Direct Verification of 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 recently-proposed reachability method on a set of high-dimensional linear system benchmarks taken from model order reduction and presented in ARCH 2016. The method uses a state-set representation called a generalized star set and the principle of superposition of linear systems to achieve scalability. It was previously shown to have promise in terms of scalability for direct analysis of large linear systems. For each benchmark, we also compare computing the basis matrix, a core part of the reachability method, using numerical simulations versus a matrix exponential formulation. The approach successfully analyzes systems with hundreds of dimensions in minutes, and can scale to systems with over 10000 dimensions with a computation time ranging from tens of minutes to tens of hours, depending on the desired time step.

1 Introduction

Reachability analysis attempts to compute, given a set of initial states, the set of states a 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 system dynamics expressed as $\dot{x} = Ax + c$. We only consider the continuous post operation, that is, we do not evaluate guards or other features of a hybrid automaton [3]. Nonetheless, continuous post is a core component of many hybrid systems reachability methods.

When faced with a high-dimensional safety verification problem, a powerful approach is abstraction [6, 4, 11]. Abstraction methods try to simplify a complex system to a simpler one, such that properties of interest, such as safety, are preserved. For example, abstraction methods can reduce certain high-dimensional systems to ones with less dimensions [21]. The reason to do this is because there are scalability limits to verification approaches, for example, the number of dimensions that can be handled in a reasonable amount of time. The methods evaluated here are orthogonal to these abstraction approaches, and deal with the direct verification problem. Nonetheless, note that abstraction is still typically followed up by direct verification on a simplified model. Finally, it is common to combine abstraction and direct verification in a loop,

DISTRIBUTION A: Approved for public release; distribution unlimited (#88ABW-2017-0429, 02 FEB 2017)

using counter-examples from direct verification to guide the abstraction process [13, 5]. To enable such counter-example abstraction refinement (CEGAR), the direct verification method must produce counter-example traces.

Reachable state set computation tools typically track states uses data structures such as polyhedra [16], zonotopes [18], support functions [19], Taylor models [12], or combinations of these [2]. Recently, a data structure and associated reachability method was proposed which stores states in a data structure called a generalized star set [15]. The reachability method we evaluate is based on this technique, which essentially uses individual executions and superposition to compute the reachable states.

The implementation we used only reasons about the states reachable at discrete time steps. It does not look at 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.

Since the dynamics are time-invariant, however, if some external method is first used to bound the states reachable at any time within first time step, then this larger set can be used as the initial states with the unmodified technique to prove properties about the system at all times [14]. For example, for small time steps the system's Lipschitz constant may be used to produce such a bound on all the states reached between the first two time steps. This aspect of the reachability problem is not the focus of this paper.

We perform our evaluation using a benchmark suite consisting of nine large-scale linear systems [20]. These benchmarks were taken from "diverse fields such as civil engineering and robotics." The difficulty of these benchmarks was rated by the benchmark authors as "low" to "challenge", and the models range from ten dimensions 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.

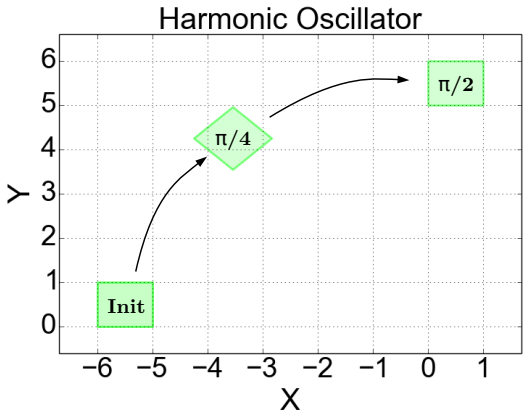
2 Computing a Star's Basis Matrix

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 [15].

