Reachable Set Over-approximation for Nonlinear Systems Using Piecewise Barrier Tubes

Hui Kong[1], Ezio Bartocci[2], Thomas A. Henzinger[1]

[1] IST Austria, Klosterneuburg, Austria
[2] TU Wien, Vienna, Austria

Abstract. We address the problem of analyzing the reachable set of a polynomial nonlinear continuous system by over-approximating the flowpipe of its dynamics. The common approach to tackle this problem is to perform a numerical integration over a given time horizon based on Taylor expansion and interval arithmetic. However, this method results to be very conservative when there is a large difference in speed between trajectories as time progresses. In this paper, we propose to use combinations of barrier functions, which we call piecewise barrier tube (PBT), to over-approximate flowpipe. The basic idea of PBT is that for each segment of a flowpipe, a coarse box which is big enough to contain the segment is constructed using sampled simulation and then in the box we compute by linear programming a set of barrier functions (called barrier tube or BT for short) which work together to form a tube surrounding the flowpipe. The benefit of using PBT is that 1) BT is independent of time and hence can avoid being stretched and deformed by time; and 2) a small number of BTs can form a tight over-approximation for the flowpipe, which means that the computation required to decide whether the BTs intersect the unsafe set can be reduced significantly. We implemented a prototype called PBTS in C++. Experiments on some benchmark systems show that our approach is effective.

1 Introduction

Hybrid systems [17] are widely used to model dynamical systems which exhibit both discrete and continuous behaviors. The reachability analysis of hybrid systems has been a challenging problem over the last few decades. The hard core of this problem lies in dealing with the continuous behavior of systems that are described by ordinary differential equations (ODEs). Although there are currently several quite efficient and scalable approaches for reachability analysis of linear systems [9,8,10,26,16,34,14,20,19], nonlinear ODEs are much harder to handle and the current approaches can be characterized into the following groups.

This research was supported by the Austrian Science Fund (FWF) under grants S11402-N23, S11405-N23 (RiSE/SHiNE) and Z211-N23 (Wittgenstein Award).

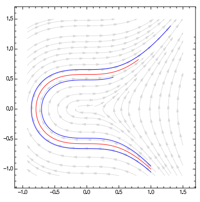
Invariant generation [22,28,39,27,18,21,36,37] An invariant I for a system S is a set such that any trajectory of S originating from I never escapes from I. Therefore, finding an invariant I such that the initial set $I_0 \subseteq I$ and the unsafe set $U \cap I = \emptyset$ indicates the safety of the system. In this way, there is no need to compute the flowpipe. The main problem with invariant generation is that it is hard to define a set of high quality constraints which can be solved efficiently.

Abstraction and hybridization [35,11,24,2,31] The basic idea of the abstraction-based approach is first constructing a linear model which over-approximates the original nonlinear dynamics and then applying techniques for linear systems to the abstraction model. However, how to construct an abstraction with the fewest discrete states and sufficiently high accuracy is still a challenging issue.

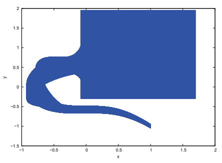
Satisfiability Modulo Theory (SMT) over reals [6,7,23] This approach encodes the reachability problem for nonlinear systems as first-order logic formulas over the real numbers. These formulas can be solved using for example $\delta-$complete decision procedures that overcome the theoretical limits in nonlinear theories over the reals, by choosing a desired precision δ. An SMT implementing such procedures can return either *unsat* if the reachability problem is unsatisfiable or δ-sat if the problem is satisfiable given the chosen precision. The δ-sat verdict does not guarantee that the dynamics of the system will reach a particular region. It may happens that by increasing the precision the problem would result *unsat*. In general the limit of this approach is that it does not provide as a result a complete and comprehensive description of the reachability set.

Bounded time flowpipe computation [4,5,1,3,32,25] The common technique to compute a bounded flowpipe is based on interval method or Taylor model. Interval-based approach is quite efficient even for high dimensional systems [29], but it suffers the wrapping effect of intervals and can quickly accumulate over-approximation errors. In contrast, the Taylor-model-based approach is more precise in that it uses a vector of polynomials plus a vector of small intervals to symbolically represent the flowpipe. However, for the purpose of safety verification or reachability analysis, the Taylor model has to be further over-approximated by intervals, which may bring back the wrapping effect. In particular, the wrapping effect can explode easily when the flowpipe segment over a time interval is stretched drastically due to a large difference in speed between individual trajectories. This case is demonstrated by the following example.

Example 1 (Running example). Consider the 2D system [30] described by $\dot{x} = y$ and $\dot{y} = x^2$. Let the initial set X_0 be a line segment $x \in [1.0, 1.0]$ and $y \in [-1.05, -0.95]$, Figure 1a shows the simulation result on three points in X_0 over time interval $[0, 6.6]$. The reachable set at $t = 6.6s$ is a smooth curve connecting the end points of the three trajectories. As can be seen, the trajectory originating from the top is left far behind the one originating from the bottom, which means that the tiny initial line segment is being stretched into a huge curve very quickly, while the width of the flowpipe is actually converging to 0. As a result, the



(a)



(b)

Fig. 1: (a) Simulation for Example 1 showing flowpipe segment being extremely stretched and deformed, (b) Interval over-approximation of the Taylor model computed by *Flow** [3].

interval over-approximation of this huge curve can be extremely conservative even if its Taylor model representation is precise, and reducing the time step size is not helpful. To prove this point, we computed with *Flow** [3] a Taylor model series for the time horizon of 6.6s which consists of 13200 Taylor models. Figure 1b shows the interval approximation of the Taylor model series, which apparently starts exploding.

In this paper, we propose to use piecewise barrier tubes (PBTs) to over-approximate flowpipes of polynomial nonlinear systems, which can avoid the issue caused by the excessive stretching of a flowpipe segment. The idea of PBT is inspired from barrier certificate [33,22]. A barrier certificate $B(\boldsymbol{x})$ is a real-valued function such that 1) $B(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x}$ in the initial set X_0; 2) $B(\boldsymbol{x}) < 0$ for all $\boldsymbol{x}$ in the unsafe set X_U; 3) no trajectory can escape from $\{\boldsymbol{x} \in \mathbb{R}^n \mid B(\boldsymbol{x}) \geq 0\}$ through the boundary $\{\boldsymbol{x} \in \mathbb{R}^n \mid B(\boldsymbol{x}) = 0\}$. A sufficient condition for this constraint is that the Lie derivative of $B(\boldsymbol{x})$ w.r.t the dynamics $\dot{\boldsymbol{x}} = \boldsymbol{f}$ is positive all over the invariant region, i.e., $\mathcal{L}_{\boldsymbol{f}} B(\boldsymbol{x}) > 0$, which means that all the trajectories must move in the increasing direction of the level sets of $B(\boldsymbol{x})$.

Barrier certificates can be used to verify safety properties without computing the flowpipe explicitly. The essential idea is to use the zero level set of $B(\boldsymbol{x})$ as a barrier to separate the flowpipe from the unsafe set. Moreover, if the unsafe set is very close to the boundary of the flowpipe, the barrier has to fit the shape of the flowpipe to make sure that all components of the constraint are satisfied. However, the zero level set of a polynomial of fixed degree may not have the power to mimic the shape of the flowpipe, which means that there may exist no solution for the above constraints even if the system is safe. This problem might be addressed using piecewise barrier certificate, i.e., cutting the flowpipe into small pieces so that every piece is straight enough to have a barrier certificate of simple form. Unfortunately, this is infeasible because we know nothing about the flowpipe locally. Therefore, we have to find another way to proceed.

