On Generating A Variety of Unsafe Counterexamples for Linear Dynamical Systems [★]

Manish Goyal and Parasara Sridhar Duggirala [*]

[*] *Department of Computer Science and Engineering, University of Connecticut, Storrs, USA. (e-mails: manish.goyal@uconn.edu, psd@uconn.edu)*

Abstract: Counterexamples encountered in formal verification are typically used as evidence for violation of specification. They also play a crucial role in CEGAR based techniques, where the counterexample guides the refinements to be performed on the abstractions. While several scalable techniques for verification have been developed for safety verification of hybrid systems, less attention has been paid to extracting the various types of counterexamples for safety violations. Since these systems are infinite state systems, the number of counterexamples for safety violations are potentially infinite and hence searching for the right counterexample becomes a challenging task. In this paper, we present a technique for providing various types of counterexamples for a safety violation of the linear dynamical system. More specifically, we develop algorithms to extract the longest counterexample — the execution that stays in the unsafe set for most time, and deepest counterexample — the execution that ventures the most into the unsafe set in a specific direction provided by the user.

Keywords: Safety verification, linear dynamical systems, counterexample, dynamic programming, linear programming.

1. INTRODUCTION

Counterexamples currently play very important role in the domain of model checking. In the early days of model checking, counterexamples that are obtained as a by product during the model checking process, were regarded as important artifacts due to their practical relevance. They provided intuition to the system designer about the reason why the system does not satisfy the specification. The introduction of Counter Example Guided Abstraction Refinement (CEGAR) Clarke et al. (2000, 2002) changed the role of counterexamples from a mere feature to an algorithmic tool. In CEGAR, the counterexample acts as a primary guide to restricting the space of the possible refinements. Techniques to uncover deep bugs, which would otherwise take a long time to uncover were developed in Bradley (2011, 2012). More recently, in the domain of automated synthesis, CEGIS Raman et al. (2015), counterexamples from verification are used for synthesizing the system that satisfies the specification.

In dynamical and hybrid systems, counterexamples are crucial for verification, primarily because the state space is uncountable. Thus, providing an “important” counterexample would provide a better insight into the system behavior and help the designer deal with the uncountable state space. In verification of hybrid systems domain, while a lot of attention was paid for extracting counterexamples for hybrid systems with timed and rectangular dynamics, not many approaches have been developed to generating various counterexamples for such systems. This is primarily because most of the model checking approaches in hybrid system verification focus on computing overapproximations. As a side effect, an additional effort is required for generating counterexamples.

Our goal to generate various types of counterexamples stems from the desire to provide intuition to the control system designer during the process of controller synthesis. To a control designer, not all counterexamples for safety violation are equivalent. For example, the control designer would want to observe counterexample trajectory that *stayed for the longest duration* in the unsafe set. Similarly, the control designer might want to explore the counterexample that *goes the farthest along a specific direction* in the unsafe set. Currently, none of the existing model checkers provides us with a technique for generating such counterexamples. We will illustrate the utility of the counterexample through an example.

Example 1. Consider the classic case of a regulation control problem where the control designer wants to make the error between the observation and the desired value to be 0. The typical profile of an execution after applying PID controller would look similar to Figure 1. In such cases, the control designer is most concerned about the amount of overshoot that occurred and also the duration for which the value of error was above the threshold. Current verification techniques, although inform the designer whether the overshoot happened or not, but do not provide her with enough support to find out the maximum value of overshoot and the duration of the overshoot.

[★] The work has been supported and funded by UTC Institute for Advanced Systems Engineering.

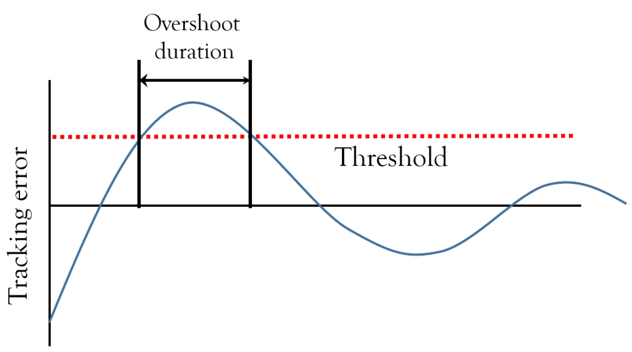


Fig. 1. Classical case of overshoot in stabilizing controllers.

In this paper, we enhance the model checking tool HyLAA to generate various types of counterexamples. More specifically, we present two types of counterexamples (longest and deepest respectively) that we believe are most important for the control designer, and present algorithms for generating these counterexamples. The primary insight in this paper is that to obtain important counterexamples for linear dynamical system, one need **not** search in the functional space of trajectories, but rather in the state space of initial states. For this purpose, our approach leverages the *superposition principle* property of the trajectories and uses the symbolic representation of generalized stars.

This paper builds on our previous work of computing reachable set Duggirala and Viswanathan (2016). To generate the counterexamples, our algorithm reuses the artifacts generated during the model checking process. Our experimental evaluation suggests that the cost of generating these counterexamples while usually being less than the safety verification time, is dependent on the duration of overlap between the reachable sets and the unsafe set.

Related Work: Generating specific type of counterexamples has been an active research topic in model checking. In the domain of hybrid systems, many CEGAR based approaches pursue various notions of counterexamples Fehnker et al. (2005a); Dierks et al. (2007); Clarke et al. (2003); Alur et al. (2003, 2006); Prabhakar et al. (2013); Fehnker et al. (2005b); Duggirala and Mitra (2011); Ratschan and She (2005); Sankaranarayanan and Tiwari (2011); Tiwari (2012); Frehse et al. (2006). Most of them are restricted to the domain of timed and rectangular hybrid systems. The current state of the art tools such as SpaceEx Frehse et al. (2011) and HyLAA Bak and Duggirala (2017a) spit out the counterexample that violates the safety specification at the first time step and the counterexample that violates at the last time step respectively.

