A Dynamic Algorithm for Approximate Flow Computations

Pavithra Prabhakar
Department of Computer Science
University of Illinois at Urbana Champaign
pprabha2@illinois.edu

Mahesh Viswanathan
Department of Computer Science
University of Illinois at Urbana Champaign
vmahesh@illinois.edu

ABSTRACT

In this paper we consider the problem of approximating the set of states reachable within a time bound T in a linear dynamical system, to within a given error bound ϵ. Fixing a degree d, our algorithm divides the interval $[0, T]$ into sub-intervals of not necessarily equal size, such that a polynomial of degree d approximates the actual flow to within an error bound of ϵ, and approximates the reach set within each sub-interval by the polynomial tube. Our experimental evaluation of the algorithm when the degree d is fixed to be either 1 or 2, shows that the approach is promising, as it scales to large dimensional dynamical systems, and performs better than previous approaches that divided the interval $[0, T]$ evenly into sub-intervals.

Categories and Subject Descriptors

D.0 [**Software**]: General

General Terms

Verification

Keywords

Approximation, Linear Dynamical Systems, Post Computation

1. INTRODUCTION

Integral to the automatic verification of safety properties is the computation of the set of reachable states of a system. In the context of hybrid systems, the key challenge in reachability computation is to compute, for a given set of states X, all states that are reachable from X under the continuous dynamics, within (some) time T. While states reachable within a bounded time can be computed for some hybrid systems with simple dynamics [3, 1, 16, 19, 27], it must typically be approximated, since the problem of computing the reachable set is undecidable for most dynamical systems.

HSCC'11, April 12–14, 2011, Chicago, Illinois, USA.

There are three principal techniques for computing an approximation to the reachable set of states. The first constructs an abstract transition system that *simulates* (in a formal sense) the dynamical system, and carries out the reachability computation on the abstract system [5, 2, 8, 7, 11, 17, 26]. In this method, the quality of the approximate solution cannot be measured, and so, often this is compensated by repeatedly refining the abstract transition system. The second approach, called *hybridization* [25, 4, 10], partitions the continuous state space, and approximates the continuous dynamics in each partition. Here one can explicitly bound the error between the reachable set of the hybrid system with simpler dynamics and the original dynamical system.

The third method is to directly compute the states reachable within time T. This has been carried out primarily for linear dynamical systems and a convex polytope as initial set X. For such systems, the algorithm proceeds as follows. First, the time interval $[0, T]$ is partitioned into equal intervals of size Δ. Then, the points reached at time Δ from the vertices of X are computed. The set of states reachable *within* time Δ is then approximated by the convex hull of the vertices of the set X and the points reached at time Δ from the vertices of X. Given Δ, T, and the dynamics, the error (or Hausdorff distance) between this convex hull and the actual set of states reachable within time Δ can be bounded. Based on this error bound, this convex hull is first "bloated" to contain all the reachable states and then approximated by a data structure of choice. Different data structures that have been considered and found useful include griddy polytopes [9], ellipsoids [18], level sets [21], polytopes [6], zonotopes [13, 14], and support functions [15]. After this, the computation for the time interval $[0, \Delta]$ is *translated* to obtain the reachable states for the time interval $[i\Delta, (i+1)\Delta]$ — the states reached at time $i\Delta$ and $(i+1)\Delta$ are obtained by translating the vertices of X and those at time Δ, respectively, and then the "bloated" convex hull of all of these points is approximated by the data structure of choice. This method has been found to scalable and successful, making the automated analysis of linear dynamical systems possible.

In this paper, we take a slightly different stance on the problem of directly approximating the reachable set. Instead of trying to bound the error of a reachability computation, we view the problem as one where given an error bound ϵ, one has to compute an over-approximation of the reachable set whose Hausdorff distance from the actual set of reachable states is bounded by ϵ. This subtle change in perspective, immediately suggests some natural changes

to the basic algorithm outlined in the previous paragraph. First, the discretization of the time interval $[0, T]$ need not be in terms of equal-sized intervals. We could change interval sizes, as long as the error of approximating the reachable states within that interval can be bounded by ϵ. Second, in the "basic algorithm", the convex hull of the points of the initial set and those at time Δ is taken to be the approximation of the set of states reachable within time Δ. Instead, we view the approximation process as first approximating the *flow* in the interval $[0, \Delta]$ by a polynomial, and then taking the "polynomial tube" defined by this dynamics to be the approximation of the reachable states — when the polynomial is taken to be a linear function, then it corresponds to taking the convex hull of the points, as done in the basic algorithm. Combining these ideas, the algorithm we plan to study in this paper can be summarized as follows. Given an error bound ϵ and the degree d of the polynomials to be considered, we first find a time (hopefully, as large as possible) t such that dynamics in $[0, t]$ when approximated by degree d polynomials is within error bound ϵ of actual dynamics. The degree d polynomial approximating the actual dynamics can be found by constructing the appropriate Bernstein polynomial [20]. We approximate the set of reachable states in the interval $[0, t]$ by the degree d polynomial tube, and then repeat the process for the time interval $[t, T]$, each time *dynamically* figuring out the appropriate discretization.

Our algorithm, when compared with the basic algorithm previously studied, has both perceptible advantages and disadvantages. On first glance, the basic algorithm seems to be computationally simpler and potentially faster. For a system with linear dynamics, computing the set of states reached at time Δ involves computing (or rather approximating) matrix exponentials. This is a significant computational overhead. In the basic algorithm, this cost is minimized, as it is performed once for the first $[0, \Delta]$ interval, and for subsequent intervals it is obtained by translation rather than direct computation. In our algorithm, this must be computed afresh for each sub-interval. Moreover, in our algorithm, the intervals need to be determined dynamically, which is an additional overhead to the computation in each sub-interval. However, on the flip side, dynamically determining intervals is likely to give us larger sub-intervals in some places and therefore result in fewer intervals overall to consider. This could be a potentially significant advantage. This is because the eventual approximation of the reach set for the interval $[0, T]$ is given as the union of basic sets (represented by the chosen data structure) that are computed for each sub-interval; thus, the number of terms in the union is as large as the number of sub-intervals. Subsequent steps in verifying safety properties (or other properties of interest) involve taking intersections, and checking emptiness and membership of states in these sets. The complexity of these set-theoretic operations depends on how many terms there are in the union — this is true no matter what the chosen data structure is. Thus, the time (and memory) used in verification is directly influenced by how many steps the time interval $[0, T]$ is divided into. Note, that the "quality" of the solution computed by the basic algorithm is not better than the one computed by our algorithm, even if it uses smaller sub-intervals and more of them, because the overall quality is determined by the quality of the solution in the "worst" interval, which is then built into the bloating factor used by both algorithms.