Instead of computing a single barrier certificate, we propose to compute barrier tubes to piecewise over-approximate the flowpipe. Concretely, in the beginning, we first construct a containing box, called **enclosure**, for the initial set using interval approach [29] and simulation, then, using linear programming, we compute a group of barrier functions which work together to form a tight tube (called barrier tube) around the flowpipe. Similarly, taking the intersection of the barrier tube and the boundary of the box as the new initial set, we repeat the previous operations to obtain successive barrier tubes step by step. The key point here is how to compute a group of tightly enclosing barriers around the flowpipe without a constraint on the unsafe set inside the box. Our basic idea is to construct a group of auxiliary state sets U around the flowpipe and then, for each $U_i \in U$, we compute a barrier certificate between U_i and the flowpipe. If a barrier certificate is found, we expand U_i towards the flowpipe iteratively until no more barrier certificate can be found; otherwise, we shrink U_i away from the flowpipe until a barrier certificate is found. Since the auxiliary sets are distributed around the flowpipe, so is the barrier tube. The benefit of such piecewise barrier tubes is that they are time independent, and hence can avoid the issue of stretched flowpipe segments caused by speed differences between trajectories. Moreover, usually a small number of BTs can form a tight over-approximation of the flowpipe, which means that less computation is needed to decide the intersection of PBT and the unsafe set.

The main contributions of this paper are as follows:

1. We transform the constraint-solving problem for barrier certificates into a linear programming problem using Handelman representation [15];
2. We introduce PBT to over-approximate the flowpipe of nonlinear systems, thus dealing with flowpipes independent of time and hence avoiding the error explosion caused by stretched flowpipe segments;
3. We implement a prototype in C++ to compute PTB automatically and we show the effectiveness of our approach by providing a comparison with the state-of-the-art tools for reachability analysis of polynomial nonlinear systems such as *CORA* [1] and *Flow** [3].

The paper is organized as follows. Section 2 is devoted to the preliminaries. Section 3 shows how to compute barrier certificates using Handelman representation, while in Section 4 we present a method to compute Piecewise Barrier Tubes. Section 5 provides our experimental results and we conclude in Section 6.

2 Preliminaries

In this section, we recall some concepts used throughout the paper. We first clarify some notation conventions. If not specified otherwise, we use boldface lower case letters to denote vectors, we use $\mathbb{R}$ for the real number field and $\mathbb{N}$ for the set of natural numbers, and we consider multivariate polynomials in $\mathbb{R}[\boldsymbol{x}]$, where the components of $\boldsymbol{x}$ act as indeterminates. In addition, for all the polynomials $B(\boldsymbol{u}, \boldsymbol{x})$, we denote by $\boldsymbol{u}$ the vector composed of all the u_i and denote by $\boldsymbol{x}$ the vector composed of all the remaining variables x_i that occur in the polynomial. We use $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{>0}$ to denote the domain of nonnegative real number and positive real number respectively.

Let $P \subseteq \mathbb{R}^n$ be a convex and compact polyhedron with non-empty interior, bounded by linear polynomials $p_1, \cdots, p_m \in \mathbb{R}[\boldsymbol{x}]$. Without lose of generality, we may assume $P = \{\boldsymbol{x} \in \mathbb{R}^n \mid p_i(\boldsymbol{x}) \geq 0, i = 1, \cdots, m\}$.

Next, we present the notation of the Lie derivative, which is widely used in the discipline of differential geometry. Let $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ be a continuous vector field such that $\dot{x}_i = f_i(\boldsymbol{x})$ where $\dot{x}_i$ is the time derivative of $x_i(t)$.

Definition 1 (Lie derivative). *For a given polynomial $p \in \mathbb{R}[\boldsymbol{x}]$ over $\boldsymbol{x} = (x_1, \ldots, x_n)$ and a continuous system $\dot{\boldsymbol{x}} = \boldsymbol{f}$, where $\boldsymbol{f} = (f_1, \ldots, f_n)$, the **Lie derivative** of $p \in \mathbb{R}[\boldsymbol{x}]$ along $\boldsymbol{f}$ of order k is defined as follows.*

$$\mathcal{L}_{\boldsymbol{f}}^k p \overset{def}{=} \begin{cases} p, & k = 0 \\ \sum_{i=1}^n \frac{\partial \mathcal{L}_{\boldsymbol{f}}^{k-1} p}{\partial x_i} \cdot f_i, & k \geq 1 \end{cases}$$

Essentially, the k-th order Lie derivative of p is the k-th derivative of p w.r.t. time, i.e., reflects the change of p over time. We write $\mathcal{L}_{\boldsymbol{f}} p$ for $\mathcal{L}_{\boldsymbol{f}}^1 p$.

In this paper, we focus on semialgebraic nonlinear systems, which are defined as follows.

Definition 2 (Semialgebraic system). *A **semialgebraic system** is a triple $M \overset{def}{=} \langle X, \boldsymbol{f}, X_0, I \rangle$, where*

1. *$X \subseteq \mathbb{R}^n$ is the state space of the system M,*
2. *$\boldsymbol{f} \in \mathbb{R}[\boldsymbol{x}]^n$ is locally Lipschitz continuous vector function,*
3. *$X_0 \subseteq X$ is the initial set, which is semialgebraic [40],*
4. *I is the invariant of the system.*

The local Lipschitz continuity guarantees the existence and uniqueness of the differential equation $\dot{\boldsymbol{x}} = \boldsymbol{f}$ locally. A trajectory of a semialgebraic system is defined as follows.

Definition 3 (Trajectory). *Given a semialgebraic system M, a **trajectory** originating from a point $\boldsymbol{x}_0 \in X_0$ to time $T > 0$ is a continuous and differentiable function $\boldsymbol{\zeta}(\boldsymbol{x}_0, t) : [0, T) \to \mathbb{R}^n$ such that 1) $\boldsymbol{\zeta}(x_0, 0) = \boldsymbol{x}_0$, and 2) $\forall \tau \in [0, T)$: $\frac{d\boldsymbol{\zeta}}{dt}\big|_{t=\tau} = \boldsymbol{f}(\boldsymbol{\zeta}(\boldsymbol{x}_0, \tau))$. T is assumed to be within the maximal interval of existence of the solution from $\boldsymbol{x}_0$.*

For ease of readability, we also use $\zeta(t)$ for $\zeta(\boldsymbol{x}_0, t)$. In addition, we use $Flow_{\boldsymbol{f}}(X_0)$ to denote the flowpipe of initial set X_0, i.e.,

$$Flow_{\boldsymbol{f}}(X_0) \overset{\text{def}}{=} \{\boldsymbol{\zeta}(\boldsymbol{x}_0, t) \mid \boldsymbol{x}_0 \in X_0, t \in \mathbb{R}_{\geq}, \dot{\boldsymbol{\zeta}} = \boldsymbol{f}(\boldsymbol{\zeta})\} \quad (1)$$