The approach presented in this paper bears some resemblance to the CEGIS based approach described in Raman et al. (2015). Here, the verification condition that the system satisfies an STL specification is formulated as a satisfiability problem for mixed-integer liner program (MILP). If the specification is violated, one can investigate the results of MILP to obtain the counterexample. In Ghosh et al. (2016), the authors extend Raman et al. (2015) and provide an intuition/reason for the system failing to satisfy the specification.

2. PRELIMINARIES

States and vectors are elements in $\mathbb{R}^n$ are denoted as x and v. Inner product between two vectors is denoted as $v_1 \cdot v_2$. Given a sequence $seq = s_1, s_2, \ldots$, the i^{th} element in the sequence is denoted as $seq[i]$. In this work, we use the following mathematical notation of a linear dynamical system.

Definition 2. A *linear dynamical system* H is defined to be a tuple $\langle X, Flow \rangle$ where:

$X \subseteq \mathbb{R}^n$ is the state space of the system.
$Flow$ determines the system dynamics using an affine differential equation $\dot{x} = Ax + B$.

The *initial set of states* Θ is a subset of X and the state $x_0 \in \Theta$ is called an *initial state*.

Definition 3. Given a linear dynamical system and an initial state x_0, the system trajectory that describes the evolution of the state with time is denoted as $\xi_H(x_0, t)$. That is, ξ_H is the solution of the initial value problem for the linear differential equation. The closed form expression for the trajectory is given as $\xi_H(x_0, t) = e^{At}x_0 + \int_0^t e^{A(t-\tau)} B d\tau$. Here $e^{At} = I + \frac{At}{1!} + \frac{(At)^2}{2!} + \ldots$.

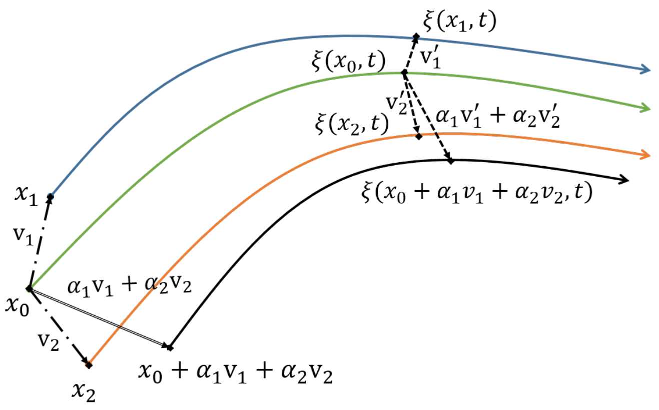
The set of states encountered by all executions that conform to the above semantics is called the *reachable set.* Even though the trajectories has a closed form expression, the trajectory might not be finitely computable. For analysis, system designers often employ numerical ODE solders that give a finite time approximation of the trajectory. Since this paper deals with finding counterexamples, we focus on counterexamples that can be generated using a specific simulation engine for linear hybrid automata. More specifically, we use the simulation engine that is described in Bak and Duggirala (2017b).

Definition 4. A sequence $\xi_H(x_0, h, \infty) = x_0, x_1, x_2, \ldots$, is a (x_0, h)-simulation of the dynamical system H from the initial set Θ if and only if $x_0 \in \Theta$ and each pair (x_i, x_{i+1}) corresponds to a continuous trajectory such that a trajectory starting from x_i would reach x_{i+1} after exactly h time units. We drop the subscript when it is clear from context. Bounded-time variants of these simulations, with time bound $k \times h$, are called (x_0, h, k)-simulations. For simulations, h is called the *step size* and k is called *time bound.* These are denoted as $\xi(x_0, h, k)$.

Observation On Simulation Algorithm: The simulation engine given in Definition 4 simulates the system only at discrete time instances. It is very hard to finitely compute and represent the execution trace for the entire continuous time interval, and hence we consider this a valid assumption. This is the closest we can get to such representation; by reducing the time step, one can further get arbitrarily close to the execution.

We now define the safety property for simulations and for a set of initial states

Definition 5. A given simulation $\xi_H(x_0, h, \infty)$ is said to be safe with respect to an unsafe set of states U if and only if $\forall x_i \in \xi_H(x_0, h, \infty)$, $x_i \notin U$. Safety for bounded time simulations are defined similarly.



$\xi(x_1,t)$
$\xi(x_0,t)$
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
$x_0 + \alpha_1 v_1 + \alpha_2 v_2$
x_2

Fig. 2. Observe that the state reached at time t from $x_0 + \alpha_1 v_1 + \alpha_2 v_2$ is identical to $\xi(x_0,t) + \alpha_1(\xi(x_0 + v_1,t) - \xi(x_0,t)) + \alpha_2(\xi(x_0+v_2,t) - \xi(x_0,t))$.

Definition 6. A dynamical system H with initial set Θ, time bound k, and unsafe set of states U is said to be safe with respect to its simulations if all simulations starting from Θ for bounded time k are safe.

Our goal in this paper is to generate two types of counterexamples namely the deepest and the longest counterexamples respectively. We give the definitions as follows.

Definition 7. Given a dynamical system H with an initial set Θ, time bound k, step h, unsafe set U, and direction v, the depth of a counterexample ξ in direction v, denoted as $depth(\xi, v) = \max\{v \cdot x_i \mid x_i \in \xi \wedge x_i \in U\}$.

The counterexample ξ with the maximum value of depth is called the deepest counterexample.