The generalized star set approach for reachability computation takes advantage of the superposition principle of linear systems in order to compute the reachable set. 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 state reachable from an arbitrary point by taking linear combinations of the states 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 where any initial point has evolved to at time t. If the dynamics are affine, $\dot{x} = Ax + c$, then one more simulation is required from the origin, in addition to the n simulations that are performed on the linear part of the system $\dot{x} = Ax$.

If we are interested in where a set of states, rather than a single point, can evolve to at time t, we can construct a linear program (LP) that encodes the constraints on the initial conditions and *how each unit vector has evolved at time* t. Linear conditions can also be added



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

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.

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 $n \times n$ *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 an example of the star-based reachability approach.

Harmonic Oscillator Example. 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 1. 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. The basis matrix gets updated at each time step, while the rest of the constraints remain the same. The initial conditions are encoded in rows 3-6. The basis matrix at time 0 is the identity matrix, $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$. After $\frac{\pi}{4}$ time, the point which started at state $\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$ evolves to $\left(\begin{smallmatrix} 0.707 \\ -0.707 \end{smallmatrix}\right)$ and the point which started at $\left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)$ goes to $\left(\begin{smallmatrix} 0.707 \\ 0.707 \end{smallmatrix}\right)$. 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 circled in red 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. A concrete error trace, then, would start at point (α_1, α_2), and, after evolving for t time, reach the unsafe point (x_1, x_2)

As mentioned before, computing the basis matrix for each time step dominates the runtime of the safety verification method, and so it is important that its computation is optimized. In the original work [15], 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}. For example, in the harmonic oscillator system above, the dynamics matrix is $A = \left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right)$, and the basis matrix at time $\frac{\pi}{4}$ evaluates to $e^{A(\pi/4)} = \left(\begin{smallmatrix} 0.707 & 0.707 \\ -0.707 & 0.707 \end{smallmatrix}\right)$, which is the same as the numerical simulation approach.

Further, rather than recomputing the basis matrix at each step, 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 multiplication to get the basis matrix at the next time step. This can be repeated until the time horizon is reached. In the next section, we compare these methods for computing the basis matrix on a set of benchmark systems.

3 Benchmark Results

The simulation-equivalent reachability method has been shown to have promise in terms of scalability [9, 10]. In this section, we evaluate the approach on a recently-proposed benchmark suite for linear systems [20], which includes a system 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 extracted from the LP which are deemed to be unsafe. Then, we perform a numerical simulation with higher accuracy parameters from the exact initial state and check how close the final point is to the expected final point. We report the relative error of these two points (`CE Error`).

Both the speed and the accuracy of the overall verification approach depends on how accurately the basis matrix is computed. For numerical simulation, we tried to choose absolute and relative tolerances in the simulation engine so that the method is close to the accuracy of the matrix exponential approach. In our case, we found that simulation tolerances of 10^{-12} yielded errors close to the double-precision matrix exponential method.

In communications with the benchmark authors, we found out that the error conditions were chosen based on a finite number of simulations of the models, while holding the inputs constant. To model this, for each input u_i we add a new variable to the model with the appropriate initial range and differential equation with $\dot{u_i} = 0$. This model transformation was automated using the Hyst model transformation tool [7]. Each model also includes a time variable with $\dot{t} = 1$. Thus, for example, while the original benchmark for the Motor system had 8 variables, 2 inputs, and a time variable, we report this as an 11-dimensional system.

For the benchmarks, we use the original time bound of 20. We consider different step sizes, from 0.1 (200 steps) down to the step size suggested by the benchmark authors of 0.001 (20000 steps). We used the `hypy` library [8] in order to script the process of running all the benchmarks with all the parameters.