In order to evaluate these competing claims, we implemented both the basic algorithm and our algorithm in Matlab. Observe that in both the basic algorithm and our algorithm, states reached at certain times must be computed, and then the convex hull or polynomial tube must be approximated by the data structure of choice. In our experimental evaluation, we choose to be agnostic about the relative merits of different data structures, and we make no claims about which data structure should be chosen. Therefore, we only compute the states reached at certain time steps, and not the data structure representing the reachable states. Once a data structure is chosen, the computational overhead in constructing the desired set will be the same whether the basic algorithm or our method is used (provided linear flows are used to approximate the actual flow in our method). Thus, our experimental setup is to evaluate under what conditions (types of matrices and time bound T) does unequal intervals plus associated computation costs beat uniform intervals with computation minimized by translation. We also try to understand, when it makes sense to use polynomials that are not linear to approximate the flow. Our results apply no matter what your favorite data structure is.

Our experimental results are surprising. We evaluated the two methods on both "natural" examples that have been studied before, and randomly generated matrices, and for different time intervals and error bounds. First we observed that our algorithm is scalable as it computes the points for both large matrices (we tried it on 100×100 matrices) and for many time steps (requiring thousands of iterations). Second, surprisingly, our algorithm, approximating flow by linear functions, *most of the time* outperforms the basic algorithm, sometimes by a few orders of magnitude. The gap in the performance between the two algorithms only widens considerably as the time bound T is increased. This can be explained by the fact that as the number of iterations increases the significant reduction in the number of intervals dominates the computation costs. Our algorithm not only uses significantly fewer number of intervals for the same precision (as would be expected) but the size of the *minimum* interval is also significantly larger than the size of the uniform interval chosen by the basic algorithm. This suggests that our algorithm reaps the benefits of dynamic computation of error bounds, over static determination of them. Next, we compare the potential benefits of using non-linear flows to approximate the actual flow. We consider polynomials of degree 2, as they are appropriate when considering ellipsoids as the data structure. Theoretically, the size of a dynamically determined sub-interval could be a factor of 2 larger when using polynomials of degree 2 when compared with linear functions. That, in turn, could translate to significant (exponential) savings in terms of the number of intervals. However, these theoretical possibilities were not observed in our experiments — the number of intervals for degree 2 polynomials were at most a factor of 2 smaller. This could be explained by our observation that most of the sub-intervals in the linear approximation tend to be small, and they are roughly of the same size. In such a scenario the theoretical benefits of using quadratic approximations don't translate to visible gains.

We would like to remark that the approximations that we compute depend on the machine precision, since the approximations are constructed by sampling the function at certain points, and are only as precise as that of the function values

computed for the sample points. There are other approaches for reachability analysis [23, 22, 19, 27] which rely on the decidability of the satisfiability problem for the first order theory of reals, which in contrast are algebraic techniques with infinite precision computation. However, for reachability analysis for the class of linear dynamical system, the above techniques need to first compute a polynomial approximation of the dynamics.

The rest of the paper is organized as follows. We begin with some preliminaries in Section 2. Next, in Section 3, we outline our algorithm to compute post for general dynamical systems by approximating flows using Bernstein polynomials. In Section 4, we describe the specific algorithm for linear dynamical systems. We then give details of our experimental results (Section 5) before presenting our conclusions.

2. PRELIMINARIES

Let $\mathbb{N}$ denote the set of natural numbers and let $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$ denote the set of real numbers and non-negative real numbers, respectively. Given $x \in \mathbb{R}^n$, let $(x)_i$ denote the projection of x onto the i-th component, that is, if $x = (x_1, \cdots, x_n)$, then $(x)_i = x_i$. Given a function $F : A \to \mathbb{R}^n$, let $F_i : A \to \mathbb{R}$ denote the function given by $F_i(a) = (F(a))_i$. Given a function $F : \mathbb{R}_{\geq 0} \to B$ and $[a, b] \subseteq \mathbb{R}_{\geq 0}$, let $F[a, b] : [0, b - a] \to B$ denote the function given by $F[a, b](c) = F(a + c)$.

We will use ∞-norms for measuring the distance between two vectors. Given $x, y \in \mathbb{R}^n$, let $\|x - y\|$ denote the distance between x and y in the ∞-norm, that is, $\|x - y\| = max_{1 \leq i \leq n} |(x)_i - (y)_i|$. Also, given two functions $G : A \to \mathbb{R}^n$ and $H : A \to \mathbb{R}^n$, the distance between the functions, denoted $\|G - H\|$, is given by $\|G - H\| = \inf_{x \in A} \|G(x) - H(x)\|$. Given two sets $A, B \subseteq \mathbb{R}^n$, the Hausdorff distance between the two sets, denoted $d_H(A, B)$, is defined as

$$d_H(A, B) = max(\sup_{x \in A} \inf_{y \in B} \|x - y\|, \sup_{x \in B} \inf_{y \in A} \|x - y\|)$$

A polynomial over a variable x of degree k, denoted $p(x)$, is a term of the form $a_0 + a_1 x^1 + \cdots + a_k x^k$, where $a_i \in \mathbb{N}$ for all $1 \leq i \leq k$ and $a_k \neq 0$. Given a $v \in \mathbb{R}$, let $p(v)$ denote the value obtained by substituting v for x in the expression $p(x)$ and evaluating the resulting expression. A function $P : [a, b] \to \mathbb{R}^n$, for some $a, b \in \mathbb{R}$, is a *polynomial function* if there exist polynomials, $p_1, \cdots, p_n$ over x, such that for all $v \in [a, b]$, $P_i(v) = p_i(v)$. The degree of the polynomial function P is the maximum degree of the polynomials representing it, that is, degree of P is maximum of the degrees of $p_1, \cdots, p_n$ (the degree is unique). Note that polynomial functions are continuous functions. A piecewise polynomial function is a function whose domain can be divided into finite number of intervals such that the function restricted to each of these intervals is a polynomial function. A *piecewise polynomial function* (*PPF*) is a continuous function $P : [a, b] \to \mathbb{R}^n$, where $a, b \in \mathbb{R}$, such that there exists a sequence $t_1, \cdots, t_k$ such that $a < t_1 < \cdots < t_k < b$ and $P[a, t_1], P[t_1, t_2], \cdots, P[t_k, b]$ are all polynomial functions.

3. POST COMPUTATION BY FLOW APPROXIMATION

In this section, we describe a general algorithm to approximate the flow of a dynamical system by a piecewise polynomial function of any fixed degree within any approximation error bound. The approximations are based on Bernstein Polynomials.

3.1 Bernstein Polynomial Approximations

Our general algorithm for approximating a flow within a given error bound is based on the well-known Weierstrass Approximation Theorem, which says that an arbitrary function over the reals with a compact domain can be approximated by a polynomial such that the distance between the two functions is within a given error bound.

THEOREM 1 (WEIERSTRASS). *Given a continuous function $F : \mathbb{R} \to \mathbb{R}$, a compact subset $[a, b]$ of $\mathbb{R}$ and an $\epsilon > 0$, there exists a polynomial function $P : \mathbb{R} \to \mathbb{R}$ such that*

$$|F(x) - P(x)| < \epsilon, \forall x \in [a, b].$$

We can use the polynomial function guaranteed by this theorem to obtain a polynomial approximation of the flow of a dynamical system. Application of this theorem for hybrid system verification by construction of ϵ-simulations can be found in [24].

