Reachability analysis for polynomial dynamical systems using the Bernstein expansion over polytopes

Thao Dang and Romain Testylier *

VERIMAG,
2 avenue de Vignate, 38610 Gières, France

Abstract. This paper is concerned with the reachability computation problem for polynomial dynamical systems. Such computations constitute a crucial component in algorithmic verification tools for hybrid systems and embedded software with polynomial dynamics, which have found applications in many engineering domains. We describe two methods for over-approximating reachable sets of such systems; these methods are based on a combination of the Bernstein expansion of polynomial functions and a representation of reachable sets by template polyhedra. Using a prototype implementation, the performance of the methods was demonstrated on a number of examples.

1 Introduction

Hybrid systems have become a common mathematical model for engineering systems exhibiting both continuous and discrete dynamics. Recently they have proved appropriate for modeling phenomena in molecular biology. Generally, a hybrid system consists

In this work we focus on safety verification of such systems, which can be roughly stated as proving that a hybrid system never enters a dangerous (i.e. unsafe) state.

Generally, a hybrid system can be thought as the interaction between a discrete and a continuous process. It consists of a collection of continuous modes each of which is associated with a continuous vector field specifying the evolution of n continuous variables within some subset of the state space $\mathcal{X} \subseteq \mathbb{R}^n$. Such sets are called the staying sets of the modes. The discrete dynamics is described by discrete transitions, which can be triggered when the continuous variables satisfy their associated guards. Between two transitions, the continuous process evolves according to the continuous vector field associated with the active mode. Various hybrid systems models have been proposed and this remains an active research area [**?**,**?**,**?**]. It is important to note the following major types of non-determinism in the behavior of a hybrid system. First, the continuous dynamics can be subject to uncertain input modelling external disturbances or under-specified control. Second, non-determinism in discrete dynamics manifests when multiple discrete transitions are simultaneously activated, or when the system simultaneously satisfies the staying condition of the current mode (that is, it can still evolve according to the current continuous dynamics) and the guard condition of a transition (that is, it can take the transition to switch to a different continuous mode). In addition, initial conditions that are not exactly known. Even starting from a single initial state, the system may generate a possibly infinite set of trajectories. Therefore, to prove that the system satisfies a property, one often needs to consider a set of solutions instead of single solutions.

A major component of a safety verification algorithm for hybrid systems is an efficient method to compute their reachable set, which is the set of all the states visited by all the trajectories. Roughly speaking, the goal of reachability analysis is to study the set of all possible trajectories. While the computation of reachable sets by discrete dynamics requires mainly Boolean operations over sets in $\mathbb{R}^n$, computing the set of states reachable by continuous dynamics requires handling and is more difficult; this has been an obstacle towards applying formal verification to real-life problems. Therefore, much research in hybrid systems verification focuses on this particular problem. Using

* This work is supported by the ANR project VEDECY.

well-established results on linear systems, numerous methods and tools for such systems and other simpler systems have been developed [1]. Nevertheless, nonlinear systems still remain a challenge.

In this work, we address the following reachability computation problem: given a set of initial states in $\mathbb{R}^n$, compute the reachable set of a discrete-time dynamical system described by the following difference equation:

$$x[k+1] = \pi(x[k]) \tag{1}$$

where $\pi : \mathbb{R}^n \to \mathbb{R}^n$ is a multivariate polynomial. Such dynamical systems could result from a numerical approximation of a continuous or hybrid system. In addition, similar equations can arise in embedded control systems, such as when a physical system is controlled by a computer program which is the implementation of some continuous (or possibly hybrid) controller using appropriate discretization. Our interest in discrete-time polynomial systems is motivated by their applicability in modeling a variety of phenomena bio-chemical networks. Indeed, discrete-time models are useful for analysis purposes, since experimental data are often measured by sampling the continuous biochemical reaction outputs, and computer based analysis and simulation depend on discrete-time data.

It is important to emphasize that the results presented in this paper can also be extended to continuous-time dynamical systems described by differential equations, since these equations can be approximated using an appropriate time discretization scheme. However, as to solve initial value problems for continuous-time ordinary differential equations, it is crucial to obtain a time discretization method that can guarantee conservativeness of the resulting reachable set approximations. This is indeed a topic of our current research.

