Simulation-Equivalent Reachability of Large Linear Systems with Inputs

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

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

Abstract. Control systems can be subject to outside inputs, environmental effects, disturbances, and sensor/actuator inaccuracy. To model such systems, linear differential equations with constrained inputs are often used, $\dot{x}(t) = Ax(t) + Bu(t)$, where the input vector $u(t)$ stays in some bound. Simulating these models is an important tool for detecting design issues. However, since there may be many possible initial states and many possible valid sequences of inputs, simulation-only analysis may also miss critical system errors. In this paper, we present a scalable verification method that computes the *simulation-equivalent reachable set* for a linear system with inputs. This set consists of all the states that can be reached by a fixed-step simulation for (i) any choice of start state in the initial set and (ii) any choice of piecewise constant inputs.

Building upon a recently-developed reachable set computation technique that uses a state-set representation called a generalized star, we extend the approach to incorporate the effects of inputs using linear programming. The approach is made scalable through two optimizations based on Minkowski sum decomposition and warm-start linear programming. We demonstrate scalability by analyzing a series of large benchmark systems, including a system with over 10,000 dimensions (about two orders of magnitude larger than what can be handled by existing tools). The method detects previously-unknown violations in benchmark models, finding complex counter-example traces which validate both its correctness and accuracy.

1 Introduction

Linear dynamical systems with inputs are a powerful formalism for modeling the behavior of systems in several disciplines such as robotics, automotive, and feedback mechanical systems. The dynamics are given as linear differential equations and the sensor noise, input uncertainty, and modeling error make up the bounded input signals. For ensuring the safety of such systems, an engineer would simulate the system using different initial states and input signals, and

DISTRIBUTION A. Approved for public release; Distribution unlimited.
(Approval AFRL PA #88ABW-2017-2082, 1 MAY 2017)

check that each simulation trace is safe. While simulations are helpful in developing intuition about the system's behavior, they might miss unsafe behaviors, as the space of valid input signals is vast.

In this paper, we present a technique to perform *simulation-equivalent reachability* and safety verification of linear systems with inputs. That is, given a linear system $\dot{x}(t) = Ax(t) + Bu(t)$, where $x(t)$ is the state of the system and $u(t)$ is the input, we infer the system to be safe if and only if all the discrete time simulations from a given initial set and an input space are safe. We restrict our attention to input signals that are piecewise constant, where the value of the input signal $u(t)$ is selected from a bounded set U every h time units. We consider piecewise constant input signals for two main reasons. First, the space of all input signals spans an infinite-dimensional space and hence is hard to analyze. To make the analysis tractable, we restrict ourselves to piecewise constant signals which can closely approximate continuous signals. Second, our approach is driven by the desire to produce concrete counterexamples if the system has an unsafe behavior. Counterexamples with input signals that are piecewise constant can be easily validated using numerical simulations.

A well-known technique for performing safety verification is to compute the *reachable set*, or its overapproximation. The reachable set includes all the states that can be reached by any trajectory of the linear system with a valid choice of inputs at each time instant. Typically, the reachable set is stored in data structures such as polyhedra [12], zonotopes [14], support functions [17], or Taylor models [8]. For linear systems, the effect of inputs can be exactly computed using the Minkowski sum operation [15]. While being theoretically elegant, a potential difficulty with this method is that Minkowski sum can greatly increase the complexity of the set representation, especially in high dimensions and after a large number of steps. This previously limited its application to reachability methods where Minkowski sum is efficient: zonotopes and support functions.

In this paper, we demonstrate that it is possible to efficiently perform the Minkowski sum operation using a recently-proposed *generalized star* representation and a linear programming (LP) formulation. The advantage of using the generalized star representation is that the reachable set of an n-dimensional linear system (without inputs) can be computed using only $n + 1$ numerical simulations [10], making it scalable and fast. The method is also highly accurate (assuming the input simulations are accurate). Furthermore, it is also capable of generating concrete counterexamples, which is generally not possible with other reachability approaches.

The contributions of this paper are threefold. First, we define the notion of simulation-equivalent reachability and present a technique to compute it using the generalized star representation, Minkowski sum, and linear programming. Second, we present two optimizations which improve the speed and scalability of basic approach. The first optimization leverages a key property of Minkowski sum and decomposes the linear program that needs to be solved for verification. The second optimization uses the result of the LP at each step to warm-start the computation at the next time step. Third, we perform a thorough evaluation, examining the effect of each of the optimizations and comparing the approach versus existing reachability tools on large benchmark systems ranging from 9 to 10914 dimensions. The new approach successfully analyzes models over two orders of magnitude larger than the state-of-the-art tools, finds previously-unknown errors in the benchmark systems, and generates highly-accurate counter-example traces with complex input sequences that are externally validated to drive the system to an error state.

2 Preliminaries

2.1 Problem Definition

A system consists of several continuous variables evolving in the space of $\mathbb{R}^n$. States denoted as x and vectors denoted as v lie in $\mathbb{R}^n$. A set of states $S \subseteq \mathbb{R}^n$.

The set $S_1 \oplus S_2 \triangleq \{x_1 + x_2 \,|\, x_1 \in S_1, x_2 \in S_2\}$ is defined to be the Minkowski sum of sets S_1 and S_2 , where $x_1 + x_2$ is addition of n-dimensional points.

The behaviors of control systems with inputs are modeled with differential equations. In this work, we consider *time-invariant affine systems with bounded inputs* given as:

$$\dot{x}(t) = Ax(t) + Bu(t); \quad u(t) \in U, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ are constant matrices, and $U \subseteq \mathbb{R}^m$ is the set of possible inputs. We assume that the system has m inputs and hence the input function $u(t)$ is given as $u : \mathbb{R}_{\geq 0} \to \mathbb{R}^m$. Given a function $u(t)$, a trajectory of the system in Equation 1 starting from the initial state x_0 can be defined as a function $\xi(x_0, u, t)$ that is a solution to the differential equation, $\frac{d}{dt}\xi(x_0, u, t) = A\xi(x_0, u, t) + Bu(t)$. If $u(t)$ is an integrable function, the closed form expression to the unique solution to the differential equation is given as:

$$\xi(x_0, u, t) = e^{At}x_0 + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau. \tag{2}$$

If $u(t)$ is a constant function set to the value of u_0, then we abuse notation and use $\xi(x, u_0, t)$ to represent the trajectory. If the input is a constant 0, we drop the term u and denote the trajectory as $\xi(x, t)$. For performing verification of linear systems, we leverage an important property often called the *superposition principle*. Given a state $x_0 \in \mathbb{R}^n$ and vectors $v_1, v_2, \ldots, v_n \in \mathbb{R}^n$, and $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{R}$, we have

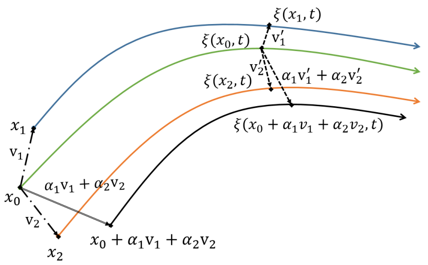

$\xi(x_1,t)$
$\xi(x_0,t)$
v_1'
v_2'
$\xi(x_2,t)$
$\alpha_1 v_1' + \alpha_2 v_2'$
$\xi(x_0+\alpha_1 v_1+\alpha_2 v_2,t)$
x_1
v_1
x_0
$\alpha_1 v_1 + \alpha_2 v_2$
v_2
x_2
$x_0 + \alpha_1 v_1 + \alpha_2 v_2$

Fig. 1: The state reached at time t from $x_0 + \alpha_1 v_1 + \alpha_2 v_2$ is identical to $\xi(x_0, t) + \alpha_1(\xi(x_0 + v_1, t) - \xi(x_0, t)) + \alpha_2(\xi(x_0 + v_2, t) - \xi(x_0, t))$.

$$\xi(x_0 + \sum_{i=1}^{n} \alpha_i v_i, t) = \xi(x_0, t) + \sum_{i=1}^{n} \alpha_i(\xi(x_0 + v_i, t) - \xi(x_0, t)). \tag{3}$$

An illustration of the superposition principle in 2-d is shown in Figure 1.