Definition 8. Given a dynamical system H with initial set Θ, time bound k, step h, and unsafe set U, a counterexample is said to be of *length* l if and only if $\exists$ consecutive states $x_i, x_{i+1}, \ldots, x_{i+l-1}$ in ξ such that $\forall i \leq j \leq i+l-1, x_j \in U$.

The counterexample of the maximum length is called the longest counterexamples.

For computing the deepest and longest counterexamples, we use the simulation equivalent reachable set approach that is presented in Duggirala and Viswanathan (2016); Bak and Duggirala (2017b).

2.1 Superposition principle, Generalized Stars, and Simulation-equivalent Reachable Set

We now present some of the building blocks in computation of the reachable set (from Bak and Duggirala (2017b)). There are three main aspects of the reachable set computation. First, is the superposition principle, second is the generalized star representation that is used for representing the set of reachable states and finally, the building block reachable set algorithm for the simulation-equivalent reachable set computation (Bak and Duggirala (2017b)).

Definition 9. Given any initial state x_0, vectors $v_1, \ldots, v_m$ where $v_i \in \mathbb{R}^n$, scalars $\alpha_1, \ldots, \alpha_m$, the trajectories of linear differential equations ξ always satisfy
$\xi(x_0 + \Sigma_{i=1}^m \alpha_i v_i, t) = \xi(x_0,t) + \Sigma_{i=1}^m \alpha_i(\xi(x_0+v_i,t) - \xi(x_0,t))$.

We exploit the superposition property of linear systems in order to compute the simulation-equivalent reachable set of states. An illustration of the superposition principle for two vectors is shown in Figure 2. Before describing the algorithm for computing the reachable set, we finally introduce the data structure called a *generalized star* that is used to represent the reachable set of states.

Definition 10. A *generalized star* (or simply star) Θ is a tuple $\langle c, V, P\rangle$ where $c \in \mathbb{R}^n$ is called the *center*, $V = \{v_1, v_2, \ldots, v_m\}$ is a set of m $(\leq n)$ vectors in $\mathbb{R}^n$ called the *basis vectors*, and $P : \mathbb{R}^n \to \{\top, \bot\}$ is a predicate.

A generalized star Θ defines a subset of $\mathbb{R}^n$ as follows.

$$[\![\Theta]\!] = \{x \mid \exists \bar{\alpha} = [\alpha_1, \ldots, \alpha_m]^T \text{ such that } x = c + \Sigma_{i=1}^n \alpha_i v_i \text{ and } P(\bar{\alpha}) = \top\}$$

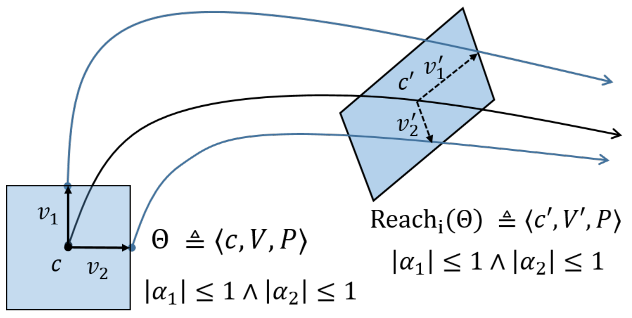
Sometimes we will refer to both Θ and $[\![\Theta]\!]$ as Θ. Additionally, we refer to the variables in $\bar{\alpha}$ as basis variables and the variables x as orthonormal variables. Given a valuation of the basis variables $\bar{\alpha}$, the corresponding orthonormal variables are denoted as $x = c + V \times \bar{\alpha}$.

Similar to Bak and Duggirala (2017b), we consider predicates P which are conjunctions of linear constraints. This is primarily because linear programming is very efficient when compared to nonlinear arithmetic. We therefore harness the power of these linear programming algorithms to improve the scalability of our approach.

Example 11. Consider a set $\Theta \subset \mathbb{R}^2$ given as $\Theta \triangleq \{(x,y) \mid x \in [4,6], y \in [4,6]\}$. The given set Θ can be represented as a generalized star in multiple ways. One way of representing the set is given as $\langle c, V, P\rangle$ where $c = (5,5)$, $V = \{[0,1]^T, [1,0]^T\}$ and $P \triangleq -1 \leq \alpha_1 \leq 1 \wedge -1 \leq \alpha_2 \leq 1$. That is, the set Θ is represented as a star with center $(5,5)$ with vectors as the orthonormal vectors in the Cartesian plane and predicate where the components along the basis vectors are restricted by the set $[-1,1] \times [-1,1]$.

Reachable Set Computation For Linear Dynamical Systems Using Simulations: Owing to space limitations, we briefly describe the algorithm for computing simulation-equivalent reachable set, this is primarily done to present some crucial observations which will later be used in the algorithms for generating specific counterexamples. Longer explanation and proofs for these observations and algorithms is available in prior work Duggirala and Viswanathan (2016); Bak and Duggirala (2017b).

At its crux, the algorithm exploits the superposition principle of linear systems and computes the reachable states using a generalized star representation. For an n-dimensional system, this algorithm requires at most $n+1$ simulations. Given an initial set $\Theta \triangleq \langle c, V, P\rangle$ with $V = \{v_1, v_2, \ldots, v_m\} (m \leq n)$, the algorithm performs a simulation starting from c (denoted as $\xi(c,h,k)$), and $\forall 1 \leq j \leq n$, performs a simulation from $c + v_j$ (denoted as $\xi(c + v_j, h, k)$). For a given time instance $i \cdot h$, the reachable set denoted as $Reach_i(\Theta)$ is defined as $\langle c_i, V_i, P\rangle$ where $c_i = \xi(c,h,k)[i]$ and $V_i = \langle v_1', v_2', \ldots, v_m'\rangle$ where



v_1'
c'
v_2'
v_1
c
v_2
$\Theta \triangleq \langle c, V, P \rangle$
$|\alpha_1| \leq 1 \wedge |\alpha_2| \leq 1$
$\mathrm{Reach_i}(\Theta) \triangleq \langle c', V', P \rangle$
$|\alpha_1| \leq 1 \wedge |\alpha_2| \leq 1$

Fig. 3. Illustration of the reachable set using sample simulations and generalized star representation. Notice that in the star representation, the predicate that defines the reachable set is same as that of the initial set.