As mentioned earlier, due to various non-determinism, to prove that the system satisfies a property, one often needs to consider a set of solutions instead of single solutions. Roughly speaking, the goal of reachability analysis is to study sets of all possible trajectories. Many existing reachability computation methods can be seen as an extension of numerical integration. That is, one has to solve the above equation (1) with sets, that is $x[k]$ and $x[k+1]$ in the equation are subsets of $\mathbb{R}^n$ (while they are points if we only need a single solution, as in numerical integration).

This problem was previously considered in the work [**?**], which was inspired by modeling techniques from Computer Aided Geometric Design (CADG) and tried to exploit special geometric properties of polynomials. The drawback of the Bézier simplex based method proposed in this work is that it requires expensive mesh computation, which restricts its application to systems of dimensions not higher than $3, 4$. In this paper, we pursue the direction which was initiated in [**?**] and make use of a special class of polyhedra. These polyhedra can be thought of as local meshes of fixed form. This enables a significant reduction of complexity. The manipulation of such polyhedra is handled by optimization techniques. In addition, by exploiting a technique from CADG, namely the Bernstein expansion, we only need to solve linear programming (LP) problems instead of polynomial optimization problems. In this paper, we describe our results achieved along this direction, in particular, a significant accuracy improvement compared to [**?**], thanks to a more precise representation of the Bernstein expansion over polyhedra.

The paper is organized as follows. In Section 2 we introduce basic definitions of reachable sets, template polyhedra and the Bernstein expansion of polynomials. We then formally state our reachability problem and describe an optimization-based solution. In order to transform the polynomial optimization problem to a linear programming (LP) problem, two methods for computing bound functions for polynomials over polyhedral sets are presented. Section 6.2 describes an algorithm summarizing the main steps of our reachability analysis method. Some experimental results, in particular the analysis of a control system and two biological systems, are reported in Section 7.

2 Preliminaries

Let $\mathbb{R}$ denote the set of reals. Throughout the paper, vectors are often written using bold letters. Exceptionally, scalar elements of multi-indices, introduced later, are written using bold letters.

[1] The reader is referred to the recent proceedings of the conference Hybrid Systems: Computation and Control HSCC.

Given a vector $\mathbf{x}$, x_i denotes its i^{th} component. Capital letters, such as A, B, X, Y, denote matrices or sets. If A is a matrix, A^i denotes the i^{th} row of A.

We use $\mathcal{B}$ to denote the unit box anchored at the origin, that is $\mathcal{B} = [0,1]^n$. We use π to denote a vector of n functions such that for all $i \in \{1,\ldots,n\}$, π_i is an n-variate polynomial of the form $\pi_i : \mathbb{R}^n \to \mathbb{R}$. In the remainder of the paper, we sometimes refer to π simply as "a polynomial".

To discuss the Bernstein expansion of polynomials, we use multi-indices of the form $\mathbf{i} = (\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_n)$ where each $\mathbf{i}_j$ is a non-negative integer. Given two multi-indices $\mathbf{i}$ and $\mathbf{d}$, we write $\mathbf{i} \leq \mathbf{d}$ if for all $j \in \{1,\ldots,n\}$, $\mathbf{i}_j \leq \mathbf{d}_j$. Also, we write $\frac{\mathbf{i}}{\mathbf{d}}$ for $(\mathbf{i}_1/\mathbf{d}_1, \mathbf{i}_2/\mathbf{d}_2, \ldots, \mathbf{i}_n/\mathbf{d}_n)$ and $\binom{\mathbf{i}}{\mathbf{d}}$ for $\binom{\mathbf{i}_1}{\mathbf{d}_1}\binom{\mathbf{i}_2}{\mathbf{d}_2}\cdots\binom{\mathbf{i}_n}{\mathbf{d}_n}$.

2.1 Reachable sets

We consider a discrete-time dynamical system:

$$x[k+1] = \pi(x[k]) \tag{2}$$

where the initial state $x[0]$ is inside some set $X_0 \subset \mathbb{R}^n$, and X_0 is called the initial set.

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

$$\pi(X) = \{(\pi_1(\mathbf{x}), \ldots, \pi_n(\mathbf{x})) \mid \mathbf{x} \in X\}.$$

The *reachable set* X_k of the system (2) at time step $k \geq 0$ is defined by the following recurrence

$$X_{k+1} = \pi(X_k).$$

2.2 Template polyhedra

When starting from X_0, a dynamical system such as (2) generates a set of solutions. To charaterize this set of solutions we use special convex polyhedra with fixed geometric form, called *template polyhedra* [**?**,**?**]; ranges [**?**] and the octagon domains [**?**] are special template polyhedra. In the following we give a brief introduction to template polyhedra.