Our evaluation uses a new tool, Hylaa[1], which is written mostly in Python. Matrix exponential is computed using `scipy` and the `sparse.linalg.expm` function, which uses a Padé approximation optimized for sparse matrices, which we found was faster on the systems in this benchmark set than the dense-matrix equivalent. Numerical simulations are done with `scipy`'s `integrate.odeint` function, which can simulate both stiff and non-stiff systems with using `lsoda` from the Fortran library `odepack`. Matrix multiplication was performed using the `numpy` library's `dot` function. The implementation parallelizes both the matrix multiplication (for ma-

[1] http://stanleybak.com/hylaa

trices larger than 150×150, which was empirically derived), as well as the simulations, across all the cores in the system. The measurements were performed on an Intel i7-3930K processor (6 cores, 12 threads) running at 3.5 GHz with 24 GB RAM.

For large matrices with a large number of time steps, it is infeasible to store the complete simulation results in memory. For example, a single basis matrix for a 10000 dimensional model takes about 800 megabytes of RAM (10000×10000 entries times 8 bytes per entry). With a time step of 0.001 and the time bound of 20, storing all 20000 of these would require over 15 terabytes of memory. To reduce the memory requirement, the simulations are not always performed for the full time-bound, and can be split into several parts. This does have the effect of slightly slowing down the computation, since the internally variable-step simulations must be reinitialized each time. The length of the simulations performed at one time is computed based on the amount of memory made available for the computation. A positive side-effect of this is that it allows for early termination when an unsafe state can be reached versus if the entire simulation needed to be computed beforehand. The matrix exponential approach, in contrast, computes one step at a time, and so will terminate on exactly the step where an unsafe state is reached.

The results for each model, safety condition, basis matrix computation method, and for the various step sizes, are shown in Table 1. The star-based reachability method is confirmed as scalable, and is able to perform analysis on all of the benchmarks, often taking only minutes for systems with hundreds of dimensions.

In terms of correctness, both 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. Examining the plot of the reachable states of this system (Figure 2), the reason becomes apparent. A fixed-step simulation with a large time step, in fact, does not reach error states, which only happens around time 0.07. Traditional reachability analysis methods, such as those implemented in SpaceEx [17], Flow* [12], or CORA [1], do reason in between time steps and would not mark such systems as safe. Nonetheless, due to the scalability and generation of counter-examples, we believe there is a place for simulation-equivalent reachability.

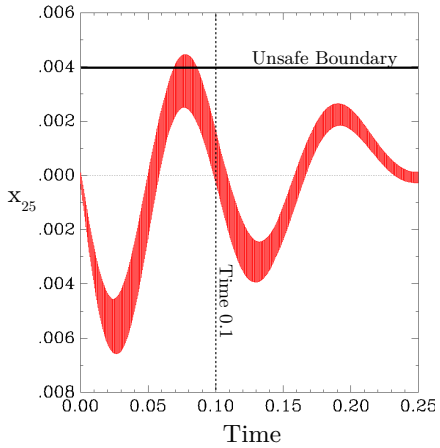


Figure 2: Plot of x_{25} over time of the first 0.25 seconds for the Building (50 dimensions) model produced in SpaceEx. Notice that fixed-time simulations with a time step of 0.1 do not enter the unsafe states ($x_{25} \geq 0.004$).

Table 1: Benchmark results. Stars (*) indicate the benchmark's original unsafe condition.

