Verifying Linear Systems with over 10000 Dimensions (Benchmark Result)

Stanley Bak[1] and Parasara Sridhar Duggirala[2]

[1] Air Force Research Laboratory
[2] University of Connecticut

Abstract

We evaluate a recent simulation-based reachability method on a set of high-dimensional linear systems benchmarks taken from model order reduction. The method we used is simulation based, and was previously shown to have promise in terms of scalability. The method can scale to systems with over 10000 dimensions with several hours of computation time, which is an encouraging result.

1 Introduction

Reachability analysis attempts to compute, given a set of initial states, which states the system can enter. Safety verification can then be done by performing intersections between the unsafe states and reachable set. In this paper, we examine reachability for affine, time-invariant systems, with the differential equations $\dot{x} = Ax + b$. We only consider the continuous post operation, that is, we do not evaluate guards or other features of a hybrid automaton [**?**]. Nonetheless, note that continuous post is a core component of many hybrid systems reachability tools.

Typically, the reachable set tools store states in data structures such as polyhedra [**?**], zonotopes [**?**], support functions [**?**], or Taylor models [**?**]. Recently, a data structure and associated reachability method was proposed which stores states in a data structure called a generalized star set [**?**]. The reachability method we evaluate is based on this technique, which essentially uses simulations and superposition to reason over which states are reachable.

The implementation we used only reasons about the states reachable at discrete time steps. Unlike other reachability methods, it does not reason about states between time steps. However, it is capable of generating counter-example traces whenever the unsafe states are deemed reachable. We call this the *simulation-equivalent reachable set*, since it consist of all the states visited by every fixed-step simulation.

We perform an evaluation of a set of nine benchmarks which consist of large-scale linear systems [**?**]. These benchmarks were taken from "diverse fields such as civil engineering and robotics." The difficulty of these benchmarks was rated as low to challenge, and they range from tens to over ten thousand dimensions. Using these benchmarks, we evaluate two approaches which can be used to compute a generalized star set's basis matrix, which is a key component when computing its reachability. This is discussed in the next section.

2 Computing a Star's Basis Matrix

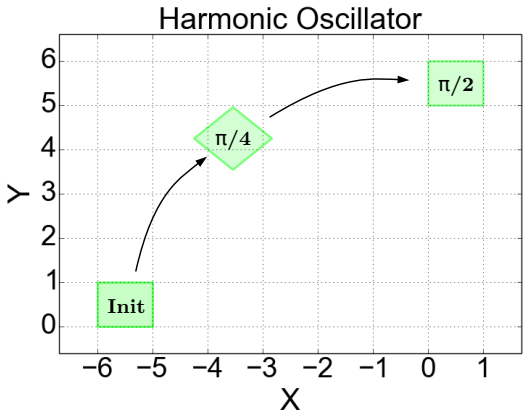
Here, we discuss the computation of the basis matrix of a generalized star set (or simply star). We do not review the full reachability computation with this approach, which is available in earlier work [**?**].

The generalized star set approach for reachability computation takes advantage of the superposition principle of linear systems in order to compute the reachable set. Basically, at each instance in time, if one knows how each orthonormal unit vector in the standard basis has evolved since the initial time, one can reconstruct the current state reachable from an arbitrary point by taking linear combinations of the state's reached by the unit vector points. For example, say in a 2-d space the initial point $\binom{1}{0}$ after some time t evolves under some linear differential equations $\dot{x} = Ax$ and reaches point $\binom{a}{b}$. The reachable state of $\binom{2}{0}$ then could be easily computed by taking twice the state that $\binom{1}{0}$ reached, which is $\binom{2a}{2b}$. Knowing how $\binom{1}{0}$ and $\binom{0}{1}$ evolve is sufficient to be able to reconstruct how any initial point evolves at time t. If the dynamics are affine, $\dot{x} = Ax + c$, then one more simulation is required from the origin. If we are interested in how a set of states can evolve at time t, one can construct a linear program (LP) which encodes the constraints on the initial conditions, how each unit vector has evolved at time t. Linear conditions can also be added to encode unsafe state conditions, and the LP will then be feasible if and only if an unsafe state is reachable from some initial state.