Definition 4 (Safety). *Given an unsafe set $X_U \subseteq X$, a semialgebraic system $M = \langle X, \boldsymbol{f}, X_0, I \rangle$ is said to be **safe** if no trajectory $\boldsymbol{\zeta}(\boldsymbol{x}_0, t)$ of M satisfies that $\exists \tau \in \mathbb{R}_{\geq 0} : \boldsymbol{x}(\tau) \in X_U$, where $\boldsymbol{x}_0 \in X_0$.*

3 Computing Barrier Certificates

Given a semialgebraic system M, a barrier certificate is a real-valued function $B(\boldsymbol{x})$ such that 1) $B(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x}$ in the initial set; 2) $B(\boldsymbol{x}) < 0$ for all $\boldsymbol{x}$ in the unsafe set; 3) no trajectory can escape from the region of $B(\boldsymbol{x}) \geq 0$. Then, the hyper-surface $\{\boldsymbol{x} \in \mathbb{R}^n \mid B(\boldsymbol{x}) = 0\}$ forms a barrier separating the flowpipe from the unsafe set. To compute such a barrier certificate, the most common approach is template based constraint solving, i.e., firstly figure out a sufficient condition for the above condition and then, set up a template polynomial $B(\boldsymbol{u}, \boldsymbol{x})$ of fixed degree, and finally solve the constraint on $\boldsymbol{u}$ derived from the sufficient condition on $B(\boldsymbol{u}, \boldsymbol{x})$. There are a couple of sufficient conditions available for this purpose [22,27,13]. In order to have an efficient constraint solving method, we adopt the following condition [33].

Theorem 1. *Given a semialgebraic system M, let X_0 and U be the initial set and the unsafe set respectively, the system is guaranteed to be safe if there exists a real-valued function $B(\boldsymbol{x})$ such that*

$$\forall \boldsymbol{x} \in X_0 : B(\boldsymbol{x}) > 0 \tag{2}$$

$$\forall \boldsymbol{x} \in I : \mathcal{L}_f B > 0 \tag{3}$$

$$\forall \boldsymbol{x} \in X_U : B(\boldsymbol{x}) < 0 \tag{4}$$

In Theorem 1, the condition (3) means that all the trajectories of the system always point in the increasing direction of the level sets of $B(\boldsymbol{x})$ in the region I. Therefore, no trajectory starting from the initial set would cross the zero level set. The benefit of this condition is that it can be solved more efficiently than other existing conditions [13,22] although it is relatively conservative. The most widely used approach is to transform the constraint-solving problem into a sum-of-squares (*SOS*) programming problem [33], which can be solved in polynomial time. However, a serious problem with *SOS* programming based approach is that automatic generation of polynomial templates is very hard to perform. We now show an example to demonstrate the reason. For simplicity, we assume that the initial set, the unsafe set and the invariant are defined by the polynomial inequalities $X_0(\boldsymbol{x}) \geq 0$, $X_U(\boldsymbol{x}) \geq 0$ and $I(\boldsymbol{x}) \geq 0$ respectively, then the *SOS* relaxation of Theorem 1 is that the following polynomials are all *SOS*

$$B(\boldsymbol{x}) - \mu_1(\boldsymbol{x}) X_0(\boldsymbol{x}) + \epsilon_1 \tag{5}$$

$$\mathcal{L}_f B - \mu_2(\boldsymbol{x}) I(\boldsymbol{x}) + \epsilon_2 \tag{6}$$

$$- B(\boldsymbol{x}) - \mu_3(\boldsymbol{x}) X_U(\boldsymbol{x}) + \epsilon_3 \tag{7}$$

where $\mu_i(\boldsymbol{x}), i = 1, \cdots, 3$ are *SOS* polynomials as well and $\epsilon_i > 0, i = 1, \cdots, 3$. Suppose the degrees of $X_0(\boldsymbol{x})$, $I(\boldsymbol{x})$ and $X_U(\boldsymbol{x})$ are all odd numbers. Then, the degree of the template for $B(\boldsymbol{x})$ must be an odd number too. The reason is that, if $deg(B)$ is an even number, in order for the first and third polynomials to be *SOS* polynomials, $deg(B)$ must be greater than both $deg(\mu_3 X_U)$ and $deg(\mu_1 X_0)$, which are odd numbers. However, since the first and third condition contain $B(\boldsymbol{x})$

and $-B(\boldsymbol{x})$ respectively, their leading monomials must have the opposite sign, which means that they cannot be *SOS* polynomial simultaneously. Moreover, the degrees of the templates for the auxiliary polynomials $\mu_1(\boldsymbol{x}), \mu_3(\boldsymbol{x})$ must also be chosen properly so that $deg(\mu_1 X_0) = deg(\mu_3 X_U) = deg(B)$, because only in this way the leading monomials (which has an odd degree) of (5) and (7) have the chance to be resolved so that the resultant polynomial can be a *SOS*. Similarly, in order to make the second polynomial a *SOS* as well, one has to choose an appropriate degree for $\mu_2(\boldsymbol{x})$ according to the degree of $\mathcal{L}_f B$ and $I(\boldsymbol{x})$. As a result, the tangled constraints on the relevant template polynomials reduce the power of *SOS* programming significantly.

Due to the above reason, inspired by the work [38], we use Handelman representation to relax Theorem 1. We assume that the initial set X_0, the unsafe set X_U and the invariant I are all convex and compact polyhedra, i.e., $X_0 = \{\boldsymbol{x} \in \mathbb{R}^n \mid p_1(\boldsymbol{x}) \geq 0, \cdots, p_{m_1}(\boldsymbol{x}) \geq 0\}$, $I = \{\boldsymbol{x} \in \mathbb{R}^n \mid q_1(\boldsymbol{x}) \geq 0, \cdots, q_{m_2}(\boldsymbol{x}) \geq 0\}$ and $X_U = \{\boldsymbol{x} \in \mathbb{R}^n \mid r_1(\boldsymbol{x}) \geq 0, \cdots, r_{m_3}(\boldsymbol{x}) \geq 0\}$, where $p_i(\boldsymbol{x}), q_j(\boldsymbol{x}), r_k(\boldsymbol{x})$ are linear polynomials. Then, we have the following theorem.

Theorem 2. *Given a semialgebraic system M, let X_0, X_U and I be defined as above, the system is guaranteed to be safe if there exists a real-valued polynomial function $B(\boldsymbol{x})$ such that*

$$B(\boldsymbol{x}) \equiv \sum_{|\boldsymbol{\alpha}| \leq M_1} \lambda_{\boldsymbol{\alpha}} p_1^{\alpha_1} \cdots p_{m_1}^{\alpha_{m_1}} + \epsilon_1 \tag{8}$$

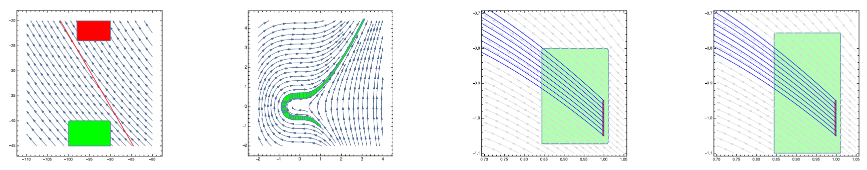
$$\mathcal{L}_f B \equiv \sum_{|\boldsymbol{\beta}| \leq M_2} \lambda_{\boldsymbol{\beta}} q_1^{\beta_1} \cdots q_{m_2}^{\beta_{m_2}} + \epsilon_2 \tag{9}$$

$$-B(\boldsymbol{x}) \equiv \sum_{|\boldsymbol{\gamma}| \leq M_3} \lambda_{\boldsymbol{\gamma}} r_1^{\gamma_1} \cdots r_{m_3}^{\gamma_{m_3}} + \epsilon_3 \tag{10}$$

where $\lambda_{\boldsymbol{\alpha}}, \lambda_{\boldsymbol{\beta}}, \lambda_{\boldsymbol{\gamma}} \in \mathbb{R}_{\geq 0}$, $\epsilon_i \in \mathbb{R}_{>0}$ and $M_i \in \mathbb{N}, i = 1, \cdots, 3$.