A *template* is a set of linear functions over $\mathbf{x} = (x_1, \ldots, x_n)$. We denote a template by an $m \times n$ matrix H, such that each row H^i corresponds to the linear function $H^i\mathbf{x}$. Given such a template H and a real-valued vector $\mathbf{c} \in \mathbb{R}^m$, a template polyhedron is defined by considering the conjunction of the linear inequalities of the form

$$\bigwedge_{i=1,\ldots,m} H^i\mathbf{x} \leq c_i.$$

We denote this polyhedron by $\langle H, \mathbf{c}\rangle$.

By varying the values of the elements of $\mathbf{c}$, one can create a family of template polyhedra corresponding to the template H. We call $\mathbf{c}$ a *polyhedral coefficient vector*. Given $\mathbf{c}, \mathbf{c'} \in \mathbb{R}^m$, if $\forall i \in \{1,\ldots,m\} \;:\; c_i \leq c'_i$, we write $\mathbf{c} \preceq \mathbf{c'}$. Given an $m \times n$ template H and two polyhedral coefficient vectors $\mathbf{c}, \mathbf{c'} \in \mathbb{R}^m$, if $\mathbf{c} \preceq \mathbf{c'}$ then the inclusion relation $\langle H, \mathbf{c}\rangle \subseteq \langle H, \mathbf{c'}\rangle$ holds, and we say that $\langle H, \mathbf{c}\rangle$ is not larger than $\langle H, \mathbf{c'}\rangle$.

The advantage of template polyhedra over general convex polyhedra is that the Boolean operations (union, intersection) and common geometric operations can be performed more efficiently [**?**]. Indeed, manipulating general convex polyhedra is expensive especially in high dimensions. This poses a major problem in continuous and hybrid systems verification approaches using polyhedral representations.

3 Reachable set approximation using template polyhedra

To compute the reachable set by a template polyhedron, at each time step, we need to compute the image of a polyhedron P by the polynomial π. The template matrix H, which is of size $m \times n$, is assumed to be given; the polyhedral coefficient vector $\mathbf{c} \in \mathbb{R}^m$ is however unknown. The problem we now focus on is thus to find $\mathbf{c}$ such that

$$\pi(P) \subseteq \langle H, \mathbf{c} \rangle. \tag{3}$$

For safety verification purposes, exact computation of reachable sets is often not possible (due to undecidablity issues for example) and one thus needs to resort to over-approximations, and when an over-approximation does not allow proving a safety property, the approximation needs to be refined.

It is not hard to see that the following condition is sufficient for (3) to hold:

$$\forall \mathbf{x} \in P \ : \ H\pi(\mathbf{x}) \leq \mathbf{c}.$$

Therefore, to determine $\mathbf{c}$, one can formulate the following optimization problems:

$$\forall i \in \{1, \ldots, m\}, c_i = \max(\Sigma_{k=1}^{n} H_k^i \pi_k(\mathbf{x})) \text{ subj. to } \mathbf{x} \in P. \tag{4}$$

where H^i is the i^{th} row of the matrix H and H_k^i is its k^{th} element. Note that the above functions to optimize are polynomials. This problem is computationally difficult, despite recent progress in the development of methods and tools for polynomial programming (see for example [**?**] and references therein). An alternative solution is to find their affine bound functions, in order to replace the polynomial optimization problem by a linear programming one, which can be solved more efficiently (in polynomial time) using well-developed techniques, such as Simplex and interior point techniques [**?**]. Indeed, the Bernstein expansion can be used to compute affine bound functions of polynomials, as shown in the next section.

3.1 The Bernstein expansion

An n-variate polynomial $\pi : \mathbb{R}^n \to \mathbb{R}^n$ can be represented using the power base as follows:

$$\pi(\mathbf{x}) = \sum_{\mathbf{i} \in I_{\mathbf{d}}} \mathbf{a_i} \mathbf{x}^{\mathbf{i}}$$

where $\mathbf{a_i}$ is a vector in $\mathbb{R}^n$; $\mathbf{i}$ and $\mathbf{d}$ are two multi-indices of size n such that $\mathbf{i} \leq \mathbf{d}$; $I_{\mathbf{d}}$ is the set of *all* multi-indices $\mathbf{i} \leq \mathbf{d}$, that is $I_{\mathbf{d}} = \{\mathbf{i} \mid \mathbf{i} \leq \mathbf{d}\}$. The multi-index $\mathbf{d}$ is called the *degree* of π.