$\forall 1 \leq j \leq m, v_j' = \xi(c + v_j, h, k)[i] - \xi(c, h, k)[i]$. Notice that the predicate does not change for the reachable set, but only the center and the basis vectors are changed.

An illustration of this reachable set computation is shown in Figure 3. Here, as the system is 2-dimensional, a total number of three simulations are performed, one from *center* c and one from $c + v_1$ and one from $c + v_2$. the reachable set after time $i \cdot h$ is given as the star with center $c' = \xi(c, h, k)[i]$, basis vectors $v_1' = \xi(c+v_1, h, k)[i] - \xi(c, h, k)[i]$, and $v_2' = \xi(c+v_2, h, k)[i] - \xi(c, h, k)[i]$ and the same predicate P as the given in the initial set.

The reachable set algorithm computeSimEquivReach returns the reachable set in the form of a sequence. The first node of the sequence is the initial set Θ. Each node in this sequence corresponds to a generalized star S_i of the form $S_i \triangleq \langle c_i, V_i, P_i \rangle$ corresponding to the set of states visited at a discrete step. Each node in the reach sequence has one *continuous successor* that corresponds to the evolution in this mode for one step. We denote the sequence form of the reachable set from an initial state Θ as *ReachSeq*.

Definition 12. Given an initial set Θ, bound T, and step h, and the simulation equivalent reachable set as *ReachSeq*, given a star $S_i \in ReachSeq$, we call a sequence of stars $\sigma = R_1, R_2, \ldots, R_m$ a *chain* starting from S_i if and only if $R_1 = S_i$ and $\forall 2 \leq j \leq m, R_j$ is a continuous successor of R_{j-1}.

Remark 13. Given a star $S_i \triangleq \langle c_i, V_i, P_i \rangle$ in *ReachSeq* and a valuation $\bar{\alpha}$ of the basis vectors such that $P_i(\bar{\alpha}) = \top$, one can use this valuation of basis variables to generate the trace starting from the initial set Θ to P_i. We call the procedure that generates this execution as $getExecution(\bar{\alpha}, S_i, ReachSeq)$. This observation is crucial for our algorithm as it reduces the problem of finding the counterexample from the functional space of trajectories ξ to the valuation of the basis variables $\bar{\alpha}$. We solve the problem of generating the appropriate counterexample by selecting the appropriate value of $\bar{\alpha}$.

Assumptions: Similar to the assumptions in our earlier work Bak and Duggirala (2017b), we assume that ODE solvers give the exact result. While theoretically unsound, such an assumption is adopted due to its practicality. Second, we use floating-point arithmetic in our computations and do not track the errors by floating point arithmetic.

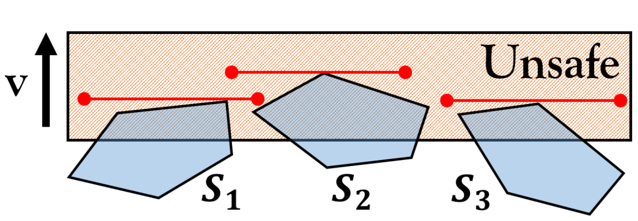


Fig. 4. Figure illustrating the deepest counterexample in the direction specified.

A user concerned about the inaccuracy of numerical simulation can either use validated simulations (Computer Assisted Proofs in Dynamic Groups [1]) or compute the linear ODE solution as a matrix exponential to an arbitrary degree of precision. The algorithms presented are oblivious to the simulation engine used.

3. DEEPEST COUNTEREXAMPLE

In this section, we will present the algorithm that would return the deepest counterexample for a safety specification and a direction. We will illustrate the way to obtain the deepest counterexample using Figure 4.

Suppose that in the ReachSeq computation, there are three stars S_1, S_2, and S_3 that overlap with the unsafe set U. Given a direction v, the procedure to compute the deepest counterexample would be the following. (1) For each of the stars S_i, compute the maximum depth $depth_i$ of star S_i as $max\ v \cdot x$ with $x \in S_i \cap U$. (2) Select the star S_j with maximum value of $depth_j$. (3) Extract the corresponding value of basis variables $\bar{\alpha}$ which achieves the maximum depth and extract the corresponding execution. The correctness of the algorithm trivially follows from Definition 7 and the correctness of the simulation-equivalent reachable set. The algorithm is presented formally in Algorithm 1

```
input : Initial Set: $\Theta$ and the simulation equivalent
        reachable sequence $ReachSeq$, direction $v$,
        Unsafe set $U$
output: Counterexample $ce$ with maximum depth
1  $depth_{max} \leftarrow -\infty$; $depthStar \leftarrow \perp$; $ce \leftarrow \perp$;
2  for each star $S_i$ in $ReachSeq$ do
3      if $S_i \cap U \neq \emptyset$ then
4          $OptProb_i \leftarrow \max \{v \cdot x\}$ given $[x \in S_i \cap U]$;
5          $depth_i \leftarrow solution(OptProb_i)$;
6          if $depth_i > depth_{max}$ then
7              $depth_{max} \leftarrow depth_i$;
8              $\bar{\alpha}_{max} \leftarrow getBasisVariables(OptProb_i)$;
9              $depthStar \leftarrow S_i$;
10         end
11     end
12 end
13 if $depth_{max} \neq -\infty$ then
14     $ce \leftarrow getExection(\bar{\alpha}_{max}, depthStar, ReachSeq)$;
15 end
16 return $ce$;
```
Algorithm 1: Algorithm that computes the deepest counterexample with respect to a direction v in the unsafe set.

