Generating Longest Counterexample: On the Cross-roads of Mixed Integer Linear Programming and SMT

Manish Goyal[1], David Bergman[2], and Parasara Sridhar Duggirala[1]

Abstract— **We present a technique for obtaining the longest counterexample – the execution that stays in the unsafe set for the longest (not necessarily contiguous) time, for a safety specification of linear hybrid systems. Given that hybrid systems are infinite state systems, the number of counterexamples for safety violations are potentially infinite. Therefore, searching for the right counterexample is very challenging. We employ two frameworks for solving this problem: first is an *Mixed Integer Linear Program* (MILP) formulation and second is to encode counterexamples using *Satisfiability Module Theory* (SMT) solvers. We evaluate these frameworks on several linear hybrid systems with up to 30 dimensions.**

I. INTRODUCTION

Counterexamples play a critical role in the domain of model checking. Perhaps most importantly, they provide intuition to the system designer regarding the reason why the system does not satisfy a specification. The introduction of *Counter-Example-Guided-Abstraction-Refinement* (CEGAR) [10] changed the role of counterexamples from a mere feature to an algorithmic tool, where the counterexample acts as a primary guide to restrict the space of possible refinements. Further, since the state space is uncountable in hybrid systems, providing an important counterexample can greatly reduce the burden of the system designer and provide a more detailed insight into system behavior.

Designing controllers for hybrid systems that satisfy stability and safety specification is challenging. For proving stability of hybrid systems, one has to come up with either a common [27], [25] or multiple [9], [37] Lyapunov function(s). One can then use a safety verification tool for checking that safety specification is satisfied. Unlike stability, the safety specification of a system would change based on operating condition. For example, a hybrid controller that is originally safe can become unsafe if the safety specification is tightened. In such circumstances, counterexamples provide a unique insight into the behavior of the hybrid system. Additionally, metrics over counterexamples can be used as a proxy for comparing performance of different hybrid controllers. Such metrics can also be useful when the cost function for designing optimal control is non-convex.

Safety specification is satisfied if all trajectories of a system avoid the set of states labelled as *unsafe*. Any trajectory that encounters an unsafe state is called a *counterexample*. In the verification of hybrid systems domain, a few approaches have been developed for generating counterexamples for hybrid systems with linear dynamics. This is primarily because most model checking approaches in affine hybrid system verification focus on computing over-approximation of the reachable set and hence establish the safety specification. Our goal to generate 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 the counterexample trajectory that *stays for the longest duration* in the unsafe set. Currently, none of the existing model checkers are equipped with a technique for generating such counterexamples.

Our approach of generating counterexamples builds on the prior work of computing a simulation-equivalent reachable set [6], which includes the set of states encountered by a simulation algorithm for hybrid systems with linear dynamics. The reachable set computation and counterexample generation algorithms leverage the superposition principle and the generalized star representation [6], [17]. Further, the algorithm presented reuses the artifacts generated during the model checking process.

The contribution of this paper is twofold. First, we provide a new definition for the longest counterexample that generalizes other variants [22], and extend it to linear hybrid systems. Second, we provide two different ways to extract such counterexamples from safety verification artifacts; one based on **S**atisfiability **M**odulo **T**heory (SMT) solvers and the other uses **M**ixed **I**nteger **L**inear **P**rogramming (MILP) solvers. Additionally, we compare and contrast the performance of these approaches on multiple benchmarks. To the best of our knowledge, such a comparison between MILP and SMT for verification has not previously been conducted. We believe that these two methods presented can be used for extracting other type of counterexamples as well.

II. RELATED WORK

Generating specific type of counterexamples has been an active research topic in model checking. In one of the recent works in this area [22], the authors provide techniques to generate longest contiguous and deepest counterexamples for linear dynamical systems. In the domain of hybrid systems, many CEGAR based approaches pursue various notions of counterexamples [19], [13], [3], [30], [16], [33], [34]. Most of them are restricted to the domain of timed and rectangular hybrid systems. The current state-of-the-art tools such as SpaceEx [20] and HyLAA [5] spit out the counterexample

[1]Department of Computer Science, University of North Carolina at Chapel Hill, NC 27516 USA. manishg@cs.unc.edu, psd@cs.unc.edu

[2]School of Business, University of Connecticut at Storrs, CT 06229, USA. david.bergman@uconn.edu

that violates the safety specification at the earliest time and at the latest time, respectively.

Counterexamples also play an important role in *falsification* techniques [18], [15]. Instead of proving that the specification is satisfied, falsification tools like S-Taliro [4] and Breach [14] search for an execution that violates the specification. Given a specification of Cyber-Physical System in Metric Temporal Logic (MTL) [24] or Signal Temporal Logic (STL) [26], falsification tools employ a variety of techniques [28], [2], [38], [12] for discovering an execution that violates the specification. Unlike the counterexamples given in this paper, the counterexamples returned by falsification techniques need not be the longest counterexamples.