The polynomial π can also be represented using the Bernstein expansion. In order to explain this, we first introduce Bernstein polynomials. For $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, the $\mathbf{i}^{th}$ Bernstein polynomial of degree $\mathbf{d}$ is defined as follows:

$$\mathcal{B}_{\mathbf{d},\mathbf{i}}(\mathbf{x}) = \beta_{\mathbf{d}_1,\mathbf{i}_1}(x_1) \ldots \beta_{\mathbf{d}_n,\mathbf{i}_n}(x_n)$$

where for a real number y, $\beta_{\mathbf{d}_j,\mathbf{i}_j}(y) = \binom{\mathbf{d}_j}{\mathbf{i}_j} y^{\mathbf{i}_j}(1 - y^{\mathbf{d}_j - \mathbf{i}_j})$.

Then, for all $\mathbf{x} \in \mathcal{B} = [0, 1]^n$, the polynomial π can be written using the Bernstein expansion as follows:

$$\pi(\mathbf{x}) = \sum_{\mathbf{i} \in I_{\mathbf{d}}} \mathbf{b_i} \mathcal{B}_{\mathbf{d},\mathbf{i}}(\mathbf{x})$$

where for each $\mathbf{i} \in I_{\mathbf{d}}$ the Bernstein coefficient $\mathbf{b_i}$ is defined as:

$$\mathbf{b_i} = \sum_{\mathbf{j} \leq \mathbf{i}} \frac{\binom{\mathbf{i}}{\mathbf{j}}}{\binom{\mathbf{d}}{\mathbf{j}}} \mathbf{a_j}. \tag{5}$$

With respect to our reachability problem that requires computing the image of a set by a polynomial, the Bernstein expansion is of particular interest. For example, using the second property, the coefficients of the Bernstein expansion can be used to over-approximate the image of the unit box $\mathcal{B}$ by the polynomial π. Furthermore, as we will show in Section 4, these coefficients can be used to efficiently compute an affine approximation of the polynomial.

It is important to note that the above expansion is valid only if $\mathbf{x}$ is inside the unit box. Even if our initial set X_0 is inside the unit box $\mathcal{B}$, after the first step, the polyhedral approximation of the reachable set can be outside the unit box. Therefore, we need to consider the problem of computing the image of a general convex polyhedron P. To this end, we first consider the case where the set P is the unit box and then show how the solution can be extended to general convex polyhedra.

4 Computing bound functions over the unit box domain

To compute bound functions, we use the method based on the Bernstein expansion, published in [**?**]. Computing convex lower bound functions for polynomials is a problem of great interest, especially in global optimization. The reader is referred to [**?**,**?**,**?**] for more detailed descriptions of these methods. It is important to note that the methods described in this section only work for the case where the variable domain is the unit box $\mathcal{B}$. The reason is that it employs the expression of the control points of the Bernstein expansion in (5) which is only valid for this unit box. Their extensions to arbitrary polyhedral domains are discussed in the next section. Therefore, in what follows, we assume that our initial polyhedron P is included in the unit box.

4.1 Using a convex hull lower facet

The first step of this method [**?**] involves computing the affine lower bound function whose corresponding hyperplane passes through this control point $\mathbf{b}^0$. Then, additionally, $(n-1)$ hyperplanes passing through n other control points are determined. This allows constructing a sequence of n affine lower bound functions $l_0, l_1, \ldots l_n$. The method ends up with l_n, a function whose corresponding hyperplane passes through a lower facet of the convex hull spanned by these control points. Note that we can easily compute an upper bound function of π by computing a lower bound functions for $-\pi$ using this method and then multiply each resulting function by -1.

4.2 Using linear least squares approximation

The essence of the second method [**?**] for computing bound functions is to find a hyperplane that is close to *all* the control points, using linear least squares approximation. This can lead to tighter bound functions since the general shape of the function graph is better captured. Then, the resulting affine function corresponds to the "median" axis of the convex hull of all the control points. It thus suffices to shift it downward so that it is guaranteed to be a lower bound function.