The above theorem is an existential theorem, and by itself does not suggest a way to obtain the polynomials. However, there exist a class of polynomials called Bernstein Polynomials, which can be constructed for a given function F, and an $\epsilon > 0$, such that the distance between the function F and the corresponding polynomial is within ϵ. The approximate polynomial is constructed by evaluating the function F at some finite number of points and using these values to compute the coefficients of the polynomial. Let $F : [a, b] \to \mathbb{R}$ be a function. Then a Bernstein polynomial of degree n approximating F, denoted by $Bern_n(F)$ is given by:

$$(Bern_n(F))(x) =$$

$$\sum_{k=0}^{n} F(a + k(b - a)/n) * \binom{n}{k} *$$

$$((x - a)/(b - a))^k (1 - (x - a)/(b - a))^{n-k},$$

for all $a \leq x \leq b$.

The next two lemmas essentially show that the approximation error introduced by the polynomial approximation can be made arbitrarily small. In particular, given an $\epsilon > 0$, we can choose an n effectively such that the distance between the two functions is bounded by ϵ. Let us denote by F_{diff}, the absolute difference between the maximum and minimum values of F in its domain, i.e, $F_{diff} = \max_{x_1, x_2 \in [a,b]} |F(x_2) - F(x_1)|$. Then, we have the following from [20]:

LEMMA 1. *Let $F : [a, b] \to \mathbb{R}$ be a continuous function and $\epsilon > 0$. Let $\delta > 0$ be such that for all $x_1, x_2 \in [a, b]$, $|x_2 - x_1| \leq \delta$ implies $|F(x_2) - F(x_1)| \leq \epsilon$. Then $|F(x) - Bern_n(F)(x)| \leq \epsilon$ if $n > F_{diff}/(\epsilon\delta^2)$.*

LEMMA 2. *Let $F : [a, b] \to \mathbb{R}$ be a continuous function satisfying the Lipschitz condition $|F(x) - F(y)| < L|x - y|$ for $x, y \in [a, b]$. Then $|F(x) - Bern_n(F)(x)| < L/(2\sqrt{(}n))$.*

Note that both the lemmas give an n such that the error or distance between F and $Bern_n(F)$ is within ϵ. In particular, given an ϵ the first lemma tells us to choose an $n > F_{diff}/(\epsilon\delta^2)$, and the second lemma tells us that a choice of $n > (L/2\epsilon)^2$ would ensure that the error is within ϵ.

3.2 General Algorithm

Our aim is to approximate a flow F over a time interval $[0,T]$ by a polynomial of very low degree, such as a *linear* or a *quadratic* polynomial. Lemmas 1 and 2 give us a polynomial of a certain degree approximating the function over the interval $[0,T]$ ensuring the desired error bound. However, the degree of the polynomial can be large. Hence instead of approximating by a single polynomial of high degree, we present an algorithm which splits the interval $[0,T]$ into smaller intervals, and approximates the flow separately in each of the smaller intervals, thereby giving a piecewise continuous polynomial approximation of a fixed degree.

Consider the following dynamical system.

$$\dot{x} = f(x), x \in \mathbb{R}^n, x(0) \in X_0,$$

where X_0 is a set of initial vectors. We will assume that f is a 'nice' function (for example, Lipschitz continuous) such that it has a unique solution $\Phi : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, satisfying $d/dt(\Phi(x_0,t)) = f(\Phi(x_0,t))$ for all $x_0 \in X_0$ and $t \in \mathbb{R}_{\geq 0}$. Note that Φ is assumed to be continuous and differentiable.

Let us fix an initial vector $x_0 \in X_0$, and a time $T \in \mathbb{R}_{\geq 0}$. Let $F : [0,T] \to \mathbb{R}^n$ be the function $F(t) = \Phi(x_0,t)$ for all $0 \leq t \leq T$. We will approximate each F_i by a piecewise polynomial function P_i of degree $\leq m$ within an error bound of ϵ. Hence $\|F - P\| < \epsilon$. The general algorithm is outlined below.

Algorithm 1 Varying Time Step Algorithm

Input: $m \in \mathbb{N}, \epsilon \in \mathbb{R}_{\geq 0}, F : [0,T] \to \mathbb{R}$
Output: Sequence of polynomials

$t := 0$
while $t < T$ **do**
  Choose $0 < t_i < T$ s.t.
  $\|Bern_m(F[t, t+t_i]) - F[t, t+t_i]\| \leq \epsilon$
  Output $Bern_m(F[t, t+t_i])$
  $t := t + t_i$
end while

Starting at time $t = 0$, find a time $0 < t_1 \leq T$ such that $\|Bern_n(F_i[0,t_1]) - F_i[0,t_1]\| < \epsilon$. There always exists such a t_1, since continuity of F implies that $F_i[0,t_1]_{diff}$ can be made arbitrarily small by taking t_1 to be sufficiently small, and therefore one can satisfy the condition $F_i[0,t_1]_{diff}/\epsilon\delta^2 < m$ in Lemma 1. This reduces the problem to finding a piecewise polynomial approximation of the function $F_i[t_1,T]$, and we proceed in the same manner to compute $t_2, t_3, \cdots$. Since the function values of the Bernstein polynomial and the function it is approximating match at the end-point, the piecewise polynomial function P, in any interval $[0, \sum_{i=1}^{k} t_i]$ is continuous. To ensure that the number of iterations is finite, we need to ensure that the we make progress. This can be guaranteed by ensuring that in each step, the t_i chosen is at least Δ, for some $\Delta > 0$. Note that there always exists a Δ, which can be chosen at any step which satisfies the condition in Lemma 1. To see this, let $\gamma = m\epsilon\delta^2$, where m is the degree of the polynomials we are considering, ϵ is the desired bound on approximation error, and δ is the parameter in the definition of continuity for F_i corresponding to ϵ. Since F_i is continuous and bounded, there exists a $\Delta > 0$ such that for all $t, t' \in [0,T]$, $|t - t'| \leq \Delta$ implies $|F_i(t) - F_i(t')| \leq \gamma$. Hence choosing Δ at any step ensures that we make progress. In order to materialize the above sketch of the algorithm, we need to be able to compute F_{diff} or some upper bound on it, which ensures progress. In the next section, we present two methods to compute the t_is for the class of linear dynamical systems.

4. APPROXIMATION OF LINEAR DYNAMICAL SYSTEMS

In this section, we consider linear dynamical systems and present our algorithm in detail. Consider the following system:

$$\dot{x} = Ax, x \in \mathbb{R}^n, x(0) \in X_0,$$

where $X_0 \subseteq \mathbb{R}^n$ is a bounded convex polyhedron. The solution of the above equation is given by:

$$\Phi(x_0,t) = e^{At}x_0, x_0 \in X_0, t \in \mathbb{R}_{\geq 0}.$$

Let us define $Post_\Phi(X,[0,T]) = \{\Phi(x,t) \mid x \in X, t \in [0,T]\}$.