The main loop of the algorithm in lines 2 - 12 iterates through all the stars in the reachable set given in the

[1] http://capd.ii.uj.edu.pl/index.php

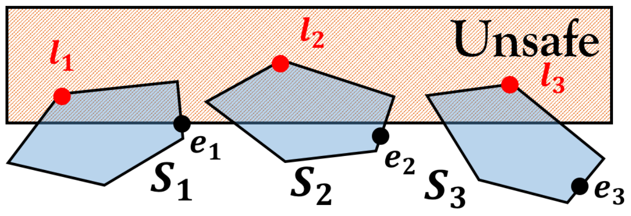


Fig. 5. Figure illustrating the longest counterexample.

sequence format as *ReachSeq* and selects the stars that overlap with the unsafe set U. The optimization problem for maximizing the value of the function $v \cdot x$ for the overlap with the unsafe set is generated in line 4, which is then solved in line 5. If the maximum value of depth computed in line 5 is greater than the current maximum value (lines 6 - 10), then the maximum value of depth is updated, the value of basis variables corresponding to the optimal solution is stored, and the star corresponding to the maximum depth is also stored. In line 14, the execution corresponding to the maximum depth is computed using the value of $\bar{\alpha}$ and returned.

4. LONGEST COUNTEREXAMPLE

In this section, we will describe the algorithm for obtaining the counterexample that spends the longest contiguous time in the unsafe set. We illustrate the problem of finding the longest counterexample through an illustration given in Figure 5. Consider the three consecutive stars in the reachable set S_1, S_2, and S_3 having overlap with the unsafe set as shown. If one picks the state $e_1 \in S_1$ as shown, then the *post* states of e_1, denoted as e_2 and e_3 in the figure respectively, do not lie in the unsafe set. However, if one picks the state $l_1 \in S_1$ as shown, then the *post* states of l_2 and l_3 lie in the unsafe set.

The key insight for the generation of longest counterexamples is that one has to select the *appropriate* state which visits the maximum number of overlaps between the unsafe set and the reachable set. In this instance, any state $x_1 \in S_1$ such that $x_1 \in S_1 \cap U$, with its successors x_2, x_3 such that $x_2 \in S_2 \cap U$, and $x_3 \in S_3 \cap U$ is the appropriate choice.

For finding such a state, we perform constraint propagation (similar to the invariant constraint propagation in Bak and Duggirala (2017b)). That is, we identify the constraints C on the basis variables $(\bar{\alpha})$ such that $\forall \bar{\alpha}$ such that $C(\bar{\alpha}) = \top$, we have, $x_1 = c_1 + V \times \bar{\alpha} \in S_1 \cap U$, $x_2 = c_2 + V_2 \times \bar{\alpha} \in S_2 \cap U$, and $x_3 = c_3 + V_3 \times \bar{\alpha} \in S_3 \cap U$.

To extract these set of constraints, we convert the unsafe set U into the center and basis vectors of each of the stars S_1, S_2, and S_3. hence $S_i \cap U = \langle c_i, V_i, P_i \wedge Q_i \rangle$. From Remark 13, we know that the set of states that reach $\langle c_i, V_i, P_i \wedge Q_i \rangle$ originate from $\langle c_0, V_0, P_i \wedge Q_i \rangle$. Hence, the set of states that would visit all the intersections of the unsafe set should originate from $\langle c_0, V_0, P_1 \wedge Q_1 \wedge P_2 \wedge Q_2 \wedge P_3 \wedge Q_3 \rangle$. Hence, if the set of constraints $P_1 \wedge Q_1 \wedge P_2 \wedge Q_2 \wedge P_3 \wedge Q_3$ is satisfiable, then the corresponding trajectory corresponding to the basis variables that satisfies these constraints visits the unsafe set in all of the stars S_1, S_2, and S_3.

Building on the above discussion, the algorithm to compute the longest counterexample would iterate as follows. We first consider contiguous sequences of stars $S_1, S_2, \ldots, S_m$ that overlap with the unsafe set U. We then compute the set of constraints C such that if C is satisfiable, then, there exists a trajectory that stays in the unsafe set for at least m duration. We find the longest sequence of stars such that the corresponding constraint C is satisfiable and provide the corresponding counterexample trace. This procedure is formally defined in Algorithm 2

```
input  : Initial Set: $\Theta$ and the simulation equivalent
         reachable sequence $ReachSeq$, Unsafe set $U$
output: Counterexample $ce$ that spends longest time in
         $U$
1  $length_{max} \leftarrow -\infty$; $lengthStar \leftarrow \perp$; $ce \leftarrow \perp$;
2  for each star $S_i$ in $ReachSeq$ do
3      if $S_i \cap U \neq \emptyset$ then
4          for each sequence of stars $\sigma$ starting with $S_i$ do
5              Transform $U$ into $\langle c_i, V_i, Q_i \rangle$ where
                 $\sigma[i] \triangleq \langle c_i, V_i, P_i \rangle$;
6              $\mathcal{C}_\sigma \leftarrow \bigwedge_{i=1}^{|\sigma|} Q_i \wedge P_i$;
7              if $\mathcal{C}_\sigma$ is feasible and $|\sigma| > length_{max}$ then
8                  $length_{max} \leftarrow |\sigma|$;
9                  $\bar{\alpha}_{len} \leftarrow feasible(\mathcal{C}_\sigma)$;
10                 $lengthStar \leftarrow S_i$;
11             end
12         end
13     end
14 end
15 if $length_{max} \neq -\infty$ then
16     $ce \leftarrow getExection(\bar{\alpha}_{len}, lengthStar, ReachSeq)$;
17 end
18 return $ce$;
```

Algorithm 2: Algorithm that computes the counterexample that stays in the unsafe set U for the longest contiguous duration.