Model	Dims	Unsafe Error Condition	Method	Step Size	Runtime	Safe?	CE Error	CE Time
Motor*	11	$x_1 \in [0.35, 0.4] \wedge x_5 \in [0.45, 0.6]$	Simulation	0.1	0.5s	✓		
				0.01	0.8s	✓		
				0.001	2.6s	✓		
			Matrix Exp	0.1	0.4s	✓		
				0.01	0.7s	✓		
				0.001	3.9s	✓		
Motor	11	$x_1 \in [0.3, 0.4] \wedge x_5 \in [0.4, 0.6]$	Simulation	0.1	0.5s		$1.6 \cdot 10^{-11}$	0.1
				0.01	0.6s		$6.2 \cdot 10^{-13}$	0.04
				0.001	0.6s		$2.2 \cdot 10^{-12}$	0.037
			Matrix Exp	0.1	0.5s		$9.3 \cdot 10^{-13}$	0.1
				0.01	0.5s		$9.9 \cdot 10^{-13}$	0.04
				0.001	0.6s		$1.4 \cdot 10^{-12}$	0.037
Building*	50	$x_{25} \geq 0.006$	Simulation	0.1	2.1s	✓		
				0.01	2.5s	✓		
				0.001	7.6s	✓		
			Matrix Exp	0.1	0.6s	✓		
				0.01	1.4s	✓		
				0.001	8.9s	✓		
Building	50	$x_{25} \geq 0.004$	Simulation	0.1	1.9s	✓		
				0.01	2.0s		$1.2 \cdot 10^{-9}$	0.07
				0.001	2.8s		$2.2 \cdot 10^{-9}$	0.07
			Matrix Exp	0.1	0.5s	✓		
				0.01	0.5s		$1.1 \cdot 10^{-9}$	0.07
				0.001	0.4s		$1.1 \cdot 10^{-9}$	0.07
PDE*	86	$y_1 \geq 12$	Simulation	0.1	0.7s	✓		
				0.01	1.9s	✓		
				0.001	13.4s	✓		
			Matrix Exp	0.1	0.7s	✓		
				0.01	2.2s	✓		
				0.001	17.9s	✓		
PDE	86	$y_1 \geq 10.75$	Simulation	0.1	0.6s		$1.2 \cdot 10^{-12}$	0.1
				0.01	0.8s		$3.2 \cdot 10^{-12}$	0.03
				0.001	4.0s		$1.3 \cdot 10^{-11}$	0.021
			Matrix Exp	0.1	0.4s		$1.3 \cdot 10^{-12}$	0.1
				0.01	0.6s		$1.3 \cdot 10^{-12}$	0.03
				0.001	0.5s		$1.2 \cdot 10^{-12}$	0.021
Heat*	202	$x_{133} \geq 0.1$	Simulation	0.1	4.1s	✓		
				0.01	7.7s	✓		
				0.001	56.6s	✓		
			Matrix Exp	0.1	2.1s	✓		
				0.01	12.0s	✓		
				0.001	1m57s	✓		
Heat	202	$x_{133} \geq 0.02$	Simulation	0.1	4.1s		$3.9 \cdot 10^{-11}$	15.7
				0.01	6.9s		$6.7 \cdot 10^{-12}$	15.67
				0.001	47.7s		$8.9 \cdot 10^{-12}$	15.67
			Matrix Exp	0.1	1.6s		$2.4 \cdot 10^{-11}$	15.7
				0.01	9.8s		$2.5 \cdot 10^{-11}$	15.67
				0.001	1m30s		$2.5 \cdot 10^{-11}$	15.67
ISS*	274	$y_3 \notin [-0.0005, 0.0005]$	Simulation	0.1	7m23s	✓		
				0.01	7m14s	✓		
				0.001	7m44s	✓		
			Matrix Exp	0.1	2.9s	✓		
				0.01	11.5s	✓		
				0.001	1m39s	✓		
ISS	274	$y_3 \notin [-0.00017, 0.00017]$	Simulation	0.1	7m11s		$1.1 \cdot 10^{-6}$	0.5
				0.01	7m8s		$7.9 \cdot 10^{-7}$	0.5
				0.001	4m10s		$6.3 \cdot 10^{-7}$	0.498
			Matrix Exp	0.1	0.7s		$1.1 \cdot 10^{-6}$	0.5
				0.01	1.2s		$7.9 \cdot 10^{-7}$	0.5
				0.001	3.2s		$6.3 \cdot 10^{-7}$	0.498