At each time step, the value of t changes, and so 'how each unit vector has evolved at time t' will change. There are n initial unit vectors, and each one reaches an n-dimensional point, so these can be combined to form what is called the star's *basis matrix*. The computation of this basis matrix at each time step dominates the runtime of the safety verification method. On the proposed benchmarks, we will evaluate two methods to compute the basis matrix. First, however, we present a more complete example of the star-based reachability approach.

Harmonic Oscillator A 2-d harmonic oscillator has the dynamics $\dot{x} = y$, $\dot{y} = -x$. We use initial states $x = [-6, -5]$, $y = [0, 1]$. Simulations of this system rotate clockwise around the origin. The simulation-equivalent reachable sets, and the associated LP formulation at time step $\frac{\pi}{4}$ is shown in Figure **??**. If there was a linear unsafe error condition, it could be added to this LP. The LP would then be feasible if, and only if, unsafe states were reachable. The basis matrix in this instance is the red encircled values in the LP formulation in the figure. These get updated at each time step, and the rest of the constraints remain the same. The initial conditions are encoded in rows 3-6. The basis matrix at time 0 is $\left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)$. After $\frac{\pi}{4}$ time, the point which started at state $\binom{1}{0}$ evolves to $\binom{0.707}{-0.707}$ and the point which states at $\binom{0}{1}$ goes to $\binom{0.707}{0.707}$. The basis matrix at time $\frac{\pi}{4}$ is therefore $\left(\begin{smallmatrix}0.707 & 0.707\\ -0.707 & 0.707\end{smallmatrix}\right)$, as shown in the Figure. At the next time step, time $\frac{\pi}{2}$, the basis matrix would be updated to $\left(\begin{smallmatrix}0 & 1\\ -1 & 0\end{smallmatrix}\right)$. Error states could be encoded by adding additional rows to the LP constraints imposing a linear condition on x_1 and x_2. If an error state is reachable, the LP would give a concrete assignment to x_1, x_2, α_1, and α_2. One concrete error trace, then, would start at point (α_1, α_2), and evolve to the unsafe point (x_1, x_2).

As mentioned before, the computation of this basis matrix at each time step dominates the runtime of the safety verification method, and so it is important to optimize this computation. In the original work [**?**], the computation of this basis matrix was done using simulations. However, notice that, since the solution to a linear system $\dot{x} = Ax$ is $x(t) = e^{At}x(0)$, the basis matrix can also be computed using the matrix exponential. In particular the basis matrix at time t is equal to e^{At}. Further, rather than recomputing the basis matrix at each step, we notice that $e^{A2t} = e^{At} \times e^{At}$. Thus, a single computation of the matrix exponential can be performed at the desired time step, and then the resultant matrix can be multiplied using standard matrix



$$\begin{pmatrix} -1 & 0 & 0.707 & 0.707 \\ 0 & -1 & -0.707 & 0.707 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ \alpha_1 \\ \alpha_2 \end{pmatrix} \begin{matrix} = \\ = \\ \leq \\ \leq \\ \leq \\ \leq \end{matrix} \begin{pmatrix} 0 \\ 0 \\ 6 \\ -5 \\ 0 \\ 1 \end{pmatrix}$$

Figure 1: Plot of states reached by any fixed-step simulation (simulation-equivalent reachable set) in the harmonic oscillator system, using a step size of $\frac{\pi}{4}$ (left), and the star's LP formulation at time $\frac{\pi}{4}$ (right). The values circled in red are the star's basis matrix at time $\frac{\pi}{4}$, which gets updated at each step.

multiplication to get the basis matrix at the next time step. This can be repeated until the time horizon is reached.

3 Benchmark Results

The simulation-equivalent reachability method has been shown to have promise in terms of scalability [**?**, **?**]. In this section, we check if it can handle a recently-proposed benchmark suite for linear systems [**?**], which includes systems with over 10000 dimensions. For each benchmark, we also considered a variant with a weakened or strengthened unsafe condition, so that each system would have both a safe and an unsafe case. In order to evaluate the accuracy of the method, when an unsafe state is reached, we output the initial state and final states which are unsafe. Then, we perform a numerical simulation with higher accuracy parameters from the initial state and check how close the final point is to the expected final point. We report the relative error of these two points.

