X_2, \ldots, X_T$ whose union, called *flowpipe*, contains all the states reachable by the system. Such a sequence can be obtained by adopting a *numerical set-integration*, which can be seen as a breath-first exploration of the reachable sets. The usual key-steps of a set-based integrator, that are similar to the traditional integrators, are:

1. Fix a set of initial conditions X_0;
2. Compute $X_{k+1} = \mathbf{f}(X_k)$;
3. Repeat Step 2 until a condition is met.

The halting conditions are typically defined in terms of thresholds on the maximum number of reachable steps (in this case we speak of *bounded-time* reachability) or the achievement of a fix-point checkable by the inclusion $X_{k+1} \subseteq X_k$. From this scheme, we can see that the key element of this kind of reachability algorithms is the computation of the image of a set $X_{k+1} = \mathbf{f}(X_k)$ (see Step 2). The hardness of this task depends on two factors: 1) the kind of set X_k that has to be transformed, and 2) the transforming function $\mathbf{f}$. For instance, to obtain the image $\mathbf{f}(X_k)$ of a polytope with respect to a linear function, we can compute the images of the vertices of X_k and, by the convexity preservation of linear functions, we can determine X_{k+1} as the convex hull of the obtained points. However, things become more complicated when nonlinear functions, as the polynomials treated in this work, are considered. In general, nonlinear functions do not preserve nice properties of sets such as convexity, and thus approximation techniques are required.

A large part of this work is concerned with the transformation of polytopes with respect to polynomial functions. If we define efficient methods to over-approximate polynomial images of sets, we can also define algorithms for computing flowpipes that contain the reachable sets of polynomial dynamical systems. Before stating the details of our contributions, let us have an overview of the existing methods for both linear and nonlinear systems.

1.3 Related Work

Linear Systems Reachability computation set of linear systems is one of the most studied problems related to formal verification of linear dynamical and hybrid systems. As previously pointed out, this class of systems has useful properties, such as convexity preservation, that simplify the computation of the transformation of sets. Not surprisingly, one of the most adopted classes of sets are polytopes (often called convex polyhedra).

It has been shown how particular polytopes, such as boxes [76] and parallelotopes [49,50] offer a good trade-off between system dimension and precision. More complex polytopes, like zonotopes [39,40,1], that have the only drawback of not being closed under intersection, have also been successfully applied to linear systems. Different techniques based on a combination of generic polytopes and optimization are [42,18–20,77,34,71].

Further approaches based on a symbolic representation of polytopes as support functions [58,59] have been successfully applied to systems with hundreds of variables. Leaving polytopes, we can find works that use ellipsoids [54,10] or symbolic semialgebraic sets [2,57] to represent and compute reachable sets.

Tools born from these ideas are CheckMate [20], HyTech [42], d/dt [3], MPT [56], PHAVer [34], SpaceEx [35], and Ellipsoidal Toolbox (ET) [55].

Nonlinear Systems If on the one hand linear reachability analysis has several efficient solutions, on the other hand nonlinear reachability remains a rather open problem for which the available methods are limited to restricted classes of systems with low dimensions (hardly more than ten variables for nonlinear systems against hundreds of dimensions for linear ones).

Some attempts for tackling nonlinearity have been made by considering subclasses of nonlinear systems, such as multi-affine systems [6,7] (where each variable appears with degree at most one) or by transforming the original systems into simpler ones using different representations (such as Bézeir simplex or Bernstein basis) [23,68,73].

Some examples of more direct methods that aim to numerically compute reachable sets are based on the manipulation of nonconvex sets, such as orthogonal polyhedra [11,27], or Taylor models [9,16]. A different family approaches based on symbolic manipulations of formulas [5] includes approximated logic semantics [32,37,13] and differential algebraic logics [64,65].

Examples of the tools based on these concepts are d/dt [3], Ariadne [5], KeYmaera [66], pyHybrid Analysis [12], dReach [48], and Flow* [17].

It is important to point out that the available methods can handle, in the optimistic case, systems with at most a dozen of variables.

1.4 Contributions

In this work we present new scalable methods for over-approximations of the reachable set of polynomial discrete-time dynamical systems (note that this is a class of nonlinear systems that has numerous applications in modeling physical phenomena). The developed reachability algorithms share at their core the transformation of polytopes with respect to polynomials, an operation that, as we will discover later, can be computed by maximizing polynomials over a polytopic domains.

The Bernstein expansion, born to represent polynomials in Bernstein basis [8,75], is an efficient tool to bound polynomials over the unit box domain, i.e., the hyperrectangle anchored at the origin having unit edge lengths. In order to be able to bound polynomials over more generic domains, we propose a technique to make use of Bernstein coefficients also on generic boxes and parallelotopes. Intuitively, the adaptation is made possible by the transformation of the unit box into the set on which the polynomial has to be bounded. This trick is the key ingredient for our set transformation methods.

Exploiting Bernstein coefficients over generic boxes and parallelotopes, we define some methods to over-approximate the image of boxes and parallelotopes with respect to polynomials. Later, we will also lift these approaches to the transformation of generic polytopes represented with a new data structure that we call *paralleltope bundle*. A parallelotope bundle is a finite set of parallelotopes whose intersection symbolically represents a polytope. In order to obtain the transformation of a polytope, one can reason on its bundle representation, considering separately each parallelotope, transforming it (with our predefined methods), and then intersecting the obtained parallelotopes. The result is a new polytope that over-approximates the image of the starting one. These three set image methods will be later used to define our reachability algorithm for polynomial dynamical systems.

All the techniques presented in this work have been implemented in a C++ tool called *Sapo* [30]. Our methods have been evaluated on several case studies arising from practical dynamical systems. As we will see in the experimental section, we have been able to apply our reachability algorithms on a quadcopter drone model composed by seventeen variables, aspect that demonstrates the scalability and quality of the techniques proposed in this work.

To summarize, the contributions of this work are the following:

- The adaptation of the bounding properties of *Bernstein coefficients* of polynomials to generic boxes and parallelotopes;
- A *box-based* set image over-approximation technique;
- A *parallelotope-based* set image over-approximation technique;
- The definition of *parallelotope bundles* for the symbolic representation of polytopes and a *parallelotope bundle-based* set image over-approximation technique;
- The definition of a *reachability algorithm* for discrete-time polynomial dynamical systems based on boxes, parallelotopes, and parallelotope bundles;
- The implementation (in a tool called *Sapo*) and the experimental evaluation of our methods on several case studies.

1.5 Paper Structure

The paper is organized as follows. In Sections 2 we introduce the reachability computation problem for dynamical systems, highlighting its hardness whenever nonlinear dynamics are involved. We will also introduce a common optimization-based technique for over-approximating the images of sets using template polyhedra (Section 2.3). We will show how Bernstein coefficients are good candidates to estimate these optima (Section 2.4) which however require some adaptation when applied to generic domains. In Section 3 we will show how some Bernstein coefficients properties can be generalized to some polytopic domain and how they can be used to define algorithms for the over-approximation of the images of sets with respects to polynomials. In Section 3.1 we will reason on boxes, in Section 3.2 on parallelotopes, and in Section 3.3 we will define and work with parallelotope bundles. Section 4 is

dedicated to the experimental evaluation of our methods. In Section 4.1 we will apply our methods on several dynamical systems of increasing complexity, while in Section 4.2 we will compare our implementation with the current state-of-the-art tool for the reachability computation of nonlinear systems. Finally, Section 5 concludes the paper with a short summary and some insights for future developments. Some additional details concerning the experimental evaluations can be found in Appendix A. The implementation of our tool and the experiments presented in this paper can be found at the link [29].

2 Reachable Set Computation

2.1 Sets and Reachability

A discrete-time dynamical system can be described by difference equations of the form:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k) \tag{2}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the vector of *state variables* and the *dynamics* $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ is a vector of n multi-variate continuous functions of the form $\mathbf{f}_i : \mathbb{R}^n \to \mathbb{R}$, for $i \in \{1, \ldots, n\}$. The *initial conidtion* (or *initial state*) $\mathbf{x}_0$ is inside some *initial set* $X_0 \subset \mathbb{R}^n$.

Given a set $X \subset \mathbb{R}^n$, the image of X by $\mathbf{f}$, denoted by $\mathbf{f}(X)$, is defined as:

$$\mathbf{f}(X) = \{(\mathbf{f}_1(\mathbf{x}), \ldots, \mathbf{f}_n(\mathbf{x})) \mid \mathbf{x} \in X\} \tag{3}$$

The set $X_k \subset \mathbb{R}^n$ reachable at time $k \in \mathbb{N}$ can be obtained by the recurrence:

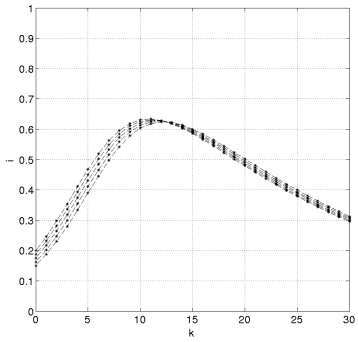
$$X_{k+1} = \mathbf{f}(X_k) \tag{4}$$

where X_0 is the initial set.

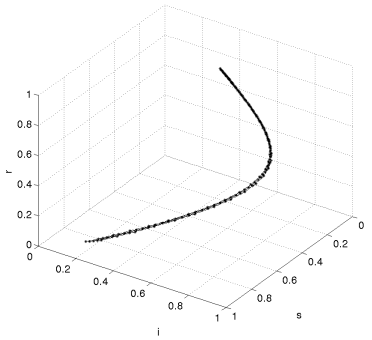
Example 1 Let us consider the SIR epidemic model [46] as running example. This is a well-known dynamical system used to describe the evolution of a disease in a population. The dynamics of the system are the following:

$$\begin{aligned} s_{k+1} &= s_k - \beta s_k i_k / N \\ i_{k+1} &= i_k + \beta s_k i_k / N - \gamma i_k \\ r_{k+1} &= r_k + \gamma i_k \end{aligned} \tag{5}$$

The system considers a population of $N \in \mathbb{R}_{\geq 0}$ individuals partitioned in three compartments: s is the group of *susceptible* individuals who have not been exposed to the disease, i is the class of *infected* individuals, and r are the *removed* individuals who recovered from the disease. The migration of individuals between compartments is regulated by two parameters: β is the probability for a susceptible individual to become infected once there is a contact with an sick person, and $1/\gamma$ is the mean infection period, i.e, the time necessary for an infected individual to migrate from the infected to the



(a) Evolution of infected i individuals in time.



(b) Evolution of susceptible s, infected i, and removed r individuals in space.

Fig. 1: Evolutions of SIR dynamical system with different initial conditions.

removed compartment. The values of the parameters β and γ must be kept constant during the simulation of the model.

For simplicity, let us consider a normalized population, i.e., $N = 1.0$, and parameters $\beta = 0.35$ and $\gamma = 0.05$. Let $X_0 = [0.80, 0.85] \times [0.15, 0.20] \times [0.0, 0.0] \subset \mathbb{R}^3$ be the initial set, i.e., $s_0 \in [0.80, 0.85]$, $i_0 \in [0.15, 0.20]$, and $r_0 \in [0.0, 0.0]$. Figure 1 shows some evolutions of the SIR system having initial conditions sampled from X_0. From the figure we can notice that different initial conditions lead to different trajectories.

The goal of this work is to compute (over-approximate) all the possible trajectories starting from an initial set. In particular, we want to develop a reachability algorithm called REACH (Algorithm 1), that incrementally computes the sets reachable by a dynamical system in a bounded amount of time.

Algorithm 1 Reachability

1: **function** REACH(X_0, t_{max}) ▷ $X_0 \subset \mathbb{R}^n$ initial set, $t_{max} \in \mathbb{N}$
2: **for** $i = 0, \ldots, t_{max}$ **do**
3: $X_{i+1} \leftarrow$REACHSTEP(X_i)
4: **end for**
5: **end function**

The algorithm receives in input an initial set $X_0 \subset \mathbb{R}^n$ and a time horizon $t_{max} \in \mathbb{N}$, and computes a sequence of sets $X_0, X_1, \ldots, X_{t_{max}}$, called *flowpipe*, whose union over-approximates the reachable set of the considered dynamical system. At time i, the set X_{i+1} is obtained by calling the function REACHSTEP(X_i) that implements a single reachability step. As we will see, a concrete implementation of REACHSTEP strongly depends on the adopted set representation and the considered dynamics.

2.2 Polytopes and Template Polyhedra

In general, the computation and representation of a set transformed by a non-linear function is hard. A common way to deal with this issue consists in over-approximating the transformed set with simpler objects, such as polytopes and template polyhedra.

Definition 1 A *polytope* $Q \subset \mathbb{R}^n$ is a closed, compact, bounded subset of $\mathbb{R}^n$ such that there is a finite set of half-spaces $H = \{h_1, \dots, h_m\}$ whose intersection is Q, that is:

$$Q = \bigcap_{i=1}^{m} h_i \tag{6}$$

where an *half-space* $h = \{\mathbf{x} \mid \mathbf{d}\mathbf{x}^T \leq c\}$ is a set characterized by a non-null normal vector $\mathbf{d} \in \mathbb{R}^n$ and an offset $c \in \mathbb{R}$.

The linear constraints that generate the half-spaces can be organized in a matrix $D \in \mathbb{R}^{m \times n}$ called *template* and a vector $\mathbf{c} \in \mathbb{R}^m$ called *offset vector*, in short *offsets*. The polytope generated by the template D and the offset vector $\mathbf{c}$ is denoted by $\langle D, \mathbf{c} \rangle$. Notice that not all the pairs $\langle D, \mathbf{c} \rangle$ define a nonempty polytope.

Template polyhedra [72,14] are a subclass of polytopes with fixed templates and variable offsets. By varying the offsets $\mathbf{c}$ of a template polyhedron $\langle D, \mathbf{c} \rangle$, it is possible to obtain a infinite number of polytopes having D as template. The advantage of template polyhedra over polytopes is that common geometric operations, such as intersection and union, can be performed more efficiently. In the following we assume that the directions of the adopted templates are given and fixed. In general, determining optimal directions for a template is a hard problem that goes outside the scope of this work. Some works that considered the issue of automatically finding ideal directions are, e.g., [15,74].

2.3 Single Step Reachability

A polytope-based reachability computation requires the ability of computing the image of a polytope Q by the dynamics $\mathbf{f}$. In the case of template polyhedra, where a template $D \in \mathbb{R}^{m \times n}$ is given, the set image over-approximation can be seen as the problem of finding an offset vector $\mathbf{c} \in \mathbb{R}^m$ such that:

$$\mathbf{f}(Q) \subseteq \langle D, \mathbf{c} \rangle \tag{7}$$

The continuity of $\mathbf{f}$ guarantees that $\mathbf{f}(Q)$ is a closed, bounded, and compact set for which always exists an over-approximating polytope $\langle D, \mathbf{c} \rangle$.

It is not difficult to see that this inclusion holds if $D\mathbf{f}(\mathbf{x}) \leq \mathbf{c}$ holds for all the $\mathbf{x} \in Q$. This suggests that offsets $\mathbf{c} = (\mathbf{c}_1, \dots, \mathbf{c}_m)$ can be determined by solving the following optimization problems:

$$\mathbf{c}_i = \max_{\mathbf{x} \in Q} D_i \mathbf{f}(\mathbf{x}) \tag{8}$$

where D_i is the i-th row of D and $i \in \{1, \ldots, m\}$. Whenever $\mathbf{f}$ are nonlinear functions, these optimization problems require nonlinear and nonconvex optimization techniques that are in general computationally expensive. A way to address this issue, is to relax the problem and seek tight bounds that can be efficiently determined. For instance, if we are able to bound a polynomial over a polytope, then we can develop a single step reachability algorithm for polynomial dynamical systems.