Beam	350	$x_{89} \geq 2100$	Simulation	0.1	4m38s	✓		
				0.01	4m47s	✓		
				0.001	7m0s	✓		
			Matrix Exp	0.1	5.2s	✓		
				0.01	28.7s	✓		
				0.001	4m24s	✓		
Beam*	350	$x_{89} \geq 1000$	Simulation	0.1	4m28s		$1.4 \cdot 10^{-12}$	16.1
				0.01	4m39s		$1.5 \cdot 10^{-12}$	16.05
				0.001	6m30s		$8.8 \cdot 10^{-13}$	16.041
			Matrix Exp	0.1	5.1s		$1.3 \cdot 10^{-11}$	16.1
				0.01	23.7s		$2.2 \cdot 10^{-11}$	16.05
				0.001	3m28s		$2.1 \cdot 10^{-12}$	16.041
MNA1*	588	$x_1 \geq 0.5$	Simulation	0.1	9m49s	✓		
				0.01	10m22s	✓		
				0.001	20m41s	✓		
			Matrix Exp	0.1	22.6s	✓		
				0.01	3m0s	✓		
				0.001	30m8s	✓		
MNA1	588	$x_1 \geq 0.2$	Simulation	0.1	9m45s		$1.2 \cdot 10^{-10}$	16.6
				0.01	10m19s		$1.2 \cdot 10^{-10}$	16.56
				0.001	18m20s		$1.2 \cdot 10^{-10}$	16.554
			Matrix Exp	0.1	20.2s		$7.4 \cdot 10^{-11}$	16.6
				0.01	2m31s		$7.4 \cdot 10^{-11}$	16.56
				0.001	24m58s		$7.0 \cdot 10^{-11}$	16.554
FOM*	1008	$y_1 \geq 45$	Simulation	0.1	7m3s	✓		
				0.01	9m39s	✓		
				0.001	45m15s	✓		
			Matrix Exp	0.1	30.3s	✓		
				0.01	4m51s	✓		
				0.001	48m49s	✓		
FOM	1008	$y_1 \geq 7$	Simulation	0.1	7m4s		$3.3 \cdot 10^{-10}$	0.2
				0.01	4m27s		$1.7 \cdot 10^{-10}$	0.07
				0.001	2m39s		$1.7 \cdot 10^{-10}$	0.069
			Matrix Exp	0.1	2.5s		$7.5 \cdot 10^{-12}$	0.2
				0.01	3.3s		$4.4 \cdot 10^{-12}$	0.07
				0.001	12.5s		$4.6 \cdot 10^{-12}$	0.069
MNA5*	10923	$x_1 \geq 0.2 \vee x_2 \geq 0.15$	Simulation	0.1	41m28s	✓		
				0.01	3h51m	✓		
				0.001	24h2m	✓		
			Matrix Exp	0.1	14h3m	✓		
				0.01	(>5d)			
				0.001	(>53d)			
MNA5	10923	$x_1 \geq 0.1 \vee x_2 \geq 0.15$	Simulation	0.1	5m57s		$9.2 \cdot 10^{-8}$	2.0
				0.01	21m40s		$2.0 \cdot 10^{-7}$	1.92
				0.001	2h11m		$2.0 \cdot 10^{-7}$	1.919
			Matrix Exp	0.1	1h25m		$9.2 \cdot 10^{-8}$	2.0
				0.01	13h29m		$2.0 \cdot 10^{-7}$	1.92
				0.1	(>5d)			

Notice that with the original error conditions, the `Beam` model (350 dimensions) can actually reach the unsafe states. This error condition was not known prior to analysis with Hylaa. Since the original error conditions were derived based on simulations, such a situation is possible where the simulation which demonstrates the error states was not tested. This shows the incompleteness of a sampled simulation-based analysis, and motivates more rigorous approaches. If one wanted to exhaustively simulate all the corners of the initial states, since 49 of the variables are initially intervals, one would need to run $2^{49} = 5.6 \cdot 10^{14}$ simulations. For this system, our

parallel implementation averaged 1-2 simulations per second, depending on the time step. With this simulation rate, exhaustive analysis would possible after 9 to 18 million years.