We refer to the dynamics without the input ($\dot{x} = Ax$) as the autonomous system and the system $\dot{x} = Ax + Bu(t)$ as the system with the inputs. As mentioned in the introduction, we restrict our attention to inputs that are piecewise constant. That is, the value of inputs are updated periodically with time period h. The inputs stay constant for the time duration $[k \times h, (k+1) \times h]$. A *simulation* of such a system records the state of the system at time instants that are multiples of h, the same as the period when the inputs get updated. A formal definition of such a simulation is given in Definition 1.

Definition 1 (Fixed-Step Simulation of a System with Inputs). *Given an initial state x_0, a sequence of input vectors u, and a time period h, the sequence $\rho(x_0, u, h) = x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} x_2 \xrightarrow{u_2} \dots$, is a **$(\mathbf{x_0}, \mathbf{u}, \mathbf{h})$-simulation** of a system in Equation 1 if and only if all $u_i \in U$, and for each x_{i+1} we have that x_{i+1} is the state of the trajectory starting from x_i when provided with constant input u_i for h time units, $x_{i+1} = \xi(x_i, u_i, h)$. Bounded-time variants are called (x_0, u, h, T)-simulations. We drop u to denote simulations where no input is provided.*

For simulations, h is called the *step size* and T is called *time bound*. The set of states *encountered* by a (x_0, u, h)-simulation is the set of states in $\mathbb{R}^n$ at the multiples of the time step, $\{x_0, x_1, \dots\}$. Given a simulation $\rho(x_0, h, T)$ as defined in Definition 1, we can use the closed-form solution in Equation 2 and substitute u to obtain the relationship between x_i and x_{i+1},

$$x_{i+1} = e^{Ah} x_i + G(A, h) B u_i \tag{4}$$

where $G(A, h) = \sum_{i=0}^{\infty} \frac{1}{(i+1)!} A^i h^{i+1}$.

Definition 2 (Simulation-Equivalent Reachable Set). *Given an initial set Θ and time step h, the **simulation-equivalent reachable set** is the set of all states that can be encountered by any (x_0, u, h)-simulation starting from any $x_0 \in \Theta$, for any valid sequence of input vectors u. This can also be extended to a time-bounded version.*

We define the system to be safe if and only if the simulation-equivalent reachable set and the unsafe set Δ are disjoint. In this paper, the initial set Θ, the space of allowed inputs U, and the unsafe set Δ are bounded polyhedra (sets of linear constraints).

2.2 Generalized Star Sets and Reachable Set Computation

Definition 3 (Generalized Star Set). *A **generalized star set** (or **generalized star**, or simply **star**) Θ is a tuple $\langle c, V, P \rangle$ where $c \in \mathbb{R}^n$ is the center, $V = \{v_1, v_2, \dots, v_m\}$ is a set of m vectors in $\mathbb{R}^n$ called the basis vectors, and $P : \mathbb{R}^m \to \{\top, \bot\}$ is a predicate. The basis vectors are arranged to form the star's $n \times m$ basis matrix. The set of states represented by the star is given as*

$$[\![\Theta]\!] = \{x \mid x = c + \Sigma_{i=1}^{m} \alpha_i v_i \text{ such that } P(\alpha_1, \dots, \alpha_m) = \top\}.$$

Sometimes we will refer to both the tuple Θ and the set of states $[\![\Theta]\!]$ as Θ. In this work, we restrict the predicates to be a conjunction of linear constraints, $P(\alpha) \triangleq C\alpha \leq d$ where, for p linear constraints, $C \in \mathbb{R}^{p\times m}$, α is the vector of m-variables i.e., $\alpha = [\alpha_1, \ldots, \alpha_m]^T$, and $d \in \mathbb{R}^{p\times 1}$.

This definition is slightly more general than the one used in existing work [10], where stars were restricted to having no more than n basis vectors. This generalization is important when computing the input effects as a star. Any set given as a conjunction of linear constraints in the standard basis can be immediately converted to the star representation by taking the center as the origin, the n basis vectors as the standard orthonormal basis vectors, and the predicate as the original conjunction linear condition with each x_i replaced by α_i. Thus, we can assume the set of initial states Θ is given as a star.

Reachable Set Computation With Stars: Due to the superposition principle, simulations can be used to accurately compute the time-bounded simulation-equivalent reachable set for an autonomous (no-input) linear system from any initial set Θ [10,6]. For an n dimensional system, only $n + 1$ simulations are necessary. The algorithm, described more fully in Appendix A, takes as input an initial set Θ, a simulation time step h, and time bound $k \times h$, and returns a tuple $\langle\Theta_1, \Theta_2, \ldots, \Theta_k\rangle$, where the sets of states that all the simulations starting from Θ can encounter at time instances $i \times h$ is given as Θ_i.

In brief, the algorithm first generates a discrete time simulation $\rho_0 = s_0[0], s_0[1], \ldots, s_0[k]$ of the system from the origin at each time step. Then, n simulations are performed from the state which is unit distance along each orthonormal vector from the origin, $\rho_j = s_j[0], s_j[1], \ldots, s_j[k]$. Finally, the reachable set at each time instant $i \times h$ is returned as a star $\Theta_i = \langle c_i, V_i, P\rangle$ where $c_i = \rho_0[i]$, $V_i = \{v_1, v_2, \ldots, v_n\}$ where $v_j = \rho_j[i] - \rho_0[i]$, and P is the same predicate as in the initial set Θ. This accuracy of this approach is dependent on the errors in the $n + 1$ input simulations, which in practice can often be made arbitrarily small.

Given an unsafe set Δ as a conjunction of linear constraints, discrete-time safety verification can be performed by checking if the intersection of each $\Theta_i \cap \Delta$ is nonempty where Θ_i is the reachable set at time instant $i \times h$. This can be done by solving for the feasibility of a linear program which encodes (1) the relationship between the standard orthonormal basis and the star's basis (given by the basis matrix), (2) the linear constraints on the star's basis variables α from the star's predicate, and (3) the linear conditions on the standard basis variables from the unsafe states Δ. An example reachable set computation using this algorithm, and the associated LP formulation, is provided in Appendix B.

If the LP is feasible, then there exists a point in the star that is unsafe. Further, the trace from the initial states to the unsafe states can be produced by taking the basis point from the feasible solution ($\alpha = [\alpha_1, \ldots, \alpha_n]^T$) and multiplying it by the basis matrix the star in every preceding time step. This will give a sequence of points (one for every multiple of the time step) in the standard basis, starting from the initial set up to a point in the unsafe states.