Algorithm 2 Single Step Reachability

```
1: function REACHSTEP(X)                          ▷ X = ⟨D, c⟩ ⊂ ℝ^n polytope
2:     for i ∈ {1, ..., m} do
3:         c'_i ← BOUND(D_i f(x), X)
4:     end for
5:     return X' = ⟨D, c'⟩
6: end function
```

REACHSTEP (Algorithm 2) implements this approach. It computes an over-approximation set $X' \supseteq \mathbf{f}(X)$ bounding the functions $D_i\mathbf{f}(\mathbf{x})$ over the polytope $X = \langle D, \mathbf{c}\rangle$. With this setup, the computation of the reachable sets depends on the ability of bounding a function over a polytope. This task can be difficult for generic $\mathbf{f}$ and X.

In the next section we introduce a technique to bound polynomials over unit boxes exploiting Bernstein coefficients (typically used to express polynomials in Bernstein form).

2.4 Bernstein Basis and Coefficients

Before defining Bernstein basis and coefficients, we introduce some notations useful to work with polynomials.

A *multi-index* is a vector $\mathbf{i} = (\mathbf{i}_1, \ldots, \mathbf{i}_n)$ where each $\mathbf{i}_j \in \mathbb{N}$. Given two multi-indices $\mathbf{i}$ and $\mathbf{d}$ of the same length, we write $\mathbf{i} \leq \mathbf{d}$ ($\mathbf{d}$ dominates $\mathbf{i}$) if for all $j \in \{1, \ldots, n\}$, $\mathbf{i}_j \leq \mathbf{d}_j$. We denote the multi-index $(\mathbf{i}_1/\mathbf{d}_1, \ldots, \mathbf{i}_n/\mathbf{d}_n)$ by $\mathbf{i}/\mathbf{d}$ and the product of the binomial coefficients $\binom{\mathbf{d}_1}{\mathbf{i}_1} \ldots \binom{\mathbf{d}_n}{\mathbf{i}_n}$ by $\binom{\mathbf{d}}{\mathbf{i}}$.

A polynomial $\pi(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ can be represented in the power basis as:

$$\pi(\mathbf{x}) = \sum_{\mathbf{i} \in I^\pi} \mathbf{a}_\mathbf{i} \mathbf{x}^\mathbf{i} \tag{9}$$

where $\mathbf{i} = (\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_n)$ is a multi-index of size $n \in \mathbb{N}$ and $\mathbf{x}^\mathbf{i}$ denotes the monomial $\mathbf{x}_1^{\mathbf{i}_1}\mathbf{x}_2^{\mathbf{i}_2} \ldots \mathbf{x}_n^{\mathbf{i}_n}$. The finite set I^π is called the *multi-index set* of π. The *degree* $\mathbf{d}$ of π is the smallest multi-index that dominates all the multi-indices of I^π, i.e., for all $\mathbf{i} \in I^\pi, \mathbf{i} \leq \mathbf{d}$. The coefficients $\mathbf{a}_\mathbf{i} \in \mathbb{R}$ range over the reals.

Example 2 Consider the polynomial $\pi(\mathbf{x}_1, \mathbf{x}_2) = 1/3\mathbf{x}_1^2 - 1/2\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$. The multi-index set of π is $I^\pi = \{(2,0), (0,1), (1,1), (0,0)\}$, the associated

coefficients are $\mathbf{a}_{(2,0)} = 1/3, \mathbf{a}_{(0,1)} = -1/2, \mathbf{a}_{(1,1)} = 1/4$, $\mathbf{a}_{(0,0)} = 1/2$, and the degree is $\mathbf{d} = (2, 1)$.

Bernstein basis polynomials of degree $\mathbf{d}$ are basis for the space of polynomials of degree at most $\mathbf{d}$ over $\mathbb{R}^n$. For $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{R}^n$, the $\mathbf{i}$-th Bernstein polynomial of degree $\mathbf{d}$ is defined as:

$$\mathcal{B}_{(\mathbf{d},\mathbf{i})}(\mathbf{x}) = \beta_{\mathbf{d}_1,\mathbf{i}_1}(\mathbf{x}_1)\beta_{\mathbf{d}_2,\mathbf{i}_2}(\mathbf{x}_2)\ldots\beta_{\mathbf{d}_n,\mathbf{i}_n}(\mathbf{x}_n) \tag{10}$$

where, for a real number $x \in \mathbb{R}$,

$$\beta_{\mathbf{d}_j,\mathbf{i}_j}(x) = \binom{\mathbf{d}_j}{\mathbf{i}_j} x^{\mathbf{i}_j}(1-x)^{\mathbf{d}_j-\mathbf{i}_j} \tag{11}$$

A polynomial $\pi(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ can be represented using Bernstein basis as:

$$\pi(\mathbf{x}) = \sum_{\mathbf{i}\in I^\pi} \mathbf{b}_\mathbf{i}\mathcal{B}_{(\mathbf{d},\mathbf{i})}(\mathbf{x}) \tag{12}$$

where, for each $\mathbf{i} \in I^\pi$, the *Bernstein coefficient* $\mathbf{b}_\mathbf{i}$, is defined as:

$$\mathbf{b}_\mathbf{i} = \sum_{\mathbf{j}\le\mathbf{i}} \frac{\binom{\mathbf{i}}{\mathbf{j}}}{\binom{\mathbf{d}}{\mathbf{j}}}\mathbf{a}_\mathbf{j} \tag{13}$$

Bernstein coefficients can be calculated from the coefficients of the monomials of the treated polynomial in power basis. The $(n+1)$-dimensional points $(\mathbf{i}/\mathbf{d}, \mathbf{b}_\mathbf{i}) \in \mathbb{R}^{n+1}$ are called *Bernstein control points*.

Example 3 Consider the polynomial $\pi(\mathbf{x}_1, \mathbf{x}_2) = 1/3\mathbf{x}_1^2 - 1/2\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$ of Example 2. The Bernstein coefficient associated to the multi-index $(1, 1)$ is:

$$\mathbf{b}_{(1,1)} = \frac{\binom{(1,1)}{(1,1)}}{\binom{(2,1)}{(1,1)}}1/4 - \frac{\binom{(1,1)}{(0,1)}}{\binom{(2,1)}{(0,1)}}1/2 + \frac{\binom{(1,1)}{(1,0)}}{\binom{(2,1)}{(1,0)}}0 + \frac{\binom{(1,1)}{(0,0)}}{\binom{(2,1)}{(0,0)}}1/2 = 0.125 \tag{14}$$

Applying the same scheme to the other multi-indices, we obtain the Bernstein coefficients $\mathbf{b}_{(0,0)} = 0.5$, $\mathbf{b}_{(0,1)} = 0.0$, $\mathbf{b}_{(1,0)} = 0.5$, $\mathbf{b}_{(1,1)} = 0.125$, $\mathbf{b}_{(2,0)} = 0.834$, and $\mathbf{b}_{(2,1)} = 0.584$.

2.4.1 Properties of Bernstein Coefficients

Bernstein coefficients present several interesting properties. Here we expose two properties that will be exploited in our reachability techniques. For further properties see, for instance, [75,33].

Lemma 1 (Range Enclosing)

$$\min_{\mathbf{i}\in I^\pi} \mathbf{b}_\mathbf{i} \le \pi(\mathbf{x}) \le \max_{\mathbf{i}\in I^\pi} \mathbf{b}_\mathbf{i}, \tag{15}$$

for all $\mathbf{x} \in [0, 1]^n$*, where* $\mathbf{b}_\mathbf{i}$*, for* $\mathbf{i} \in I^\pi$*, are the Bernstein coefficients of* π*.*

Lemma 2 (Sharpness)

$$\mathbf{b_i} = \pi(\mathbf{i}/\mathbf{d}) \tag{16}$$

for all $\mathbf{i} \in \mathcal{V}_\mathbf{d}$*, where* $\mathbf{b_i}$ *are the Bernstein coefficients of* π *and* $\mathcal{V}_\mathbf{d}$ *is the set of vertices of the box* $[0, \mathbf{d}_1] \times [0, \mathbf{d}_2] \times \ldots [0, \mathbf{d}_n]$.

These two properties provide us some useful information about the image of the polynomial π over the unit box $[0,1]^n$. In particular:

1. The range enclosing property (Lemma 1) states that the minimum and maximum Bernstein coefficients are a lower bound and an upper bound of the image of π over the unit box domain, respectively;
2. The sharpness property (Lemma 2) says that the Bernstein coefficients at the vertices of the box domain, match exactly the values of the polynomial at some points.

Example 4 Consider the polynomial $\pi(\mathbf{x}_1, \mathbf{x}_2) = 1/3\mathbf{x}_1^2 - 1/2\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$ and its Bernstein coefficients (from Example 3). Figure 2 shows the image of π over the unit box (gray area) and its control points (black dots).

The coefficients $\mathbf{b}_{(1,1)} = 0.125$ and $\mathbf{b}_{(2,0)} = 0.834$ are a lower bound and an upper bound of $\pi([0,1]^2)$ (range enclosing property) and the control points lying on the vertices of the unit box match exactly the values of $\pi([0,1]^2)$ (sharpness property).

The following lemma [26] limits the error between the actual optimums and the bounds provided by Bernstein coefficients.

Lemma 3 *Let* $C_\pi : \mathbb{R}^n \to \mathbb{R}$ *be the piecewise linear function defined by the Bernstein control points of the polynomial* $\pi : \mathbb{R}^n \to \mathbb{R}$*, with respect to the box* $[0,1]^n$*. For all* $\mathbf{x} \in [0,1]^n$

$$| \pi(\mathbf{x}) - C_\pi(\mathbf{x}) | \leq \max_{\mathbf{x} \in [0,1]^n; i,j \in \{1,\ldots,n\}} | \partial_i \partial_j \pi(\mathbf{x}) | \tag{17}$$

where $| \cdot |$ *is the infinity norm on* $\mathbb{R}^n$.

Several convergent subdivision procedures for reducing the gap between bounds and optimums have been proposed [38,62,61].

2.4.2 Computation of upper and lower bounds

The range enclosing property (Lemma 1) can be used to determine upper and lower bounds of polynomials over the unit box. We now define the algorithm maxBernCoeff (Algorithm 3) that implements this idea.

Given a polynomial $\pi : \mathbb{R}^n \to \mathbb{R}$, maxBernCoeff computes the set B_π of the Bernstein coefficients of $\pi(\mathbf{x})$ (that can be obtained using Equation (13)), scans all the coefficients, and determines their maximum $\overline{b}$ that is an upper bound for the polynomial $\pi(\mathbf{x})$ over the unit box.

If needed, in a similar way, we can define the algorithm minBernCoeff that returns a lower bound of $\pi(\mathbf{x})$. minBernCoeff can be easily obtained by extracting the minimum coefficient from the set B_π. Note that we can also introduce a test for the sharpness property (see Lemma 2) that checks whether the computed bound matches the exact optimum.

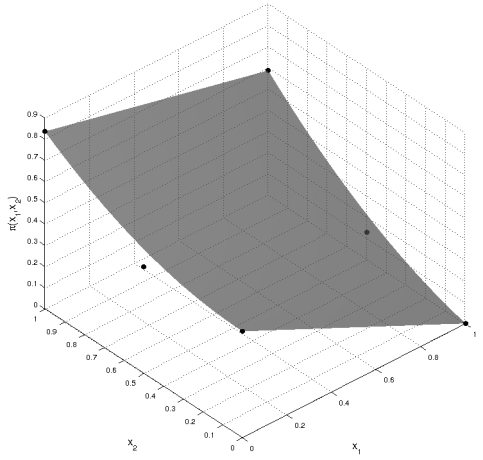


Fig. 2: The polynomial $\pi(\mathbf{x}_1, \mathbf{x}_2) = 1/3\mathbf{x}_1^2 - 1/2\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$ over the unit box (in gray) and its control points (in black). The coefficients $\mathbf{b}_{(1,1)} = 0.125$ and $\mathbf{b}_{(2,0)} = 0.834$ are a lower bound and an upper bound of $\pi([0,1]^2)$ (range enclosing property) and the control points lying on the vertices of the unit box match exactly the values of $\pi([0,1]^2)$ (sharpness property).

Algorithm 3 Compute maximum Bernstein Coefficient

1: **function** MAXBERNCOEFF(π) ▷ $\pi(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$
2: $B_\pi \leftarrow$ BERNCOEFFS(π) ▷ Compute Bernstein coefficients
3: $\overline{b} \leftarrow \max_{\mathbf{b_i} \in B_\pi} \mathbf{b_i}$ ▷ Extract the maximum
4: **return** $\overline{b}$
5: **end function**

3 Bounding Polynomials Over Polytopes

In the previous section we have seen how the over-approximation of the image of a set can be reduced to a number of function optimizations. We have also introduced Bernstein coefficients, that can be used to bound polynomials exclusively on unit box domains. In this section we will develop some methods that take advantage of the Bernstein coefficients properties on different subclasses of polytopes, precisely on generic boxes (or hyperrectangles), parallelotopes (the generalization of parallelograms to higher dimensions), and parallelotope bundles (a new data structure for representing polytopes). For each class of sets, the goal is to develop an algorithm BOUND(π, X) that returns an upper bound of the maximum of π over X. This algorithm will be used by REACHSTEP (Algorithm 2) to compute over-approximations of images

of sets and indirectly by REACH (Algorithm 1) to compute the reachable set of polynomial dynamical systems.

3.1 Boxes

We begin by considering the class of *boxes*, also known as *hyperrectangles*.

Definition 2 (Box) A set $B \subset \mathbb{R}^n$ is a *box* if and only if it can be expressed as the product of n intervals, that is:

$$B = [\underline{x}_1, \overline{x}_1] \times \dots [\underline{x}_n, \overline{x}_n] = \prod_{i=1}^{n} [\underline{x}_i, \overline{x}_i] \tag{18}$$

where $\underline{x}_i, \overline{x}_i \in \mathbb{R}$, for $i \in \{1, \dots, n\}$.

It is easy to see that a box $B = [\underline{x}_1, \overline{x}_1] \times \dots [\underline{x}_n, \overline{x}_n]$ is a polytope since it can be represented by a template $D \in \mathbb{R}^{2n \times n}$ and offsets $\mathbf{c} \in \mathbb{R}^{2n}$ where:

$$D = \begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \\ -1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & -1 \end{pmatrix} \quad \mathbf{c} = \begin{pmatrix} \overline{x}_1 \\ \vdots \\ \overline{x}_n \\ -\underline{x}_1 \\ \vdots \\ -\underline{x}_n \end{pmatrix} \tag{19}$$

3.1.1 Bounding over Boxes

Let us now focus on the problem of extending the properties of Bernstein coefficients, that are valid only for unit box domains, to generic boxes.

Let $\pi : \mathbb{R}^n \to \mathbb{R}$ be a polynomial and $X = \langle D, \mathbf{c} \rangle = [\underline{x}_1, \overline{x}_1] \times \dots \times [\underline{x}_n, \overline{x}_n]$ be a box. We begin by defining a linear transformation $\mathbf{v}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^n$ that maps the unit box to X. Such a map can be defined as follows:

$$\mathbf{v}(\mathbf{x}) = \begin{pmatrix} \overline{x}_1 - \underline{x}_1 & 0 & \dots & 0 \\ 0 & \overline{x}_2 - \underline{x}_2 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & & \dots & 0 \quad \overline{x}_n - \underline{x}_n \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} + \begin{pmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \vdots \\ \underline{x}_n \end{pmatrix} \tag{20}$$

Composing $\mathbf{v}(\mathbf{x})$ with $\pi(\mathbf{x})$ we observe that the equality $\pi(X) = \pi(\mathbf{v}([0,1]^n))$ holds, which suggests that we can use the Bernstein coefficients of $\pi(\mathbf{v}(\mathbf{x}))$, to indirectly bound $\pi(\mathbf{x})$ over X. We recall the the Bernstein coefficients can be computed by MAXBERNCOEFF (Algorithm 3).