Theorem 2 provides us with an alternative to *SOS* programming to find barrier certificate $B(\boldsymbol{x})$ by transforming it into a linear programming problem. The basic idea is that we first set up a template $B(\boldsymbol{u}, \boldsymbol{x})$ of fixed degree as well as the appropriate $M_i, i = 1, \cdots, 3$ that make the both sides of the three identities (8)–(10) have the same degree. Since (8)–(10) are identities, the coefficients of the corresponding monomials on both sides must be identical as well. Thus, we derive a system S of linear equations and inequalities over $\boldsymbol{u}, \lambda_{\boldsymbol{\alpha}}, \lambda_{\boldsymbol{\beta}}, \lambda_{\boldsymbol{\gamma}}$. Now, finding a barrier certificate is just to find a feasible solution for S, which can be solved by linear programming. Compared to *SOS* programming based approach, this approach is more flexible in choosing the polynomial template as well as other parameters. We consider now a linear system to show how it works.

Example 2. Given a 2D system defined by $\dot{x} = 2x + 3y, \dot{y} = -4x + 2y$, let $X_0 = \{(x, y) \in \mathbb{R}^2 \mid p_1 = x + 100 \geq 0, p_2 = -90 - x \geq 0, p_3 = y + 45 \geq 0, p_4 = -40 - y \geq 0\}$, $I = \{(x, y) \in \mathbb{R}^2 \mid q_1 = x + 110 \geq 0, q_2 = -80 - x \geq 0, q_3 = y + 45 \geq 0, q_4 = -20 - y \geq 0\}$ and $X_U = \{(x, y) \in \mathbb{R}^2 \mid r_1 = x + 98 \geq 0, r_2 = -90 - x \geq$



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

Fig. 2: (a) Linear barrier certificate (straight red line) for Example 2. Rectangle in green: initial set, rectangle in red: unsafe set. (b) PBT for the running Example 5, consisting of 45 BTs. (c) Enclosure (before bloating) for flowpipe of Example 3 (green shadow region). (d) Enclosure (after bloating) for flowpipe of Example 3.

$0, r_3 = y + 24 \geq 0, r_4 = -20 - y \geq 0\}$. Assume $B(\boldsymbol{u}, \boldsymbol{x}) = u_1 + u_2x + u_3y$, $M_i = \epsilon_i = 1$ for $i = 1, \cdots, 3$, then we obtain the following polynomial identities according to Theorem 2

$$u_1 + u_2x + u_3y - \sum_{i=1}^{4} \lambda_{1i}p_i - \epsilon_1 \equiv 0$$

$$u_2(2x + 3y) + u_3(-4x + 2y) - \sum_{j=1}^{4} \lambda_{2j}q_j - \epsilon_2 \equiv 0$$

$$-(u_1 + u_2x + u_3y) - \sum_{k=1}^{4} \lambda_{3k}r_k - \epsilon_3 \equiv 0$$

where $\lambda_{ij} \geq 0$ for $i = 1, \cdots, 3$, $j = 1, \cdots, 4$. By collecting the coefficients of x, y in the above polynomials, we obtain a system S of linear polynomial equations and inequalities over u_i, λ_{jk}. By solving S using linear programming, we obtain a feasible solution and Figure 2a shows the computed linear barrier certificate. Note that, for the aforementioned reason, it is impossible to find a linear barrier certificate using *SOS* programming for this example.

4 Piecewise Barrier Tubes

In this section, we introduce how to construct PBTs for nonlinear polynomial systems. The basic idea of constructing PBT is that, for each segment of the flowpipe, an enclosure box is first constructed and then, a BT is constructed to form a tighter over-approximation for the flowpipe segment inside the box.

4.1 Constructing an enclosure box

Given an initial set, the first task is to construct an enclosure box for the initial set and the following segment of the flowpipe. As pointed out in Section 1, one principle to construct an enclosure box is to simplify the shape of the flowpipe segment, or in other words, to approximately bound the twisting of trajectories by some θ in the box, where the *twisting* of a trajectory is defined as follows.

Definition 5 (Twisting of a trajectory). *Let M be a continuous system and $\zeta(t)$ be a trajectory of M. Then, $\zeta(t)$ is said to have a twisting of θ on the time interval $I = [T_1, T_2]$, written as $\xi_I(\zeta)$, if it satisfies that $\xi_I(\zeta) = \theta$, where $\xi_I(\zeta) \overset{def}{=} \sup_{t_1,t_2 \in I} \arccos\left(\frac{\langle \dot{\zeta}(t_1), \dot{\zeta}(t_2)\rangle}{\|\dot{\zeta}(t_1)\| \|\dot{\zeta}(t_2)\|}\right)$.*

The basic idea to construct an enclosure box is depicted in Algorithm 1.

Algorithm 1: Algorithm to construct an enclosure box

```
input  : $M$: dynamics of the system; $n$: dimension of system; $X_0$: initial set
         $\theta_1$: twisting of simulation; $d$: maximum distance of simulation;
output: E: an enclosure box containing $X_0$; P: plane where flowpipe exits ;
        G: range of intersection of $Flow_f(X_0)$ with plane $P$ by simulation
1  sample a set $S_0$ of points from $X_0$;
2  select a point $\boldsymbol{x}_0 \in S_0$;
3  find a time step size $\Delta T_0$ by $(\theta, d)$-bounded simulation for $\boldsymbol{x}_0$;
4  $\Delta T \longleftarrow \Delta T_0$;
5  while $\Delta T > \epsilon$ do
6      $[found, \mathsf{E}] \longleftarrow$ find an enclosure box by interval arithmetic using $\Delta T$;
7      if $found$ then
8          do a simulation for all $\boldsymbol{x}_i \in S_0$, select the plane $P$ which intersects
           with the most of simulations; generate G;
9          bloat E s.t $Flow_f(X_0)$ gets out of E only through the facet in P;
10         break;
11     else
12         $\Delta T \longleftarrow 1/2 * \Delta T$;
```

Remark 1. In Algorithm 1, we use interval arithmetic [29] and simulation to construct an enclosure box E for a given initial set and its following flowpipe segment. Meanwhile, we obtain a coarse range of the intersection of the flowpipe and the boundary of the enclosure, which helps to accelerate the construction of barrier tube. To be simple, the enclosure is constructed in a way such that the flowpipe gets out of the box through a single facet. Given an initial set X_0, we first sample a set S_0 of points from X_0 for simulation. Then, we select a point $\boldsymbol{x}_0$ from S_0 and do (θ, d)-simulation on $\boldsymbol{x}_0$ to obtain a time step ΔT. A (θ, d)-simulation is a simulation that stops either when the twisting of the simulation reaches θ or when the distance between x_0 and the end point reaches d. On the one hand, by using a small θ, we aim to achieve a straight flowpipe segment. On the other hand, by specifying a maximal distance d, we make sure that the simulation can stop for a long and straight flowpipe. At each iteration of the *while* loop in line 5, we first try to construct an enclosure box by interval arithmetic over ΔT. If such an enclosure box is created, we then perform a simulation (see line 8) for all the points in S_0 to find out the plane P of facet which intersects with the most of the simulations. The idea behind line 9 is that in order to better over-approximate the intersection of the flowpipe with the boundary of the box using intervals, we push the other planes outwards to make P the only plane where the flowpipe get out of the box. Certainly, simply by simulation we cannot guarantee that the flowpipe does not intersect the other facets. Therefore, we have the following theorem for the decision.