Both the speed and the accuracy of the verification result depends on how accurately the basis matrix is computed. Therefore, it is inappropriate to consider one metric without considering the other. In order to compute the basis matrix, we consider four different approaches: two based on numeric simulations, and two based on matrix exponential and matrix multiplication. For matrix exponential, we consider using both single precision and double precision numbers, which give different speed / accuracy values. For numeric simulation, we try to choose absolute and relative tolerance in the simulation engine so that the method is close to the accuracy of the matrix exponential approaches. In our case, we found that a simulation tolerance of 10^{-6} produced errors similar to the single-precision matrix exponential method, and simulation tolerance 10^{-12} yielded errors close to the double-precision matrix exponential approach.

For the benchmarks, we use the original time bound of 20. We consider different step sizes. While simulation methods are expected to take slightly longer with more intermediate steps, we expect more time steps to have more impact on the matrix exponential approaches, where the number of matrix multiplications is exactly equal to the number of time steps.

Our evaluation uses the Hylaa tool, which is written mostly in Python. Matrix exponential is computed using `scipy` and the `expm` function, which uses a Padé approximation. Numerical simulations are done with `scipy`'s `odeint` function, which can simulate both stiff and non-stiff systems with using `lsoda` from the Fortran library `odepack`. The implementation parallelizes both the matrix multiplication (for matrices larger than 150×150, which was empirically derived), as well as the simulations. For large matrices, the simulations are performed a few steps at a time, rather than for the entire time bound, so as to reduce memory consumption (a single basis matrix for a 10000 dimensional model takes about 800MB of RAM).

The results are shown in Table **??**.

Table 1: Benchmark results. Stars (*) indicate original unsafe error conditions.

Model	Dims	Error Condition	Method	Step	Time	Safe?	CE Error	CE Time
Motor*	11	$x_1 \in [0.35, 0.4] \wedge x_5 \in [0.45, 0.6]$	Sim (1e-6)	0.1	0.6s	✓		
				0.05	0.6s	✓		
				0.01	0.7s	✓		
				0.005	0.9s	✓		
				0.001	2.3s	✓		
			ExpM (32-bit)	0.1	0.7s	✓		
				0.05	1.0s	✓		
				0.01	2.7s	✓		
				0.005	4.6s	✓		
				0.001	20.7s	✓		
			Sim (1e-12)	0.1	0.6s	✓		
				0.05	0.6s	✓		
				0.01	0.8s	✓		
				0.005	0.9s	✓		
				0.001	2.3s	✓		
			ExpM (64-bit)	0.1	0.8s	✓		
				0.05	1.0s	✓		
				0.01	2.6s	✓		
				0.005	4.7s	✓		
				0.001	20.1s	✓		
Motor	11	$x_1 \in [0.3, 0.4] \wedge x_5 \in [0.4, 0.6]$	Sim (1e-6)	0.1	0.6s		$1.6 \cdot 10^{-6}$	0.1
				0.05	0.5s		$3.4 \cdot 10^{-7}$	0.05
				0.01	0.6s		$1.7 \cdot 10^{-6}$	0.04
				0.005	0.7s		$1.7 \cdot 10^{-6}$	0.04
				0.001	0.6s		$9.7 \cdot 10^{-7}$	0.037
			ExpM (32-bit)	0.1	0.6s		$1.9 \cdot 10^{-5}$	0.1
				0.05	0.5s		$1.0 \cdot 10^{-5}$	0.05
				0.01	0.6s		$1.1 \cdot 10^{-5}$	0.04
				0.005	0.5s		$2.0 \cdot 10^{-5}$	0.04
				0.001	0.6s		$7.4 \cdot 10^{-6}$	0.037
			Sim (1e-12)	0.1	0.5s		$1.2 \cdot 10^{-9}$	0.1
				0.05	0.6s		$7.3 \cdot 10^{-11}$	0.05
				0.01	0.6s		$3.7 \cdot 10^{-12}$	0.04
				0.005	0.7s		$4.0 \cdot 10^{-12}$	0.04
				0.001	0.8s		$2.2 \cdot 10^{-11}$	0.037
			ExpM (64-bit)	0.1	0.6s		$1.2 \cdot 10^{-9}$	0.1
				0.05	0.5s		$7.2 \cdot 10^{-11}$	0.05
				0.01	0.6s		$4.5 \cdot 10^{-12}$	0.04
				0.005	0.6s		$4.5 \cdot 10^{-12}$	0.04
				0.001	0.5s		$2.5 \cdot 10^{-11}$	0.037
Building*	50	$x_{25} \geq 0.006$	Sim (1e-6)	0.1	1.1s	✓		
				0.05	1.2s	✓		
				0.01	1.6s	✓		
				0.005	1.9s	✓		
				0.001	6.7s	✓		
			ExpM (32-bit)	0.1	0.9s	✓		
				0.05	1.1s	✓		