We consider the problem of computing an over approximation of $Post_\Phi(X_0,[0,T])$ such that the error in the approximation is within an ϵ. More precisely, we wish to find a set $\widehat{Post_\Phi}(X_0,[0,T])$ such that $\widehat{Post_\Phi}(X_0,[0,T])$ is an over approximation, that is, $Post_\Phi(X_0,[0,T]) \subseteq \widehat{Post_\Phi}(X_0,[0,T])$, and the Hausdorff distance between the two sets is bounded by ϵ, that is, $d_H(Post_\Phi(X_0,[0,T]), \widehat{Post_\Phi}(X_0,[0,T])) \leq \epsilon$.

First we show that the flow function for a linear system preserves convexity and hence it suffices to approximate only the flows starting from the vertices of X_0.

PROPOSITION 1. *Let $x = \alpha_1 x_1 + \cdots + \alpha_k x_k$ where $x_i \in \mathbb{R}^n$ and $\sum_{i=1}^{k} \alpha_k = 1$. Then $\Phi(x,t) = \alpha_1\Phi(x_1,t) + \cdots + \alpha_k\Phi(x_k,t)$.*

Let $Vertices(X_0)$ denote the set of vertices of X_0. Let us fix a time T. Given a $v \in Vertices(X_0)$, let $F_v : [0,T] \to \mathbb{R}^n$ be the function $F_v(t) = \Phi(v,t)$ for all $t \in [0,T]$. For each $v \in Vertices(X_0)$, let $\hat{F}_v$ denote a function such that $\|\hat{F}_v - F_v\| \leq \epsilon$. Let $\hat{R} = \{\alpha_1\hat{F}_{v_1}(t) + \cdots + \alpha_k\hat{F}_{v_k}(t) \mid \alpha_1 + \cdots + \alpha_k = 1, t \in [0,T]\}$. The next lemma says that the Hausdorff distance between the exact post set and $\hat{R}$ is bounded by ϵ.

LEMMA 3.

$$d_H(Post_\Phi(X_0,[0,T]), \hat{R}) \leq \epsilon.$$

The above proposition tells us that it suffices to approximate the flows starting at the vertices of the polyhedron. More precisely, if we approximate the flows at the vertices within an error bound of ϵ in an interval $[0,T]$, then at any time $t \in [0,T]$, the Hausdorff distance between the actual and approximate sets is with ϵ.

In the literature, various methods have been proposed to compute $\widehat{Post_\Phi}$. These methods can be seen as consisting of the following two steps.

- Depending on the ϵ, a time step Δ is chosen. Let $V_0 = Vertices(X_0)$ and $V_i = Post_\Phi(V_0,[i\Delta, i\Delta])$ for $i > 0$ be the set of points reached from the vertices of X_0 after i time steps of size Δ. First $V_1 = Post_\Phi(V_0,[\Delta,\Delta])$ is computed. Then the convex hull C_0 of V_0 and V_1 is bloated by ϵ, and the resulting set is enclosed by a data structure of a certain form to obtain an overapproximation C_0' of $Post_\Phi(X_0,[0,\Delta])$.

- Similarly, to obtain an overapproximation of $Post_\Phi(X_0, [i\Delta, (i+1)\Delta])$, the convex hull C_i of V_i and V_{i+1} is bloated by ϵ, and enclosed in a data structure. However, instead of computing V_i directly from V_0, it is computed iteratively from V_{i-1}, that is, V_i is computed from V_{i-1} by a linear transformation using the matrix e^{Δ}.

We think of the above algorithm as first computing an approximation of the flow function, which is the piecewise linear function obtained by joining the corresponding points in V_is, and then enclosing the reach set given by the approximated flow function by a set of a certain form. The above algorithms compute a piecewise linear approximation of the flow function by dividing the interval $[0, T]$ into equal intervals of size Δ. Our main contribution is a novel algorithm for computing an approximation of the flow function, which does not divide the interval uniformly, but dynamically computes the next time step. The obvious advantage is the reduction in the number of times steps, since a time step chosen by the dynamic algorithm is always larger than the constant time step Δ chosen by the uniform time step algorithm. This in turn implies that the size of the final representation of the post set would be smaller, and the size plays a crucial role in further analysis. However, there is a overhead involved with the dynamic algorithm, which is in computing the set of vertices V_is at various time points, since these V_is can no more be computed iteratively by multiplication using a fixed matrix. Since the timesteps Δ keep changing, there does not exist a fixed matrix e^{Δ} which can be used to obtain V_i from V_{i-1} for every i. So the new algorithm involves computing a new matrix exponential e^{Δ_i} at each step. However, as we will see in the next section, our experimental evaluations show that the overhead introduced due to computation of a new matrix exponential at each time step becomes negligible due to the huge decrease in the number of steps. In other words, the cost of doing the large number of matrix multiplications in the constant time step algorithm is greater than the cost of computing new matrix exponentials followed by matrix multiplications for a small number of timesteps in the dynamic algorithm.

To compute the approximation of the flow function for a linear dynamical system, we instantiate Algorithm 1 to obtain an effective algorithm for linear dynamical systems. As mentioned in the discussion of Algorithm 1, we need to present a method to compute the t_is in each step such that progress is ensured. Next we present two methods for computing t_is.

4.1 Computing t_i: First method

In this section we use Lemma 2 to compute the bound t_i. Let us fix an $x_0 \in \mathbb{R}^n$ and an $n \times n$ matrix A. Let $F : [0, T] \to \mathbb{R}^n$ be the function $F(t) = e^{At}x_0$. First we show that F satisfies the Lipschitz condition, and the Lipschitz constant can be bounded by a function of T.

LEMMA 4. *Let $F : [0, T] \to \mathbb{R}^n$ be as defined above. Then for each $1 \le i \le n$, F_i is Lipschitz continuous with the Lipschitz constant $L = \|A\|e^{\|A\|T}\|x_0\|$.*

The next lemma gives us a lower bound on the time step t_i that can be chosen at each step such that the approximate polynomial is within distance ϵ from the original function.

LEMMA 5. *Let $F : [0, T] \to \mathbb{R}^n$ be as defined above. For $t_1 = \log_e(2\sqrt{m}\epsilon/\|A\|\|x_0\|)/\|A\|$, $\|F[0, t_1] - B_m(F[0, t_1])\| \le \epsilon$.*

Using the t_i in the definition of Lemma 5 is desirable since it gives a closed form expression for computing the t_i. However, the problem with the above expression is that the expression being computed might not result in a positive number in which case we are in trouble. Next we present another method for computing lower bound for t_i, which always gives a positive answer.