2.3 Reachability of Linear Systems with Inputs

The reachable set of a linear system with inputs can be exactly written as the Minkowski sum of two sets, the first accounting for the autonomous system (no-input) and the second accounting for the effect of inputs [18]. From Equation 4, this relationship is expressed as $\Theta_{i+1} = e^{Ah}\Theta_i \oplus G(A,h)BU$. Here, the $e^{Ah}\Theta_i$ represents the evolution of the autonomous system and $G(A,h)BU$ represents the effect of inputs for the time duration h. Representing $\mathcal{U} = G(A,h)BU$ and expanding the above equation, we have

$$\Theta_{i+1} = e^{A(i+1)h}\Theta \oplus e^{A(i)h}\mathcal{U} \oplus e^{A(i-1)h}\mathcal{U} \oplus \ldots \oplus e^{Ah}\mathcal{U} \oplus \mathcal{U}. \quad (5)$$

Here $e^{A(i+1)h}\Theta$ is the set reached by the autonomous system and the rest of the summation represents the accumulated effects of the inputs.

The performance of the algorithm based on Equation 5 critically depends on the efficiency of the Minkowski sum operation of the set representation that is used. In particular, representations such as polytopes were dismissed because of the high complexity associated with computing their Minkowski sum [18], driving researchers to instead use zonotopes and support functions.

3 Reachability of Linear Systems With Inputs using Stars

In this section, we first present the basic approach for adapting Equation 5 for use with generalized stars. We then present two optimizations which greatly improve the efficiency of the approach when used for safety verification.

3.1 Basic Approach

Recall that the expression for the reachable set given in Equation 5 is

$$\Theta_i = e^{Ai\times h}\Theta \oplus e^{A(i-1)\times h}\mathcal{U} \oplus e^{A(i-2)\times h}\mathcal{U} \oplus \ldots \oplus e^{Ah}\mathcal{U} \oplus \mathcal{U}.$$

where $e^{Ai\times h}\Theta$ is the reachable set of the autonomous system and the remainder of the terms characterize the effect of inputs. Consider the j^{th} term in the remainder, namely, $e^{A(j-1)\times h}\mathcal{U}$. This term is exactly same as the reachable set of states starting from an initial set $\mathcal{U}$ after $(j-1)\times h$ time units, and evolving according to the autonomous dynamics $\dot{x} = Ax$.

Furthermore, the set $\mathcal{U} = G(A,h)BU$ can be represented as a star $\langle c, V, P\rangle$ with m basis vectors, for an n-dimensional system with m inputs. This is done by taking the origin as the center c, the set $G(A,h)B$ as the star's $n\times m$ basis matrix V, and using the linear constraints U as the predicate P, replacing each input u_i with α_i. With this, a simulation-based algorithm for computing the reachable set with inputs is given in Algorithm 1, which makes use of the AutonomousReach function that is the autonomous (no-input) reachability technique described in Section 2.2.

```
input : Initial state: $\Theta_0$, influence of inputs: $\mathcal{U}_0 = G(A,h)BU$, time bound: $k \times h$
output: Reachable states at each time step: $(\Omega_0, \Omega_1 \ldots, \Omega_k)$
1 $\langle \Theta_1, \Theta_2, \ldots, \Theta_k \rangle \leftarrow$ AutonomousReach($\Theta_0, h, k \times h$) ;
2 $\langle \mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_k \rangle \leftarrow$ AutonomousReach($\mathcal{U}_0, h, k \times h$) ;
3 $S \leftarrow \mathcal{U}_0$;
4 for $i = 1$ to $k$ do
5     $\Omega_i \leftarrow \Theta_i \oplus S$;
6     $S \leftarrow S \oplus \mathcal{U}_i$;
7 end
8 return $(\Omega_1 \ldots, \Omega_k)$;
```

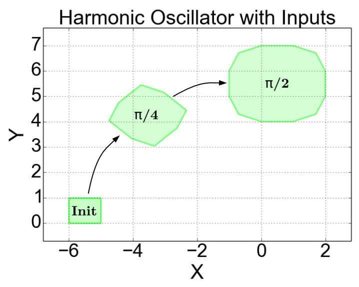
Algorithm 1: The Basic approach computes the reachable set of states at each time step up to time $k \times h$, where AutonomousReach is the reachable set computation technique presented in Section 2.2. For safety verification each of the returned stars should be checked for intersection with the unsafe states using LP.

Algorithm 1, which we refer to as the Basic approach, computes the reachable set of the autonomous part for initial set Θ in line 1 and the effect of the input $\mathcal{U}$ in line 2. The simulation-based AutonomousReach computation avoids the need to compute and multiply by matrix exponential at every iteration. The variable S in the loop from lines 4 to 7 computes the Minkowski sum of $\mathcal{U}_i \oplus \mathcal{U}_{i-1} \oplus \ldots \oplus \mathcal{U}_1 \oplus \mathcal{U}_0$. The correctness of this algorithm follows from the expression for the reachable set given in Equation 5. Note that although the computations of the $\mathcal{U}_i$ sets can be thought of as using an independent call to AutonomousReach, they can be computed more efficiently by reusing to simulations used to compute the Θ_i sets. Finally, Algorithm 1 needs to perform Minkowski sum with stars, for which we propose the following approach:

Minkowski Sum with Stars. Given two stars $\Theta = \langle c, V, P \rangle$ with m basis vectors and $\Theta' = \langle c', V', P' \rangle$ with m' basis vectors, their Minkowski is a new star $\overline{\Theta} = \langle \overline{c}, \overline{V}, \overline{P} \rangle$ with $m + m'$ basis vectors and (i) $\overline{c} = c + c'$, (ii) $\overline{V}$ is the list of $m + m'$ vectors produced by joining the list of basis vectors of Θ and Θ', (iii) $\overline{P}(\overline{\alpha}) = P(\alpha_m) \wedge P'(\alpha_{m'})$. Here $\alpha_m \in \mathbb{R}^m$ denotes the variables in Θ, $\alpha_{m'} \in \mathbb{R}^{m'}$ denotes the variables for Θ', and $\overline{\alpha} \in \mathbb{R}^{m+m'}$ denotes the variables for $\overline{\Theta}$ (with appropriate variable renaming).

Notice that both the number of variables in the star and the number of constraints grow with each Minkowski sum operation. In an LP formulation of these constraints, this would mean that both the number of columns and the number of rows grows at each step in the algorithm. However, even though the constraint matrix is growing in size, the number of non-zero entries added to the matrix at each step is constant, so, for LP solvers that use a sparse matrix representation, this may not be as bad as it first appears.