Let us formalize this procedure in the function BOUND (Algorithm 4) that receives in input a polynomial $\pi : \mathbb{R}^n \to \mathbb{R}$ and a box $X \subset \mathbb{R}^n$, and returns an upper bound $\overline{b} \in \mathbb{R}$ of $\pi(\mathbf{x})$ such that $\overline{b} \geq \max_{\mathbf{x} \in [0,1]^n} \pi(\mathbf{v}(\mathbf{x})) = \max_{\mathbf{x} \in X} \pi(\mathbf{x})$.

Algorithm 4 Bound polynomial over box

```
1: function BOUND(π, X)                          ▷ X ⊂ ℝ^n box
2:     v(x) ←MAPUNITBOXTO(X)                     ▷ Map [0,1]^n to X
3:     b̄ ←MAXBERNCOEFF(π(v(x)))                  ▷ Compute maximum coefficient
4:     return b̄
5: end function
```

The algorithm BOUND, using the function MAPUNITBOXTO based on Equation 20, computes the transformation $\mathbf{v}(\mathbf{x})$ that maps the unit box to the given box X. Then, it computes the Bernstein coefficients of $\pi(\mathbf{v}(\mathbf{x}))$ and returns their maximum. This bounding algorithm is the basic brick of our first box-based reachability algorithm.

Example 5 We now illustrate the computation of the single step reachability algorithm based on boxes. We consider the SIR dynamical system introduced in Example 1. Let $X \subset \mathbb{R}^3$ with $s_0 \in [0.80, 0.85], i_0 \in [0.15, 0.20]$, and $r_0 \in [0.0, 0.0]$ be a box whose constraint representation is $X = \langle D, \mathbf{c} \rangle$ where:

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \qquad \mathbf{c} = \begin{pmatrix} 0.85 \\ 0.20 \\ 0.00 \\ -0.80 \\ -0.15 \\ 0.00 \end{pmatrix} \tag{21}$$

The algorithm REACHSTEP begins by composing the first direction D_1 of D with the system dynamics, obtaining the function:

$$D_1\mathbf{f}(s,i,r) = \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} s - 0.35si \\ i + 0.35si - 0.05i \\ r + 0.05i \end{pmatrix} = s - 0.35si \tag{22}$$

Then it proceeds by bounding $D_1\mathbf{f}(\mathbf{x})$ over X calling $\text{BOUND}(D_1\mathbf{f}(\mathbf{x}), X)$ that computes the map $\mathbf{v}(\mathbf{x})$ from the unit box to X:

$$\mathbf{v}(\mathbf{x}) = \begin{pmatrix} 0.85 - 0.80 & 0 & 0 \\ 0 & 0.20 - 0.15 & 0 \\ 0 & 0 & 0.0 - 0.0 \end{pmatrix} \begin{pmatrix} s \\ i \\ r \end{pmatrix} + \begin{pmatrix} 0.80 \\ 0.15 \\ 0.0 \end{pmatrix} \tag{23}$$

and composes it with the function to bound $D_1\mathbf{f}(\mathbf{x})$, generating the polynomial:

$$D_1\mathbf{f}(\mathbf{v}(\mathbf{x})) = s/20 - (7(i/20 + 3/20)(s/20 + 4/5))/20 + 4/5 \tag{24}$$

Finally the set $B_{D_1\mathbf{f}(\mathbf{v}(\mathbf{x}))}$ of Bernstein coefficients of $D_1\mathbf{f}(\mathbf{v}(\mathbf{x}))$ is computed:

$$D_1\mathbf{f}(\mathbf{v}(\mathbf{x})) = \{0.7580, 0.7440, 0.8054, 0.7905\} \tag{25}$$

and the maximum coefficient $\overline{b} = 0.8054$ is returned. Fixing $\overline{b}$ as the offset $\mathbf{c}'_1$ for the direction D_1, we obtain the first half-space of the new over-approximating

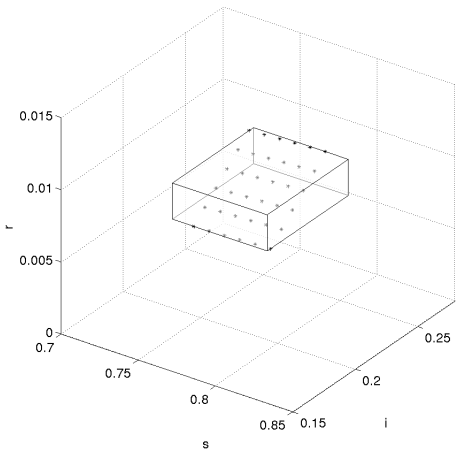


Fig. 3: Box-based set image approximation. The constructed box (in white) and some reachable points (in black).

box. Repeating the procedure for all the directions of D we obtain the box $X' = \langle D, \mathbf{c}' \rangle$ where:

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \qquad \mathbf{c}' = \begin{pmatrix} 0.8054 \\ 0.2495 \\ 0.0100 \\ -0.7440 \\ -0.1845 \\ -0.0075 \end{pmatrix} \tag{26}$$

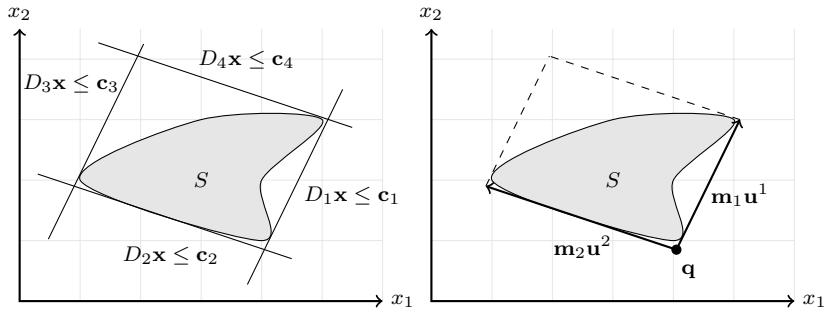
The constructed box X' is shown in Figure 3 (in white) together with some reachable points (in black) computed sampling initial conditions in X. Note how X contains all the sampled reachable points.

3.2 Parallelotopes

In this section we extend our set image approximation method to *parallelotopes*, i.e., the n-dimensional generalization of parallelograms. The use of parallelotopes makes the reachability method more flexible as far as the choice of the initial set is concerned and it allows one to obtain better approximations.

A parallelotope is a centrally symmetric convex polytope whose opposite facets are parallel. As all the polytopes, it can be represented as a collection of linear constraints.

Definition 3 (Parallelotope Constraint Representation) Let $D \in \mathbb{R}^{2n \times n}$ be a template such that $D_i = -D_{i+n}$ for each $i \in \{1, 2, \ldots, n\}$, and let



(a) Constraint representation. (b) Generator representation.

Fig. 4: A set S and two enclosing parallelotopes.

$\mathbf{c} \in \mathbb{R}^{2n}$ be an offset vector. The parallelotope $P \subset \mathbb{R}^n$ generated by D and $\mathbf{c}$ is $P = \langle D, \mathbf{c} \rangle$.

We refer to the above representation as the *constraint representation* (see Figure 4a). Note that a parallelotope is a polytope and that a box is a parallelotope (see Equation (19)). Another way to characterize a parallelotope, similar to the one adopted for zonotopes [21], is to fix a point of origin and use vectors to define it.

Definition 4 (Parallelotope Generator Representation) Let $U = \{\mathbf{u}^1, \ldots, \mathbf{u}^n\}$ be a set of n linearly independent normalized vectors in $[0,1]^n$, $\mathbf{m} \in \mathbb{R}^n_{\geq 0}$, and $\mathbf{q} \in \mathbb{R}^n$. The parallelotope $P \subset \mathbb{R}^n$ generated by $U, \mathbf{m}$, and $\mathbf{q}$ is:

$$P = \{\mathbf{y} \mid \mathbf{y} = \gamma_U(\mathbf{q}, \mathbf{m}, \mathbf{x}), \mathbf{x} \in [0,1]^n\} \tag{27}$$

where the function $\gamma_U : \mathbb{R}^n \times \mathbb{R}^n_{\geq 0} \times \mathbb{R}^n \to \mathbb{R}^n$ is defined as:

$$\gamma_U(\mathbf{q}, \mathbf{m}, \mathbf{x}) = \mathbf{q} + \sum_{j=1}^{n} \mathbf{m}_j \mathbf{u}^j \mathbf{x}_j \tag{28}$$

This representation is called *generator representation*. The vectors $\mathbf{u}^1, \ldots, \mathbf{u}^n$ are the *generators* of the parallelotope, $\mathbf{q}$ is the *base vertex*, and $\mathbf{m}$ are the *magnitudes* of the generators. Intuitively, the base vertex is the point on which the generators are anchored, the generators defines the orientation of the parallelotope, while the magnitudes its scale (see Figure 4b). This notation emphasizes the aspect that a parallelotope can be seen as the affine transformation of the unit box (note in Equation (27) that $\mathbf{x} \in [0,1]^n$). This suggests that there might be a way to combine Bernstein coefficients with parallelotopes.

Before discussing the parallelotope-based set image approximation technique, we show how a parallelotope can be equivalently represented using constraints and generators. In our reachability algorithm we will use one of the two representations depending on the operation we want to perform.

3.2.1 Representation Conversion

From Constraints to Generators Given a parallelotope $P = \langle D, \mathbf{c} \rangle$ in constraint representation, we want to find a generator set U, a base vertex $\mathbf{q}$, and magnitudes $\mathbf{m}$ such that $\gamma_U(\mathbf{q}, \mathbf{m}, [0,1]^n) = \langle D, \mathbf{c} \rangle$. First, we rewrite the inequalities given by the template D and offsets $\mathbf{c}$ in form:

$$-\mathbf{c}_{n+i} \leq D_i \mathbf{x} \leq \mathbf{c}_i \tag{29}$$

for all $i \in \{1, \ldots, n\}$. The based vertex $\mathbf{q}$ and the coordinates of the vertex $\mathbf{i}_i$ that lies on the straight line passing through the i-th generator vector applied to the vertex $\mathbf{q}$, are the solutions of the linear systems:

$$\begin{pmatrix} D_1 \\ \vdots \\ D_n \end{pmatrix} \mathbf{x} = \begin{pmatrix} -\mathbf{c}_{n+1} \\ \vdots \\ -\mathbf{c}_{2n} \end{pmatrix} \qquad \begin{pmatrix} D_1 \\ \vdots \\ D_i \\ \vdots \\ D_n \end{pmatrix} \mathbf{x} = \begin{pmatrix} -\mathbf{c}_{n+1} \\ \vdots \\ \mathbf{c}_i \\ \vdots \\ -\mathbf{c}_{2n} \end{pmatrix} \tag{30}$$

Let $\mathbf{g}^i$ be the vector anchored on the base vertex $\mathbf{q}$ that points to the vertex $\mathbf{v}_i$, i.e., $\mathbf{g}^i = \mathbf{v}_i - \mathbf{q}$. The generator $\mathbf{u}^i$ and the magnitude $\mathbf{m}_i$ such that $\mathbf{g}^i = \mathbf{m}_i \mathbf{u}^i$ are given by $\mathbf{m}_i = \|\mathbf{g}^i\|$ and $\mathbf{u}^i = \dfrac{\mathbf{g}^i}{\|\mathbf{g}^i\|}$.

From Generators to Constraints We now consider the inverse conversion: given a generator function $\gamma_U(\mathbf{q}, \mathbf{m}, \mathbf{x})$, find a template D and an offset vector $\mathbf{c}$ such that $\langle D, \mathbf{c} \rangle = \gamma_U(\mathbf{q}, \mathbf{m}, [0,1]^n)$.

Let $G = \{\mathbf{g}^i \mid \mathbf{g}^i = \mathbf{m}_i \mathbf{u}^i, \mathbf{m}_i \in \mathbf{x}, \mathbf{u}^i \in U\}$ be the set of generators scaled by their correspondent magnitudes. At first, we calculate the points $\mathbf{p}^1, \ldots, \mathbf{p}^n$ that are traversed by the hyperplanes correspondent to the constrains. Each $\mathbf{p}^i$ is obtained by adding the vector $\mathbf{g}^i$ to the base vertex $\mathbf{q}$, i.e., $\mathbf{p}^i = \mathbf{q} + \mathbf{g}^i$. The i-th constraint of the parallelotope lies on the hyperplane (whose equation is $h_i = \mathbf{a}_i \mathbf{x} + \mathbf{c}_i$) passing through the points $\mathbf{q}, \mathbf{p}^1, \ldots, \mathbf{p}^{i-1}, \mathbf{p}^{i+1}, \ldots, \mathbf{p}^n$. The equation $h_{i+n} = \mathbf{a}_{i+n} \mathbf{x} + \mathbf{c}_{i+n}$ of the hyperplane parallel to h^i can be found by translating the vertices used to compute h_i by the vector $\mathbf{g}^i$, i.e., h^{i+n} is the hyperplane passing through the points $\mathbf{q} + \mathbf{g}^i, \mathbf{p}^1 + \mathbf{g}^i, \ldots, \mathbf{p}^{i-1} + \mathbf{g}^i, \mathbf{p}^{i+1} + \mathbf{g}^i, \ldots, \mathbf{p}^n + \mathbf{g}^i$. Let $\underline{\mathbf{c}}_i$ and $\overline{\mathbf{c}}_i$ be defined as $\underline{\mathbf{c}}_i = \min\{d_i, d_{i+n}\}$ and $\overline{\mathbf{c}}_i = \max\{d_i, d_{i+n}\}$.

Since h_i and h_{i+n} are parallel it must hold $\mathbf{a}_i = \mathbf{a}_{i+n}$. Hence, the portion of the parallelotope included between h_i and h_{i+n} is the solution of the inequalities $\underline{\mathbf{c}}_i \leq \mathbf{a}_i \mathbf{x} \leq \overline{\mathbf{c}}_i$, which means that the i-th and $(i+n)$-th rows of the template matrix are $D_i = \mathbf{a}_i$ and $D_{i+n} = -\mathbf{a}_i$, while the i-th and $(i+n)$-th offset elements are $\mathbf{c}_i = \overline{\mathbf{c}}_i$ and $\mathbf{c}_{i+n} = -\underline{\mathbf{c}}_i$.

3.2.2 Bounding over Parallelotopes

Let us now focus on the set image computation and in particular on the polynomial bounding problem. Let $X = \langle D, \mathbf{c} \rangle \subset \mathbb{R}^n$ be a parallelotope in constraint representation whose generator function is $\gamma_U(\mathbf{q}, \mathbf{m}, \mathbf{x})$. We are interested in computing a parallelotope $X' \subset \mathbb{R}^n$ such that $X' \supseteq \mathbf{f}(X)$. Adopting the template D of X, we can obtain X' by determining the offsets $\mathbf{c}' \in \mathbb{R}^{2n}$ such that $\mathbf{f}(X) \subseteq \langle D, \mathbf{c}' \rangle = X'$. We recall that in order for a parallelotope $X' = \langle D, \mathbf{c}' \rangle$ to over-approximate the set $\mathbf{f}(X)$ it must hold that:

$$\mathbf{c}'_i \geq \max_{\mathbf{x} \in X} D_i \mathbf{f}(\mathbf{x}) \tag{31}$$

for all $i \in \{1, \ldots, 2n\}$.

This condition can be equivalently rewritten using the generator representation as:

$$\mathbf{c}'_i \geq \max_{\mathbf{x} \in [0,1]^n} \mathbf{h}_i(\mathbf{x}) \tag{32}$$

for all $i \in \{1, \ldots, 2n\}$, where $\mathbf{h}_i(\mathbf{x}) = D_i \mathbf{f}(\gamma_U(\mathbf{q}, \mathbf{m}, \mathbf{x}))$ for some fixed $U \subset [0,1]^n$, $\mathbf{q} \in \mathbb{R}^n$, and $\mathbf{m} \in \mathbb{R}^n$. Note that $\mathbf{h}_i(\mathbf{x})$ is a polynomial function of $\mathbf{x}$ that we want to maximize over the unit box. Therefore, we can use Bernstein coefficients to compute an upper bound $\mathbf{c}'_i \in \mathbb{R}$ of the function $\mathbf{h}_i(\mathbf{x})$ for $\mathbf{x} \in [0,1]^n$.