5 Computing affine bound functions over polyhedral domains

As mentioned earlier, the methods to compute affine bound functions for polynomials in Section 4 can be applied only when the set P is inside the unit box $\mathcal{B}$ anchored at the origin. To extend it to polyhedral domains, we transform the polyhedra to the unit box by two methods: (1) via an (oriented) box approximation, and (2) by rewriting the polynomials using a change of variables.

5.1 Using a box approximation

If we over-approximate P with a box B, it is then possible to derive a formula expressing the Bernstein coefficients of π over B. However, this formula is complex and its representation and evaluation can become expensive.

We alternatively consider the composition of the polynomial π with an affine transformation τ that maps the unit box to B. The functions resulting from this composition are still polynomials, for which we can compute their bound functions over the unit box, using the formula (5) of the Bernstein expansion. This is explained more formally in the following.

Let B be the bounding box of the polyhedron P, that is, the smallest box that includes P. The affine function τ that maps the unit box $\mathcal{B}$ to B can be easily defined as: $\tau(\mathbf{x}) = diag(\lambda)\mathbf{x} + \mathbf{g}$ where $\mathbf{g} \in \mathbb{R}^n$ such that $g_i = l_i$, and $diag(\lambda)$ is a $n \times n$ diagonal matrix with the elements on the diagonal defined as follows: for each $i \in \{1, \ldots, n\}$, $\lambda_i = h_i - l_i$.

The composition $\gamma = (\pi \; o \; \tau)$ is defined as $\gamma(\mathbf{x}) = \pi(\tau(\mathbf{x}))$.

Note that $\gamma = \pi \; o \; \tau$. Then, $\pi(P) \subseteq \gamma(\mathcal{B})$.

We remark that the above is still valid for any affine function τ. This means that instead of an axis-aligned bounding box, we can over-approximate P more precisely with an oriented (i.e. non-axis-aligned) bounding box. The directions of an oriented bounding box can be computed using Principal Component Analysis (PCA) [**?**]. A detailed description of the method can be found in [**?**].

5.2 Using a change of variables

The polyhedron P can also be map to the unit box $\mathcal{B}$ by a change of variables as follows. We assume that the polyhedron P is bounded and let $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_l\}$ be the set of its vertices. We first express the coordinates of a point x inside the polyhedron P as a linear combination of the vertices of P, that is

$$\mathbf{x} = \sum_{j=1}^{l} \alpha_j \mathbf{v}_j = \nu(\alpha_1, \ldots, \alpha_l)$$

such that

$$\forall j \in \{1, \ldots, l\} \; \alpha_j \geq 0 \tag{6}$$

$$\sum_{j=1}^{l} \alpha_j = 1. \tag{7}$$

We then substitute $\mathbf{x}$ in π with $\nu(\alpha_1, \ldots, \alpha_l)$ to yield a new polynomial in $\alpha_1, \ldots, \alpha_l$.

We denote $\mu = \pi \; o \; \nu$, that is $\pi(\mathbf{x}) = \mu(\alpha_1, \ldots, \alpha_l)$. Furthermore, in order to retain the relation between α_j expressed in the constraint (7) we can use

$$\alpha_l = 1 - \sum_{j=1}^{l-1} \alpha_j$$

to substitute α_l in μ by the above sum, in order to obtain a polynomial with $(l-1)$ variables, denoted by $\xi(\beta)$ where $\tilde{\alpha} = (\alpha_1, \ldots, \alpha_{l-1})$.

Note that the constraints (6-7) indicate that γ is inside the unit box $B_{\tilde{\alpha}}$ in $\mathbb{R}^{l-1}$. This implies that a bound function computed for the polynomial $\xi(\tilde{\alpha})$ on this unit box is also a bound function for the original polynomial π on the polyhedron P without additional error, unlike in the above-described case of box approximations. It then suffices to compute the bound functions for π over the polyhedron P using the Bernstein expansion of ξ over the $B_{\tilde{\alpha}}$.