The effects of the time step are also apparent in the counter example time (`CE Time` column). In the MNA1 model (588 dimensions), using a time-step of 0.1 results in a counter-example at time 16.6, which is reduced to 16.56 for a time step of 0.01, and then finally a time 16.554 for a 0.001 time step. The counter-example time, however, is unaffected by the basis matrix computation method on these models, which confirms that they are basically computing the same thing.

In terms of the computation time of the basis matrix, both approaches are viable for most of the systems. The runtime of the simulation method is sensitive to the internal variable time-steps the simulation engine can use, which in turn depends on the system's dynamics. This effect can be seen with the ISS model (274 dimensions), which is completed much faster using the matrix exponential approach. For the largest, MNA5 system (10923 dimensions), however, the matrix exponential method becomes extremely slow when there are a large number of steps. This is because matrix multiplication, which is done at each step to compute the basis matrix, takes about 230 seconds with our parallel implementation. The estimated times in parenthesis in the table are based on this value, multiplied by the number of steps required. With this estimation method, the 0.1 time step variant (200 steps) should take about $200 \cdot 230$ seconds, which equals 12.8 hours, which is close to the measured value of about 14 hours.

While a simulation method is expected to take slightly longer with more intermediate steps, we expect the number of 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. This trend is apparent for the larger systems, such as the safe variant of the MNA1 model (588 dimensions). Here, the matrix exponential approach increases from 20 seconds with a time step of 0.1, to 151 seconds (2m31s) with a time step of 0.01, to 1498 seconds (18m20s) with a time step of 0.001. This is close to a factor of ten increase in runtime each time the number of steps increases by a factor of ten. The simulation method does take longer with more intermediate steps on this system, but the relationship is less than linear, increasing from 589 seconds (9m49s) to 622 seconds (10m22s) to 1241 seconds (20m41s) as the time step is reduced.

The counter-example accuracy is not too affected by the time step. This is because the simulation engine will adjust its internal step size based on the (fixed) accuracy parameter. It is possible in the matrix exponential method that numerical errors could accumulate during repeated matrix multiplications when there are a large number of time steps, but for these systems, we did not observe this making a difference on the counter-example accuracy. Note that the counter-example reported in the `CE Error` column is relative error, and it is small in all cases. Additionally, some of the input models are provided with less than 5 digits of precision in the ODEs, which may also lead to errors in the analysis result even if the approach was exact.

In terms of models which do reach unsafe states, the matrix exponential approach usually takes less time to find a counter-example. This is because, as an optimization, simulations are run in batches over many steps rather than a single step at a time. The matrix exponential methods, however, perform one step at a time. For larger systems, where the simulation length is limited for memory reasons as described earlier, we do observe that the analysis time is reduced for the simulation approach when compared with the case where no unsafe states are reachable. This is why the unsafe variant of the FOM model (1008 dimensions) takes less time to compute as the time step is reduced (the simulation lengths are reduced since there are more intermediate time steps).

Finally, since the runtime is dominated by the computation of the basis matrix, and this

computation is independent of the initial states or error states, its result could be reused. This would allow the user to make arbitrary changes to the initial states and unsafe conditions, and then quickly compute simulation-equivalent reachability without having to do the expensive basis matrix computation. This was useful in order to find the strengthened error conditions where there are reachable unsafe states for each of the models, since it allowed us to perform many runs of the tool in a short amount of time. For the measurements, however, we ran all the computations from scratch.

4 Conclusion

We analyzed nine high-dimensional benchmark systems using simulation-equivalent reachability analysis. Our main result confirms that the recently proposed technique for generalized star-based analysis of linear systems is capable of scaling to extremely large systems. Further, the counter-example error traces constructed are of high accuracy, as measured by post-analysis, high-accuracy simulations. We also evaluated two methods for the computation of the basis matrix, which dominates the runtime of the approach. We compared using a matrix exponential approach versus numerical simulations. Generally, both can be used, although for the largest systems with many steps, numerical simulations were generally faster.