4.2 Computing t_i: Second method

In this section, we use Lemma 1 to compute a bound on the t_is.

LEMMA 6. *Let $F : [0, T] \to \mathbb{R}^n$ be as defined above. Let t_1 be such that $e^{3\|A\|t_1}t_1 < m\epsilon^3/\|A\|^3\|x_0\|^3$. Then we have that $\|B_m(F[0, t_1]) - F[0, t_1]\| \le \epsilon$.*

There always exists a positive real number $t_1 \le t$ satisfying the inequality $e^{at_1}t_1 < b$ where $a = 3\|A\|$ and $b = m\epsilon^3/\|A\|^3\|x_0\|^3$, since the function $e^{ax}x \to 0$ as $x \to 0$. Computing a t_1 such that $e^{at_1}t_1 = b$ might not be possible, instead one can obtain an upper bound on this value. For example, we know that $t_1 \le t$. Hence we can consider $t_1 = b/e^{at}$. We use the following alternative bound. If $at_1 < 1$, then we can upper bound e^{at_1} by $1/(1-at_1)$. This gives us a bound $t_1 < b/(1+ab)$. Hence $t_1 < \min\{b/(1+ab), 1/2a_1\}$ is a positive bound for t_1.

The algorithm for computing a piecewise polynomial approximation of a linear dynamical system is given in Algorithm 2.

Algorithm 2 Post Computation Algorithm for Linear Dynamical Systems

Input: $m \in \mathbb{N}, \epsilon \in \mathbb{R}_{\ge 0}, V_0 \subseteq_{Fin} \mathbb{R}^n, A \in \mathbb{R}^{n\times n}, T > 0$
Output: Sequence of a set of n polynomials

Let $F^x : [0, T] \to \mathbb{R}^n$ be $F^x(t) = e^{At}x$
for all $v \in V_0$ **do**
  $t := 0$
  $x := v$
  while $t < T$ **do**
    Choose $\tau_1 > 0$ s.t
    $e^{3\|A\|\tau_1}\tau_1 < m\epsilon^3/\|A\|^3\|x\|^3$
    Let $\tau_2 = \log_e(2\sqrt{m}\epsilon/\|A\|\|x\|)/\|A\|$
    Let $t_i = \max \tau_1, \tau_2$
    Output $Bern_m(F_j^v[t, t + t_i])$, for each $1 \le j \le n$
    $t := t + t_i$
    $x := e^{At}v$
  end while
end for

4.3 Termination of the Algorithm

In each step, we take as the next time step the maximum of the values obtained by methods in Lemma 5 and Lemma 6. This time step is always going to be positive, since Method 2 always gives a positive answer. Next we show that the time step we choose in any iteration has a positive lower bound. Hence, the algorithm always terminates.

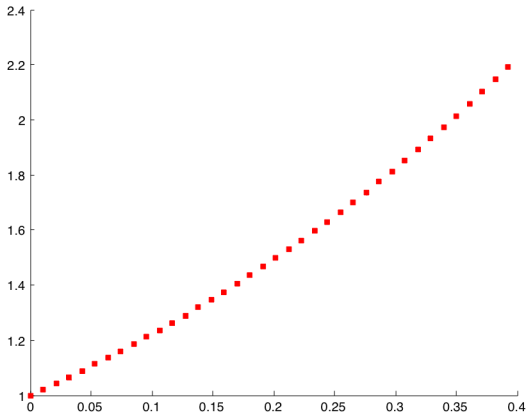

Figure 1: Constant time step Algorithm

Assume that in each step of method 2, we choose $t_i = \min\{b/(1+ab), 1/2a_1\}$.

$$b/(1+ab) = \frac{(m\epsilon^3/(\|A\|^3\|x_i\|^3))}{(1+(am\epsilon^3/(\|A\|^3\|x_i\|^3)))}$$

$$= \frac{(m\epsilon^3/(\|A\|^3))}{(\|x_i\|^3+(am\epsilon^3/(\|A\|^3)))}$$

$$\geq \frac{(m\epsilon^3/(\|A\|^3))}{(e^{aT}\|x_0\|^3+(am\epsilon^3/(\|A\|^3)))}$$

Therefore, the time steps t_i are lower bounded by a positive number.

Figure 1 and Figure 4.5 illustrate the difference between the constant step and varying step algorithms. For each algorithm, the points (t, y) are plotted, where t ranges over the times $\sum_{j=0}^{i} t_i$, where $t_1, t_2, \cdots$ is the sequence of time steps chosen by the algorithm, and y is given by $e^{At}x$. Observe that in the case of timestep varying algorithm, initially larger steps are chosen, and when the time approaches close to T, the timesteps taken by the constant time step algorithm and the varying time step algorithm become identical. Notice that the sampling rate of the varying time step algorithm depends on the value of the derivative of the function at various points, where as the constant time step algorithm makes no such distinction.

4.4 Function Evaluation Errors

Observe that we need to compute the value of the functions at several time points. In practice, these values can seldom be computed exactly. One can only hope to compute approximations within arbitrary error bounds. Let the function values be computed with in an error bound of γ. The approximate values of the samples can be thought of as the samples of a new function $\hat{F}$ such that $\|\hat{F} - F\| \leq \gamma$. Given an ϵ, let P be the piecewise polynomial approximation of F constructed using our algorithm, and let γ be a bound on the error in evaluating the function F. Then the error $\|P - F\| \leq \|P - \hat{F}\| + \|\hat{F} - F\| \leq \epsilon + \gamma$. Hence, by reducing γ, we can get as precise an approximation as desired.

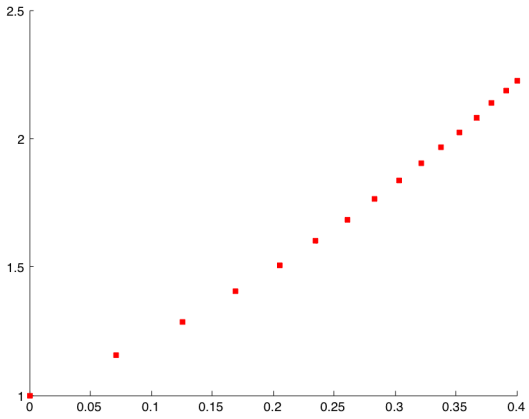

Figure 2: Varying time step Algorithm

4.5 Comparison with other polynomial approximations

In this paper, we considered Bernstein polynomials to approximate an arbitrary function by a polynomial function. Another popular technique to obtain polynomial approximations is to use Taylor's series expansion of the function and truncate the infinite sum after some points to obtain a polynomial. There are a few caveats in using Taylor's approximation in general. First it assumes that the function is smooth, and the derivatives can be computed. Secondly, it does not give a closed form expression for m, the number of terms in the Taylor's expansion that should be considered to obtain an ϵ bound on the approximation error.