Example 1 (Harmonic Oscillator with Inputs). Consider a system with dynamics $\dot{x} = y + u_1$, $\dot{y} = -x + u_2$, where u_1, u_2 are inputs in the range $[-0.5, 0.5]$ that can



$$\left(\begin{array}{cccccccc} -1 & 0 & 0.0 & 1.0 & 0.707 & 0.293 & 0.293 & 0.707 \\ 0 & -1 & -1.0 & 0.0 & -0.293 & 0.707 & -0.707 & 0.293 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right) \begin{pmatrix} x \\ y \\ \alpha_1 \\ \alpha_2 \\ u_1 \\ u_2 \\ u_1' \\ u_2' \end{pmatrix} \begin{array}{c} = \\ = \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq \\ \leq \end{array} \begin{pmatrix} 0 \\ 0 \\ 6 \\ -5 \\ 0 \\ 1 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix}$$

Fig. 2: A plot of the simulation-equivalent reachable states for Example 1 using a $\frac{\pi}{4}$ step size, and the associated linear constraints representing the set at time $\frac{\pi}{2}$ (after two steps).

vary at each time step, and the initial states are $x = [-6, -5]$, $y = [0, 1]$. A plot of the simulation-equivalent reachable states of this system is shown in Figure 2 (left). The trajectories of this system generally rotate clockwise over time, although can move towards or away from the origin depending on the values of the inputs. The LP constraints which define the reachable states at time $\frac{\pi}{2}$ are given in Figure 2 (right). Simulations are used to determine the values of the autonomous star's basis matrix (the red encircled values in the matrix). At each time step, the input-free basis matrix gets updated exactly as in the case where there were no inputs. Rows 3-6 in the constraints come from the conditions on the initial states. Additionally, at each step, two columns are added to the LP in order to account for the effects of the two inputs in the model. Rows 7-10 are the conditions on the inputs from the first step, and rows 11-14 are from the second step. The blue dotted values are the each step's input star's basis matrix, $\mathcal{U} = G(A, h)B$. The basis matrix of the combined star is the 2 by 6 matrix constructed by combining the matrices of the basis matrices from the autonomous star and each of the input-effect stars, each of which are 2 by 2. Notice that, at each step, both the number of rows and the number of columns in the LP constraints gets larger, although the number of non-zero entries added to the matrix is constant. To extract a counter-example trace to a reachable state, the LP would be solved in order to give specific assignments to each of the variables. The values of x and y would be the final position that is reachable, that could be, for example, minimized or maximized in the LP. Then, α_1 and α_2 indicate the initial starting position, u_1' and u_2' are the inputs to apply at the first step, and u_1 and u_2 are the inputs to apply at the second step.

3.2 Minkowski Sum Decomposition for Efficient Safety Verification

Algorithm 1 computes the reachable set Ω_i at each step [18] in line 5 based on Equation 5 for safety verification with respect to an unsafe set Δ. As noted in Section 3.1, the number of variables in Ω_i increases linearly with i and hence for high dimensional systems, checking whether $\Omega_i \cap \Delta = \emptyset$ using linear programming becomes increasingly difficult. To improve the scalability of safety verification, we observe that it is not necessary to compute Ω_i in the star representation and then check for intersection with Δ. Consider a specific case where Δ is defined as a half-space $v \cdot x \geq a$. Checking safety of Ω_i with respect to Δ is equivalent to computing the maximum value of the cost function $v \cdot x$ over the set Ω_i and checking if $\max_{v \cdot x}(\Omega_i) \geq a$. As Ω_i is a Minkowski sum of several sets, computing $\max_{v \cdot x}(\Omega_i)$ is equivalent to computing the maximum value of $v \cdot x$ for each of the sets in the Minkowski sum and adding these maximum values. This property is described rigorously in Proposition 1. This observation was first made in [18] for zonotopes and the authors leverage this property to avoid computing Minkowski sum for safety verification. In this paper, we explicitly state the property for Minkowski sum of any two sets and any linear cost function and note that it is independent of the data structure used for representing the sets.

Proposition 1. *If $S = S_1 \oplus S_2$, then $\max_v(S) = \max_v(S_1) + \max_v(S_2)$, where $\max_v$ is defined as the maximum value of the cost function $v \cdot x$, v is any n-dimensional vector and $v \cdot x$ is the dot product.*

Proof. Let p_1 and p_2 be states in S_1 and S_2 respectively which maximize the dot product. From the definition of Minkowski sum, $p_1 + p_2 = p \in S$. Since p is in S, $\max_v(S) \geq p \cdot v$, and therefore $\max_v(S) \geq p \cdot v = p_1 \cdot v + p_2 \cdot v = \max_v(S_1) + \max_v(S_2)$.

For the other inequality, let q be the point in S which maximizes the dot product. By the definition of Minkowski sum, there exist two points $q_1 \in S_1$ and $q_2 \in S_2$ where $q = q_1 + q_2$. Therefore, $q \cdot v = q_1 \cdot v + q_2 \cdot v$. Notice now that $\max_v(S_1) \geq q_1 \cdot v$ and $\max_v(S_2) \geq q_2 \cdot v$, and so we can substitute to get $\max_v(S) = q \cdot v \leq \max_v(S_1) + \max_v(S_2)$.

Since both inequalities must hold, $\max_v(S) = \max_v(S_1) + \max_v(S_2)$. □

As $\Omega_i = \Theta_i \oplus \mathcal{U}_{i-1} \oplus \mathcal{U}_1 \oplus \mathcal{U}$ (Equation 5), it follows from Proposition 1 that, for safety verification in the case of unsafe set $\Delta \triangleq v \cdot x \geq a$, it suffices to compute $\max_{v \cdot x} \mathcal{U}_j$ for all j and $\max_{v \cdot x} \Theta_i$; add these values; and compare the summation with a. In the general case, the unsafe states Δ may be a conjunction of half-planes, say $(v_1 \cdot x \geq a_1) \wedge (v_2 \cdot x \geq a_2) \wedge \ldots \wedge (v_l \cdot x \geq a_l)$. For such instances, if $\Omega_i \cap \Delta \neq \emptyset$, it is a necessary condition that $\forall j, \max_{v_j \cdot x}(\Omega_i) \geq a_j$. Thus, we do not compute the full Minkowski sum representation of Ω_i until the above necessary condition is satisfied. Informally, we perform *lazy computation* of Ω_i.

Additionally, all the $\mathcal{U}_j$'s appearing in Minkowski sum formulation of Ω_i also appear in Ω_{i+1}. Therefore, we keep a running sum of the maximum values for each $\mathcal{U}_j$ for all the linear functions and check the necessary conditions for all Ω_i. Only after checking the necessary conditions, we computing the Minkowski sum and formulate the linear program for checking $\Omega_i \cap \Delta = \emptyset$. We refer to this approach, shown in Algorithm 2, as the Decomp method.