Let $B_{\mathbf{h}_i} = \{\mathbf{b_j} \mid \mathbf{j} \in I^{\mathbf{h}_i}\}$ be the set of Bernstein coefficients of the function $\mathbf{h}_i(\mathbf{x})$ and let $\mathbf{c}' = (\mathbf{c}'_1, \ldots, \mathbf{c}'_{2n})$ be defined as:

$$\mathbf{c}'_i = \max\{\mathbf{b_j} \mid \mathbf{j} \in I^{\mathbf{h}_i}\} \tag{33}$$

for all $i \in \{1, \ldots, 2n\}$. It is easy to see that the vector $\mathbf{c}'$ satisfies the inclusion $\mathbf{f}(X) \subseteq \langle D, \mathbf{c}' \rangle$. This result follows directly from the range enclosing property of Bernstein coefficients (Lemma 1) and the bounding condition of Equation (32).

At this point we have all the ingredients to define an algorithm that bounds a polynomial (the $\mathbf{h}_i(\mathbf{x})$ functions) over a parallelotpe (the set to be transformed). Let us formalize these ideas overloading the algorithm BOUND already defined on boxes in Section 3.1. The function BOUND (Algorithm 5) receives in input a polynomial $\pi : \mathbb{R}^n \to \mathbb{R}$ and a parallelotope $X = \langle D, \mathbf{c} \rangle$ in constraint representation, and returns an upper-bound $\overline{b} \in \mathbb{R}$ of $\pi(\mathbf{x})$ over X.

Algorithm 5 Bound polynomial over parallelotope

```
1: function BOUND(π, X)                          ▷ X = ⟨D, c⟩ ⊂ ℝ^n parallelotope
2:     γ_U(q, m, x) ←CON2GEN(X)                  ▷ Compute generator function
3:     b̄ ←MAXBERNCOEFF(π(γ_U(q, m, x))           ▷ Compute maximum coefficient
4:     return b̄
5: end function
```

The first step for BOUND is to compute the generator function $\gamma_U(\mathbf{q}, \mathbf{m}, \mathbf{x})$ for the parallelotope X as described in Section 3.2.1. The resulting function is composed with π, and the Bernstein coefficients of $\pi(\gamma_U(\mathbf{q}, \mathbf{m}, \mathbf{x}))$ are computed. Finally, the maximum Bernstein coefficient, constituting an upper-bound of π over X, is returned.

Plugging the new algorithm BOUND into REACHSTEP (Algorithm 2), which in turn is used by REACH (Algorithm 1), we obtain a parallelotope-based reachability algorithm.

Example 6 We now show a parallelotope-based single step reachability on the SIR epidemic model of Example 1. Let $X = \langle D, \mathbf{c} \rangle$ be the parallelotope in constraint representation with:

$$D = \begin{pmatrix} -1 & 0 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{c} = \begin{pmatrix} -0.80 \\ -0.95 \\ 0.00 \\ 0.85 \\ 1.00 \\ 0.00 \end{pmatrix} \tag{34}$$

The algorithm REACHSTEP picks the first direction D_1 of the template D and, calling BOUND, bounds the polynomial $D_1\mathbf{f}(\mathbf{x})$ over X. The bounding function computes the generator function $\gamma_U(\mathbf{q}, \mathbf{m}, \mathbf{x})$ of X, that is:

$$\gamma_U(\mathbf{q}, \mathbf{m}, \mathbf{x}) = \begin{pmatrix} \mathbf{q}_1 + 0.7070\mathbf{m}_1 s \\ \mathbf{q}_2 - 0.7070\mathbf{m}_1 s + \mathbf{m}_2 i \\ \mathbf{q}_3 + r\mathbf{m}_3 \end{pmatrix} \tag{35}$$

with $\mathbf{q} = (0.80, 0.15, 0.00)$ and $\mathbf{m} = (0.0707, 0.0500, 0.0000)$. Composing $D_1\mathbf{f}(\mathbf{x})$ with $\gamma_U(\mathbf{q}, \mathbf{m}, \mathbf{x})$, the algorithm generates the polynomial:

$$D_1\mathbf{f}(\gamma_U(\mathbf{q}, \mathbf{m}, \mathbf{x})) = ((\mathbf{q}_1 + 0.7070\mathbf{m}_1 s)0.35(\mathbf{q}_1 + 0.7070\mathbf{m}_1 s)(\mathbf{q}_2 0.7070\mathbf{m}_1 s + \mathbf{m}_2 i)) \tag{36}$$

that instantiated on the base vertex $\mathbf{q}$ and magnitudes $\mathbf{m}$, leads to the Bernstein coefficients:

$$B_{D_1\mathbf{f}(\gamma_U(\mathbf{q},\mathbf{m},\mathbf{x}))} = \{-0.7580, -0.7440, -0.7887, -0.7743, -0.8203, -0.8054\} \tag{37}$$

The maximum coefficient -0.7440 is the offset $\mathbf{c}'_1$ that associated with the direction D_1 generates the first over-approximating half-space $D_1\mathbf{x} \leq \mathbf{c}'_1$. Repeating the procedure for all the directions, we obtain the offset vector $\mathbf{c}' = (-0.7440, -0.9425, -0.0050, 0.8203, 0.9925, 0.0100)$ that coupled with the template D leads to the over-approximating parallelotope $X' = \langle D, \mathbf{c}' \rangle \supseteq \mathbf{f}(X)$.

The constructed parallelotope X' and some reachable points computed sampling initial conditions in X are shown in Figure 5 (in white and black, respectively). Note how X' contains all the computed reachable points.

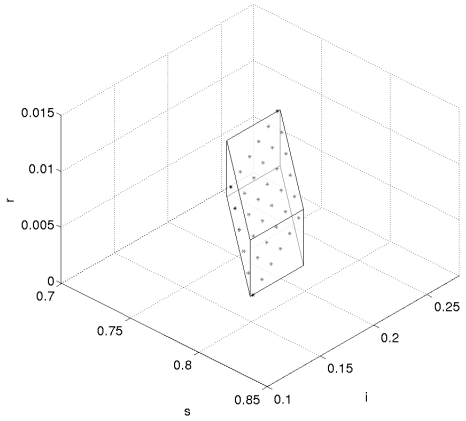


Fig. 5: Parallelotope-based set image approximation. The constructed parallelotope (in white) and some reachable points (in black).

3.3 Parallelotope Bundles

Combining Bernstein coefficients with generic polytopes is hard, since it requires the transformation of the unit box into a polytope. This can be done considering higher dimensional unit boxes [74], but it sensibly increases the computation complexity of the procedure. What we propose here is a new data structure called *parallelotope bundles* to symbolically represent polytopes as intersections of parallelotopes. The core idea behind parallelotope bundles is to exploit the techniques previously developed on parallelotopes to define a bundle-based image algorithm.

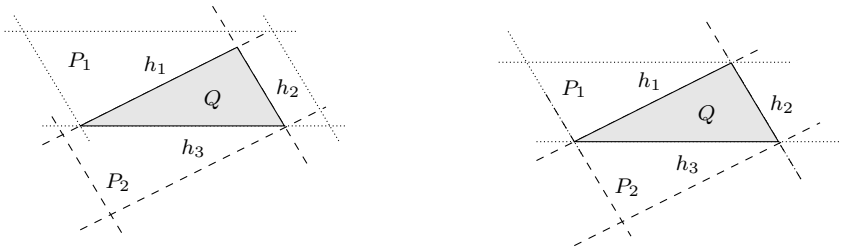
Definition 5 (Parallelotope Bundle) A *parallelotope bundle* $B = \{P_1, \dots, P_b\}$ is a finite set of parallelotopes whose intersection, denoted by $\mathcal{I}(B) = \cap_{i=1}^{b} P_i$, generates a polytope.

The polytope $\mathcal{I}(B)$ represented by a parallelotope bundle B can be represented as the set of all the templates and offsets of the parallelotopes that constitute B.

Lemma 4 (Polytope Decomposition) *Any polytope $Q \subset \mathbb{R}^n$ defined by m constrains can be represented by a bundle involving $\lceil m/n \rceil$ parallelotopes.*

Proof We first demonstrate that for any polytope $Q \subset \mathbb{R}^n$ there exists a parallelotope bundle B such that $Q = \mathcal{I}(B)$. Any set of parallelotopes $P_1, \dots, P_b$ whose hyperplanes union is a cover[1] of the the hyperplanes of Q, is a bundle $B = \{P_1, \dots, P_b\}$ such that $Q = \mathcal{I}(B)$.

[1] A set of nonempty subsets of X whose union contains the given set X is called a *cover* of X.



(a) A polytope Q and a possible decomposing bundle $B = \{P_1, P_2\}$, i.e., $Q = \mathcal{I}(B)$.

(b) A bundle $B = \{P_1, P_2\}$ in canonical form, i.e., $B = \mathcal{S}(B)$.

Fig. 6: Examples of parallelotope bundles.

Next, we show that $\lceil m/n \rceil$ parallelotopes are sufficient to decompose a polytope $Q \subset \mathbb{R}^n$ defined by m constrains. Let $H_Q = \{h_1, \ldots, h_k\}$ be the set of hyperplanes of Q and $H_B = \cup_{i=1}^{b} \{h_1^i, \ldots, h_{2n}^i\} = \{h'_1, \ldots, h'_{k'}\}$ be the union of all the hyperplanes of the parallelotopes of B, for some $k' \in \mathbb{N}, k \leq k' \leq b2n$.

The polytope generated by B is then $\cap_{i=1}^{b} \cap_{j=1}^{2n} h_j^i = \cap_{i=1}^{k'} h'_i$. Since H_B covers H_Q, it holds that $\{h'_1, \ldots, h'_{k'}\} = \{h_1, \ldots, h_k\}$ and then $\cap_{i=1}^{k'} h'_i = \cap_{i=1}^{k} h_i = Q$.

A single parallelotope can match at least n hyperplanes of Q. Then, the total number of sufficient parallelotopes to decompose Q is equal to the number of facets of Q divided by the worst case maximum number of facet matchable by a single parallelotope, i.e., $\lceil m/n \rceil$.

Lemma 4 states that any polytope can be represented by a parallelotope bundle and establishes the maximum number of parallelotopes sufficient to represent a polytope.

Example 7 Figure 6a depicts a polytope Q (in gray) together with a possible bundle $B = \{P_1, P_2\}$ such that $Q = \mathcal{I}(B)$. In this case $m = 3$ and $n = 2$, thus $\lceil 3/2 \rceil = 2$ are sufficient to decompose Q (in our case P_1 and P_2).

A bundle representing a polytope may not be "minimal" in the sense that one or more paralleloptopes can be shrunk while the resulting bundle still represents the same polytope (see, e.g., the parallelotopes of Figure 6a). Thus we can define a shrinking process that removes parts of parallelotopes that are not in the polytope. As we will see later, the shrinking reduces the error when the image over-approximation is performed on shrunk parallelotopes.

Definition 6 (Shrinking) Let $B = \{P_1, \ldots, P_b\}$ be a parallelotope bundle. The *shrinking* $B' = \mathcal{S}(B)$ of B produces a parallelotope bundle $B' = \{P'_1, \ldots, P'_b\}$ such that for all $P'_i = \langle D, \mathbf{c}' \rangle \in B'$ and $P_i = \langle D, \mathbf{c} \rangle \in B$ it holds that:

$$\mathbf{c}'_j = \max_{\mathbf{x} \in \mathcal{I}(B)} D_j \mathbf{x} \tag{38}$$

for $j \in \{1, \ldots, 2n\}$.

Roughly speaking, the shrinking places the hyperplanes of the parallelotopes of a bundle tangent to its polytope. This operation can be done by solving $2n$ linear programs. A bundle that remains unchanged after a shirking is said to be in *canonical form*.

Definition 7 (Canonical From) A bundle B is in canonical from if and only if $\mathcal{S}(B) = B$.

A bundle in canonical form is a "minimal" representation of the polytope with respect to a given set of directions, since all the offsets are shifted towards the constraints of the polytope. The advantage of dealing with bundles in canonical form will become clearer on image approximations.

Example 8 Figure 6b shows the shrinking of the bundle of Figure 6a. Note how all the halfspaces of the parallelotopes are tangent to the polytope Q. The shown bundle is in canonical form.

Note that after shrinking, the hyperplanes of different paralleloptopes might overlap (specifically, those with the same directions, like for instance the hyperplane h_2 of Figure 6b shared by P_1 and P_2). Moreover, the constraints of the paralleloptopes are pairwise parallel, meaning that for a given hyperplane we can obtain the direction of its parallel one by reversing the original direction sign. These observations suggest us that in a bundle there is a lot of redundant information and instead of storing separately each parallelotope, we might think of a data structure that compactly represents bundles in canonical form.

Definition 8 (Bundle Representation) A parallelotope bundle in canonical from can be compactly represented by the tuple $\langle D, \overline{\mathbf{c}}, \underline{\mathbf{c}}, T\rangle$ where:

- $D \in \mathbb{R}^{k\times n}$ is the *direction matrix* that contains the directions used to build the parallelotopes. The i-th row D_i of D represents a direction;
- $\overline{\mathbf{c}} \in \mathbb{R}^k$ is the *upper offsets* vector. The i-th element of $\overline{\mathbf{c}}$ associated with the i-th direction D_i constitutes the halfspace $D_i\mathbf{x} \leq \overline{\mathbf{c}}_i$;
- $\underline{\mathbf{c}} \in \mathbb{R}^k$ is the *lower offsets* vector. The i-th element of $\underline{\mathbf{c}}$ associated with the i-th direction D_i constitutes the halfspace $-D_i\mathbf{x} \leq \underline{\mathbf{c}}_i$ (note the change of sing in the direction);
- $T \in \{1,\dots,k\}^{b\times n}$ is the *template matrix*. Each element in T refers to a direction in D and some offsets in $\overline{\mathbf{c}}$ and $\underline{\mathbf{c}}$. A row in T points to a set of halfspaces that generate a parallelotope.

With a slight abuse of notation we write $B = \{P_1,\dots,P_b\} = \langle D, \overline{\mathbf{c}}, \underline{\mathbf{c}}, T\rangle$ to indicate that the bundle $B = \{P_1,\dots,P_b\}$ is represented by the tuple $\langle D, \overline{\mathbf{c}}, \underline{\mathbf{c}}, T\rangle$.

Example 9 Consider the bundle $B = \{P_1, P_2\}$ in canonical from of Figure 6b where $P_1 = \langle D^1, \mathbf{c}^1\rangle$ and $P_2 = \langle D^2, \mathbf{c}^2\rangle$ with:

$$D^1 = \begin{pmatrix} 1.6 & 1 \\ 0 & 1 \\ -1.6 & -1 \\ 0 & -1 \end{pmatrix} \mathbf{c}^1 = \begin{pmatrix} 10 \\ 3.1 \\ -1 \\ -1 \end{pmatrix} \qquad D^2 = \begin{pmatrix} 1.6 & 1 \\ -0.5 & 1 \\ -1.6 & -1 \\ 0.5 & -1 \end{pmatrix} \mathbf{c}^2 = \begin{pmatrix} 10 \\ 1 \\ -1 \\ 1.7 \end{pmatrix} \tag{39}$$

The bundle $B = \{P_1, P_2\}$ can be represented by the tuple $\langle D, \overline{\mathbf{c}}, \underline{\mathbf{c}}, T\rangle$ where:

$$D = \begin{pmatrix} 1.6 & 1 \\ 0 & 1 \\ -0.5 & 1 \end{pmatrix} \quad \overline{\mathbf{c}} = \begin{pmatrix} 10 \\ 3.1 \\ 1 \end{pmatrix} \quad \overline{\mathbf{c}} = \begin{pmatrix} -1 \\ -1 \\ 1.7 \end{pmatrix} \quad T = \begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix} \tag{40}$$

3.3.1 Bounding over Bundles

Similarly to boxes and parallelotopes, also here we aim to use bundles to over-approximate the image of polytopes. In particular, given a bundle $B = \langle D, \overline{\mathbf{c}}, \underline{\mathbf{c}}, T\rangle$ and a polynomial $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$, we want to find a vector $\mathbf{c}' \in \mathbb{R}^{2k}$ such that $\mathbf{f}(\mathcal{I}(B)) \subseteq \langle D', \mathbf{c}'\rangle$, where D' is a template, possibly composed by the directions of the bundle. Once we compute the polytope $\langle D', \mathbf{c}'\rangle$ we might want to decompose it into a new bundle. Let us start with the problem of bounding a direction over the image of a bundle.