				0.01	3.3s	✓		
				0.005	5.7s	✓		
				0.001	26.8s	✓		
			Sim (1e-12)	0.1	2.1s	✓		
				0.05	2.1s	✓		
				0.01	2.7s	✓		
				0.005	3.3s	✓		
				0.001	7.6s	✓		
			ExpM (64-bit)	0.1	0.9s	✓		
				0.05	1.1s	✓		
				0.01	3.3s	✓		
				0.005	5.8s	✓		
				0.001	27.2s	✓		
Building	50	$x_{25} \geq 0.004$	Sim (1e-6)	0.1	1.0s	✓		
				0.05	1.1s	✓		
				0.01	0.9s		$1.8 \cdot 10^{-4}$	0.07
				0.005	1.0s		$1.8 \cdot 10^{-4}$	0.07
				0.001	2.1s		$1.8 \cdot 10^{-4}$	0.07
			ExpM (32-bit)	0.1	0.9s	✓		
				0.05	1.2s	✓		
				0.01	0.6s		$9.8 \cdot 10^{-6}$	0.07
				0.005	0.5s		$2.8 \cdot 10^{-5}$	0.07
				0.001	0.7s		$7.1 \cdot 10^{-5}$	0.07
			Sim (1e-12)	0.1	2.2s	✓		
				0.05	2.1s	✓		
				0.01	2.0s		$1.2 \cdot 10^{-8}$	0.07
				0.005	2.2s		$1.1 \cdot 10^{-8}$	0.07
				0.001	3.0s		$9.0 \cdot 10^{-9}$	0.07
			ExpM (64-bit)	0.1	0.8s	✓		
				0.05	1.2s	✓		
				0.01	0.5s		$1.2 \cdot 10^{-8}$	0.07
				0.005	0.6s		$1.0 \cdot 10^{-8}$	0.07
				0.001	0.7s		$8.5 \cdot 10^{-9}$	0.07
PDE*	86	$y_1 \geq 12$	Sim (1e-6)	0.1	0.7s	✓		
				0.05	0.8s	✓		
				0.01	2.0s	✓		
				0.005	3.1s	✓		
				0.001	12.6s	✓		
			ExpM (32-bit)	0.1	0.8s	✓		
				0.05	1.1s	✓		
				0.01	3.1s	✓		
				0.005	5.7s	✓		
				0.001	25.1s	✓		
			Sim (1e-12)	0.1	0.7s	✓		
				0.05	0.9s	✓		
				0.01	1.6s	✓		
				0.005	3.2s	✓		
				0.001	12.7s	✓		
			ExpM (64-bit)	0.1	0.9s	✓		
				0.05	1.1s	✓		
				0.01	3.2s	✓		
				0.005	5.8s	✓		
				0.001	26.3s	✓		
PDE	86	$y_1 \geq 10.75$	Sim (1e-6)	0.1	0.7s		$5.8 \cdot 10^{-8}$	0.1
				0.05	0.6s		$2.8 \cdot 10^{-6}$	0.05
				0.01	1.0s		$1.9 \cdot 10^{-6}$	0.03
				0.005	1.4s		$3.9 \cdot 10^{-6}$	0.025
				0.001	4.2s		$1.3 \cdot 10^{-6}$	0.021
			ExpM (32-bit)	0.1	0.6s		$4.1 \cdot 10^{-7}$	0.1
				0.05	0.7s		$4.1 \cdot 10^{-7}$	0.05
				0.01	0.6s		$5.2 \cdot 10^{-7}$	0.03
				0.005	0.6s		$5.3 \cdot 10^{-7}$	0.025
				0.001	0.6s		$2.0 \cdot 10^{-7}$	0.021
			Sim (1e-12)	0.1	0.6s		$4.8 \cdot 10^{-12}$	0.1
				0.05	0.6s		$2.6 \cdot 10^{-11}$	0.05
				0.01	1.0s		$1.7 \cdot 10^{-10}$	0.03
				0.005	1.4s		$1.2 \cdot 10^{-11}$	0.025