The algorithm starts by extracting each constraint hyperplane's normal direction and value in lines 1 and 2. Lines 9 and 10 compute the maximum over all the normal directions in the sets Θ_i and $\mathcal{U}_i$. The tuple μ accumulates the maximum over all inputs up to the current iteration, whereas λ is recomputed

```
   input : Initial state: $\Theta$, influence of inputs: $\mathcal{U}$, time bound: $k \times h$, unsafe set: $\Delta$
   output: SAFE or UNSAFE
1  $\langle v_1, v_2, \dots, v_l \rangle \leftarrow$ NormalDirections($\Delta$);
2  $\langle a_1, a_2, \dots, a_l \rangle \leftarrow$ NormalValues($\Delta$);
3  $\langle \Theta_1, \Theta_2, \dots, \Theta_k \rangle \leftarrow$ AutonomousReach($\Theta, h, k \times h$);
4  $\langle \mathcal{U}_1, \mathcal{U}_2, \dots, \mathcal{U}_k \rangle \leftarrow$ AutonomousReach($\mathcal{U}, h, k \times h$);
5  $\mathcal{U}_0 \leftarrow \mathcal{U}$ ;
6  $\Theta_0 \leftarrow \Theta$ ;
7  $\mu \leftarrow \langle 0, 0, \dots, 0 \rangle$;
8  for $i = 1$ to $k$ do
9      $\lambda \leftarrow \langle \max_{v_1}(\Theta_i), \max_{v_2}(\Theta_i), \dots, \max_{v_l}(\Theta_i) \rangle$;
10     $\mu \leftarrow \langle \mu[1] + \max_{v_1}(\mathcal{U}_{i-1}), \mu[2] + \max_{v_2}(\mathcal{U}_{i-1}), \dots, \mu[l] + \max_{v_l}(\mathcal{U}_{i-1}) \rangle$;
11     if $\forall j \in \{1, \dots, l\},\ \lambda[j] + \mu[j] \geq a_j$ then
12         if $\Theta_i \oplus \mathcal{U}_{i-1} \oplus \mathcal{U}_{i-2} \oplus \dots \oplus \mathcal{U}_0 \cap \Delta \neq \emptyset$ then
13             return UNSAFE;
14         end
15     end
16 end
17 return SAFE;
```

Algorithm 2: The Decomp algorithm uses Minkowski sum decomposition to avoid needing to solve the full LP at each iteration.

at each step based on the autonomous system's current generalized star. Only if all the linear constraints can exceed their constraint's value (the check on line 11), will the full LP be formulated and checked on line 12.

3.3 Optimizing with Warm-Start Linear Programming

In this section, we briefly outline the core principle behind warm-start optimization and explain why it is effective in safety verification using generalized stars. Consider a linear program given as `maximize` $c^T y$, `Subject to:` $Hy \leq g$ where $y \in \mathbb{R}^m$. To solve this LP, we use a two-phase simplex algorithm [19]. First, the algorithm finds a feasible solution and second, it traverses the vertices of the polytope defined by the set of linear conditions in the m-dimensional space to reach the optimal solution. Finding the feasible solution is performed using slack variables and the traversal among feasible solutions is done by relaxing and changing the set of active constraints. The time taken for the simplex algorithm to terminate is directly proportional to the time taken to find a feasible solution and the number of steps in the traversal from the feasible solution to the optimal solution.

The warm-start optimization allows the user (perhaps using a solution to an earlier linear program) to explicitly specify the initial set of active constraints. Internally, the slack variable associated with these constraints are assigned to be 0. This can speed-up the running time of simplex in two ways. If the set of active constraints gives a feasible solution, then the first phase of simplex terminates without any further computation. Second, if the user-provided active constraints correspond to a feasible solution is close to the optimal solution, the steps needed during the second phase are reduced.

Similar to [6], we have used warm-start optimization in this paper for improving the efficiency of safety verification. Warm-start optimization works in this context because we use the generalized star representation for sets. Consider two consecutive reachable sets of the autonomous system $\Theta_i = \langle c_i, V_i, P \rangle$ and $\Theta_{i+1} = \langle c_{i+1}, V_{i+1}, P \rangle$ represented as generalized stars. Consider solving a linear program for maximizing a cost function $v \cdot x$ over Θ_i and Θ_{i+1}. These two stars differ in the value of the *center* and the set of *basis vectors*, but the *predicate* remains unchanged. Moreover, if we choose have small time-steps, the difference between the values of center and basis vectors is also small. Therefore, the set of active constraints in the predicate P is often identical for the optimal solutions. Even in the cases where the active constraints are not identical, the corresponding vertices are often close, reducing the work needed.

In this paper, we feed the active constraints of the first linear program i.e., maximizing $v \cdot x$ over Θ_i as a warm-start to the second linear program, i.e., maximizing $v \cdot x$ over Θ_{i+1}. We apply the same principle for the input stars $\mathcal{U}_i$ and $\mathcal{U}_{i+1}$. Hence, the warm-start optimization can also be used together with the Minkowski sum decomposition optimization.

4 Evaluation

We encoded the techniques developed in this paper into a tool named Hylaa (HYbrid Linear Automata Analyzer). Using this tool, we first evaluate the effects of each of the proposed optimizations on the runtime of reachability computation. Then, we evaluate the overall approach on a benchmark set of systems ranging from 9 to 10914 coupled continuous variables. All of the measurements are run on a desktop computer running Ubuntu 16.04 x64 with an Intel i7-3930K processor (6 cores, 12 threads) running at 3.5 GHz with 24 GB RAM.

4.1 Optimization Evaluation

We examine the effects of each of our proposed optimizations for computing reachability for linear-time invariant systems with inputs. We compare the Basic algorithm from Section 3.1, against the Decomp approach described in Section 3.2. The Warm method is the enhancement of the Basic approach with warm-start optimization as described in Section 3.3, and Hylaa is the approach used in our tool, which uses both Minkowski sum decomposition and LP warm-start. For reference, we also include measurements for the no-input system (NoInput), which could be considered a lower-bound for the simulation-based methods if the time to handle the inputs could be eliminated completely. Finally, we compared the approach with both optimizations (Hylaa) with other state-of-the-art tools which handle time-varying inputs. We used the support function scenario in the SpaceEx [13] tool (SpaceEx), and the linear ODE mode with Taylor model order 1 (fastest speed) in the Flow* [8] tool (Flow*).

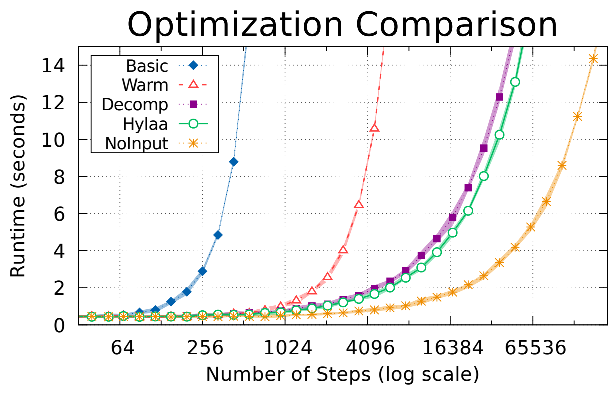


Fig. 3: The performance of the generalized star and linear programming approach for reachability computation (Basic), is improved by the warm-start linear programming optimization (Warm), but not as much as when the Minkowski sum decomposition optimization is used (Decomp). Combining both optimizations works even better (Hylaa). The reachability time for the system without inputs (NoInput) is a lower bound.

For evaluation, we use the harmonic oscillator with input system, as described in Example 1. Recall that the full LP grows at each step both in terms of the number of columns and the number of rows. The unsafe condition used is $x + y \geq 100$, which is never reached but must be checked at each step.

We varied the number of steps in the problem by changing the step size and keeping the time bound fixed at 2π. We then measured the runtime for each of the methods, recording 10 measurements in each case. Figure 3 shows the results with both the average runtime (lines), and the runtime ranges over all 10 runs (slight shaded regions around each line). Each optimization is shown to improve the performance of the method, with Minkowski sum decomposition having a larger effect on this example compared with warm-start LP. The fully optimized approach (Hylaa) is not too far from the lower bound computed by ignoring the inputs in the system (NoInput). In the tool comparison shown in Figure 4, the approach is shown to be comparable to the other reachability tools, and outperforms both SpaceEx and Flow* when the model has a large number of steps.