5. EXPERIMENTAL EVALUATION

In this section, we explain our experimental set up for evaluating the performance of our algorithm and comparison with other methods. We implemented our algorithm is Matlab 7.4.0, and the experiments were conducted on Mac OS X Version 10.4.11, with a 2.16 GHz Intel Core 2 Duo processor and 1GB SDRAM. We performed our experiments on linear and quadratic approximations. We will report and explain our results for both the approximations in the following sections.

5.1 Linear approximation

Our experimental evaluation of the two algorithms chooses to be agnostic of the relative benefits of different data structures, and attempts to highlight the relative advantages of each algorithm, independent of the chosen data structure. No matter what the chosen data structure, the algorithms require computing the states reached at certain time steps. Once these points are computed, the data structure approximating a convex hull of the points needs to be computed. Thus, in our experimental evaluation we only compared the computational costs of finding the states reached at the required times using the two algorithms. In the basic algorithm, the interval size is fixed to be Δ, and then the states reached at time Δ are computed by multiplying initial set of states with matrix exponential e^{Δ}, but for subsequent times $i\Delta$ they are computed by translation that involves only multiplication with e^{Δ} which is evaluated only once.

Matrix	ϵ	T	n	m	RT_V	RT_C	t_{max}	t_{min}	Constant
$2DR$	4.93E-03	1	3.34E+02	8.53E+02	1.80E-01	2.79E-02	5.54E-03	1.78E-03	1.17E-03
$2DR$	6.78E-02	2	1.80E+01	1.60E+03	1.77E-02	1.15E-01	1.96E-01	7.81E-02	1.25E-03
$2DR$	3.60E-01	5	1.20E+01	4.27E+03	1.68E-02	4.99E-01	4.83E-01	4.83E-01	1.17E-03
$2DR$	1.85E+00	10	9.20E+01	8.47E+03	5.64E-02	1.51E+00	4.60E-01	9.42E-03	1.18E-03
$5DR$	4.92E-01	1	1.00E+01	5.65E+02	1.49E-02	4.69E-02	1.23E-01	1.07E-01	1.77E-03
$5DR$	3.14E+00	2	1.50E+01	1.14E+03	1.96E-02	1.15E-01	1.44E-01	1.44E-01	1.76E-03
$5DR$	2.11E+02	5	3.30E+01	2.99E+03	2.95E-02	4.97E-01	1.57E-01	1.57E-01	1.67E-03
$5DR$	3.04E+05	10	6.20E+01	6.15E+03	4.81E-02	1.62E+00	1.65E-01	1.65E-01	1.63E-03
$100DR$	2.01E-02	1	3.70E+02	9.16E+02	4.81E+00	5.33E-01	2.76E-03	2.61E-03	1.09E-03
$100DR$	2.56E-02	2	3.17E+02	1.84E+03	4.42E+00	2.93E+00	6.57E-03	5.93E-03	1.09E-03
$100DR$	6.13E-02	5	8.20E+01	4.59E+03	1.24E+00	2.14E+01	7.69E-02	4.97E-02	1.09E-03
$100DR$	2.65E-01	10	1.30E+01	9.17E+03	4.05E-01	9.23E+01	8.65E-01	8.22E-01	1.09E-03

Figure 3: Random matrices

Matrix	ϵ	T	n	m	RT_V	RT_C	t_{max}	t_{min}	Constant
$Z2$	1E-01	1	1.30E+02	7.47E+02	7.43E-02	2.55E-02	1.13E-02	5.96E-03	1.34E-03
$Z2$	1E-01	2	1.86E+02	6.69E+03	1.04E-01	9.95E-01	2.69E-02	5.96E-03	2.99E-04
$Z2$	1E-01	3	2.42E+02	4.49E+04	1.33E-01	2.78E+01	2.69E-02	5.96E-03	6.68E-05
$Z5$	1E-01	1	1.04E+02	5.61E+02	6.22E-02	4.00E-02	1.34E-02	7.91E-03	1.78E-03
$Z5$	1E-01	2	1.52E+02	5.02E+03	8.75E-02	1.04E+00	3.82E-02	7.91E-03	3.98E-04
$Z5$	1E-01	3	1.87E+02	3.38E+04	1.07E-01	4.33E+01	3.82E-02	7.91E-03	8.89E-05
Nav	1E+00	1	7.00E+00	5.00E+00	1.47E-02	2.30E-02	1.92E-01	1.92E-01	1.92E-01
Nav	1E+00	2	1.20E+01	9.30E+01	1.65E-02	2.30E-02	1.92E-01	1.92E-01	2.14E-02
Nav	1E+00	3	1.70E+01	6.37E+03	1.95E-02	1.45E+00	1.92E-01	1.92E-01	4.71E-04

Figure 4: Standard Examples for total time $T = 1, 2, 3$

In contrast, in our algorithm, the states reached at each of the designated time steps is computed from scratch using matrix exponentials, and the size of the next interval is found dynamically. Recall from Section 4, that when the initial set is a convex polyhedron, reach set computation involves computing this approximated flow for each vertex. Thus, in our experimental evaluation, we start from a single point, as this will faithfully reflect the costs of starting from a polyhedron. Finally, the error bound chosen for the varying time step algorithm was the one guaranteed by the fixed time step algorithm.

To determine feasibility of the algorithms, we first ran them on some randomly generated matrices. The entries of the matrices were random values in the interval $[-1, 1]$. The results of our experiments are shown in Figure 3. The rows labelled $2DR$ report results for 2×2 matrices, those labelled $5DR$ for 5×5 matrices, and finally those labelled $100DR$ for 100×100 matrices. The columns reported in the table are as follows: ϵ gives the error bound; T gives the time bound chosen for the experiment; m and n are the number of sub-intervals used by the constant timestep algorithm and varying timestep algorithms, respectively; RT_C and RT_V are the running times of the constant timestep and varying timestep algorithms, respectively; t_{max} and t_{min} are the largest and smallest time intervals considered by the varying timestep algorithm; "Constant" is the size of the interval used by the constant timestep algorithm. For these matrices, we chose a time step for the constant timestep algorithm to be of the order of 10^{-3}, and used the resulting error bound as ϵ. The results show that the varying timestep algorithm is scalable and has a running time comparable to the constant timestep algorithm; in many cases the varying timestep algorithm is faster by 2 orders of magnitude. The number of sub-intervals used by the varying timestep algorithm (n) is always less than that used by the constant timestep method (m) by either a magnitude or two orders of magnitude. The other surprising observation is that the smallest time interval used by the varying timestep method is in all cases larger than the interval used by the constant timestep algorithm.