The algorithm proceeds as follows: the main loop (lines 2 - 14) iterates over all the stars that have a overlap with the unsafe set U. The inner loop (lines 4 - 12) enumerates all the contiguous sequences σ starting with S_i and computes the set of constraints $\mathcal{C}_\sigma$ for the sequence. If the constraints are feasible, then the valuation of the basis variables that satisfies these constraints is stored, the star that begins the start of the longest counterexample is stored, and finally, the length of the longest counterexample is updated. In line 16, the execution corresponding to the longest counterexample is obtained.

Theorem 14. The execution returned by Algorithm 2 returns the longest counterexample.

Proof. We prove this by contradiction. Suppose that the given initial set $\Theta \triangleq \langle c_0, V_0, P_0 \rangle$ and the longest counterexample $\xi = x_0, x_1, \ldots, x_k$ which spends duration m in the unsafe set U. Consider that the states $x_j, x_{j+1}, \ldots, x_{j+m-1}$ in the execution ξ lie in the unsafe set. Additionally, suppose that the execution returned by Algorithm 2 returns a counterexample of length strictly less than m.

From the soundness and completeness result of simulation equivalent reachability Bak and Duggirala (2017b), we have that $\exists$ stars $S_j, S_{j+1}, \ldots, S_{j+m-1}$ in *ReachSeq* such that $\forall j \leq r \leq j+m-1, x_r \in S_r$. Therefore, it should be the case that $\forall r, j \leq r \leq j+m-1, U \cap S_r \neq \emptyset$. Additionally, since the trajectory ξ passes through $U \cap S_r$, it should be the case that $\xi \in \langle c_0, V_0, P_r \wedge Q_r \rangle$ where $S_r \triangleq \langle c_r, V_r, P_r \rangle$ and $U \triangleq \langle c_r, V_r, Q_r \rangle$. Therefore, the constraint $\mathcal{C}_\sigma$ that is computed for the sequence $\sigma = S_j, S_{j+1}, \ldots, S_{j+m-1}$ should be feasible and would be updated as the longest counterexample in lines 7 - 11. Which is a contradiction.

Analysis and Optimizations: In our implementation, we consider only those sequence of stars where all of them overlap with unsafe set U. One of the optimization that can be performed is to adopt a procedure that is similar to a binary search. That is, given a star S_i that overlaps with U, we can generate the maximum contiguous sequence of stars, σ starting from S_i. Instead of generating constraint $\mathcal{C}_\sigma$ incrementally, we can generate constraints in a binary search fashion so as to arrive at the longest feasible subsequence by checking feasibility of only $O(logm)$ constraints generated.

5. EXPERIMENTS

The proposed algorithm has been implemented in a Python based verification tool named HyLAA; although, some of the computational libraries used may be written in other languages. Simulations for reachable sets are performed using `scipy's odeint` function, which can handle stiff and non-stiff differential equations using the FORTRAN library `odepack's lsoda` solver. Linear programming is performed using the `GLPK` library, and matrix operations are performed using `numpy`. The measurements were performed on a system running Ubuntu 16.04 with an 2.70GHz Intel i7-6820HQ CPU with 8 cores and 16 GB RAM.

HyLAA has a provision to perform verification in *aggregation* mode for better performance. However, for our experiments, we run HyLAA in *deaggregation* mode. By default, HyLAA terminates as soon as it finds a counterexample. But, we let the tool run for the entire duration because we perform our analysis on all the stars intersecting with the unsafe set.

The benchmarks for our study are taken from *The Benchmarks of continuous and hybrid systems*[2]. These linear continuous systems benchmarks - Harmonic Oscillator, Vehicle Platoon 1 and Vehicle Platoon 2 are simulated for maximum 100 time steps with step size 0.2 sec.

Each benchmark is originally safe. Since our objective is to find the counterexamples and to classify them, we choose unsafe set in a manner that the reachable set intersects with the unsafe set at multiple time steps. We further adjust the size of the unsafe set and observe that the intersection window of the reachable set with the unsafe set differs proportionally. The values in a duration interval are time steps not the real time values.

For each benchmark, the unsafe set is varied in such a way that with increase in it's size, the number of stars intersecting with the unsafe set increases. This, in turn, may lead to longer counter-examples. The increase in the number of error stars with growth in the unsafe set size translates directly into the counterexample generation time. The reason being every new star adds to the analysis time during counterexample generation.

[2] https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/

Model (Dims)	LCE	Actual Intersection Duration	LCE Duration	Verification, LCE Gen. Time (sec)
Harmonic Oscillator (2)				
SU	[-5.373 0.0]	[5 10]	[6 10]	0.17, 0.01
MU	[-5.0 0.3968]	[4 10][33 44] [66 74]	[33 44]	0.22, 0.03
LU	[-5 0.296]	[3 10][29 49] [59 100]	[59 100]	0.28, 0.17
Vehicle Platoon 1 (15)				
SU	$x_8 = 1.0475$ $x_{2,5} = 1.1$ $x_i = 0.9$	[27 41]	[29 41]	1.82, 0.18
MU	$x_{6,9} = 1.1$ $x_{12} = 1.0761$ $x_i = 0.9$	[27 73]	[27 73]	2.90, 1.40
LU	-do-	[27 100]	[27 100]	3.51, 3.78
Vehicle Platoon 2 (30)				
SU	$x_9 = 0.9223$ $x_5 = 1.0204$ $x_i \in 0.9, 1.1$	[42 48]	[44 48]	4.86, 0.23
MU	$x_{19} = 1.0501$ $x_i \in \{0.9, 1.1\}$	[42, 53]	[45 53]	5.20, 0.43
LU	$x_i = 0.9$	[36 100]	[36 100]	10.73, 9.81