				0.001	4.1s		$2.2 \cdot 10^{-11}$	0.021
			ExpM (64-bit)	0.1	0.6s		$5.0 \cdot 10^{-12}$	0.1
				0.05	0.6s		$3.4 \cdot 10^{-11}$	0.05
				0.01	0.6s		$1.7 \cdot 10^{-10}$	0.03
				0.005	0.6s		$1.7 \cdot 10^{-11}$	0.025
				0.001	0.6s		$1.9 \cdot 10^{-11}$	0.021
Heat*	202	$x_{133} \geq 0.1$	Sim (1e-6)	0.1	1.6s	✓		
				0.05	1.9s	✓		
				0.01	5.9s	✓		
				0.005	11.0s	✓		
				0.001	54.0s	✓		
			ExpM (32-bit)	0.1	2.2s	✓		
				0.05	3.3s	✓		
				0.01	11.7s	✓		
				0.005	22.3s	✓		
				0.001	1m58s	✓		
			Sim (1e-12)	0.1	3.2s	✓		
				0.05	3.8s	✓		
				0.01	7.0s	✓		
				0.005	11.2s	✓		
				0.001	54.1s	✓		
			ExpM (64-bit)	0.1	1.7s	✓		
				0.05	2.7s	✓		
				0.01	10.0s	✓		
				0.005	19.0s	✓		
				0.001	1m34s	✓		
Heat	202	$x_{133} \geq 0.02$	Sim (1e-6)	0.1	1.7s		$4.4 \cdot 10^{-6}$	15.7
				0.05	1.8s		$4.1 \cdot 10^{-6}$	15.7
				0.01	5.3s		$4.7 \cdot 10^{-6}$	15.67
				0.005	9.4s		$4.5 \cdot 10^{-6}$	15.67
				0.001	46.4s		$3.3 \cdot 10^{-6}$	15.67
			ExpM (32-bit)	0.1	2.0s		$2.5 \cdot 10^{-4}$	15.7
				0.05	2.8s		$2.5 \cdot 10^{-4}$	15.7
				0.01	9.2s		$7.4 \cdot 10^{-4}$	15.65
				0.005	17.5s		$7.1 \cdot 10^{-4}$	15.65
				0.001	1m31s		$5.4 \cdot 10^{-4}$	15.652
			Sim (1e-12)	0.1	3.1s		$9.5 \cdot 10^{-10}$	15.7
				0.05	3.5s		$9.6 \cdot 10^{-10}$	15.7
				0.01	6.4s		$9.4 \cdot 10^{-10}$	15.67
				0.005	10.3s		$8.4 \cdot 10^{-10}$	15.67
				0.001	45.6s		$8.0 \cdot 10^{-10}$	15.67
			ExpM (64-bit)	0.1	1.6s		$9.8 \cdot 10^{-10}$	15.7
				0.05	2.3s		$9.8 \cdot 10^{-10}$	15.7
				0.01	8.3s		$9.7 \cdot 10^{-10}$	15.67
				0.005	16.1s		$8.7 \cdot 10^{-10}$	15.67
				0.001	1m16s		$8.9 \cdot 10^{-10}$	15.67
ISS*	274	$y_3 \notin [-0.0005, 0.0005]$	Sim (1e-6)	0.1	46.5s	✓		
				0.05	46.6s	✓		
				0.01	50.7s	✓		
				0.005	54.3s	✓		
				0.001	1m17s	✓		
			ExpM (32-bit)	0.1	2.9s	✓		
				0.05	4.6s	✓		
				0.01	10.6s	✓		
				0.005	20.4s	✓		
				0.001	1m36s	✓		
			Sim (1e-12)	0.1	6m1s	✓		
				0.05	5m58s	✓		
				0.01	6m11s	✓		
				0.005	6m14s	✓		
				0.001	6m25s	✓		
			ExpM (64-bit)	0.1	2.7s	✓		
				0.05	4.3s	✓		
				0.01	10.7s	✓		
				0.005	20.1s	✓		
				0.001	1m35s	✓		
ISS	274	$y_3 \notin [-0.00017, 0.00017]$	Sim (1e-6)	0.1	45.6s		$2.7 \cdot 10^{-4}$	0.5