We also experimented on benchmark examples considered in [13]. Nav is the navigation benchmark first suggested in [12], while $Z2$ and $Z5$ are the 2 dimensional and 5 dimensional examples from [13]. The matrices describing their dynamics is as follows.

$$Nav = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1.2 & 0.1 \\ 0 & 0 & 0.1 & -1.2 \end{bmatrix}, Z2 = \begin{bmatrix} -0.1 & -0.4 \\ 0.4 & -0.1 \end{bmatrix}$$

$$Z5 = \begin{bmatrix} -0.1 & -0.4 & 0 & 0 & 0 \\ 0.4 & -0.1 & 0 & 0 & 0 \\ 0 & 0 & -0.3 & 0.1 & 0 \\ 0 & 0 & -0.1 & -0.3 & 0 \\ 0 & 0 & 0 & 0 & -0.2 \end{bmatrix}$$

We tried to study the effect of increasing the time bound T on the running time of these algorithms and so we considered $T = 1, 2, 3$. Figure 4 shows our results for these benchmark examples and varying time. It shows that as T increases, the varying timestep algorithm's relative performance improves both in terms of the number of sub-intervals considered and

Matrix	t_{min}	t_{max}	RT_V	n	t^Q_{min}	t^Q_{max}	RT_Q	n_Q
$Z2$	5.96E-03	1.13E-02	9.88E-02	1.30E+02	7.21E-03	2.20E-02	4.74E-02	6.70E+01
$Z2$	5.96E-03	2.69E-02	1.28E-01	1.86E+02	1.26E-05	5.10E-02	5.98E-02	9.60E+01
$Z2$	1.00E-03	2.69E-02	1.60E-01	2.42E+02	1.18E-02	5.10E-02	7.49E-02	1.24E+02
$Z5$	2.48E-03	1.34E-02	1.21E-01	1.04E+02	1.56E-02	2.55E-02	4.10E-02	5.30E+01
$Z5$	1.60E-03	3.82E-02	1.12E-01	1.52E+02	1.56E-02	6.45E-02	5.17E-02	7.80E+01
$Z5$	7.91E-03	3.82E-02	1.31E-01	1.87E+02	1.56E-02	7.02E-02	6.25E-02	9.70E+01
Nav	1.92E-01	1.92E-01	3.54E-02	7.00E+00	1.92E-01	1.92E-01	1.10E-02	7.00E+00
Nav	1.92E-01	1.92E-01	3.84E-02	1.20E+01	1.92E-01	1.92E-01	1.44E-02	1.20E+01
Nav	1.92E-01	1.92E-01	4.09E-02	1.70E+01	1.92E-01	1.92E-01	1.57E-02	1.70E+01

Figure 5: Comparison of Quadratic and Linear Approximations

the running time, with the gap increasing to as much two orders of magnitude.

5.2 Quadratic approximation

We also implemented the varying time step algorithm for approximation by piecewise quadratic approximation. Theoretically, a single timestep of the quadratic approximation could be twice as much as that of the linear approximation. Interestingly, this could lead to a huge reduction in the number of total time steps. Consider an exponentially growing function, for which the time steps chosen by the linear approximation decrease by a constant factor in consecutive time steps. For example, consider the following sequence of timesteps $1, 1/2, 1/2^2, \cdots, 1/2^k$. A doubling of the time step in the quadratic approximation could lead to the skipping of k timesteps in the above example. However, we did not observe this phenomenon in our experiments which are tabulated in Figure 5. The columns t_{min}, t_{max}, RT_V, and n report the smallest time interval, largest time interval, running time, and number of intervals when using a linear approximations; the columns with superscript or subscript Q refer to the same quantities for the quadratic approximation. The improvement for the quadratic approximation was not as dramatic as we hoped it might, and in the best case was better by a factor of 2.

6. CONCLUSIONS

We presented a new algorithm for approximating the set of states reachable within time bound T in a linear dynamical system to within an arbitrary error bound ϵ. The main innovations in our algorithm over previous approaches is that the interval $[0, T]$ is dynamically subdivided into unequal sized intervals and then the flow in each interval is approximated by a polynomial of fixed degree. Our experimental evaluation of our algorithm reveals that the approach is scalable to high dimensional system, and is both faster and yields fewer sub-intervals than previous approaches that considered statically dividing the interval $[0, T]$ into equal sized sub-intervals.

7. ACKNOWLEDGEMENTS

We would like to thank Geir Dullerud for pointing us to the use of Bernstein Polynomials for approximations, and Parasara Sridhar Duggirala and Sayan Mitra for various helpful discussions. This research was support by grants NSF CCF 0448178, NSF CCF 1016989, and NSF CNS 1016791.

8. REFERENCES

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[2] R. Alur, T. Dang, and F. Ivancic. Counter-Example Guided Predicate Abstraction of Hybrid Systems. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 208–223, 2003.

[3] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[4] E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43(7):451–476, 2007.

[5] A. Chutinan and B.H. Krogh. Infinite state transition system verification using approximate quotient transition systems. *IEEE Transactions on Automatic Control*, 46:1401–1410, 2001.

[6] A. Chutinan and B.H. Krogh. Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control*, 48(1):64–75, 2003.

[7] E.M. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems. *International Journal on Foundations of Computer Science*, 14(4):583–604, 2003.

[8] E.M. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Verification of Hybrid Systems Based on Counterexmple-Guided Abstraction Refinement. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 192–207, 2003.

[9] T. Dang and O. Maler. Reachability analysis via face lifting. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 96–109, 1998.

[10] T. Dang, O. Maler, and R. Testylier. Accurate hybridization of nonlinear systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 11–20, 2010.

[11] A. Fehnker, E.M. Clarke, S. Jha, and B. Krogh. Refining Abstractions of Hybrid Systems using Counterexample Fragments. In *Proceedings of the*

International Conference on Hybrid Systems: Computation and Control, pages 242–257, 2005.

[12] A. Fehnker and F. Ivancic. Benchmarks for hybrid systems verification. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 326–341, 2004.

[13] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 291–305, 2005.

[14] A. Girard and C.L. Guernic. Zonotope/Hyperplane intersecion for hybrid systems reachability analysis. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 215–228, 2008.

[15] C.L. Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *Proceedings of the International Conference on Computer Aided Verification*, pages 540–554, 2009.

[16] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the ACM Symposium on Theory of Computation*, pages 373–382, 1995.

[17] S.K. Jha, B.H. Krogh, J.E. Weimer, and E.M. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 287–300, 2007.

[18] A.B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 202–214, 2000.

[19] G. Lafferriere, G.J. Pappas, and S. Sastry. O-minimal Hybrid Systems. *Mathematics of Control, Signals, and Systems*, 13(1):1–21, 2000.

[20] G.G. Lorentz. *Bernstein Polynomials*. University of Toronto Press, 1953.

[21] I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 310–323, 2000.

[22] Venkatesh Mysore, Carla Piazza, and Bud Mishra. Algorithmic Algebraic Model Checking II: Decidability of Semi-algebraic Model Checking and Its Applications to Systems Biology. In *atva*, pages 217–233, 2005.

[23] Carla Piazza, Marco Antoniotti, Venkatesh Mysore, Alberto Policriti, Franz Winkler, and Bud Mishra. Algorithmic Algebraic Model Checking I: Challenges from Systems Biology. In *Proceedings of the International Conference on Computer Aided Verification*, pages 5–19, 2005.