6 Reachable set computation

6.1 Image computation

We now show how the above affine bound functions can be used to solve the optimisation problems (4) in order to determine the coefficients of a template polyhedron over-approximating the

reachable set. The functions to optimize in (4) can be seen as the compositions of polynomials π_k. Since every coefficient H_k^i is constant, each

$$\sigma^i(\mathbf{x}) = \Sigma_{k=1}^n H_k^i \pi_k(\mathbf{x})$$

is simply a polynomial and we can compute its bound functions. The template polyhedral coefficients can hence be computed by solving the following optimization problems:

$$\forall i \in \{1, \ldots, m\}, c_i = max(\sigma^i(\mathbf{x})) \text{ subj. to } \mathbf{x} \in P; \tag{8}$$

However, such compositions often result in polynomials with more monomial terms and thus more Bernstein coefficients to consider. In the following we propose a way to bound each element of the sum separately, which costs less computation time but induces greater overall error. For each $k \in \{1, \ldots, m\}$, let $u_k(\mathbf{x})$ and $l_k(\mathbf{x})$ respectively be an upper bound function and a lower bound function of $\pi_k(\mathbf{x})$ w.r.t. the initial polyhedron P.

We consider the following optimization problem:

$$\forall i \in \{1, \ldots, m\}, c_i = \Sigma_{k=1}^n H_k^i \omega_k. \tag{9}$$

where the term $H_k^i \omega_k$ is defined as follows:

- If the element $H_k^i > 0$, $H_k^i \omega_k = H_k^i \max u_k(\mathbf{x})$ subj. to $\mathbf{x} \in P$;
- If the element $H_k^i \leq 0$, $H_k^i \omega_k = H_k^i \min l_k(\mathbf{x})$ subj. to $\mathbf{x} \in P$.

The following lemma is a direct result of (9).

Lemma 1. *If a polyhedral coefficient vector $\mathbf{c} \in \mathbb{R}^m$ satisfies (9). Then $\pi(P) \subseteq \langle H, \mathbf{c} \rangle$.*

Proof. It is indeed not hard to see that the solution c_i of the optimization problems (9) is greater than or equal to the solution of (4). Hence, if $\mathbf{c}$ satisfies (9), then

$$\forall i \in \{1, \ldots, m\} \ \forall \mathbf{x} \in P : \ \Sigma_{k=1}^n H_k^i \pi_k(\mathbf{x}) \leq c_i.$$

This implies that $\forall \mathbf{x} \in P : \ H\pi(\mathbf{x}) \leq \mathbf{c}$, that is the image $\pi(P)$ is included in the template polyhedron $\langle H, \mathbf{c} \rangle$.

We remark that if all the bound functions in (9) are affine and P is a convex polyhedron, $\mathbf{c}$ can be computed by solving $2n$ linear programming problems.

6.2 Reachable set computation algorithm

Algorithm 1 summarizes the main steps of our approach for over-approximating the reachable set of the system (2) where the initial set X_0 is a bounded polyhedron in $\mathbb{R}^n$. The template is an input of the algorithm. In the current implementation of the algorithm, either templates fixed a-priori by the user or templates forming regular sets are used.

To unify two methods of mapping a polyhedron to the unit box in the same abstract algorithm, we use β to denote both of the transformations using either a box approximation or a change of variables.

The procedure *UnitBoxMap* is used to determine the function β. This function is then composed with the polynomial π, the result of which is the polynomial γ. The affine lower and upper bound functions l and u of γ are then computed, using the Bernstein expansion of γ over the corresponding unit box. The function *PolyApp* determines the polyhedral coefficient vector $\mathbf{c}$ by solving the linear programs where the optimization domain is the unit box. The polyhedral coefficient vector $\bar{\mathbf{c}}$ is then used to define a template polyhedron X_{k+1}.

Based on the analysis so far, we can state the correctness of Algorithm 1.

Theorem 1. *Let $\langle H, \bar{\mathbf{c}} \rangle$ be the template polyhedron returned by Algorithm 1. Then $\pi(P) \subseteq \langle H, \bar{\mathbf{c}} \rangle$.*

We remark that, when using a box approximation, u and l are upper and lower bound functions of γ with respect to the unit box $\mathcal{B}$. It is not hard to see that $\tau^{-1}(X_k) \subseteq \mathcal{B}$ where τ^{-1} is the inverse of τ. Using the property of bound functions, u and l are also bound functions of γ with respect to $\tau^{-1}(X_k)$. Hence, if we solve the optimization problems over the domain $\tau^{-1}(X_k)$ (which is often smaller than $\mathcal{B}$), using Lemma 1, the resulting polyhedron is still an over-approximation of $\pi(X_k)$. This remark can be used to obtain more accurate results.