Table 1. Longest Counterexample (LCE) Experiments. Dims is the no. of dimensions, SU, MU, LU are the variations of the unsafe set - **S**mall, **M**edium and **L**arge. LCE is a point in the initial set, simulation from which stays the longest in the unsafe set. x_i represents all the variables whose values are not explicitly given. $x_i \in \{0.9, 1.1\}$ signifies that the value is either 0.9 or 1.1. Actual Intersection Duration is the set of time step intervals when reachable set intersects with the unsafe set. LCE Duration is the interval for the longest counterexample. Verification Time is the time HyLAA takes for verification, LCE Gen. Time is the time it takes to generate the longest counterexample.

The longest counterexample generation can be slower than the overall verification (Refer LU Vehicle Platoon 1 benchmark). As explained in the algorithm in Section 4, the combined number of constraints to be solved can become fairly large, which increases the counterexample generation time. It is worth noticing that the length of a counterexample is not necessarily same as the actual duration for which the reachable set intersects with the unsafe set. This is the direct consequence of our approach: if a system of constraints during certain time interval is not feasible, we prune the list and again check for it's feasibility until we find a solution. Refer Figure 7.

As shown in the Tables 1 and 2, variations in the unsafe set size can provide us varying counterexamples. Similarly, change in the direction while computing the deepest counterexample may give us a different counterexample. Refer Figure 6

Model (Dims)	DCE	Direction	Depth	Verification, DCE Gen. Time (sec)
Harmonic Oscillator (2)				
SU	[-5.459 0.188]	$x_1 = 1$	2.0	0.17, 0.00
MU	[-6 0.8829]	$x_2 = 1$	5.0	0.22, 0.00
LU	[-6 1]	$x_2 = 1$	5.288	0.28, 0.01
Vehicle Platoon 1 (15)				
SU	$x_1 = 1.071$ $x_2 = 0.993$ $x_i \in \{0.9, 1.1\}$	$x_2 = 1$	-0.1825	1.82, 0.11
MU	$x_i \in \{0.9, 1.1\}$	$x_2 = 1$	0.0170	2.9, 0.39
LU	$x_i \in \{0.9, 1.1\}$	$x_2 = 1$	0.0170	3.51, 0.40
Vehicle Platoon 2 (30)				
SU	$x_5 = 0.9005$ $x_{23} = 1.0473$ $x_i \in \{0.9, 1.1\}$	$x_5 = 1$	-0.26347	4.86, 0.12
MU	$x_2 = 0.91327$ $x_4 = 0.9389$ $x_5 = 1.1$, $x_i = 0.9$	$x_5 = 1$	-0.2217	5.20, 0.27
LU	$x_i \in \{0.9, 1.1\}$	$x_5 = 1$	0.01745	10.73, 1.87

Table 2. Deepest Counterexample (DCE) Experiments. DCE is a point in the initial set, simulation from which goes the deepest in the given direction in the unsafe set. $x_i \in \{0.9, 1.1\}$ designates that the variable's value is either 0.9 or 1.1. Verification Time is the time HyLAA takes for verification, DCE Gen. Time is the time it takes to generate the deepest counterexample.

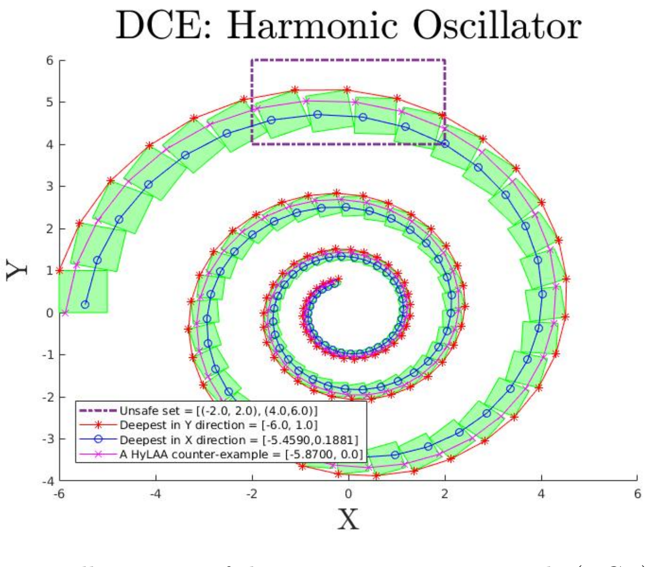


Fig. 6. Illustration of the Deepest counterexample (DCE): The figure corresponds to the SU iteration of Damped Oscillator benchmark. It illustrates two counterexamples that go deepest in different directions. The counterexample in solid red is the deepest in the y direction and the counterexample in solid blue is the deepest in the x direction. The default counterexample by HyLAA is shown in pink. The figure illustrates that change in the depth direction may lead to different counterexamples. The figure also emphasizes that the counterexample originally generated by HyLAA can differ from the deepest counterexample.