4.2 High-Dimensional Benchmark Evaluation

We evaluated the proposed approach using a benchmark suite for reachability problems for large-scale linear systems [21]. This consists of nine reachability benchmarks for linear systems with inputs of various sizes, taken from "diverse fields such as civil engineering and robotics." 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. For all the sys-

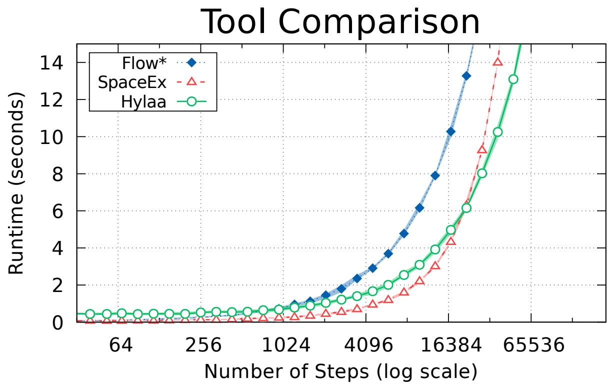


Fig. 4: The performance of our optimized generalized star representation reachability approach (Hylaa) is compared to state-of-the art tools which use support functions (SpaceEx) and Taylor Models (Flow*) as the state set representation. As the number of steps gets larger, our approach surpasses the other tools on this example.

tems, we used a step size of 0.005 and the original time bound of 20. The results are shown in Table 1.

Our Hylaa tool was able to successfully verify or disprove invariant safety conditions for all of the models, including the `MNA5` model, which has 10914 dimensions. To the best of our knowledge, this is significantly (about two orders of magnitude) larger than any system that has been verified without using any type of abstraction methods to reduce the system's dimensionality.

We also attempted to run SpaceEx (0.9.8f) and Flow* (2.0.0) on the benchmark models[3]. With Flow*, in order to successfully run the `Motor` (9 dimensions) benchmark, we needed to use a smaller step size (0.0002), and set the Taylor Model order parameter to 20. For SpaceEx, we used the support function scenario which only requires a step size parameter. In the `Motor` (9 dimensions) benchmark, however, this required us to halve the step size to 0.0025 in order for SpaceEx to be able to prove the unsafe error states were not reachable with the original safety condition. Consistent with the earlier analysis [21], SpaceEx's computation only succeeded for the `Motor` (9 dimensions) and `Building` (49 dimensions) benchmarks.

We also ran the benchmarks toggling the different optimizations, and could show cases where warm-start greatly improves performance, and even outperforms Decomp. In the `Beam` model, for instance, Warm completes the safe case in about 4 minutes, whereas Decomp takes 12 minutes (using both optimizations takes about 1.4 minutes).

[3] Performance comparisons are difficult since runtime depends on the parameters used. Since the submission of this work, the `Building` model has been used as part of a reachability tools competition [1], in which both SpaceEx and Flow* participated. Using the parameters from the competition, which were hand-tuned by the tool authors, is likely to produce better runtimes than what we achieved in this paper.

Table 1: Benchmark Results. Stars (*) indicate original specifications.

Model	Dims	Unsafe Error Condition	Tool	Time (s)	Safe?	CE Error (Abs/Rel)	CE Time
Motor*	9	$x_1 \in [0.35, 0.4] \wedge x_5 \in [0.45, 0.6]$	Hylaa	2.3s	✓	-	-
			SpaceEx	6.8s	✓	-	-
			Flow*	13m19s	✓	-	-
Motor	9	$x_1 \in [0.3, 0.4] \wedge x_5 \in [0.4, 0.6]$	Hylaa	0.4s		$2.5{\cdot}10^{-7}/2.4{\cdot}10^{-7}$	0.04
			SpaceEx	9.6s		-	-
			Flow*	14m11s		-	-
Building*	49	$x_{25} \geq 0.006$	Hylaa	2.7s	✓	-	-
			SpaceEx	59.8s	✓	-	-
Building	49	$x_{25} \geq 0.004$	Hylaa	0.9s		$4.4{\cdot}10^{-8}/1.8{\cdot}10^{-6}$	0.07
			SpaceEx	59.2s		-	-
PDE*	85	$y_1 \geq 12$	Hylaa	3.8s	✓	-	-
PDE	85	$y_1 \geq 10.75$	Hylaa	1.2s		$1.5{\cdot}10^{-8}/6.7{\cdot}10^{-8}$	0.025
Heat*	201	$x_{133} \geq 0.1$	Hylaa	11.8s	✓	-	-
Heat	201	$x_{133} \geq 0.02$	Hylaa	10.1s		$5.8{\cdot}10^{-8}/1.6{\cdot}10^{-7}$	15.67
ISS	271	$y_3 \notin [-0.0007, 0.0007]$	Hylaa	1m28s	✓	-	-
ISS*	271	$y_3 \notin [-0.0005, 0.0005]$	Hylaa	1m23s		$8.5{\cdot}10^{-6}/1.3{\cdot}10^{-5}$	13.71
Beam	349	$x_{89} \geq 2100$	Hylaa	1m23s	✓	-	-
Beam*	349	$x_{89} \geq 1000$	Hylaa	1m19s		$2.0{\cdot}10^{-5}/4.2{\cdot}10^{-9}$	16.045
MNA1*	579	$x_1 \geq 0.5$	Hylaa	4m4s	✓	-	-
MNA1	579	$x_1 \geq 0.2$	Hylaa	3m49s		$1.9{\cdot}10^{-6}/4.9{\cdot}10^{-7}$	16.555
FOM	1007	$y_1 \geq 185$	Hylaa	4m10s	✓	-	-
FOM*	1007	$y_1 \geq 45$	Hylaa	1m7s		$1.0{\cdot}10^{-6}/5.6{\cdot}10^{-7}$	0.29
MNA5*	10914	$x_1 \geq 0.2 \vee x_2 \geq 0.15$	Hylaa	6h23m	✓	-	-
MNA5	10914	$x_1 \geq 0.1 \vee x_2 \geq 0.15$	Hylaa	37m27s		$1.4{\cdot}10^{-6}/1.8{\cdot}10^{-6}$	1.92

One important difference to keep in mind is that SpaceEx and Flow* overapproximate reachability at all times, which is slightly different than simulation-equivalent reachability computed by Hylaa. Unlike Hylaa, they may catch error cases that occur between time steps. The cost of this is that there may also be false-positives; they do not produce counterexample traces when a system is deemed unsafe. For example, choosing too small of a Taylor Model order or too large of a time step in Flow* can easily result in all states, $[-\infty, \infty]$, to be computed as potentially reachable for all variables, which is not useful.

Another important concern is the accuracy of result. Since the proposed approach uses numerical simulations that may not be exact as well as floating-point computations and a floating-point LP solver, there may be errors that accumulate in the computation. To address the issue of accuracy, we examine the counterexamples produced when the error condition is reachable.