				0.05	47.5s	$2.6 \cdot 10^{-4}$	0.5
				0.01	47.4s	$3.0 \cdot 10^{-4}$	0.5
				0.005	47.7s	$1.7 \cdot 10^{-4}$	0.5
				0.001	22.9s	$4.9 \cdot 10^{-4}$	0.498
			ExpM (32-bit)	0.1	0.9s	$1.3 \cdot 10^{-6}$	0.5
				0.05	1.1s	$1.2 \cdot 10^{-6}$	0.5
				0.01	1.3s	$4.1 \cdot 10^{-6}$	0.5
				0.005	1.6s	$8.3 \cdot 10^{-6}$	0.5
				0.001	3.4s	$4.8 \cdot 10^{-5}$	0.498
			Sim (1e-12)	0.1	5m58s	$1.1 \cdot 10^{-6}$	0.5
				0.05	6m1s	$9.7 \cdot 10^{-7}$	0.5
				0.01	6m18s	$7.9 \cdot 10^{-7}$	0.5
				0.005	6m4s	$7.9 \cdot 10^{-7}$	0.5
				0.001	2m28s	$6.3 \cdot 10^{-7}$	0.498
			ExpM (64-bit)	0.1	0.8s	$1.1 \cdot 10^{-6}$	0.5
				0.05	0.9s	$9.7 \cdot 10^{-7}$	0.5
				0.01	1.1s	$7.9 \cdot 10^{-7}$	0.5
				0.005	1.3s	$7.9 \cdot 10^{-7}$	0.5
				0.001	3.2s	$6.3 \cdot 10^{-7}$	0.498
Beam	350	$x_{89} \geq 2100$	Sim (1e-6)	0.1	37.5s	✓	
				0.05	39.4s	✓	
				0.01	49.6s	✓	
				0.005	1m4s	✓	
				0.001	2m56s	✓	
			ExpM (32-bit)	0.1	4.6s	✓	
				0.05	7.0s	✓	
				0.01	26.3s	✓	
				0.005	49.5s	✓	
				0.001	3m57s	✓	
			Sim (1e-12)	0.1	4m5s	✓	
				0.05	4m16s	✓	
				0.01	4m36s	✓	
				0.005	4m32s	✓	
				0.001	6m33s	✓	
			ExpM (64-bit)	0.1	4.2s	✓	
				0.05	6.7s	✓	
				0.01	25.1s	✓	
				0.005	47.4s	✓	
				0.001	3m45s	✓	
Beam*	350	$x_{89} \geq 1000$	Sim (1e-6)	0.1	37.5s	$5.2 \cdot 10^{-8}$	16.1
				0.05	38.1s	$5.0 \cdot 10^{-8}$	16.05
				0.01	47.6s	$6.1 \cdot 10^{-8}$	16.05
				0.005	1m0s	$9.3 \cdot 10^{-8}$	16.045
				0.001	2m25s	$4.0 \cdot 10^{-8}$	16.041
			ExpM (32-bit)	0.1	4.4s	$9.8 \cdot 10^{-3}$	16.1
				0.05	6.2s	$9.7 \cdot 10^{-3}$	16.1
				0.01	21.4s	$9.4 \cdot 10^{-4}$	16.05
				0.005	40.1s	$1.4 \cdot 10^{-3}$	16.035
				0.001	3m7s	$1.0 \cdot 10^{-2}$	16.099
			Sim (1e-12)	0.1	4m8s	$8.0 \cdot 10^{-12}$	16.1
				0.05	4m14s	$8.5 \cdot 10^{-12}$	16.05
				0.01	4m30s	$8.4 \cdot 10^{-12}$	16.05
				0.005	4m45s	$8.5 \cdot 10^{-12}$	16.045
				0.001	5m44s	$8.2 \cdot 10^{-12}$	16.041
			ExpM (64-bit)	0.1	4.2s	$4.8 \cdot 10^{-11}$	16.1
				0.05	6.0s	$4.7 \cdot 10^{-11}$	16.05
				0.01	20.7s	$1.0 \cdot 10^{-11}$	16.05
				0.005	39.0s	$9.9 \cdot 10^{-12}$	16.045
				0.001	3m8s	$1.7 \cdot 10^{-11}$	16.041
MNA1*	588	$x_1 \geq 0.5$	Sim (1e-6)	0.1	1m17s	✓	
				0.05	1m21s	✓	
				0.01	2m10s	✓	
				0.005	3m7s	✓	
				0.001	11m21s	✓	
			ExpM (32-bit)	0.1	49.6s	✓	
				0.05	1m30s	✓	