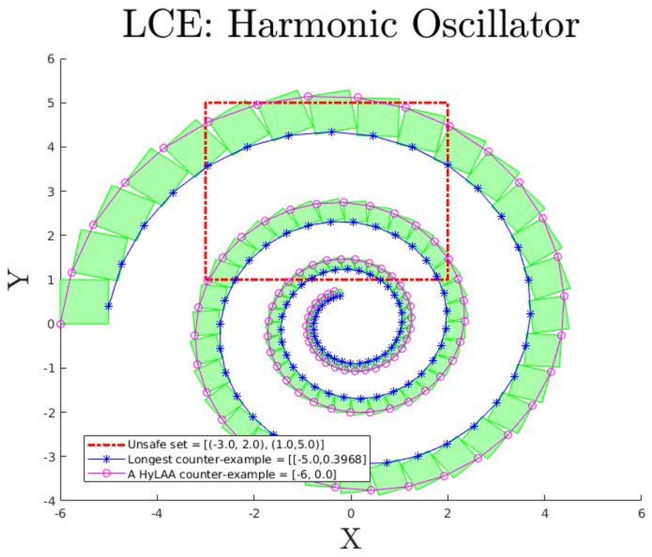


Fig. 7. Illustration of the Longest counterexample (LCE): The figure corresponds to the MU iteration of Harmonic Oscillator benchmark. It graphically represents the system simulation with the help of reachable sets (stars) at discrete time steps. The longest counterexample is shown in blue and the default counterexample generated by HyLAA is shown in pink. It shows that the actual duration of the intersection with the unsafe set ([4 10][33 44][66 74]) can be different from the duration of the longest counterexample ([33 44]). The figure also emphasizes that the counterexample originally generated by HyLAA may differ from the longest counterexample.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we provided an approach for the generating a variety of counterexamples. We defined two new types of counterexamples namely deepest and longest counterexample and developed algorithms for generating the same. The primary insight behind our technique being for generating these counterexamples, one need not search in the space of trajectories, but rather in the space of basis variables that define the initial set. Our technique leverages the superposition principle and the generalized star representation. We note that using linear stars helps us in using linear programming solvers and hence contribute to the efficiency of our procedure. We also observe that the variations in unsafe set size and optimizing direction may generate different counterexamples. To the best of our knowledge, this is the first work that focuses on generating various counterexamples for unsafe violation for linear systems.

The next step is to extend the counterexample generation approach to linear hybrid systems. We would like to generate such counterexamples and use template based methods for automatically generating the set of unsafe sets that are safe for a given initial set. We believe that these counterexamples give a special insight into the behavior of the system and hence can be useful in counterexample guided synthesis techniques for linear systems.

REFERENCES

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

Alur, R., Dang, T., and Ivančić, F. (2003). Counterexample guided predicate abstraction of hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, 208–223. Springer.

Bak, S. and Duggirala, P.S. (2017a). Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, 173–178. ACM.

Bak, S. and Duggirala, P.S. (2017b). Rigorous simulation-based analysis of linear hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 555–572. Springer.

Bradley, A.R. (2011). Sat-based model checking without unrolling. In *Vmcai*, volume 6538, 70–87. Springer.

Bradley, A.R. (2012). Ic3 and beyond: Incremental, inductive verification. In *CAV*, 4.

Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. (2000). Counterexample-guided abstraction refinement. In *Computer Aided Verification*, 154–169.

Clarke, E., Jha, S., Lu, Y., and Veith, H. (2002). Tree-like counterexamples in model checking. In *lics*, 19–29.

Clarke, E., Fehnker, A., Han, Z., Krogh, B., Stursberg, O., and Theobald, M. (2003). Verification of hybrid systems based on counterexample-guided abstraction refinement. In *Tools and Algorithms for the Construction and Analysis of Systems*, 192–207. Springer.

Dierks, H., Kupferschmid, S., and Larsen, K. (2007). Automatic Abstraction Refinement for Timed Automata. In *Proceedings of the International Conference on Formal Modelling and Analysis of Timed Systems*, 114–129.

Duggirala, P.S. and Mitra, S. (2011). Abstraction-refinement for stability. In *Proceedings of 2nd IEEE/ACM International Conference on Cyber-physical systems (ICCPS 2011).* Chicago, IL.

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

Fehnker, A., Clarke, E., Jha, S., and Krogh, B. (2005a). Refining Abstractions of Hybrid Systems using Counterexample Fragments. In *Proceedings of the International Conference on Hybrid Systems Computation and Control*, 242–257.

Fehnker, A., Clarke, E.M., Jha, S.K., and Krogh, B.H. (2005b). Refining abstractions of hybrid systems using counterexample fragments. In *HSCC'2005*, 242–257.

Frehse, G., Krogh, B.H., and Rutenbar, R.A. (2006). Verifying analog oscillator circuits using forward/backward abstraction refinement. In *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings*, DATE '06, 257–262. European Design and Automation Association, 3001 Leuven, Belgium, Belgium. URL http://dl.acm.org/citation.cfm?id=1131481.1131553.

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

Ghosh, S., Sadigh, D., Nuzzo, P., Raman, V., Donzé, A., Sangiovanni-Vincentelli, A.L., Sastry, S.S., and Seshia, S.A. (2016). Diagnosis and repair for synthesis from signal temporal logic specifications. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, 31–40. ACM.

Prabhakar, P., Duggirala, P.S., Mitra, S., and Viswanathan, M. (2013). Hybrid automata-based cegar for rectangular hybrid systems. In *Verification, Model Checking, and Abstract Interpretation*, 48–67. Springer.

Raman, V., Donzé, A., Sadigh, D., Murray, R.M., and Seshia, S.A. (2015). Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, 239–248. ACM.

Ratschan, S. and She, Z. (2005). Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *Hybrid Systems: Computation and Control*, 573–589. Springer.

Sankaranarayanan, S. and Tiwari, A. (2011). Relational abstractions for continuous and hybrid systems. In *Computer Aided Verification*, 686–702. Springer.

Tiwari, A. (2012). Hybridsal relational abstracter. In *Computer Aided Verification*, 725–731. Springer.