There are other aspects that may be a factor when choosing the basis matrix computation method. From an interface perspective, using simulations allows more fine-grained control of the error than the matrix exponential. However, the matrix exponential method may be superior if a quick safety check is desired at a known specific time instant, rather than at all multiples of the time step. A hybrid approach is also possible, where simulations are used to compute the first step's basis matrix, and then matrix multiplication is used for the remaining steps. The runtime of this method is likely similar to the matrix exponential methods, since the runtime is dominated by the matrix multiplication portion, especially when there are a large number of steps.

References

[1] M. Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[2] M. Althoff and G. Frehse. Combining zonotopes and support functions for efficient reachability analysis of linear systems. In *Proc. of the 55th IEEE Conference on Decision and Control*, 2016.

[3] 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*, 138:3–34, 1995.

[4] R. Alur, T. Dang, and F. Ivančić. Progress on reachability analysis of hybrid systems using predicate abstraction. In *International Workshop on Hybrid Systems: Computation and Control*, pages 4–19. Springer, 2003.

[5] R. Alur, T. Dang, and F. Ivančić. Counterexample-guided predicate abstraction of hybrid systems. *Theoretical Computer Science*, 354(2):250–271, 2006.

[6] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[7] S. Bak, S. Bogomolov, and T. T. Johnson. HyST: A source transformation and translation tool for hybrid automaton models. In *18th International Conference on Hybrid Systems: Computation and Control*, Seattle, Washington, Apr. 2015. ACM.

[8] S. Bak, S. Bogomolov, and C. Schilling. High-level hybrid systems analysis with hypy. In *ARCH16: Proc. of the 3rd Workshop on Applied Verification for Continuous and Hybrid Systems*, 2016.

[9] S. Bak and P. S. Duggirala. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, HSCC '17, 2017.

[10] S. Bak and P. S. Duggirala. Rigorous simulation-based analysis of linear hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2017.

[11] S. Bak, A. Greer, and S. Mitra. Hybrid cyberphysical system verification with simplex using discrete abstractions. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS '10, pages 143–152, Washington, DC, USA, 2010. IEEE Computer Society.

[12] X. Chen, E. Abraham, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Proceedings of the 2012 IEEE 33rd Real-Time Systems Symposium*, RTSS '12, pages 183–192, Washington, DC, USA, 2012. IEEE Computer Society.

[13] E. 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 of foundations of computer science*, 14(04):583–604, 2003.

[14] T. Dang. *Verification et synthese des systemes hybrides*. PhD thesis, INPG, Oct 2000.

[15] P. S. Duggirala and M. Viswanathan. Parsimonious, simulation based verification of linear systems. In *International Conference on Computer Aided Verification*, pages 477–494. Springer, 2016.

[16] G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *HSCC*, pages 258–273, 2005.

[17] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV)*, LNCS. Springer, 2011.

[18] A. Girard. Reachability of uncertain linear systems using zonotopes. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control*, LNCS. Springer, 2005.

[19] A. Girard, C. Le Guernic, et al. Efficient reachability analysis for linear systems using support functions. In *Proc. of the 17th IFAC World Congress*, pages 8966–8971, 2008.

[20] H.-D. Tran, L. V. Nguyen, and T. T. Johnson. Large-scale linear systems from order-reduction (benchmark proposal). In *3rd Applied Verification for Continuous and Hybrid Systems Workshop (ARCH)*, Vienna, Austria, 2016.

[21] H.-D. Tran, L. V. Nguyen, W. Xiang, and T. T. Johnson. Order-reduction abstractions for safety verification of high-dimensional linear systems. *arXiv preprint arXiv:1602.06417*, 2016.