[24] P. Prabhakar, V. Vladimerou, M. Viswanathan, and G.E. Dullerud. Verifying tolerant systems using polynomial approximations. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 181–190, 2009.

[25] A. Puri, V.S. Borkar, and P. Varaiya. ϵ-approximation of differential inclusions. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 362–376, 1995.

[26] M. Segelken. Abstraction and Counterexample-guided Construction of Omega-Automata for Model Checking of Step-discrete linear Hybrid Models. In *Proceedings of the International Conference on Computer Aided Verification*, pages 433–448, 2007.

[27] V. Vladimerou, P. Prabhakar, M. Viswanathan, and G.E. Dullerud. STORMED hybrid systems. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, pages 136–147, 2008.

9. APPENDIX

9.1 Proofs of the upper bounds

9.1.1 Proof of Lemma 4

PROOF. First let us recall the following identity. Given $n \times n$ matrices X and Y,

$$\|e^{X+Y} - e^X\| \leq \|Y\| e^{\|X\|} e^{\|Y\|}.$$

W.l.o.g assume $t_2 > t_1$.

$$\frac{\|e^{At_2}x_0 - e^{At_1}x_0\|}{\|t_2 - t_1\|}$$

$$= \frac{\|e^{At_1 + A(t_2-t_1)}x_0 - e^{At_1}x_0\|}{|t_2 - t_1|}$$

$$\leq \frac{\|A(t_2-t_1)\| e^{\|At_1\|} e^{\|A(t_2-t_1)\|}\|x_0\|}{|t_2 - t_1|}$$

$$= \frac{\|A\||t_2 - t_1| e^{\|A\||t_1|} e^{\|A\||t_2-t_1|}\|x_0\|}{|t_2 - t_1|}$$

$$= \|A\| e^{\|A\|e(|t_1|+|t_2-t_1|)}\|x_0\|$$

$$= \|A\| e^{\|A\|T}\|x_0\|$$

$$L \geq \max_{t_1,t_2} \frac{\|e^{At_2}x_0 - e^{At_1}x_0\|}{\|t_2 - t_1\|}$$

$$\geq \|A\| e^{\|A\|T}\|x_0\|$$

□

9.1.2 Proof of Lemma 5

PROOF. The Lipschitz constant L for the function $F[0, t_1]$ is given by $L = \|A\| e^{\|A\|t_1}\|x_0\|$ from Lemma 4.

$\|F[0, t_1] - B_m(F[0, t_1])\| < L/(2\sqrt{m})$ from Lemma 2.

$L/(2\sqrt{m}) \leq \epsilon$ implies $\|A\| e^{\|A\|t_1}\|x_0\| \leq 2\sqrt{m}\epsilon$.

Hence for $t_1 \leq \log_e(2\sqrt{m}\epsilon/\|A\|\|x_0\|)/\|A\|$,

$\|F[0, t_1] - B_n(F[0, t_1])\| \leq \epsilon$. □

9.1.3 Proof of Lemma 6

PROOF. From Lemma 1, we have that $\|B_n(F[0,t_1]) - F[0,t_1]\| \leq \epsilon$ if $n > F_{diff}/\epsilon\delta^2$.

We will find bounds on the values of F_{diff} and δ as a function of t_1.

$$F_{diff} = \max_{x,y\in[0,t_1]} \|F(x) - F(y)\|$$

$$= \max_{x,y\in[0,t_1]} \|e^{Ax}x_0 - e^{Ay}x_0\| \leq \|A\|e^{\|A\|t_1}\|x_0\|t_1$$

(See proof of Lemma 5.)

Next we need to find a lower bound on δ such that

$$\forall x, y \in [0,t_1], |x-y| \leq \delta \implies \|F(x) - F(y)\| \leq \epsilon.$$

Or equivalently

$$\max_{x,y\in[0,t_1],|x-y|\leq\delta} \|F(x) - F(y)\| \leq \epsilon.$$

However,

$$\max_{x,y\in[0,t_1],|x-y|\leq\delta} \|F(x) - F(y)\| \leq \|A\|e^{\|A\|t_1}\|x_0\|\delta$$

(again from the proof of Lemma 5).

Hence it suffices to choose a δ which ensures

$$\|A\|e^{\|A\|t_1}\|x_0\|\delta \leq \epsilon.$$

Hence we can choose

$$\delta = \epsilon/(\|A\|e^{\|A\|t_1}\|x_0\|).$$

We want to choose a t_1 so as to satisfy $m > F_{diff}/\epsilon\delta^2$. It suffices to satisfy

$$m > \frac{\|A\|e^{\|A\|t_1}\|x_0\|t_1)}{(\epsilon(\epsilon/(\|A\|e^{\|A\|t_1}\|x_0\|))^2)}.$$

Or,

$$m\epsilon^3 > \|A\|^3 e^{3\|A\|t_1}\|x_0\|^3 t_1.$$

For t_1 such that

$$e^{3\|A\|t_1}t_1 < n\epsilon^3/\|A\|^3\|x_0\|^3,$$

we have $\|F(x) - B_m(F(x))\| \leq \epsilon$ for all $0 \leq x \leq t_1$. □

9.1.4 Proof of Lemma 3

PROOF. Given $x \in Post_\Phi(X_0,[0,T])$ we will find an $x' \in \hat{R}$ such that $\|x - x'\| \leq \epsilon$ and vice versa.

Let $x \in Post_\Phi(X_0,[0,T])$.

Then $x = e^{At}x_0$ for some $x_0 \in X_0$ and $t \in T$.

Let $Vertices(X_0) = \{v_1, \cdots, v_k\}$.

Since X_0 is a bounded convex polyhedron,

$x_0 = \alpha_1 v_1 + \cdots + \alpha_k v_k$,

for some $\alpha_1 + \cdots + \alpha_k = 1$.

Then $x = e^{At}x_0 = e^{At}(\alpha_1 v_1 + \cdots + \alpha_k v_k)$

$= (\alpha_1 e^{At}v_1 + \cdots + \alpha_k e^{At}v_k)$.

Let $x' = \alpha_1\hat{F}_{v_1}(t) + \cdots + \alpha_k\hat{F}_{v_k}(t)$.

Then $|x - x'| =$

$|(\alpha_1 e^{At}v_1 + \cdots + \alpha_k e^{At}v_k) - (\alpha_1\hat{F}_{v_1}(t) + \cdots + \alpha_k\hat{F}_{v_k}(t))|$

$\leq \alpha_1|e^{At}v_1 - \hat{F}_{v_1}(t)| + \cdots + \alpha_k|e^{At}v_k - hatF_{v_k}(t)|$

$\leq \alpha_1\epsilon + \cdots + \alpha_k\epsilon = \epsilon$.

Similarly given an $x \in \hat{R}$, we can find a $x' \in Post_\Phi(X_0,[0,T])$ such that $|x - x'| \leq \epsilon$. □