Algorithm 1 Reachable set computation

/ Inputs: convex polyhedron X_0, polynomial π, templates H */*

$k = 0$
repeat
  $\beta = UnitBoxMap(X_k)$ */* Compute the function mapping the unit box $\mathcal{B}$ to the polyhedron X_k */*
  $\gamma = \pi\ o\ \beta$
  $(u, l) = BoundFunctions(\gamma)$ */* Compute the affine bound functions */*
  $\bar{\mathbf{c}} = PolyApp(u, l, H)$ */* Compute the polyhedral coefficient vector */*
  $X_{k+1} = \langle H, \bar{\mathbf{c}} \rangle$ */* Construct the template polyhedron and return it */*
  $k++$
until $k = k_{max}$

7 Experimental results

We have implemented our methods in a prototype tool. The implementaion uses the library **lp-solve**[2] for linear programming. The tool can be combined with a reachability analysis algorithms to verify hybrid systems with polynomial continuous dynamics. In the following, we demonstrate the methods with three examples: a control system (modeled as a hybrid system) and a biological system (modeled as a continuous system). The time efficiency of the tool was also evaluated by considering a number of randomly generated polynomials.

7.1 A control system

The first example we present is the Duffing oscillator taken from [**?**,**?**]. This is a nonlinear oscillator of second order, the continuous-time dynamics is described by

$$\ddot{y}(t) + 2\zeta\dot{y}(t) + y(t) + y(t)^3 = u(t)$$

where $y \in \mathbb{R}$ is the state variable and $u \in \mathbb{R}$ is the control input. The damping coefficient $\zeta = 0.3$. In [**?**], using a forward difference approximation with a sampling period $h = 0.05$ time units, this system is approximated by the following discrete-time model:

$$\begin{aligned} x_1[k+1] &= x_1[k] + hx_2[k] \\ x_2[k+1] &= -hx_1[k] + (1 - 2\zeta h)x_2[k] + hu[k] - hx_1^3[k] \end{aligned}$$

In [**?**], an optimal predictive control law $u(k)$ was computed by solving a parametric polynomial optimization problem.

We model this control law by the following switching law with 3 modes:

$$\begin{aligned} u[k] &= 0.5 * k \quad \text{if } 0 \le k \le 10 \\ u[k] &= 5 - 0.5 * (k - 10)/3 \quad \text{if } 10 < k \le 40 \\ u[k] &= 0 \qquad \text{if } k > 40 \end{aligned}$$

The controlled system is thus modeled as a hybrid automaton [**?**] with 3 discrete states. The initial set is a rectangle such that $2.49 \le x_1 \le 2.51$ and $1.49 \le x_2 \le 1.51$.

The results obtained using the two methods are shown in Figure **??** which are coherent with the phase portrait in [**?**]. We can see that the method using a change of variables achieved better precision since the reachable set it computed is include in the set computed by the other method. However, the method using a change of variables is less time-efficient. For 70 steps, the computation time of the method using a box approximation is $1.25s$ while that of the method using a change of variables is $1.18s$. We also used this example to compare the two methods of computing bound functions and observed that they produced equally accurate results, as shown in Figure 1.

Fig. 1. The Duffing oscillator: the reachable set computed using a change of variable is more accurate than the one computed using a box approximation.

dim	degree d	nb monomials of degree d	nb templates	time (s) method BA	time (s) method CV
2	2	4	4	0.004	0.001
2	3	6	4	0.002	0.008
2	4	8	4	0.005	0.01
3	2	6	6	0.009	0.011
3	3	9	6	0.023	0.043
3	4	12	6	0.068	0.158
4	2	8	8	0.041	0.065
4	3	12	8	0.184	0.62
4	4	16	8	0.87	6.11167
5	2	10	10	0.265	0.501
5	3	15	10	15.44	1.48444
6	2	12	12	1.031	4.508
7	2	14	14	5.889	51.334

Fig. 2. Computation time for randomly generated polynomial systems in various dimensions and degrees. The second column contains the degree d of the polynomials, and third column contains their number of monomial of degree d.

7.2 Randomly generated systems

In order to evaluate the performance of our methods, we tested them on a number of randomly generated polynomials in various dimensions and maximal degrees (the maximal degree is the largest degree for all variables). For a fixed dimension and degree, we generated different examples to estimate an average computation time. In the current implementation, polynomial composition is done symbolically, and we do not yet exploit the possibility of sparsity of polynomials (in terms of the number of monomials). The computation times in seconds for the method using a box approximation (abbreviated to BA) and the method using a change of variables (abbreviated to CV) are shown in the table in Figure 2.