Theorem 3. *Given a semialgebraic system M and an initial set X_0, a box E is an enclosure of X_0 and F_i is a facet of E. Then, $(Flow_f(X_0) \cap E) \cap F_i = \emptyset$ if there exists a barrier certificate $B_i(\boldsymbol{x})$ for X_0 and F_i inside E.*

Remark 2. According to the definition of barrier certificate, the proof of Theorem 3 is straightforward, which is ignored here. Therefore, to make sure that the flowpipe does not intersect the facet F_i, we only need to find a barrier certificate, which can be done using the approach presented in Section 3. Moreover, if no barrier certificate can be found, we further bloat the facet. Next, we still use the running Example 1 to demonstrate the process of constructing an enclosure.

Example 3 (running example). Consider the system in Example 1 and the initial set $x = 1.0, -1.05 \leq y \leq -0.95$, let the bounding twisting of simulation be $\theta = \pi/18$, then the time step size we computed for interval evaluation is $\Delta T = 0.2947$. The corresponding enclosure computed by interval arithmetic is shown in Figure 2c. Furthermore, by simulation, we know that the flowpipe can reach both left facet and top facet. Therefore, we have two options to bloat the facet: bloat the left facet to make the flowpipe intersects the top facet only or bloat the top facet to make the flowpipe intersects left facet only. In this example, we choose the latter option and the bloated enclosure is shown in Figure 2d. In this way, we can over-approximate the intersection of the flowpipe and the facet by intervals if we can obtain its boundary on every side. This can be achieved by finding barrier tube.

4.2 Compute a barrier tube inside a box

An important fact about the flowpipe of continuous system is that it tends to be straight if it is short enough, given that the initial set is straight as well (otherwise, we can split it). Suppose there is a small box E around a straight flowpipe, it will be easy to compute a barrier certificate for a given initial set and unsafe set inside E. A barrier tube for the flowpipe in E is a group of barrier certificates which form a tube around a flowpipe inside E. Formally,

Definition 6 (Barrier Tube). *Given a semialgebraic system M, a box E and an initial set $X_0 \subseteq E$, a barrier tube is a set of real-valued functions $BT = \{B_i(\boldsymbol{x}), i = 1, \cdots, m\}$ such that for all $B_i(\boldsymbol{x}) \in BT$: 1) $\forall \boldsymbol{x} \in X_0 : B_i(\boldsymbol{x}) > 0$ and, 2) $\forall \boldsymbol{x} \in E : \mathcal{L}_f B_i > 0$.*

According to Definition 6, a barrier tube BT is defined by a set of real-valued functions and every function inequality $B_i(\boldsymbol{x}) > 0$ is an invariant of M in E and so do their conjunction. The property of a barrier tube BT is formally described in the following theorem.

Theorem 4. *Given a semialgebraic system M, a box E and an initial set $X_0 \subseteq E$, let $BT = \{B_i(\boldsymbol{x}) : i = 1, \cdots, m\}$ be a barrier tube of M and $\Omega = \{\boldsymbol{x} \in \mathbb{R}^n \mid \bigwedge B_i(\boldsymbol{x}) > 0, B_i \in BT\}$, then $Flow_{\boldsymbol{f}}(X_0) \cap E \subseteq \Omega \cap E$.*

Remark 3. Theorem 4 states that an arbitrary barrier tube is able to form an over-approximation for the reach pipe in the box E. Compared to a single barrier certificate, multiple barrier certificates could over-approximate the flowpipe more precisely. However, since there is no constraint on unsafe sets in Definition 6, a barrier tube satisfying the definition could be very conservative. In order to obtain an accurate approximation for the flowpipe, we choose to create additional auxiliary constraints.

Auxiliary Unsafe Set (AUS) To obtain an accurate barrier tube, there are two main questions to be answered: 1) How many barrier certificates are needed? and 2) How do we control their positions to make the tube well-shaped to better over-approximate the flowpipe? The answer for the first question is quite simple: the more, the better. This will be explained later on. For the second question, the answer is to construct a group of properly distributed auxiliary state sets (AUSs). Each set of the AUSs is used as an unsafe set U_i for the system and then we compute a barrier certificate B_i for U_i according to Theorem 2. Since the zero level set of B_i serves as a barrier between the flowpipe and U_i, the space where a barrier could appear is fully determined by the position of U_i. Roughly speaking, when U_i is far away from the flowpipe, the space for a barrier to exist is wide as well. Correspondingly, the barrier certificate found would usually locate far away from the flowpipe as well. Certainly, as U_i gets closer to the flowpipe, the space for barrier certificates also contracts towards the flowpipe accordingly. Therefore, by expanding U_i towards the flowpipe, we can get more precise over-approximations for the flowpipe.

Why multiple AUS ? Although the accuracy of the barrier certificate over-approximation can be improved by expanding the AUS towards the flowpipe, the capability of a single barrier certificate is very limited because it can erect a barrier which only matches a single profile of the flow pipe. However, if we have a set U of AUSs which are distributed evenly around the flowpipe and there is a barrier certificate B_i for each $U_i \in U$, these barrier certificates would be able to over-approximate the flowpipe from a number of profiles. Therefore, increasing the number of AUSs can increase the quality of the over-approximation as well. Furthermore, if all these auxiliary sets are connected, all the barriers would form a tube surrounding the flowpipe. Therefore, if we can create a series of boxes piecewise covering the flowpipe and then construct a barrier tube for every piece of the flowpipe, we obtain an over-approximation for the flowpipe by PBT.

Based on the above idea, we provide Algorithm 2 to compute barrier tube.