Upon finding that an unsafe error state is reachable, Hylaa uses the approach described in Example 1 to determine the initial point and inputs to use

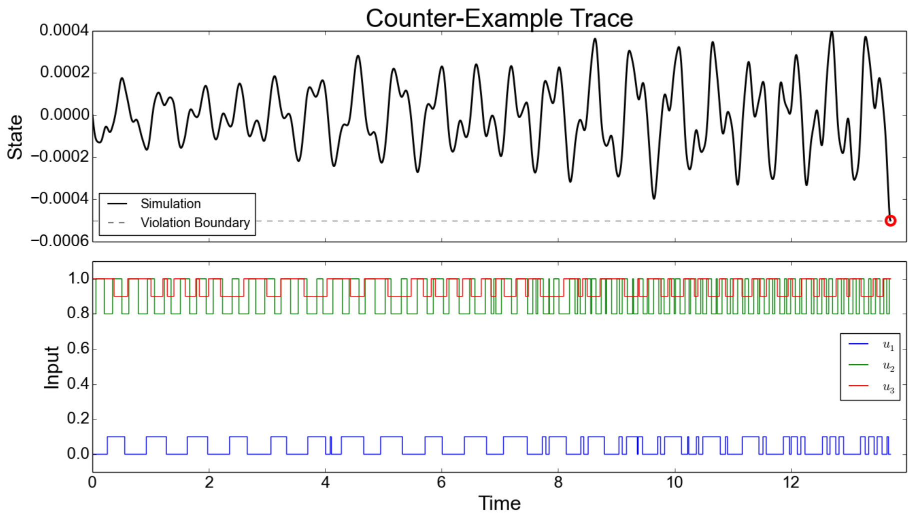


Fig. 5: Hylaa found a counterexample trace in the `ISS` model, generating a very specific set of inputs to use at each point in time. The post-verification analysis (external, high-accuracy simulation) confirms that the found violation is a true violation of the system.

at each step in order to reach the unsafe state. It creates a Python script using this start point and inputs to perform a simulation with high accuracy parameters in order to try to reproduce the counterexample error trace. We perform this check for each of the benchmark systems where an error state is found, and compute the l_2-norm of the difference between the expected final point given by Hylaa and the post-analysis high-accuracy simulation point. The values in the `CE Error` column in Table 1 indicate that the counterexamples are highly-accurate on all the models, both in terms of absolute and relative error.

Consider the structural model of component 1R (Russian Service Module) of the International Space Station (`ISS` model, 271 dimensions). The high-accuracy simulation of the counterexample trace found this system is shown in Figure 5. Here, the final point in the post-analysis simulation and the Hylaa's predicted final point differ, in terms of l_2-norm, by around 10^{-5}. The figure shows the state of the output value (top) and the values of the three inputs at each point in time, computed by Hylaa. This is an extremely complicated set of inputs that would be difficult to find using random simulations.

We attempted to use a falsification tool on this system to try find the violation. A falsification tool [3,9,20] performs stochastic optimization to minimize the difference between simulation runs and the violation region. We ran S-Taliro [3] on this system for 2000 simulations, which took 4.5 hours, but it did not find a violation.

Exhaustive testing for this case would require checking simulations from each of the corner points of the initial states (270 of the dimensions can be in an initial interval range), multiplied by 8 combinations of choices of the 3 inputs at each of the 2742 steps before a counterexample was found (Hylaa found the counterexample at time 13.71). This would be $2^{270} \cdot 8^{2742} = 3.5 \cdot 10^{2557}$ individual simulations, an unfathomably large number.

In communications with the benchmark authors, the original safety specifications for all the models were chosen based on simulations of the system while holding the inputs constant. For some of the benchmarks, Hylaa was able to find cases where the original safety specification was violated. For example, in the `ISS` model, the shown error trace in Figure 5 violates the original specification. This was a bit unexpected, but possible since we are considering inputs which can vary over time. When we kept the inputs constant in the `ISS` model, no violation was found. In other systems, for example the `Beam` model, the counterexample trace Hylaa finds has the same input values at every time step. This shows the incompleteness (and danger) of pure simulation-based analysis, where the safety specification was derived based on a finite set of simulations that did not include the violation. As far as we are aware, the generalized star reachability approach is the first to find violations in the benchmark's original safety conditions, as well as the first approach to verify systems of this size.

5 Related Work

Early work on handling inputs for linear systems was done for the purpose of creating abstractions that overapproximate behaviors of systems with nonlinear dynamics [4]. In this approach, a bound on the input is used to bloat a ball to account for the effect of any possible input. While sound, such an approach does not give a tight result and could not be used to generate concrete traces. Later, a formulation was given which explicitly used the Minkowski sum operation [15], along with a reachability algorithm based on zonotopes, which is a set representation that can efficiently compute both linear transformation and Minkowski sum. This allowed for more precise tracking of the reachable set of states, although the complexity of the zonotopes grew quadratically with the number of discrete time steps performed, so a reduction step was used to limit the expansion of the order of the zonotopes, leading to overapproximation. An important improvement removed this overapproximation by using two zonotopes, one to track the time-invariant system and one to track the effects of inputs [18]. The two zonotopes could be combined at any specific time step in the computation to perform operations on the reachable set at that time instant such as guard checks, but the time-elapse operation was done on the two zonotopes separately. A similar approach was applied for a support functions representation rather than zonotopes [17], which also allows for efficient Minkowski sum. The methods proposed in this paper also use this general approach, using the generalized star set data structure rather than zonotopes. The difficulty with this is that generalized star sets are similar to polytopes specified using hyperplane constraints, and the number of hyperplanes necessary to represent the Minkowski sum can become extremely large in high dimensions [11,22]. For systems with nonlinear dynamics, a different method using Taylor models can be used for the time-elapse operation, with inputs given as bounded time-varying uncertainties [7]. This method essentially replaces time-varying parameters by their interval enclosure at every integration step.

Building on the time-elapse operation, a hybrid automaton reachability algorithm needs to perform intersections with guard sets. For zonotopes, this can be done by performing conversions to/from polytopes [16,2], although this process may introduce overapproximation error. For support functions, this can be done by converting to/from template polyhedra [13]. In non-linear reachability computation with Taylor models, guard intersections can leverage domain-contraction and range-overapproximation (converting to other representations such as polytopes) [8]. Although not explored in this work, we believe this operation can be done using generalized star sets, without conversions that may introduce error.

6 Conclusion

In this paper, we described a new approach for computing reachability for linear systems with inputs, using a generalized star set representation and linear programming. Our approach is simulation-equivalent, which means that we can detect an unsafe state is reachable if and only if a fixed-step simulation exists to the unsafe states. Furthermore, upon reaching an unsafe state, a counter-example trace is generated for the system designer to use.

The proposed method has unprecedented scalability. On the tested benchmarks, we successfully analyzed a system with 10914 dimensions, whereas the current state-of-the-art tools did not succeed on any model larger than 48 dimensions. Such large models frequently arise by discretizing partial differential equations (PDEs). For example, a 100×100 grid over a PDE model results in a 10,000 dimensional model. Thus, we believe the proposed approach opens the door to the computer-aided verification of PDE models with ranges of possible initial conditions, inputs, and uncertainties.