				0.01	7m18s	✓		
				0.005	14m47s	✓		
				0.001	1h14m	✓		
			Sim (1e-12)	0.1	8m34s	✓		
				0.05	8m41s	✓		
				0.01	9m37s	✓		
				0.005	10m43s	✓		
				0.001	20m14s	✓		
			ExpM (64-bit)	0.1	19.6s	✓		
				0.05	36.0s	✓		
				0.01	2m46s	✓		
				0.005	5m32s	✓		
				0.001	27m17s	✓		
MNA1	588	$x_1 \geq 0.2$	Sim (1e-6)	0.1	1m17s		$1.9{\cdot}10^{-5}$	16.6
				0.05	1m19s		$1.9{\cdot}10^{-5}$	16.6
				0.01	2m3s		$1.7{\cdot}10^{-5}$	16.56
				0.005	2m54s		$2.0{\cdot}10^{-5}$	16.555
				0.001	9m26s		$8.9{\cdot}10^{-6}$	16.554
			ExpM (32-bit)	0.1	42.3s		$1.3{\cdot}10^{-3}$	16.6
				0.05	1m15s		$1.3{\cdot}10^{-3}$	16.55
				0.01	6m3s		$1.1{\cdot}10^{-3}$	16.56
				0.005	12m18s		$1.1{\cdot}10^{-3}$	16.555
				0.001	1h1m		$1.5{\cdot}10^{-3}$	16.544
			Sim (1e-12)	0.1	8m38s		$3.3{\cdot}10^{-9}$	16.6
				0.05	8m35s		$3.3{\cdot}10^{-9}$	16.6
				0.01	9m31s		$3.3{\cdot}10^{-9}$	16.56
				0.005	10m30s		$3.3{\cdot}10^{-9}$	16.555
				0.001	16m57s		$3.3{\cdot}10^{-9}$	16.554
			ExpM (64-bit)	0.1	15.8s		$3.5{\cdot}10^{-9}$	16.6
				0.05	30.1s		$3.5{\cdot}10^{-9}$	16.6
				0.01	2m16s		$3.5{\cdot}10^{-9}$	16.56
				0.005	4m30s		$3.5{\cdot}10^{-9}$	16.555
				0.001	22m5s		$3.5{\cdot}10^{-9}$	16.554
FOM*	1008	$y_1 \geq 45$	Sim (1e-6)	0.1	3m6s	✓		
				0.05	3m17s	✓		
				0.01	8m41s	✓		
				0.005	15m1s	✓		
				0.001	53m24s	✓		
			ExpM (32-bit)	0.1	35.2s	✓		
				0.05	1m6s	✓		
				0.01	4m52s	✓		
				0.005	9m26s	✓		
				0.001	47m29s	✓		
			Sim (1e-12)	0.1	6m43s	✓		
				0.05	6m31s	✓		
				0.01	12m44s	✓		
				0.005	18m29s	✓		
				0.001	1h0m	✓		
			ExpM (64-bit)	0.1	35.6s	✓		
				0.05	53.9s	✓		
				0.01	4m6s	✓		
				0.005	8m15s	✓		
				0.001	39m18s	✓		
FOM	1008	$y_1 \geq 7$	Sim (1e-6)	0.1	3m13s		$2.5{\cdot}10^{-4}$	0.2
				0.05	3m8s		$2.6{\cdot}10^{-4}$	0.2
				0.01	2m23s		$1.3{\cdot}10^{-4}$	0.07
				0.005	2m9s		$1.3{\cdot}10^{-4}$	0.07
				0.001	1m35s		$1.3{\cdot}10^{-4}$	0.069
			ExpM (32-bit)	0.1	8.4s		$1.1{\cdot}10^{-6}$	0.2
				0.05	8.4s		$1.1{\cdot}10^{-6}$	0.2
				0.01	8.2s		$1.1{\cdot}10^{-6}$	0.07
				0.005	8.6s		$1.1{\cdot}10^{-6}$	0.07
				0.001	15.1s		$1.9{\cdot}10^{-6}$	0.069
			Sim (1e-12)	0.1	6m43s		$1.9{\cdot}10^{-9}$	0.2
				0.05	6m34s		$1.9{\cdot}10^{-9}$	0.2
				0.01	3m36s		$1.3{\cdot}10^{-9}$	0.07
				0.005	3m10s		$1.3{\cdot}10^{-9}$	0.07