Algorithm 2: Algorithm to compute barrier tube

```
input  : $M$: dynamics of the system; $X_0$: Initial set;
         $\mathsf{E}$: interval enclosure of initial set;
         $\mathsf{G}$: interval approx. of $(\partial\mathsf{E} \cap Flow_f(X_0))$ by simulation;
         $\mathsf{P}$: plane where flowpipe exits from box;
         $\mathsf{D}$: candidate degree list for template polynomial;
         $\epsilon$: difference in size between AUS (auxiliary unsafe set)
output: $\mathsf{BT}$: barrier tube; $X_0'$: interval over-approximation of $(\mathsf{BT} \cap E)$
1  foreach $G_{ij}$: an facet of G do
2      $found \longleftarrow false$ ;
3      foreach $d \in D$ do
4          AUS $\longleftarrow$ CreateAUS(G, P, $G_{ij}$);
5          while true do
6              $[found, B_{ij}] \longleftarrow$ ComputeBarrierCert($X_0$, E, AUS, $d$) ;
7              if $found$ then AUS' $\longleftarrow$ Expand (AUS);
8              else AUS' $\longleftarrow$ Contract (AUS) ;
9              if Diff(AUS', AUS) $\leq \epsilon$ then break;
10             else AUS' $\longleftarrow$ AUS;
11     if $found$ then BT $\longleftarrow$ Push(BT, $B_{ij}$); break;
12     else return FAIL;
13 return SUCCEED;
```

Remark 4. In Algorithm 2, for an n-dimensional flowpipe segment, we aim to build a barrier tube composed of $2(n-1)$ barrier certificates, which means we need to construct $2(n-1)$ AUSs. According to Algorithm 1, we know that the plane P is the only exit of the flowpipe from the enclosure E and G is roughly the region where they intersect. Let F^G be the facet of E that contains G, then for every facet F_{ij}^G of F^G, we can take an $(n-1)$-dimensional rectangle between F_{ij}^G and G_{ij} as an AUS, where G_{ij} is the facet of G adjacent to F_G^{ij}. Therefore, enumerating all the facets of G in line 1 would produce $2(n-1)$ positions for AUS. The loop in line 3 is attempting to find a polynomial barrier certificate of different degrees in D. In the while loop 5, we iteratively compute the best barrier certificate by adjusting the width of AUS through binary search until the difference in width between two successive AUSs is less than the specified threshold ϵ.

Example 4 (Running example). Consider the initial set and the enclosure computed in Example 3, we use Algorithm 2 to compute a barrier tube. The initial set is $X_0 = [1.0, 1.0] \times [-1.05, -0.95]$ and the enclosure of X_0 is $E = [0.84, 1.01] \times [-1.1, -0.75]$, $G = [0.84, 0.84] \times [-0.91, -0.80]$, the plane P is $x = 0.84$, $D = \{2\}$ and $\epsilon = 0.001$. The barrier tube consists of two barrier certificates. As shown in Figure 3, each of the barrier certificates is derived from an AUS (red line segment) which is located respectively on the bottom-left and top-left boundary of E.

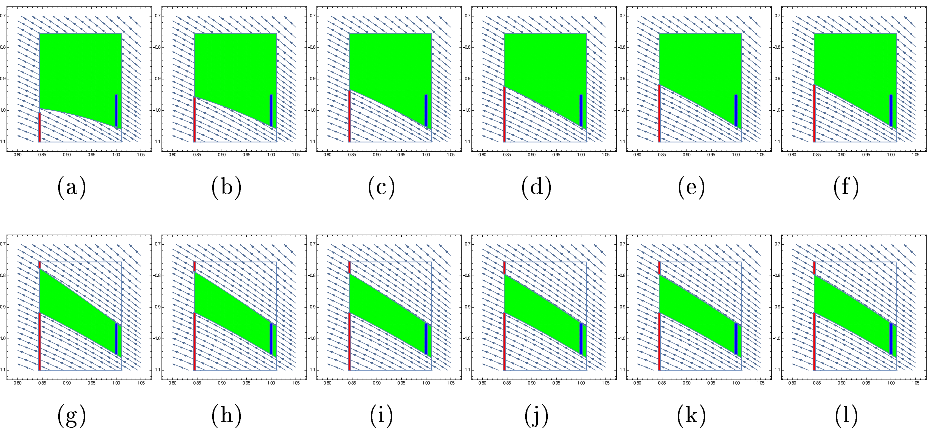


Fig. 3: Computing process of BT for Example 4. blue line segment: initial set, red line segment: `AUS`. Figure 3a–3l show how intermediate barrier certificates changed with the width of the `AUS`s and Figure 3l shows the final BT (shadow region in green).

4.3 Compute Piecewise Barrier Tube

During the computation of a barrier tube by Algorithm 2, we create a series of `AUS`s around the flowpipe, which build up a rectangular enclosure for the intersection of the flowpipe and the facet of the enclosure box. As a result, such a rectangular enclosure can be taken as an initial set for the following flowpipe segment and then Algorithm 2 can be applied repeatedly to compute a PBT. The basic procedure to compute PBT is presented in Algorithm 3.

Remark 5. In Algorithm 3, initially a box that contains the initial set X_0 is constructed using Algorithm 1. The loop in line 2 consists of 3 major parts: 1) In lines 3–6, a barrier tube BT is firstly computed using Algorithm 2. The **while** loop keeps shrinking the box until a barrier tube is found; 2) In line 8, the initial set X_0 is updated for the next box; 3) In line 9, a new box is constructed to contain X_0 and the process is repeated.

Example 5 (Running example). Let us consider again the running example. We set the length of PBT to 45 and the PBT we obtained is shown in Figure 2b. Compared to the interval over-approximation of the Taylor model obtained using *Flow**, the computed PBT consists of a significantly reduced number of segments and is more precise for the absence of stretching.

Safety verification based on PBT The idea of safety verification based on PBT is straightforward. Given an unsafe set X_U, for each intermediate initial set X_0 and the corresponding enclosure box E, we first check whether $X_U \cap E = \emptyset$. If not empty, we would further find a barrier certificate between X_U and the flowpipe of X_0 inside E. If empty or barrier found, we continue to compute

Algorithm 3: Algorithm to compute PBT

```
input  : $M$: dynamics of the system; $X_0$: Initial set;
         $N$: length of piecewise barrier tube
output: PBT: piecewise barrier tube
1 E ← construct an initial box containing $X_0$;
2 for $i \leftarrow 1$ to $N$ do
3     [$Found$, BT] ← findBarrierTube (E,$X_0$) ;
4     while not $Found$ do
5         E ← Shrink (E) ;
6         [$Found$, BT] ← findBarrierTube (E,$X_0$) ;
7     if $Found$ then
8         $X_0$ ← OverApprox(BT ∩ Facet(E)) ;
9         E ← construct the next box containing $X_0$;
```

Table 1: Model Definitions

Model	Dynamics	Initial Set X_0	Time Horizon(TH)
Controller 2D	$\dot{x} = xy + y^3 + 2$ $\dot{y} = x^2 + 2x - 3y$	$x \in [29.9, 30.1]$ $y \in [-38, -36]$	0.0125
Van der Pol Oscillator	$\dot{x} = y$ $\dot{y} = y - x - x^2 y$	$x \in [1, 1.5]$ $y \in [2.0, 2.45]$	6.74
Lotka-Volterra	$\dot{x} = x(1.5 - y)$ $\dot{y} = -y(3 - x)$	$x \in [4.5, 5.2]$ $y \in [1.8, 2.2]$	3.2
Controller 3D	$\dot{x} = 10(y - x)$ $\dot{y} = x^3$ $\dot{z} = xy - 2.667z$	$x \in [1.79, 1.81]$ $y \in [1.0, 1.1]$ $y \in [0.5, 0.6]$	0.51

longer PBT. The refinement of PBT computation can be achieved by using smaller E and higher d for template polynomial.