Let $X = \mathcal{I}(B)$ be the polytope represented by the bundle $B = \{P_1, \ldots, P_b\} = \langle D, \overline{\mathbf{c}}, \underline{\mathbf{c}}, T\rangle$ and let D' be a template. We recall that in order for a polytope $X' = \langle D', \mathbf{c}'\rangle$ to over-approximate $\mathbf{f}(X)$ it must hold that:

$$\mathbf{c}'_i \geq \max_{\mathbf{x}\in X} D'_i \mathbf{f}(\mathbf{x}) \tag{41}$$

for all $i \in \{1, \ldots, 2k\}$. As pointed out in Section 2.3, since X is a generic polytope, we may not be able to efficiently solve this optimization problem. However, since $X = \mathcal{I}(\{P_1, \ldots, P_b\}) \subseteq P_i$, for all $i \in \{1, \ldots, b\}$, it holds that:

$$\mathbf{f}(X) \subseteq \bigcap_{i=1}^{b} \mathbf{f}(P_i) \tag{42}$$

This means that for a given direction D'_i, it holds that:

$$\max_{\mathbf{x}\in X} D'_i \mathbf{f}(\mathbf{x}) \leq \max_{\mathbf{x}\in P_j} D'_i \mathbf{f}(\mathbf{x}) \tag{43}$$

for all $j \in \{1, \ldots, b\}$. In particular, we can obtain a tight upper-bound $\overline{\mathbf{c}'_i}$ of $\max_{\mathbf{x}\in X} D'_i \mathbf{f}(\mathbf{x})$ looking at the parallelotopes of a bundle:

$$\overline{\mathbf{c}'_i} = \min_{j\in\{1,\ldots,b\}} \max_{\mathbf{x}\in P_j} D'_i \mathbf{f}(\mathbf{x}) \tag{44}$$

Example 10 Let $B = \{P_1, P_2\}$ be a bundle, $X = \mathcal{I}(B)$ be its polytope, and $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ be a continuous function. Figure 7 depicts the transformations $\mathbf{f}(X)$, $\mathbf{f}(P_1)$, and $\mathbf{f}(P_2)$. Notice that since $X \subseteq P_1 \cap P_2$, the inclusion $\mathbf{f}(X) \subseteq \mathbf{f}(P_1) \cap \mathbf{f}(P_2)$ holds. Let $D_i \in \mathbb{R}^n$ be the direction that we want to bound over $\mathbf{f}(X)$. Since we are not able to directly bound D_i over $\mathbf{f}(X)$, we bound D_i over $\mathbf{f}(P_1)$ and $\mathbf{f}(P_2)$ and then we take the minimum bound. For instance, let $\mathbf{c}_i^1 \geq \max_{\mathbf{x}\in P_1} D_i \mathbf{f}(\mathbf{x})$ and $\mathbf{c}_i^2 \geq \max_{\mathbf{x}\in P_2} D_i \mathbf{f}(\mathbf{x})$ (these bounds can be obtained with the algorithm BOUND defined on parallelotopes in Section 3.2.2). Suppose that $\mathbf{c}_i^2 \leq \mathbf{c}_i^1$ (as depicted in Figure 7). Then $\mathbf{c}_i^2$ is the best candidate to be associated with the direction D_i that leads to the hyperplane $D_i\mathbf{x} \leq \mathbf{c}_i^2$ that tightly includes $\mathbf{f}(X)$.

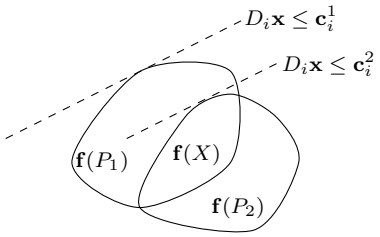


Fig. 7: Bounding the direction D_i over the transformation of a bundle $B = \{P_1, P_2\}$.

From Example 10 it easy to see that we obtain hyperplanes that are closer to the polytope image by considering more parallelotopes. Of course, this has a price in terms of computational complexity. Given a parallelotope bundle $B = \{P_1, \dots, P_b\}$ we might let the user chose on which parallelotopes to bound a direction, specifying the correspondent indices. Let us formalize the bounding algorithm.

Algorithm 6 Bound polynomial over parallelotope bundle

```
1: function BOUND(π, {P_1, ..., P_b})          ▷ {P_1, ..., P_b} parallelotope bundle
2:     for i ∈ {1, ..., b} do
3:         b_i ← BOUND(π, P_i)
4:     end for
5:     b̄ ← min{b_1, ..., b_b}
6:     return b̄
7: end function
```

Let us overload the already defined algorithm BOUND on paralellotope bundles. The new BOUND (Algorithm 6) takes in input a polynomial $\pi : \mathbb{R}^n \to \mathbb{R}^n$ and a parallelotope bundle $B = \{P_1, \dots, P_b\}$, and returns an upper-bound $\bar{b} \in \mathbb{R}$ of $\pi(\mathbf{x})$ over $\mathcal{I}(B)$. For each parallelope P_i with $i \in \{1, \dots, b\}$, the algorithm calls the function BOUND defined on parallelotopes (Algorithm 5) that returns a bound $\mathbf{b}_i$ of $\pi(\mathbf{x})$ over P_i. Once that all the paralleloptopes have been considered, the algorithm returns the minimum bound $\bar{b}$ that is an upper-bound of $\pi(\mathbf{x})$ over the polytope $\mathcal{I}(B)$.

3.3.2 Bundle-based Image Over-approximation

The function BOUND can be used to over-approximate the image of a polytope represented by a bundle. In particular, we can consider a template, bound its directions over the polytope $X = \mathcal{I}(B)$, and then construct the over-approximation polytope $X' \supseteq \mathbf{f}(X)$. If we want to reapply our bundle-based approximation scheme to X', we need to decompose X' in a new bundle. Specifically, we have to define a new templates matrix that leads to X'. In order to use bundles in the reachability algorithm, we need to slightly adapt

the REACHSTEP algorithm (Algorithm 2) considering two main aspects: 1) the directions to be bounded and the parallelotopes of the bundle on which the directions have to be bounded, and 2) the decomposition of the computed polytope into a new bundle. Let us begin by considering the first issue.

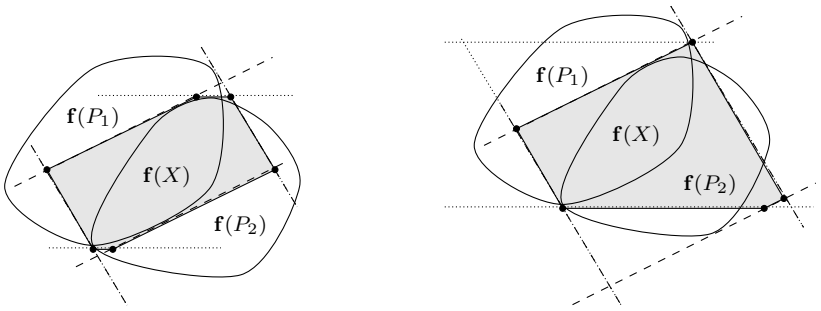
Let $B = \{P_1, \ldots, P_b\} = \langle D, \overline{\mathbf{c}}, \underline{\mathbf{c}}, T \rangle$ be the bundle to be transformed by $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$. We assume that for the over-approximating polytope we use a template D' composed by the same directions of B, i.e., $D' = \left(\begin{smallmatrix} D \\ -D \end{smallmatrix}\right)$. With these ingredients, the completest over-approximation technique that we can develop consists in bounding all the directions of D' over all the paralleloptopes $\{P_1, \ldots, P_b\}$. REACHSTEP (Algorithm 7) formalizes this idea. The algorithm takes in input a bundle $B = \langle D, \overline{\mathbf{c}}, \underline{\mathbf{c}}, T \rangle$ and goes through all the directions of D. The composition of each direction D_i with the function $\mathbf{f}(\mathbf{x})$ (and its parallel version $-D_i$ with $\mathbf{f}(\mathbf{x})$) is bounded over all the parallelotopes of the bundle with the function BOUND generating the offset vectors $\overline{\mathbf{c}}'$ and $\underline{\mathbf{c}}'$. At this point, the polytope $\langle D', \mathbf{c}' \rangle$, with $\mathbf{c}' = \left(\begin{smallmatrix} \overline{\mathbf{c}}' \\ \underline{\mathbf{c}}' \end{smallmatrix}\right)$ is an over-approximation of $\mathbf{f}(\mathcal{I}(B))$. We refer to this approach as the *all-for-one (AFO)* transformation, since we bound all the directions on each single parallelope. This method requires $\Theta(2kb)$ bounding computations.

Algorithm 7 Bundle-based reach step (AFO technique)

```
1: function REACHSTEP(B)                                   ▷ B = {P_1, ..., P_b} bundle
2:     for i ∈ {1, ..., k} do
3:         c̄'_i ←BOUND(D_i f(x), {P_1, ..., P_b})
4:         c̲'_i ←BOUND(−D_i f(x), {P_1, ..., P_b})
5:     end for
6:     B' ← DECOMPOSE(⟨D, c̄', c̲', T⟩)                      ▷ optional
7:     return B'
8: end function
```

Since we might want to apply again a bundle-based image over-approximation, we should decompose the polytope $\langle D', \mathbf{c}' \rangle$ into a new bundle B'. A static (and fast) approach consists in keeping the same parallelotope templates of B also for B', i.e., REACHSTEP can return the bundle $B' = \langle D, \overline{\mathbf{c}}', \underline{\mathbf{c}}', T \rangle$ without any decomposition. Otherwise, we have to define the composition function DECOMPOSE.

Before focusing on the decomposition, we define a faster but rougher over-approximation method. The technique consists in considering only a subset of parallelotopes when bounding the directions of the bundle. Specifically, for each parallelope P_i, we bound only the directions that compose P_i over itself. This corresponds to independently over-approximate each parallelotope by its own template. This method can be easily obtained by slightly modifying Algorithm 7. We refer to this technique as *one-for-one (OFO)* transformation, since the directions of one parallelope are bounded only with respect to one parallelope. In doing so, the number of bound computations is $\Theta(2nb)$. The OFO technique requires less optimizations than AFO (note that $k \geq n$) but



(a) AFO already shrunk. (b) OFO after shrinking.

Fig. 8: AFO and OFO transformations.

it might produce coarser results. Note that the OFO over-approximation can be seen as the independent over-approximation of each parallelotope of the bundle that generates a new bundle which can be shrunk/canonized and then represented with our data structure.

Example 11 Figure 8 shows the AFO and OFO transformations using the directions of the bundle of Figure 6 and the transformed parallelotopes of Figure 7. Figure 8a depicts the AFO transformation, where all the directions are bounded over all the parallelopes and the smallest offsets are kept. The gray area is the polytope resulting from the AFO transformation.

Figure 8b shows the OFO transformation, where the directions of each parallelotope are bounded over the image of the parallelotope itself. The gray area is the polytope obtained by shrinking the result of OFO transformation.

Note how the AFO over-approximation polytope (Figure 8a) is included in the OFO one (Figure 8b).

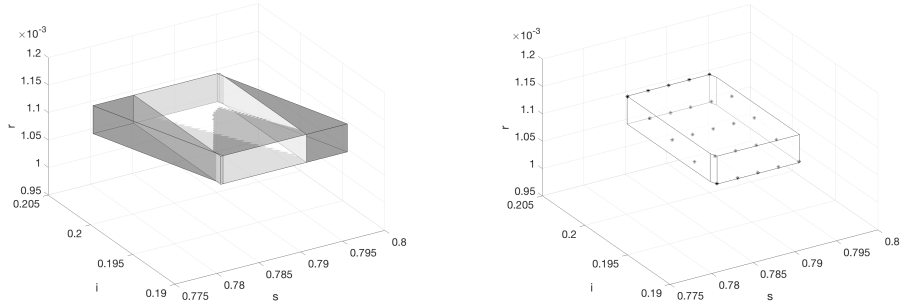
Example 12 As an illustrative example, we now show a bunlde-based single step reachability of the SIR epidemic model presented in Example 1. Let us consider the AFO transformation, i.e., the approach in which each direction is bounded over each parallelotope of the bundle (see Section 3.3.2).

Let $X \subset \mathbb{R}^n$ be a set represented by the bundle $B = \langle D, \overline{\mathbf{c}}, \underline{\mathbf{c}}, T \rangle$ with:

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0.5 \end{pmatrix} \qquad \overline{\mathbf{c}} = \begin{pmatrix} 0.8 \\ 0.2 \\ 0.0 \\ 1.0 \end{pmatrix} \qquad \underline{\mathbf{c}} = \begin{pmatrix} -0.79 \\ 0.19 \\ 0.0 \\ 0.0 \end{pmatrix} \qquad T = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \end{pmatrix} \quad (45)$$

Intuitively, the set $X = \mathcal{I}(B)$ is the box with $s \in [0.79, 0.80], i \in [0.19, 0.20]$, and $r = 0.00$. We want to determine new offset vectors $\overline{\mathbf{c}}', \underline{\mathbf{c}}'$ for a new parallelotope bundle B' such that $\mathcal{I}(B') \supseteq \mathbf{f}(X)$.

To do so, we apply the AFO transformation algorithm, that bounds all the directions of the matrix D over the parallelotopes described by the templates matrix T, and keeps the tightest bound. The bounding operations are



(a) The parallelotopes of the constructed bundle (in white and gray).

(b) The symbolic polytope (in white) and some reachable points (in black).

Fig. 9: Bundle-based set image approximation.

carried out using the methods previously described (see Section 3.2.2). For instance, by bounding the direction D_3 over the two parallelotopes, we obtain the upper-bounds 0.89 and 0.99, respectively. Thus, the algorithm keeps the tightest bound that is the offset $\overline{\mathbf{c}}'_3 = 0.89$ for the over-approximating bundle under construction. Repeating this operations for all the directions and parallelotopes, we obtain the offset vectors $\overline{\mathbf{c}}' = (0.79, 0.20, 0.001, 0.89)$ and $\underline{\mathbf{c}} = (-0.78, -0.19, -0.001, -0.88)$ that grouped with the direction and template matrices constitute the new bundle $B' = (D, \overline{\mathbf{c}}', \underline{\mathbf{c}}', T)$.

Figure 9a shows the parallelotopes that compose the new bundle (in white and gray). Figure 9a shows the symbolic polytope generated by the intersection of the two parallelotopes (in white) with some reachable points computed by sampling initial conditions in X. Note how all the points fall in the computed bundle.