As expected, the computation time grows linearly w.r.t. the number of steps. This can be explained by the use of template polyhedra where the number of constraints can be chosen according to required precisions and thus the complexity of the polyhedral operations can be better controlled, compared to general convex polyhedra. Indeed, when using general polyhedra, the operations, such as the convex hull, may increase their geometric complexity (roughly described by the number of vertices and constraints).

On the other hand, we also compared the two methods for computing bound functions: using a lower convex hull facet (abbreviated to CHF) and using the least squares approximation (abbreviated to LSA). The average computation time for one step of reachability computation is shown in Table 3. In this experiment we used box templates and we generate random quadratic polynomial systems with 5 terms. Moreover, the computation time for polynomial composition is not included, since the computation of bound functions is not a dominant part of the total computation time. We were not able to test systems of dimensions higher than 9 because polynomial composition becomes prohibitively costly. This issue can be handled by computing the Bernstein coefficients without explicit polynomial composition, which is indeed a topic of our current research. We have observed that the method using the least squares approximation would be more performant than the one using a lower convex hull facet for systems of dimension beyond 9. The latter requires solving n systems of linear equations in dimensions increasing from 1 to n. The former requires solving only one linear system in dimension $(n+1)$. Using Gaussian elimination to solve a system

[2] http://lpsolve.sourceforge.net/

of n equations for n unknowns has complexity of $O(n^3)$. Thus, the complexity of the method using a lower convex hull facet is roughly $O((n-1)^2n^2/4)$ while the complexity of the other is $O((n+1)^2)$.

dim	time (s) method LSA	time (s) method CHF
2	0.00005	0.0005
3	0.00016	0.00016
4	0.00275	0.00263
5	0.0117	0.0116
6	0.0463	0.0441
7	0.1497	0.1191
8	0.8012	0.4837
9	4.755	1.591

Fig. 3. The computation times of computing a bound function on randomly generated polynomial systems using the LSA method (second column) and the CHF method (third column). In this experiment, the polynomials are quadratic with 5 second-order monomials). The

8 Related work

Our reachability analysis approach is similar to a number of existing ones for continuous and hybrid systems in the use of linear approximation. Its novelty resides in the efficient way of computing linear approximations. Indeed, a common method to approximate a non-linear function by a piecewise linear one, as in the hybridization approach [**?**] for hybrid systems, requires non-linear optimization. Our approach exploits the Bernstein expansion of polynomials to replace expensive polynomial programming by linear programming.

A similar idea, which involves using the coefficients of the Bézier simplex representation, was used in [**?**] to compute the image of a convex polyhedron. If using the methods proposed in this paper with a sufficient number of templates to assure the same precision as the convex hull in our previous Bézier method [**?**], then the convergence of both methods are quadratic. However the Bézier method requires expensive triangulation operations, and geometric complexity of resulting sets may grow step after step. Combining template polyhedra and bound functions allows a good accuracy-cost compromise.

Besides constrained global optimization, other important applications of the Bernstein expansion include various control problems [**?**] (in particular, robust control). The approximation of the range of a multivariate polynomial over a box and a polyhedron is also used in program analysis and optimization (for example [**?**,**?**]). In the hybrid systems verification, polynomial optimization is used to compute barrier certificates [**?**]. Algebraic properties of polynomials are used to compute polynomial invariants [**?**] and to study the computability of image computation in [**?**].

9 Conclusion

The reachability computation methods we proposed in this paper combine the ideas from optimization and the Bernstein expansion. These results can be readily applicable to hybrid systems with polynomial continuous dynamics.

The performance of the methods was demonstrated using a number of randomly generated examples. These encouraging results also show an important advantage of the methods: thanks to the use of template polyhedra, the complexity and precision of the method are more controllable than those using polyhedra as symbolic set representations.

There are a number interesting directions to explore. Indeed, different tools from geometric modeling could be exploited to improve the efficiency of the method. For example, polynomial composition can be done for sparse polynomials more efficiently using the blossoming technique [**?**]. In addition to more experimentation on other hybrid systems case studies, we intend to explore a new application domain, which is verification of embedded control software. In fact, multivariate polynomials arise in many situations when analyzing programs that are automatically generated from practical embedded controllers.