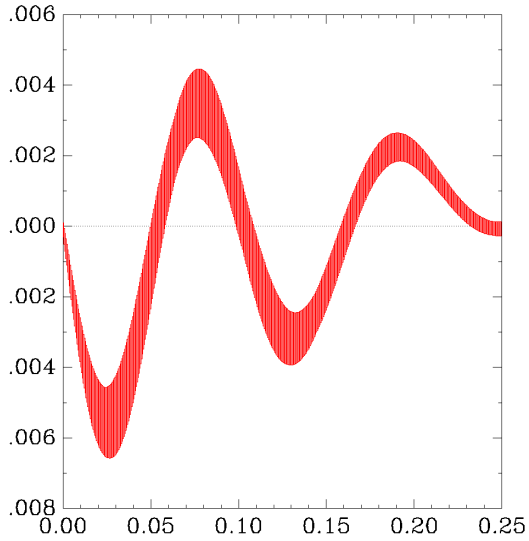


Figure 2: Plot of the first 0.25 seconds of the reachable states for the Building (50 dimensions) model produced in SpaceEx [**?**]. Notice that fixed-time simulations with a time step of 0.1 or 0.05 do not enter the unsafe states ($x_{25} > 0.004$).

	0.001	2m20s	$1.3 \cdot 10^{-9}$	0.069
ExpM (64-bit)	0.1	7.6s	$2.2 \cdot 10^{-9}$	0.2
	0.05	7.6s	$2.2 \cdot 10^{-9}$	0.2
	0.01	7.2s	$1.4 \cdot 10^{-9}$	0.07
	0.005	7.6s	$1.4 \cdot 10^{-9}$	0.07
	0.001	12.4s	$1.5 \cdot 10^{-9}$	0.069

The star-based reachability method is confirmed as scalable, and finishes on all of the benchmarks, often taking only minutes for systems with hundreds of dimensions.

All four approaches, as well as all values of the time step are able to correctly verify or find counter-examples traces to the unsafe error states for all the models, with the exception of the Building (50 dimensions) model when using a time step of 0.1 or 0.05. Examining the plot of the reachable states of this system (Figure **??**), the reason becomes apparent.

4 Conclusion

We analyzed nine benchmark systems using reachability analysis. Our main result is that the recently proposed technique for linear systems is capable of scaling to large systems, and can detect errors in models with high-dimensional initial sets of states. In the future, we plan to look into hybrid systems verification using the simulation-based continuous post approach.