3.3.3 Polytope Decomposition

As earlier pointed out, in our bundle-based over-approximation algorithm we may be interested in decomposing a polytope into a bundle (see Algorithm 7). Hence, we now define the function DECOMPOSE that receives in input a bundle B (whose polytope $\mathcal{I}(B)$ has to be decomposed) and reorganizes its the templates matrix creating a new collection of parallelotopes around the polytope $\mathcal{I}(B)$. The goal of the decomposition is to create a set of small parallelotopes whose intersection is $\mathcal{I}(B)$. There are two reasons why we want small parallelotopes:

1. Smaller parallelotopes P_i lead to a smaller bundle image $\{\mathbf{f}(P_1), \ldots, \mathbf{f}(P_d)\}$ and then to a more accurate over-approximation of $\mathbf{f}(\mathcal{I}(B))$;
2. The shorter the largest side length of P_i, the more accurate the over-approximation introduced by the Bernstein coefficients (see Lemma 3).

Thus, the aspects to be taken into account in the construction of the parallelotopes are volume and maximum side length. Moreover, we do not have to forget that the set of the parallelotope directions must cover the directions of the polytope to be decomposed (see Lemma 4). Finding the best decomposition in terms of volume and maximum length minimization is computationally expensive and might not be possible (recall that the set cover problem is NP-hard [45]).

In order to efficiently find a good decomposition, we propose a heuristic that constructs the parallelotopes while trying to minimize the volumes and maximum side lengths. The proposed heuristic starts from a decomposition, applies a series of random changes to the templates matrix, and keeps only the best one accordingly to an evaluation function that we will soon define. The procedure is repeated until a fixed number of iterations is reached.

Given a bundle $B = \{P_1, \dots, P_b\}$, the evaluation function should take into account the volumes and side lengths of the parallelotopes P_i, for $i \in \{1, \dots, b\}$. The exact computation of the volume of a parallelotope is rather expensive, since it is equal to the determinant of a $n \times n$ matrix. To lighten the computation, we approximate the volume of $P = \langle D, \mathbf{c} \rangle$ with the product of the distances of its constraints:

$$\tilde{v}(P) = \prod_{i=1}^{n} \delta(D_i \mathbf{x} \leq \mathbf{c}_i, D_{i+n} \mathbf{x} \leq \mathbf{c}_{i+n}) \tag{46}$$

where $\delta(D_i \mathbf{x} \leq \mathbf{c}_i, D_{i+n} \mathbf{x} \leq \mathbf{c}_{i+n}) = |\mathbf{c}_i - \mathbf{c}_{i+n}| / \|D_i\|$ and $\|\cdot\|$ is the Euclidean norm.

The computation of the side lengths of a parallelotope passes inevitably through the determination of its vertices, an operation that can be computationally expensive. Instead of calculating the exact lengths, we opt for a faster heuristic that guesses the lengths of a parallelotope from the angles of the directions of its constraints. Intuitively, in the two-dimensional case, having fixed two parallel lines, the lengths of the edges not lying on the two fixed lines are minimal when the added directions and the fixed ones are orthogonal. Thus, we define the notion of *orthogonal proximity* $\theta(D_i, D_j) = \widehat{D_i, D_j} \pmod{\pi/2}$, where $\widehat{D_i, D_j}$ is the angle between D_i and D_j, i.e., $\widehat{D_i, D_j} = \arccos \dfrac{D_i D_j}{\|D_i\| \|D_j\|}$. The orthogonal proximity of a parallelotope $P = \langle D, \mathbf{c} \rangle$ is defined as:

$$\Theta(P) = \max_{i,j \in \{1,\dots,2n\}} \theta(D_i, D_j). \tag{47}$$

Exploiting the notions of approximated volume $\tilde{v}$ and orthogonal proximity Θ, we define the evaluation function w for a bundle as:

$$w(\{P_1, \dots, P_b\}) = \max_{i \in \{1,\dots,b\}} \alpha \tilde{v}(P_i) + (1 - \alpha) \Theta(P_i) \tag{48}$$

where $\alpha \in [0, 1]$ is a tunable parameter.

4 Experimental Evaluation

In this section we experimentally evaluate the reachability techniques presented in this paper. We take into account several dynamical systems and observe how different methods influence flowpipe construction and computational times. As expected, the larger the number of adopted templates and directions, the preciser the constructed flowpipe and the longer the computation are.

Our methods have been gathered in the tool called *Sapo* [29].[2] The tool is in C++ and it requires two external libraries: GiNaC[3] (GiNaC is Not a CAS) for handling symbolic polynomials, and GLPK[4] (GNU Linear Programming Kit) for solving linear programs. For more details on Sapo, the reader can refer to [30].

The experimental evaluation begins with Section 4.1, where several dynamical systems, sorted by increasing dimension, are presented. We will study the Van der Pol oscillator (2d) [67], the Rössler attractor (3d) [69], the SIR epidemic model (3d) [46], a generalized Lotka-Volterra model (5d) [79], a phosphorelay systems (7d) [47], and a quadcopter drone model (17d) [22]. Each model will be analyzed focusing on different aspects that influence the reachability computation, such as the number of bundle directions and templates used to construct the flowpipes, or the set transformation method (OFO or AFO; see Section 3.3.2). All the obtained computational times and details on models and reachability configurations are summarized in Table 1. For simplicity, the direction and template matrices used to construct the bundles that constitute the computed flowpipes are collected in Appendix A.

In Section 4.2 we compare our tool Sapo with Flow* [17], the state-of-the-art tool for the reachability analysis on nonlinear dynamical and hybrid systems. We will discuss the main differences of the results provided by the tools and their running times. The comparison is summarized in Table 1.

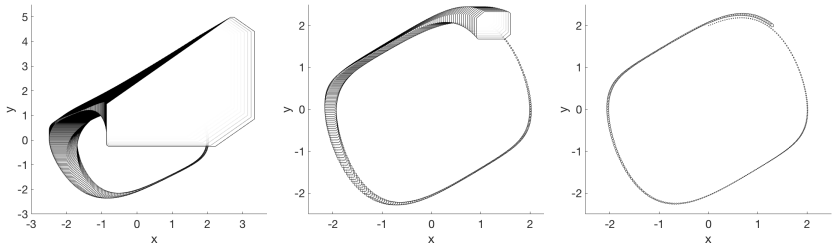
All the experiments have been performed on a MacBook Pro (3.1 GHz, 16 GB DDR3 RAM).

4.1 Case Studies

4.1.1 Van der Pol Oscillator

The first studied model is the Van der Pol oscillator [67], an important two dimensional nonlinear system that found application in many physical and biological models. A commonly used form of the discrete-time Van der Pol

[2] http://tommasodreossi.github.io/sapo/
[3] http://www.ginac.de
[4] https://www.gnu.org/software/glpk/



(a) 4 dirs/2 temps (0.70s) (b) 4 dirs/4 temps (1.79s) (c) 4 dirs/6 temps (3.48s)

Fig. 10: Reachable sets of Van der Pol oscillator with increasing number of templates (350 steps).

oscillator is given by the following dynamics:

$$\begin{aligned} x_{k+1} &= x_k + (y_k)\Delta \\ y_{k+1} &= y_k + (\mu(1 - x_k^2)y_k - x_k)\Delta \end{aligned} \tag{49}$$

where μ is a scalar parameter indicating the nonlinearity and the strength of the damping of the system, and Δ is the discretization step.

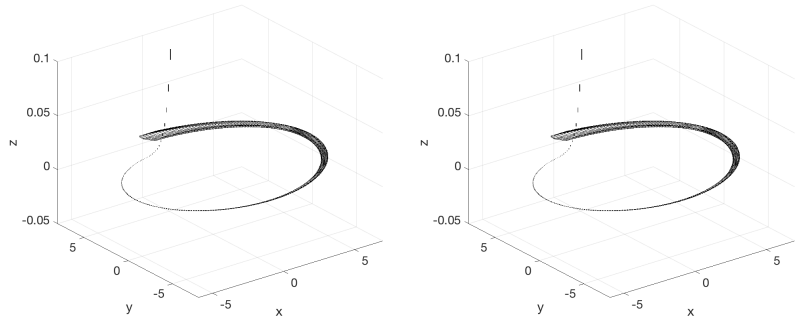
For our experiments we set $\mu = 0.5$ and $\Delta = 0.02$. The constructed bundles use four directions and from one to six templates (for details, see Appendix A.1). For each experiment, we increase the number of templates that constitute the bundle. For instance, in the first experiment, we consider a single template represented by the first direction of the templates matrix, in the second, the first two directions, and so on.

The set of initial conditions is the box with $x_0 \in [0.00, 0.01]$ and $y_0 \in [1.99, 2.00]$. We computed the flowpipe for 300 steps, corresponding to a full cycle of the oscillator. Some computed flowpipes are depicted in Figure 12. In particular, Figure 10a shows the computed flowpipe using 4 directions combined in 2 templates (0.70s), Figure 10b 4 directions in 4 templates (3.48s), and Figure 10c 4 directions in 6 templates (1.79s). Notice how adding templates leads to finer flowpipes at the expense of longer running times.

4.1.2 Rössler Attractor

We now consider the Rössler attractor [69,70], a three dimensional nonlinear dynamical system defined to study chaotic phenomena and that has found application in modeling equilibria of chemical reactions. The dynamics of the attractor are the following:

$$\begin{aligned} x_{k+1} &= x_k + (-y_k - z_k)\Delta \\ y_{k+1} &= y_k + (x_k + a y_k)\Delta \\ z_{k+1} &= z_k + (b + z_k(x_k - c))\Delta \end{aligned} \tag{50}$$



(a) OFO transformation (0.77s) (b) AFO transformation (0.99s)

Fig. 11: Reachable sets of Rössler attractor (5 dirs/3 temps, 250 steps).

where a, b, and c are scalar parameters and Δ is the discretization step. We chose the common parameter values $a = 0.1$, $b = 0.1$, $c = 14$, and step $\Delta = 0.025$. The considered bundles are composed by five directions and three templates (for details, see Appendix A.2).

We computed two flowpipes using the same bundle, but different transformation methods OFO and AFO (see Section 3.3.2). The set of initial conditions is the box with $x_0 \in [0.09, 0.10], y_0 \in [4.99, 5.00]$, and $z_0 \in [0.09, 0.10]$ and the total number of steps is 250.

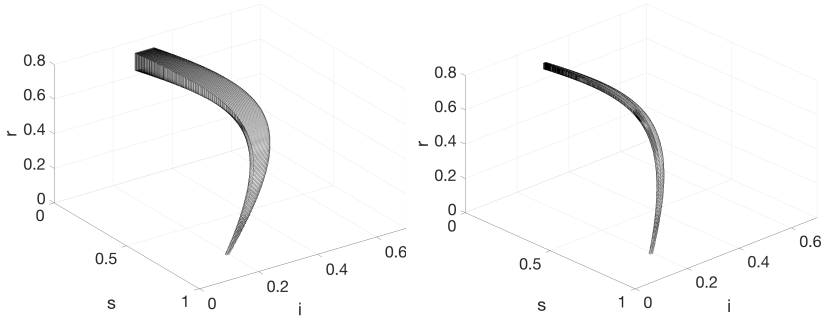
Figures 11a and 11b show the computed flowpipes using the OFO (0.77s) and AFO (0.99s) transformations, respectively. As expected, the OFO transformation is faster than the AFO one, but it interesting to notice that for this case the difference between the two computed flowpipes is very small. However, it is important to remark that this is a lucky case. In general, as we will see later, the difference between OFO and AFO can be sensible.

4.1.3 SIR Epidemic Model

Let us consider the running example adopted along this paper, that is the SIR epidemic model [46], a three dimensional dynamical system that describes the evolution of a disease in a population. We recall the dynamics of the system:

$$\begin{aligned} s_{k+1} &= s_k - (\beta s_k i_k / N)\Delta \\ i_{k+1} &= i_k + (\beta s_k i_k / N - \gamma r_k)\Delta \\ r_{k+1} &= r_k + (\gamma i_k)\Delta \end{aligned} \tag{51}$$

The model partitions a population of $N \in \mathbb{R}_{\geq 0}$ individuals in three compartments: s_k the people susceptible to the disease, i_k the infected individuals, and r_k the individuals removed from the systems. The migrations of individuals between different compartments are regulated by the parameters β, that is the contraction rate, and γ, where $1/\gamma$ is the average infection period.



(a) 3 dirs/1 temps (0.19s) (b) 5 dirs/3 temps (1.78s)

Fig. 12: Reachable sets of SIR epidemic model with increasing number of directions and templates (300 steps).

For our experiments we fix the parameters $\beta = 0.34$, $\gamma = 0.05$, and $\Delta = 0.1$. The set of initial conditions is the box with $s_0 \in [0.79, 0.80]$, $i_0 \in [0.19, 0.20]$, and $r_0 \in [0.00, 0.00]$.

We perform two experiments: the first where the bundle consists in a single box template, the second where five different directions are grouped in one box template and two parallelotopic ones (for details, see Appendix A.3). In both cases we computed 300 reachable steps using the AFO transformation. The obtained flowpipes are depicted in Figures 12a and 12b, respectively. The box-based analysis required 0.19s against the 1.78s of the bundle-based one but, also in this case, a more complex bundle leads to a sensibly finer result.

4.1.4 Generalized Lotka-Volterra Model

Growing in dimension, we now consider a five-dimensional generalization of the well-known Lotka-Volterra model [79] (sometimes called the predator-prey model [60,78]), that is an important dynamical system used to describe the evolutions of biological systems in which different species interact. The original model involves only two spices; here we model five interacting spices [79]. The model's dynamics are the following:

$$\begin{aligned} v_{k+1} &= v_k + (v_k(1 - (v_k + \alpha w_k + \beta z_k)))\Delta \\ w_{k+1} &= w_k + (w_k(1 - (w_k + \alpha x_k + \beta v_k)))\Delta \\ x_{k+1} &= x_k + (x_k(1 - (x_k + \alpha y_k + \beta w_k)))\Delta \\ y_{k+1} &= y_k + (y_k(1 - (y_k + \alpha z_k + \beta x_k)))\Delta \\ z_{k+1} &= z_k + (z_k(1 - (z_k + \alpha v_k + \beta y_k)))\Delta \end{aligned} \tag{52}$$

where α and β are the interaction parameters and Δ is the discretization step. We fix the parameter values as $\alpha = 0.85$, $\beta = 0.50$, and $\Delta = 0.01$. The chosen

set of initial conditions is the box whit $v_0, w_0, x_0, y_0, z_0 \in [0.95, 1.00]$, i.e., each variable spans in the interval $[0.95, 1.00]$.

For this model we run several experiments, starting from a simple box template and then adding constraints and parallelotopeic templates (for details, see Appendix A.4). For the multiple template experiment we apply both the OFO and AFO transformations (for the single one, the two methods are equivalent). In all the cases we compute the reachable up to 500 steps. The obtained running times are exposed in Table 1.

Surprisingly, the flowpipes constructed with different methods are the same, meaning that for this model adding directions and templates does not improve the precision of the results. However, this experiments gives us an idea of how our method scales in the bundle size on a medium sized dynamical system.