5 Implementation and Experiments

We have implemented the proposed approach as a C++ prototype called Piecewise Barrier Tube Solver (*PBTS*), choosing *Gurobi* [12] as our internal linear programming solver. We have also performed some experiments on a benchmark of four nonlinear polynomial dynamical systems (described in Table 1) to compare the efficiency and the effectiveness of our approach w.r.t. other tools. Our experiments were performed on a desktop computer with a 3.6GHz *Intel Core i7-7700* 8 Core CPU and 32 GB memory. The results are presented in Table 2.

Remark 6. There are a number of outstanding tools for flowpipe computation [4,5,1,3]. Since our approach is to perform flowpipe computation for polynomial nonlinear systems, we pick two of the most relevant state-of-the-art tools for comparison: *CORA* [1] and *Flow** [3]. Note that a big difference between our

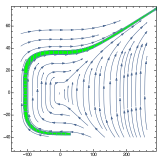
Table 2: Tool Comparison on Nonlinear Systems. #var: number of variables; T: computing time; NFS: number of flowpipe segments; DEG: candidate degrees for template polynomial (only for *PBTS*); TH: time horizon for flowpipe (only for *Flow** and *CORA*). FAIL: failed to terminate under 30min.

Model	#var	PBTS			TH	Flow*		CORA	
		T	NFS	DEG		T	NFS	T	NFS
Controller 2D	2	5.62	46	2	0.0125	22.17	6250	FAIL	-
Van der Pol	2	13.38	110	2,3	6.74	15.28	337	212.51	12523
Lotka-Volterra	2	6.65	30	3,4	3.2	10.59	3200	35.84	2903
Controller 3D	3	83.65	15	4	0.51	11.61	5100	65.18	6767

approach and the other two approaches is that *PBTS* is time-independent, which means that we cannot compare PBTS with *CORA* or *Flow** over the exactly same time horizon. To be fair enough, for *Flow** and *CORA*, we have used the same time horizon for the flowpipe computation, while we have computed a slightly longer flowpipe using *PBTS*. To guide the reader, we have also used different plotting colors to visualize the difference between the flowpipes obtained from the three different tools.

Evaluation As pointed out in Section 1, a common problem with the bounded-time integration based approaches is that the flowpipe segment of a dynamics system can be extremely stretched with time so that the interval over-approximation of the flowpipe segment is very conservative and usually the solver has to stop prematurely due to the error explosion. This fact can be found easily from the figures Fig. 4– 7. In particular, for *Controller 2D*, *Flow** can give quite nice result in the beginning but started producing an exploding flowpipe very quickly (Note that *Flow** offers options to produce better plotting which however is expensive and was not used for safety verification. *CORA* even failed to give a result after over 30 minutes of running). This phenomenon reappeared with both *Flow** and *CORA* for *Controller 3D*. Notice that most of the time horizons used in the experiment are basically the time limits that *Flow** and *CORA* can reach, i.e., a slightly larger value for the time horizon would cause the solvers to fail. In comparison, our tool has no such problem and can survive a much longer flowpipe before exploding or even without exploding as shown in Fig. 4a.

Another important factor of the approaches is the efficiency. As is shown in Table 2, our approach is more efficient on the first three examples but slower on the last example than the other two tools. The reason for this phenomenon is that the degree d of the template polynomial used in the last example is higher than the others and increasing d led to an increase in the number of decision variables in the linear constraint. This suggests that using smaller d on shorter flowpipe segment would be better. In addition, we can also see in Table 2 that the number of the flowpipe segments produced by *PBTS* is much fewer than that produced by *Flow** and *CORA*. In this respect, *PBTS* would be more efficient on safety verification.


(a) PBTS

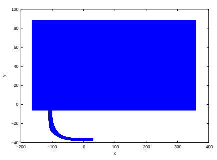
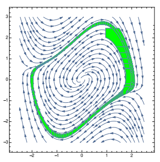
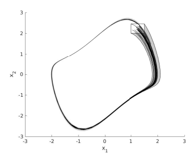

(b) Flow*

Fig. 4: Flowpipe for Controller 2D.


(a) PBTS


(b) CORA

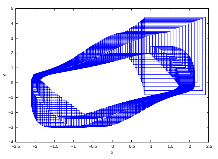
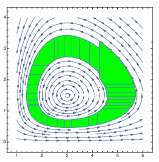
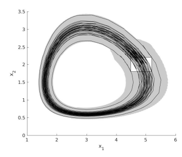

(c) Flow*

Fig. 5: Flowpipe for Van der Pol Oscillator.


(a) PBTS


(b) CORA

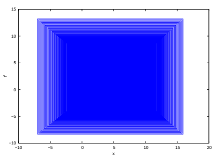
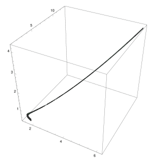
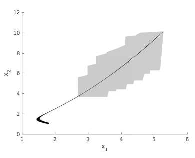

(c) Flow*

Fig. 6: Flowpipe for Lotka-Volterra.


(a) PBTS


(b) CORA (projection)

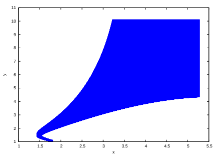

(c) Flow* (projection)

Fig. 7: Flowpipe (projection) for Controller 3D.

6 Conclusion

We have presented PBTS, a novel approach to over-approximate flowpipes of nonlinear systems with polynomial dynamics. The benefit of using BTs is that they are time-independent and hence cannot be stretched or deformed by time. Moreover, this approach only results in a small number of BTs which are sufficient to form a tight over-approximation for the flowpipe, hence the safety verification with PBT can be very efficient.

References

1. Matthias Althoff and Dmitry Grebenyuk. Implementation of interval arithmetic in CORA 2016. In *Proc. of ARCH@CPSWeek 2016: the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, volume 43 of *EPiC Series in Computing*, pages 91–105. EasyChair, 2017.
2. Eugene Asarin, Thao Dang, and Antoine Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43(7):451–476, 2007.
3. Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proc. of CAV 2013: the 25th International Conference on Computer Aided Verification*, volume 8044 of *LCNS*, pages 258–263. Springer, 2013.
4. Thao Dang, Colas Le Guernic, and Oded Maler. Computing reachable states for nonlinear biological models. In *Proc. of CMSB 2009: the 7th International Conference on Computational Methods in Systems Biology*, volume 5688 of *LNCS*, pages 126–141. Springer, 2009.
5. Parasara Sridhar Duggirala, Sayan Mitra, Mahesh Viswanathan, and Matthew Potok. C2E2: A verification tool for stateflow models. In *Proc. of TACAS 2015: the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 9035 of *LNCS*, pages 68–82. Springer, 2015.
6. Martin Fränzle and Christian Herde. Hysat: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30(3):179–198, 2007.
7. Martin Fränzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert. Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *JSAT*, 1(3-4):209–236, 2007.
8. Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In *Proc. of CAV 2011: the 23rd International Conference on Computer Aided Verification*, volume 6806 of *LNCS*, pages 379–395. Springer, Springer, 2011.
9. Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Proc. of HSCC 2005: the 8th International Workshop on Hybrid Systems: Computation and Control*, volume 3414 of *LNCS*, pages 291–305. Springer, 2005.
10. Antoine Girard and Colas Le Guernic. Efficient reachability analysis for linear systems using support functions. *Proc. of IFAC World Congress*, 41(2):8966–8971, 2008.
11. Radu Grosu, Grégory Batt, Flavio H. Fenton, James Glimm, Colas Le Guernic, Scott A. Smolka, and Ezio Bartocci. From cardiac cells to genetic regulatory networks. In *Proc. of CAV 2011: the 23rd International Conference Computer Aided Verification*, volume 6806 of *LNCS*, pages 396–411. Springer, 2011.
12. Zonghao Gu, Edward Rothberg, and Robert Bixby. Gurobi optimizer reference manual. `http://www.gurobi.com/documentation/7.5/refman/refman.html`, 2017.
13. S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *Proc. of CAV 2008: the 20th International Conference on Computer Aided Verification*, volume 5123 of *LNCS*, pages 190–203. Springer, 2008.
14. Amit Gurung, Rajarshi Ray, Ezio Bartocci, Sergiy Bogomolov, and Radu Grosu. Parallel reachability analysis of hybrid systems in xspeed. *International Journal on Software Tools for Technology Transfer*, 2018.