References

1. M. Althoff, S. Bak, D. Cattaruzza, X. Chen, G. Frehse, R. Ray, and S. Schupp. ARCH-COMP category report: Continuous and hybrid systems with linear continuous dynamics. In *4th Applied Verification for Continuous and Hybrid Systems Workshop (ARCH)*, 2017.
2. M. Althoff, O. Stursberg, and M. Buss. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear Analysis: Hybrid Systems*, 4(2):233–249, 2010.
3. Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2011.
4. E. Asarin, T. Dang, and A. Girard. Reachability analysis of nonlinear systems using conservative approximation. In *In Oded Maler and Amir Pnueli, editors, Hybrid Systems: Computation and Control, LNCS 2623*, pages 20–35. Springer-Verlag, 2003.

5. 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.
6. 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.
7. X. Chen. *Reachability Analysis of Non-Linear Hybrid Systems Using Taylor Models*. PhD thesis, RWTH Aachen University, Mar. 2015.
8. 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.
9. A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*, pages 167–170. Springer, 2010.
10. 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.
11. P. Florian. Optimizing reachabiliy analysis for non-autonomous systems using ellipsoids. Master's thesis, RWTH Aachen University, Germany, 2016.
12. G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *HSCC*, pages 258–273, 2005.
13. 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.
14. A. Girard. Reachability of uncertain linear systems using zonotopes. In M. Morari and L. Thiele, editors, *Hybrid Systems: Computation and Control*, LNCS. Springer, 2005.
15. A. Girard. Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.
16. A. Girard and C. L. Guernic. Zonotope/hyperplane intersection for hybrid systems reachability analysis, 2008.
17. 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.
18. A. Girard, C. Le Guernic, and O. Maler. *Efficient Computation of Reachable Sets of Linear Time-Invariant Systems with Inputs*, pages 257–271. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.
19. J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.
20. T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancić, A. Gupta, and G. J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, pages 211–220. ACM, 2010.
21. 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.
22. T. Zaslavsky. *Facing up to Arrangements: Face-Count Formulas for Partitions of Space by Hyperplanes: Face-count Formulas for Partitions of Space by Hyperplanes*. American Mathematical Society: Memoirs of the American Mathematical Society. American Mathematical Society, 1975.

A Reachability of Autonomous Systems using Stars

Here, we expand on the algorithm for computing simulation-equivalent reachability for an autonomous (no-input) system. Given an initial set Θ as a conjunction of linear predicates P, the algorithm generates one simulation from the center *origin* and one simulation from state $ortho_i$ where $ortho_i$ is unit distance along the i^{th} vector in the orthonormal basis. The reachable set at a time instance is computed as a new star $\langle c', V', P \rangle$ where the new center and the basis vectors are calculated based on the simulations, but the predicate remains the same. This simulation-based approach is also extremely easy to parallelize. It has been shown to be quite scalable, and is capable of analyzing an certain affine 1000-dimensional systems in 10-20 minutes [5,6].

```
input  : Initial set $\Theta \triangleq P$, time step: $h$, time bound: $k \cdot h$
output: Reachable states at each time step: $(\Theta_0, \Theta_1 \ldots, \Theta_k)$
1  $Sim_0 \leftarrow \rho(origin, h, k \cdot h)$;
2  for $j = 1$ to $n$ do
3      $Sim_j \leftarrow \rho(ortho_j, h, k \cdot h)$;
4  end
5  for $i = 0$ to $k$ do
6      $c_i \leftarrow Sim_0[i]$;
7      for $j = 1$ to $n$ do
8          $v_j \leftarrow Sim_j[i] - Sim_0[i]$;
9      end
10     $V_i \leftarrow \{v_1, \ldots, v_n\}$;
11     $\Theta_i \leftarrow \langle c_i, V_i, P \rangle$;
12 end
13 return $(\Theta_0, \Theta_1 \ldots, \Theta_k)$;
```

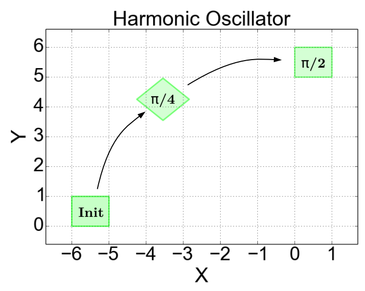
Algorithm 3: Computes the simulation-equivalent reachable set up to time $k \cdot h$ from $n + 1$ simulations, for linear system without inputs.

The procedure is given in Algorithm 3. In the algorithm, Sim_0 represents the simulation starting from the origin and Sim_j represents the simulation starting at unit distance along the j^{th} orthonormal vector. Given a simulation Sim, $Sim[i]$ represents the i^{th} state in the simulation, i.e, the state reached after $i \cdot h$ time units. Algorithm 3 computes the reachable set of the set Θ at time instance $i \cdot h$, returned as Θ_i as a generalized star with the center as $Sim_0[i]$, the j^{th} basis vector as $Sim_j[i] - Sim_0[i]$ and the same predicate P as the initial set. Observe that for all Θ_i, the predicate in the star representation is the same, only the center and the basis vectors change. Applying the Equation 2, for the closed loop system without the inputs, we have that $\Theta_i = e^{A i \cdot h}\Theta$. The correctness of this algorithm is due to the superposition principle of linear systems [10].

B Example of Autonomous System Reachability

We go through an example computation using the autonomous (no-input) reachability algorithm, and provide the associated LP formulation.

Example 2 (Harmonic Oscillator). Consider the 2-d harmonic oscillator with dynamics $\dot{x} = y$, $\dot{y} = -x$, and initial states $x = [-6, -5]$, $y = [0, 1]$. The trajectories of this system rotate clockwise around the origin. A plot of the simulation-equivalent reachable states of this system and the LP formulation at $\frac{\pi}{4}$ is shown in Figure 6. Given these constraints, linear programming can quickly determine if unsafe states, provided as a conjunction of linear constraints, intersect with the reachable states. As described above, simulations are used to determine the values of the basis matrix (the red encircled values in the matrix), which gets updated at each time step, while the rest of the constraints remains unchanged. Rows 3-6 in the constraints come from the conditions on the initial states (the predicate in the initial state star). The initial basis matrix is $\left(\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right)$, where each column is the difference between a concrete simulation and the origin simulation. Since this is a 2-dimensional system ($n = 2$), 3 ($n+1$) simulations are needed, one from the origin, one from $\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$, and one from $\left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)$. In this case, the simulation from the origin always stays at $\left(\begin{smallmatrix} 0 \\ 0 \end{smallmatrix}\right)$. After $\frac{\pi}{4}$ time, the simulation from state $\left(\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right)$ goes to $\left(\begin{smallmatrix} 0.707 \\ -0.707 \end{smallmatrix}\right)$ and the simulation from state $\left(\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right)$ goes to $\left(\begin{smallmatrix} 0.707 \\ 0.707 \end{smallmatrix}\right)$. Thus, the basis matrix at time $\frac{\pi}{4}$, which is the one shown the figure, is $\left(\begin{smallmatrix} 0.707 & 0.707 \\ -0.707 & 0.707 \end{smallmatrix}\right)$. At time $\frac{\pi}{2}$, the basis matrix in the constraints would be $\left(\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right)$.



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

Fig. 6: Plot of the simulation-equivalent reachable states for the system in Example 2 with a step size of $\frac{\pi}{4}$, and the associated LP formulation at $\frac{\pi}{4}$. The red circled values are the star's basis matrix, which changes at each step. Rows 3-6 come from the initial state constraints. Additional rows could be added to check for intersection with the unsafe states.