4.1.5 Phosphorelay Systems

Let us now consider a model arising from molecular biology that describes signal transduction, i.e., the activation of a receptor located inside a cell or on its surface triggered by an extracellular signal. Specifically, we study the phosphorelay signaling pathways system presented in [47]. The discrete-time version of the system consists of the following seven difference equations:

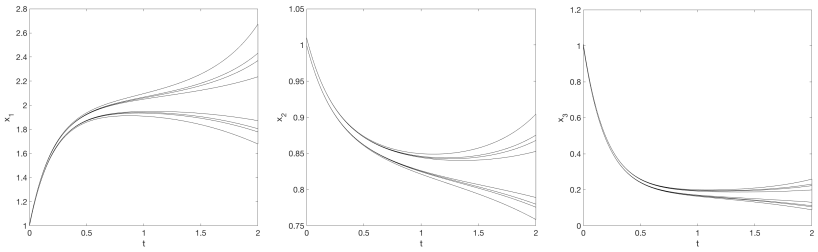
$$\begin{aligned}
Sln1_{k+1} &= Sln1_k + (-k_1 \cdot Sln1_k + k_3 \cdot Sln1AP_k \cdot Ypd1_k)\Delta \\
Sln1HP_{k+1} &= Sln1HP_k + (k_1 \cdot Sln1_k - k_2 \cdot Sln1HP_k)\Delta \\
Sln1AP_{k+1} &= Sln1AP_k + (k_2 \cdot Sln1HP_k - k_3 \cdot Sln1AP_k \cdot Ypd1_k)\Delta \\
Ypd1_{k+1} &= Ypd1_k + (k_4 \cdot Ypd1P_k \cdot Ssk1_k - k_3 \cdot Sln1AP_k \cdot Ypd1_k)\Delta \\
Ypd1P_{k+1} &= Ypd1P_k + (-k_4 \cdot Ypd1P_k \cdot Ssk1_k + k_3 \cdot Sln1AP_k \cdot Ypd1_k)\Delta \\
Ssk1_{k+1} &= Ssk1_k + (k_5 \cdot Ssk1P_k - k_4 \cdot Ypd1P_k \cdot Ssk1_k)\Delta \\
Ssk1P_{k+1} &= Ssk1P_k + (-k_5 \cdot Ssk1P_k + k_4 \cdot Ypd1P_k \cdot Ssk1_k)\Delta
\end{aligned} \tag{53}$$

where k_1, k_2, k_3, k_4, k_5 are parameters and Δ is the discretization step. We adopt the parameter values proposed in [47], i.e., $k_1 = 0.4, k_2 = 1.0, k_3 = 5.0, k_4 = 5.0, k_5 = 0.5$, and step $\Delta = 0.01$. The set of initial conditions is the box where all the variables span in the interval $[1.00, 1.01]$.

In this experiment we analyze the scalability of our methods applying always the AFO transformation and increasing the number of directions and templates that compose the bundle used to construct the over-approximation flowpipe. We compute the reachable set for 200 steps.

We perform four experiments, starting from seven directions grouped in a single box template, and adding, at each time, a new direction and template. At the end, we will obtain a bundle composed by ten directions and four parallelotopic templates (for details, see Appendix A.5).

Some illustrative projections of the constructed flowpipes are depicted in Figure 13. Specifically, the projections over time of the variables $Sln1$, $Sln1HP$, and $Sln1AP$ are shown in Figures 13a, 13b, and 13c, respectively. Each plot



(a) $Sln1$ (b) $Sln1HP$ (c) $Sln1AP$

Fig. 13: Projections of seven dimensional biochemical model (200 steps) with increasing number of directions and templates (7/1, 8/2, 9/3, 10/4 dirs/temps). Computation times (sec): 0.85, 2.75, 6.56, 15.51, respectively.

overlaps the projections of the flowpipes computed with different directions and templates.

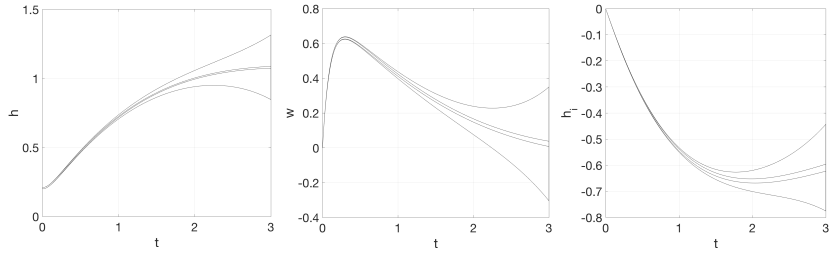
Differently from some previous experiments, each time we add a new direction and template, we obtain a finer over-approximation (see Section 4.1.4). Of course this has a cost (between 7 directions in 1 template and 10 directions in 3 templates there is a gap of 15s) but the precision gained from additional parallelotopes is remarkable. All the running times are reported in Table 1.

4.1.6 Quadcopter Drone

As last case study, we consider a seventeen dimensional model of a quadcopter drone. The goal of this study is to show the scalability of our methods in terms of system's dimension. Moreover, we will see how a single additional template can lead to a fine flowpipe. The studied quadcopter drone model [22] is composed by 17 variables, 13 of which represent the drone plant and 4 the drone controller. The plant state variables include the inertia position of the drone (p_n, p_e, h), its linear velocities (u, v, w), the drone orientation described by Euler angles expressed using quaternions (q_0, q_1, q_2, q_3), and the angular velocities (p, q, r). The controller variables (h_I, u_I, v_I, ψ_I) involve some parameters of position, speed, and orientation. For a given reference height h_r, horizontal speeds u_r, v_r, and nose orientation ψ_r, the task of the controller is to stabilize the drone from a configuration to the one specified by the reference values. The quadratic dynamics of the model and its detailed description can be found in [22]. The parameter chosen for our experiments, like mass, axis moment of inertia, propeller masses, etc.) are taken from the real quadcopter CrazyFlie Nano by Bitcraze.[5]

The chosen set of initial conditions is the box $h_0 \in [0.20, 0.21]$, $q_0 = 1.00$, and all the other variables set to zero. The reference height is $h_r = 1.00$ and speed and orientation are null, i.e., $u_r = v_r = \psi_r = 0$. We computed the

[5] https://www.bitcraze.io/



(a) Height (h) (b) Vertical speed (w) (c) Controller height (h_I)

Fig. 14: Projections of seventeen dimensional quadcopter model (300 steps) with increasing number of directions and templates (17/1, 18/2 dirs/temps). Computation times (sec): 5.67, 12.29, respectively.

reachable set for 300 steps with a discretization step $\Delta = 0.01$, corresponding to 3s of flight. Two experiments have been carried one: in the first we adopted a single box template, in the second we added a parallelotope whose non-axis aligned hyperplanes are not aligned with the dimensions that more vary during the flight, such as height, vertical speed, angle quaternions, and controller height (for details, see Appendix A.6). Both the computed flowpipes have been calculated using the AFO transformation method. The first experiment involving 17 directions and 1 template took 5.67s, the second experiment based on 18 directions and 2 templates took 12.29s.

Figure 14 shows some projections over time of the computed flowpipes. In particular, Figure 14a depicts the height h, Figure 14b the vertical speed w, and Figure 14c the height h_I computed by the controller. The figures highlight the gain of precision provided by a single additional direction and template. Note how the projections of the second flowpipe are sensibly thinner than the first ones and the wrapping effect is notably reduced.

4.2 Comparison

To conclude, we compare our tool Sapo with Flow* [17], the state-of-the-art tool for the computation of reachable set of nonlinear dynamical and hybrid systems. Flow* represents and computes flowpipes as the integration of finite sets of Taylor models [9]. Making a fair comparison between Sapo and Flow* is difficult since, for instance, the latter gives the possibility to specify error bounds on the computed flowpipes, feature that Sapo does not have yet. However, Sapo's errors can be bounded by Lemma 3 and reduced by exploiting convergent subdivision techniques [38,62]. In the following experiments no subdivision or splitting techniques have been applied by Sapo. Another important difference between Sapo and Flow* is that the first works only with discrete-time dynamical systems, while the second deals with continuous-time ones. To make the comparison as fair as possible, we discretized the original

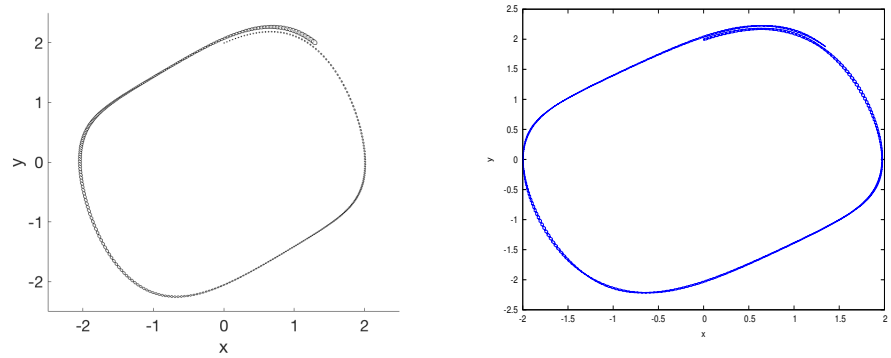
Model			Sapo			Flow*		
Name	Vars	Steps	Dirs/Temps	Trans	Time	TM	ϵ	Time
Van der Pol [67]	2	300	4/2	AFO	0.70	4	10^{-4}	1.34
			4/4	AFO	1.79			
			4/6	AFO	3.48			
Rössler [69]	3	250	5/3	OFO	0.77	4	10^{-4}	0.94
			5/3	AFO	0.99			
SIR [46]	3	300	3/1	AFO	0.19	4	10^{-4}	1.54
			5/3	AFO	1.78			
Lotka-Volterra [79]	5	500	5/1	AFO	1.78	4	10^{-4}	42.47
			7/2	OFO	24.83			
			7/2	AFO	49.70			
			7/3	OFO	46.92			
			7/3	AFO	89.67			
Phosphorelay [47]	7	200	7/1	AFO	0.85	4	10^{-1}	-
			8/2	AFO	2.75			
			9/2	AFO	6.56			
			10/3	AFO	15.51			
Quadcopter [22]	17	300	17/1	AFO	5.67	4	10^{-1}	-
			18/2	AFO	12.29			

Table 1: Reachability methods evaluation on dynamical systems. Model: model's name; Vars: model's dimension; Steps: reachability steps; Dirs/Temps: number of used directions and templates; Trans: bundle transformation method (OFO, one-for-one; AFO, all-for-one); Time: computation time (in seconds).

continuous dynamical systems with the Euler method with a fixed size step. The discretized models have been given in input to Sapo, while the continuous ones, together with the chosen discretization step, have been given in input to Flow*. For details on the discretization steps, see the model descriptions in Section 4.1.

As a benchmark, we run both the tools on the dynamical systems presented in Section 4.1. In particular, for each model, Sapo has been tested using the different configurations (described in Section 4.1 and Appendix A), while Flow* parameters has been fixed as suggested by the user manual. Specifically, `fixed orders` $= 4$, `cutoff threshold` $= 10^{-10}$, `precision` $= 53$, and `identity precondition`. The Taylor model `fixed orders` and `reminder estimation` vary depending on the experiment as shown in Table 1. Also, the schemes for polynomial ODEs are `poly ode 1` for systems with a at most 3 variables, `poly ode 2` with at most 5, and `poly ode 3` for more than 5.

Table 1 summarizes the obtained experimental results. For each case study, the table reports the model's name (Name), the model's dimension in number of variables/dynamics (Vars), and the total number of computed reachable steps (Steps). For Sapo, the table indicates the number of adopted directions and templates used to construct the parallelotope bundles (Dirs/Temps), the kind of bundle transformation (Trans), and the running time. For Flow*, the table reports the Taylor model order (TM), the reminder estimation (ϵ), and the running times. All the computational times are in seconds.



(a) Sapo (3.48s)

(b) Flow* (1.34s)

Fig. 15: Reachable sets of Van der Pol oscillator obtained with about the same computational times.

From the experiments, we can see how for large models Sapo is overall faster than Flow*. With the chosen configuration, Flow* was not able to compute the reachable set of models with more than seven variables (the error message returned after some reachablility steps was: "`The reminder estimation is not large enough`"). Again, this comparison does not take into account the precision of the results provided by the tools. However, juxtaposing the flowpipes produced by the tools on the Van der Pol oscillator (see Figure 15), we can get an idea of how they behave on small models using the same amount of time. From the figure, we can observe that the flowpipes are similar, even if for longer rechable computations, Sapo is more likely to accumulate over-approximation error. On the other hand, the strength of Sapo is that it can be applied to models whose dimensions are double the size of those handled by Flow* and it can still produce reasonably fine flowpipes (see, e.g., Section 4.1.6).

5 Conclusion

5.1 Summary

This work presents three methods for the computation of bounded time reachable sets of polynomial dynamical systems. The goal of the developed algorithms is to produce a flowpipe that over-approximates the reachable set of a dynamical system starting from a set of initial conditions. The three methods differ from the basic sets used to construct the flowpipes. We developed algorithms for the reachability computation based on boxes (i.e., hyperrectangles), parallelotopes (i.e., n-dimensional parallelograms), and parallelotope bundles (i.e., polytopes represented as symbolic intersections of parallelotopes). All these methods are based on the idea of fixing a template for the

over-approximation set and lifting the template's constraints as close as possible to the actual (possibly non-convex) reachable set. Whenever nonlinear dynamical systems are involved, this operation consists in solving a nonlinear optimization problem. We propose to determine upper bounds to these optimization problems using a particular property of Bernstein coefficients of polynomials. Intuitively, instead of involving nonlinear optimizations, one can compute the Bernstein coefficients of a polynomial and extract their maximum, that, for a specific property of Bernstein coefficients, is an upper bound of the polynomial. However, this property holds only for unit box domains. A consistent part of our work was to lift this property to more generic domains, such as boxes, parallelotopes, and symbolic polytopes. From this basic operation, we defined algorithms to over-approximate the image of boxes, parallelotopes, and parallelotope bundles with respect to polynomials, which eventually led us to the definition of a reachability algorithm for polynomial dynamical systems.

All our techniques have been gathered in a C++ tool called Sapo and tested on several case studies. Considering dynamical systems from two to seventeen dimensions, we got an idea of how our techniques scale in terms of both system dimension and complexity of the adopted templates. Moreover, we compared Sapo to Flow*, the state-of-the-art tool for nonlinear reachability analysis.

5.2 Future Work

We intend to improve and extend the methods presented in this work in different directions. First of all, we plan to have a better control on the wrapping effect introduced by the over-approximations during the reachability computation. This task can be achieved in two manners: by improving the upper bounds provided by Bernstein coefficients and by selecting set templates that better wrap the real reachable sets. Several subdivision techniques for obtaining tighter bounds from Bernstein coefficients already exist [38,62]. It is our intention to integrate and automatize these techniques in our algorithms. The selection of good templates is well-known for being a hard problem. However, form our experiments, we observed that the definition of directions with non-null components in the dimensions that more vary during the evolution of the system, generally increases the precision of the computed flowpipes. Hence, we intend to better investigate the relationship between the system's dynamics and the template definition.

There are also interesting ways in which we plan to extend this work. A natural continuation could be towards hybrid automata, that are models that describe systems characterized by the alternation of continuous and discrete behaviors. The set image techniques developed in this work can be adapted for the reachability analysis of hybrid automata. Moreover, in our works [31,24, 25] we considered the parameter synthesis problem for polynomial dynamical systems, that is the problem of finding sets of parameters under which the system satisfies a given specification. The methods that we developed are based only on boxes and parallelotopes. Thus, we intend to study the parameter

synthesis problem also involving paralleltope bundles with the goal of obtaining less restrictive sets of parameters. Note that these techniques can also be exploited to synthesize inputs, which differ from parameters since they can vary during the evolution of the system. Indeed, it is our intention to consider also the input synthesis problem for polynomial dynamical systems. Finally, we want to emphasize that several components of our algorithms can be easily parallelized. For instance the Bernstein coefficients of different dynamics can be independently computed as well as the different parallelotopes of a bundle can be independently transformed at each reachable step. It might be interesting to investigate a parallel version of our algorithms exploiting ad-hoc tools for parallel computation [41,63].