15. David Handelman. Representing polynomials by positive linear functions on compact convex polyhedra. *Pacific J. Math.*, 132(1):35–62, 1988.
16. Arnd Hartmanns and Holger Hermanns. The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification. In *Proc. of TACAS'14: the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *LNCS*, pages 593–598. Springer, 2014.
17. T.A. Henzinger. The theory of hybrid automata. In *Proc. IEEE Symp. Logic in Computer Science*, pages 278–292, 1996.
18. Zhenqi Huang, Chuchu Fan, Alexandru Mereacre, Sayan Mitra, and Marta Z. Kwiatkowska. Invariant verification of nonlinear hybrid automata networks of cardiac cells. In *Proc. of CAV 2014: the 26th International Conference on Computer Aided Verification*, volume 8559 of *LNCS*, pages 373–390. Springer, 2014.
19. Yu Jiang, Yixiao Yang, Han Liu, Hui Kong, Ming Gu, Jiaguang Sun, and Lui Sha. From stateflow simulation to verified implementation: A verification approach and a real-time train controller design. In *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2016 IEEE*, pages 1–11. IEEE, 2016.
20. Yu Jiang, Hehua Zhang, Zonghui Li, Yangdong Deng, Xiaoyu Song, Ming Gu, and Jiaguang Sun. Design and optimization of multiclocked embedded systems using formal techniques. *IEEE transactions on industrial electronics*, 62(2):1270–1278, 2015.
21. Hui Kong, Sergiy Bogomolov, Christian Schilling, Yu Jiang, and Thomas A. Henzinger. Safety verification of nonlinear hybrid systems based on invariant clusters. In *Proc. of HSCC 2017: the 20th International Conference on Hybrid Systems: Computation and Control*, pages 163–172. ACM, 2017.
22. Hui Kong, Fei He, Xiaoyu Song, William NN Hung, and Ming Gu. Exponential-condition-based barrier certificate generation for safety verification of hybrid systems. In *Proc. of CAV 2013: the 25th International Conference on Computer Aided Verification*, volume 8044 of *LNCS*, pages 242–257. Springer, 2013.
23. Soonho Kong, Sicun Gao, Wei Chen, and Edmund M. Clarke. dReach: δ-reachability analysis for hybrid systems. In *Proc. of TACAS'15: the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 9035 of *LNCS*, pages 200–205. Springer, 2015.
24. Tomas Krilavicius. *Hybrid Techniques for Hybrid Systems.* PhD thesis, University of Twente, Enschede, Netherlands, 2006.
25. Ratan Lal and Pavithra Prabhakar. Bounded error flowpipe computation of parameterized linear systems. In *Proc. of EMSOFT 2015: the International Conference on Embedded Software*, pages 237–246. IEEE, 2015.
26. Colas Le Guernic and Antoine Girard. Reachability analysis of hybrid systems using support functions. In *Proc. of CAV 2009: the 21st Intern. Conference on Computer Aided Verification*, volume 5643 of *LNCS*, pages 540–554. Springer, 2009.
27. J. Liu, N. Zhan, and H. Zhao. Computing semi-algebraic invariants for polynomial dynamical systems. In *Proc. of EMSOFT 2011: the 11th International Conference on Embedded Software*, pages 97–106. ACM, 2011.
28. Nadir Matringe, Arnaldo Vieira Moura, and Rachid Rebiha. Generating invariants for non-linear hybrid systems by linear algebraic methods. In *Proc. of SAS 2010: the 17th International Symposium on Static Analysis*, volume 6337 of *LNCS*, pages 373–389. Springer, 2010.
29. Nedialko S Nedialkov. Interval tools for ODEs and DAEs. In *Proc. of SCAN 2006: the 12th GAMM - IMACS International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*, pages 4–4. IEEE, 2006.

30. Markus Neher, Kenneth R Jackson, and Nedialko S Nedialkov. On taylor model based integration of ODEs. *SIAM Journal on Num. Analysis*, 45(1):236–262, 2007.
31. Pavithra Prabhakar and Miriam García Soto. Hybridization for stability analysis of switched linear systems. In *Proc. of HSCC 2016: of the 19th Intern. Conference on Hybrid Systems: Computation and Control*, pages 71–80. ACM, 2016.
32. Pavithra Prabhakar and Mahesh Viswanathan. A dynamic algorithm for approximate flow computations. In *Proc. of HSSC 2011: the 14th International Conference on Hybrid Systems: Computation and Control*, pages 133–142. ACM, 2011.
33. S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. *Proc. of HSCC 2004: the 7th International Workshop on Hybrid Systems: Computation and Control*, 2993:271–274, 2004.
34. Rajarshi Ray, Amit Gurung, Binayak Das, Ezio Bartocci, Sergiy Bogomolov, and Radu Grosu. Xspeed: Accelerating reachability analysis on multi-core processors. In *Proc. of HVC 2015: the 11th International Haifa Verification Conference on Hardware and Software: Verification and Testing*, volume 9434 of *LNCS*, pages 3–18. Springer, 2015.
35. Nima Roohi, Pavithra Prabhakar, and Mahesh Viswanathan. Hybridization based CEGAR for hybrid automata with affine dynamics. In *Proc. of TACAS 2016: the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 9636 of *LNCS*, pages 752–769. Springer, 2016.
36. S. Sankaranarayanan. Automatic invariant generation for hybrid systems using ideal fixed points. In *Proc. of HSCC 2010: the 13th ACM International Conference on Hybrid Systems: Computation and Control*, pages 221–230. ACM, 2010.
37. S. Sankaranarayanan, H. Sipma, and Z. Manna. Constructing invariants for hybrid systems. In *Proc. of HSCC 2004: the 7th Intern. Workshop on Hybrid Systems: Computation and Control*, volume 2993 of *LNCS*, pages 539–554. Springer, 2004.
38. Sriram Sankaranarayanan, Xin Chen, et al. Lyapunov function synthesis using handelman representations. *IFAC Proceedings Volumes*, 46(23):576–581, 2013.
39. Andrew Sogokon, Khalil Ghorbal, Paul B Jackson, and André Platzer. A method for invariant generation for polynomial continuous systems. In *Proc. of VMCAI 2016: the 17th International Conference on Verification, Model Checking, and Abstract Interpretation*, volume 9583 of *LNCS*, pages 268–288. Springer, 2016.
40. Gilbert Stengle. A Nullstellensatz and a Positivstellensatz in semialgebraic geometry. *Mathematische Annalen*, 207(2):87–97, 1974.