References

1. Althoff, M., Le Guernic, C., Krogh, B.H.: Reachable set computation for uncertain time-varying linear systems. In: Hybrid Systems: Computation and Control, HSCC, pp. 93–102. ACM (2011)
2. Anai, H., Weispfenning, V.: Reach set computations using real quantifier elimination. In: Hybrid Systems: Computation and Control, HSCC, pp. 63–76 (2001)
3. Asarin, E., Bournez, O., Dang, T., Maler, O.: Approximate reachability analysis of piecewise-linear dynamical systems. In: Hybrid Systems: Computation and Control, HSCC, pp. 20–31. Springer (2000)
4. Ashraf, Q., Galor, O.: Cultural diversity, geographical isolation, and the origin of the wealth of nations. Tech. rep., National Bureau of Economic Research (2011)
5. Balluchi, A., Casagrande, A., Collins, P., Ferrari, A., Villa, T., Sangiovanni-Vincentelli, A.L.: Ariadne: a framework for reachability analysis of hybrid automata. In: Mathematical Theory of Networks and Systems, MTNS. Citeseer (2006)
6. Batt, G., Yordanov, B., Weiss, R., Belta, C.: Robustness analysis and tuning of synthetic gene networks. Bioinformatics **23**(18), 2415–2422 (2007)
7. Berman, S., Halász, Á., Kumar, V.: Marco: a reachability algorithm for multi-affine systems with applications to biological systems. In: Hybrid Systems: Computation and Control, HSCC, pp. 76–89. Springer (2007)
8. Bernstein, S.N.: Démonstration du théorème de weierstrass fondée sur le calcul des probabilités. Communications de la Société Mathématique de Kharkov 2 **1**(4/5), 1–2 (1912)
9. Berz, M., Makino, K.: Verified integration of odes and flows using differential algebraic methods on high-order taylor models. Reliable Computing **4**(4), 361–369 (1998)
10. Botchkarev, O., Tripakis, S.: Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In: Hybrid Systems: Computation and Control, HSCC, pp. 73–88. Springer (2000)
11. Bournez, O., Maler, O., Pnueli, A.: Orthogonal polyhedra: Representation and computation. In: Hybrid Systems: Computation and Control, HSCC, pp. 46–60. Springer (1999)
12. Casagrande, A., Dreossi, T.: pyhybrid analysis: A package for semantics analysis of hybrid systems. In: Digital System Design, DSD, pp. 815–818 (2013). DOI 10.1109/DSD.2013.143. URL http://dx.doi.org/10.1109/DSD.2013.143
13. Casagrande, A., Dreossi, T., Fabriková, J., Piazza, C.: ϵ-semantics computations on biological systems. Inf. Comput. **236**, 35–51 (2014). DOI 10.1016/j.ic.2014.01.011. URL http://dx.doi.org/10.1016/j.ic.2014.01.011
14. Chen, L., Miné, A., Wang, J., Cousot, P.: Interval polyhedra: An abstract domain to infer interval linear relationships. In: Static Analysis Symposium, SAS, pp. 309–325 (2009)

15. Chen, X., Ábrahám, E.: Choice of directions for the approximation of reachable sets for hybrid systems. In: International Conference on Computer Aided Systems Theory, EUROCAST, pp. 535–542. Springer (2011)
16. Chen, X., Abraham, E., Sankaranarayanan, S.: Taylor model flowpipe construction for non-linear hybrid systems. In: Real-Time Systems Symposium, RTSS, pp. 183–192. IEEE (2012)
17. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Computer Aided Verification, CAV, pp. 258–263 (2013)
18. Chutinan, A., Krogh, B.H.: Computing polyhedral approximations to flow pipes for dynamic systems. In: Conference on Decision and Control, CDC, vol. 2, pp. 2089–2094. IEEE (1998)
19. Chutinan, A., Krogh, B.H.: Computing approximating automata for a class of linear hybrid systems. In: Hybrid Systems V, pp. 16–37. Springer (1999)
20. Chutinan, A., Krogh, B.H.: Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In: Hybrid Systems: Computation and Control, HSCC, pp. 76–90. Springer (1999)
21. Coxeter, H.S.M.: Regular polytopes. Courier Corporation (1973)
22. da Cunha, A.E.C.: Benchmark: Quadrotor attitude control. In: Applied Verification for Continuous and Hybrid Systems, ARCH (2015)
23. Dang, T.: Approximate reachability computation for polynomial systems. In: Hybrid Systems: Computation and Control, HSCC, pp. 138–152. Springer (2006)
24. Dang, T., Dreossi, T., Piazza, C.: Parameter synthesis using parallelotopic enclosure and applications to epidemic models. In: Hybrid Systems and Biology, HSB, pp. 67–82 (2014)
25. Dang, T., Dreossi, T., Piazza, C.: Parameter synthesis through temporal logic specifications. In: Formal Methods, FM, pp. 213–230 (2015)
26. Dang, T., Testylier, R.: Reachability analysis for polynomial dynamical systems using the Bernstein expansion. Reliable Computing **17**(2), 128–152 (2012)
27. Dang, T.X.T.: Verification and synthesis of hybrid systems. Ph.D. thesis, Institut National Polytechnique de Grenoble-INPG (2000)
28. Davis, P.J., Rabinowitz, P.: Methods of numerical integration. Courier Corporation (2007)
29. Dreossi, T.: Sapo (2016). URL http://tommasodreossi.github.io/sapo/
30. Dreossi, T.: Sapo: Reachability computation and parameter synthesis of polynomial dynamical systems (2016)
31. Dreossi, T., Dang, T.: Parameter synthesis for polynomial biological models. In: Hybrid Systems: Computation and Control, HSCC, pp. 233–242 (2014)
32. Eggers, A., Ramdani, N., Nedialkov, N.S., Fränzle, M.: Improving the sat modulo ode approach to hybrid systems analysis by combining different enclosure methods. Software & Systems Modeling **14**(1), 121–148 (2012)
33. Farouki, R.T.: The Bernstein polynomial basis: a centennial retrospective. Computer Aided Geometric Design **29**(6), 379–419 (2012)
34. Frehse, G.: Phaver: Algorithmic verification of hybrid systems past hytech. In: Hybrid Systems: Computation and Control, HSCC, pp. 258–273. Springer (2005)
35. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: Computer Aided Verification, CAV, pp. 379–395. Springer (2011)
36. Galor, O.: Discrete dynamical systems. Springer Science & Business Media (2007)
37. Gao, S.: Computable analysis, decision procedures, and hybrid automata: A new framework for the formal verification of cyber-physical systems. Ph.D. thesis, PhD thesis, Carnegie Mellon University (2012)
38. Garloff, J., Smith, A.P.: Investigation of a subdivision based algorithm for solving systems of polynomial equations. Nonlinear Analysis: Theory, Methods & Applications **47**(1), 167–178 (2001)
39. Girard, A.: Reachability of uncertain linear systems using zonotopes. In: Hybrid Systems: Computation and Control, HSCC, pp. 291–305. Springer (2005)
40. Girard, A., Le Guernic, C., Maler, O.: Efficient computation of reachable sets of linear time-invariant systems with inputs. In: Hybrid Systems: Computation and Control, HSCC, pp. 257–271. Springer (2006)

41. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the mpi message passing interface standard. Parallel computing **22**(6), 789–828 (1996)
42. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: Hytech: A model checker for hybrid systems. In: Computer Aided Verification, CAV, pp. 460–463. Springer (1997)
43. Hildebrand, F.B.: Introduction to numerical analysis. Courier Corporation (1987)
44. Jódar, L., Villanueva, R.J., Arenas, A.J., González, G.C.: Nonstandard numerical methods for a mathematical model for influenza disease. Mathematics and Computers in simulation **79**(3), 622–633 (2008)
45. Karp, R.M.: Reducibility among combinatorial problems. Springer (1972)
46. Kermack, W.O., McKendrick, A.G.: A contribution to the mathematical theory of epidemics. In: Royal Society of London A: Mathematical, Physical and Engineering Sciences, vol. 115, pp. 700–721. The Royal Society (1927)
47. Klipp, E., Herwig, R., Kowald, A., Wierling, C., Lehrach, H.: Systems biology in practice: concepts, implementation and application. John Wiley & Sons (2008)
48. Kong, S., Gao, S., Chen, W., Clarke, E.: dreach: δ-reachability analysis for hybrid systems. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS, pp. 200–205. Springer (2015)
49. Kostousova, E.: State estimation for dynamic systems via parallelotopes optimization and parallel computations. Optimization Methods and Software **9**(4), 269–306 (1998)
50. Kostousovat, E.K.: Control synthesis via parallelotopes: optimzation and parallel compuations. Optimization Methods and Software **4**(14), 267–310 (2001)
51. Kot, M.: Discrete-time travelling waves: ecological examples. Journal of mathematical biology **30**(4), 413–436 (1992)
52. Kot, M., Schaffer, W.M.: Discrete-time growth-dispersal models. Mathematical Biosciences **80**(1), 109–136 (1986)
53. Krommer, A.R.: Numerical Integration: On Advanced Computer Systems, vol. 848. Springer Science & Business Media (1994)
54. Kurzhanski, A.B., Varaiya, P.: Ellipsoidal techniques for reachability analysis: internal approximation. Systems & control letters **41**(3), 201–211 (2000)
55. Kurzhanskiy, A.A., Varaiya, P., et al.: Ellipsoidal toolbox. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-46 (2006)
56. Kvasnica, M., Grieder, P., Baotić, M., Morari, M.: Multi-parametric toolbox (mpt). In: Hybrid Systems: Computation and Control, HSCC, pp. 448–462. Springer (2004)
57. Lafferriere, G., Pappas, G.J., Yovine, S.: Symbolic reachability computation for families of linear vector fields. Journal of Symbolic Computation **32**(3), 231–253 (2001)
58. Le Guernic, C.: Reachability analysis of hybrid systems with linear continuous dynamics. Ph.D. thesis, Université Joseph-Fourier-Grenoble I (2009)
59. Le Guernic, C., Girard, A.: Reachability analysis of linear systems using support functions. Nonlinear Analysis: Hybrid Systems **4**(2), 250–262 (2010)
60. Lotka, A.: Elements of physical biology (1925)
61. Mourrain, B., Pavone, J.P.: Subdivision methods for solving polynomial equations. J. Symb. Comput. **44**(3), 292–306 (2009). DOI 10.1016/j.jsc.2008.04.016. URL http://dx.doi.org/10.1016/j.jsc.2008.04.016
62. Nataraj, P., Arounassalame, M.: A new subdivision algorithm for the Bernstein polynomial approach to global optimization. International journal of automation and computing **4**(4), 342–352 (2007)
63. Nvidia, C.: Programming guide (2008)
64. Platzer, A.: Differential dynamic logic for verifying parametric hybrid systems. In: Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX, pp. 216–232 (2007)
65. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reasoning **41**(2), 143–189 (2008). DOI 10.1007/s10817-008-9103-8. URL http://dx.doi.org/10.1007/s10817-008-9103-8
66. Platzer, A., Quesel, J.: Keymaera: A hybrid theorem prover for hybrid systems (system description). In: International Joint Conference on Automated Reasoning, IJCAR, pp. 171–178 (2008)
67. Van der Pol, B.: Lxxxviii. On "relaxation-oscillations". The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **2**(11), 978–992 (1926)

68. Prabhakar, P., Viswanathan, M.: A dynamic algorithm for approximate flow computations. In: Hybrid Systems: Computation and Control, HSCC, HSCC '11, pp. 133–142. ACM, New York, NY, USA (2011). DOI 10.1145/1967701.1967722. URL http://doi.acm.org/10.1145/1967701.1967722
69. Rössler, O.E.: An equation for continuous chaos. Physics Letters A **57**(5), 397–398 (1976)
70. Rössler, O.E.: An equation for hyperchaos. Physics Letters A **71**(2), 155–157 (1979)
71. Sankaranarayanan, S., Dang, T., Ivančić, F.: Symbolic model checking of hybrid systems using template polyhedra. In: Tools and Algorithms for the Construction and Analysis of Systems, TACAS, pp. 188–202. Springer (2008)
72. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Scalable analysis of linear systems using mathematical programming. In: Verification, Model Checking, and Abstract Interpretation, VMCAI, pp. 25–41 (2005)
73. Sassi, M.A.B., Sankaranarayanan, S.: Bernstein polynomial relaxations for polynomial optimization problems. arXiv preprint arXiv:1509.01156 (2015)
74. Sassi, M.A.B., Testylier, R., Dang, T., Girard, A.: Reachability analysis of polynomial systems using linear programming relaxations. In: Automated Technology for Verification and Analysis, ATVA, pp. 137–151 (2012)
75. Shisha, O.: The Bernstein form of a polynomial. Journal of Research of the National Bureau of Standards: Mathematics and mathematical physics. B **70**, 79 (1966)
76. Stursberg, O., Krogh, B.H.: Efficient representation and computation of reachable sets for hybrid systems. In: Hybrid Systems: Computation and Control, HSCC, pp. 482–497. Springer (2003)
77. Varaiya, P.: Reach set computation using optimal control. In: M. Inan, R. Kurshan (eds.) Verification of Digital and Hybrid Systems, *NATO ASI Series*, vol. 170, pp. 323–331. Springer Berlin Heidelberg (2000)
78. Volterra, V.: Variazioni e fluttuazioni del numero d'individui in specie animali conviventi. C. Ferrari (1927)
79. Wildenberg, J., Vano, J., Sprott, J.: Complex spatiotemporal dynamics in lotka–volterra ring systems. ecological complexity **3**(2), 140–147 (2006)

Appendix A Experiment Details

A.1 Van der Pol

$$D = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 1 \\ 1 & 1 \end{pmatrix} \quad T = \begin{pmatrix} 0\ 1 \\ 2\ 3 \\ 0\ 2 \\ 1\ 3 \\ 0\ 3 \\ 1\ 2 \end{pmatrix} \tag{54}$$

A.2 Rössler attractor

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0.5 & 0 \\ 0.5 & 0 & 0.5 \end{pmatrix} \quad T = \begin{pmatrix} 0\ 1\ 2 \\ 1\ 2\ 3 \\ 2\ 3\ 4 \end{pmatrix} \tag{55}$$

A.3 SIR Epidemi Model

$$D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0.5 & 0 \\ 0.5 & 0 & 0.5 \end{pmatrix} \quad T = \begin{pmatrix} 0\,1\,2 \\ 1\,2\,3 \\ 2\,3\,4 \end{pmatrix} \tag{56}$$

A.4 Generalized Lotka-Volterra Model

$$D = \begin{pmatrix} 1 & 0\,0 & 0 & 0 \\ 0 & 1\,0 & 0 & 0 \\ 0 & 0\,1 & 0 & 0 \\ 0 & 0\,0 & 1 & 0 \\ 0 & 0\,0 & 0 & 1 \\ 1 & 1\,1 & 0 & 0 \\ -1 & 0\,0 & -1 & 1 \end{pmatrix} \quad T = \begin{pmatrix} 0\,1\,2\,3\,4 \\ 1\,2\,3\,5\,6 \\ 2\,3\,4\,5\,6 \end{pmatrix} \tag{57}$$

A.5 Phosphorelay Systems

$$D = \begin{pmatrix} 1\,0\,0\,0\,0\,0\,0 \\ 0\,1\,0\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0\,0\,0 \\ 0\,0\,0\,1\,0\,0\,0 \\ 0\,0\,0\,0\,1\,0\,0 \\ 0\,0\,0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,0\,0\,1 \\ 0\,0\,1\,1\,0\,0\,0 \\ 0\,0\,0\,0\,1\,1\,0 \\ 0\,0\,1\,1\,1\,1\,0 \end{pmatrix} \quad T = \begin{pmatrix} 0\,1\,2\,3\,4\,5\,6 \\ 0\,1\,2\,4\,5\,6\,7 \\ 0\,1\,2\,5\,6\,7\,8 \\ 0\,1\,2\,5\,6\,7\,9 \end{pmatrix} \tag{58}$$

A.6 Quadcopter Drone

$$D_{(i,i)} = i \quad D_{(17,j)} = (0\ 0\ 0.5\ 0\ 0\ 0.5\ 0.5\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0.25) \tag{59}$$

for $i, j = 0, 1, \ldots, 16$ and

$$T_{(i,j)} = \begin{cases} 17 & \text{if } i = 1 \text{ and } j = 5 \\ i & \text{otherwise} \end{cases} \tag{60}$$

for $i = 0, 1, \ldots, 16$ and $j = 0, 1$.