In the domain of automated synthesis, the *Counterexample Guided Inductive Synthesis* (CEGIS) framework [36], as the name suggests, leverages counterexamples from verification for synthesis. The approach presented in this paper bears some resemblance to the CEGIS-based approach described in [32], [31]. Here, the verification condition that the system satisfies an STL [32] specification is encoded as an MILP. If the specification is violated, one can investigate the results of the MILP to obtain counterexamples. In [21], the authors extend the previous work and provide an intuition for the system failing to satisfy the specification. These work, in contrast to the presented work, do not compute reachable set for generating the counterexamples.

III. PRELIMINARIES

States and vectors are elements in $\mathbb{R}^n$ are denoted as x and v. The inner product of two vectors, v_1 and v_2, is denoted by $v_1^T v_2$. Given a sequence $seq = s_1, s_2, \ldots$, the i^{th} element in the sequence is denoted as $seq[i]$. Given a finite set S, we denote its cardinality as $|S|$. In this work, we use the following mathematical notation of a linear hybrid system.

Definition 1: A *linear hybrid system* H is defined to be a tuple $\langle Loc, X, Flow, Inv, Trans, Guard\rangle$ where:

Loc is a finite set of locations (also called modes).

$X \subseteq \mathbb{R}^n$ is the state space of the behaviors.

$Flow : Loc \rightarrow \mathit{AffineDeq}(X)$ assigns an affine differential equation $\dot{x} = A_l x + B_l$ for location l of the hybrid automaton.

$Inv : Loc \rightarrow 2^{\mathbb{R}^n}$ assigns an invariant set for each location of the hybrid system.

$Trans \subseteq Loc \times Loc$ is the set of discrete transitions.

$Guard : Trans \rightarrow 2^{\mathbb{R}^n}$ defines the set of states where a discrete transition is enabled.

For a linear hybrid system, the invariants and guards are given as the conjunction of linear constraints.

The *initial set of states* Θ is a subset of $Loc \times 2^{\mathbb{R}^n}$, where second element in the pair is a conjunction of linear constraints. An *initial state* q_0 is a pair (Loc_0, x_0), such that $x_0 \in X$, and $(Loc_0, x_0) \in \Theta$. The unsafe set of states is a subset of state space, $U \subseteq \mathbb{R}^n$.

Definition 2: Given a hybrid system and an initial set of states Θ, an *execution* of the hybrid system is a sequence of trajectories and transitions $\xi_0 a_1 \xi_1 a_2 \ldots$ such that (i) the first state of ξ_0 denoted as q_0 is in the initial set, i.e., $q_0 = (Loc_0, x_0) \in \Theta$, (ii) each ξ_i is the solution of the differential equation of the corresponding location Loc_i, (iii) all the states in the trajectory ξ_i respect the invariant of the location Loc_i, and (iv) the state of the trajectory before each transition a_i satisfies $Guard(a_i)$.

The set of states encountered by all executions that conform to the above semantics is called the *reachable set*. The closed form expression for the trajectory in each mode is given as $\xi_i(t) = e^{A_{Loc_i} t}\xi_i(0) + \int_0^t e^{A_{Loc_i}(t-\mu)} B_{Loc_i} d\mu$ where A_{Loc_i} and B_{Loc_i} define the affine dynamics of the mode Loc_i. As the closed form expression of an execution involves matrix exponential, typically a simulation engine is used to generate simulation as a proxy for such execution. For our work, we use the simulation engine that is described in [6]. This simulation engine also accounts for non-determinism induced due to discrete transitions. For a unit time (also called the *step size*), the hybrid system simulation starting from state q_0 is denoted as $\xi_H(q_0)$. We present the definition here for completeness.

Definition 3: A sequence $\xi_H(q_0) = q_0, q_1, q_2, \ldots$, where each $q_i = (Loc_i, x_i)$, is a (q_0)-simulation of the hybrid system H with initial set Θ if and only if $q_0 \in \Theta$ and each pair (q_i, q_{i+1}) corresponds to either: (i) a continuous trajectory in location Loc_i with $Loc_i = Loc_{i+1}$ such that a trajectory starting from x_i would reach x_{i+1} after exactly unit time with $x_i \in Inv(Loc_i)$, or (ii) a discrete transition from Loc_i to Loc_{i+1} (with $Loc_{i-1} = Loc_i$) where $\exists a \in Trans$ such that $x_i = x_{i+1}$, $x_i \in Guard(a)$ and $x_{i+1} \in Inv(Loc_{i+1})$. Bounded-time variants of these simulations, with time bound T, are called (q_0, T)-simulations. If the pair (q_i, q_{i+1}) corresponds to a continuous trajectory, q_{i+1} is called the continuous successor of q_i, otherwise q_{i+1} is the discrete successor of q_i.

While talking about the continuous or discrete behaviors of simulations, we abuse notation and use x_i, the continuous component of the state instead of q_i.

Observations On Simulation Algorithm: We would like to note that our simulation algorithm has three features. First, the discrete transitions happen only at time instances that are multiples of the step size. Second, the invariant is also checked at these discrete instances of time. Third, the invariant can be violated at the instance of taking the discrete transition. As a result, the simulations never encounter zeno executions. Industrial grade simulation engine tools like Simulink/Stateflow deploy similar simulation algorithms. Our rationale for picking this specific semantics for simulation engine is provided in [6].

We now define the safety property for simulations and for a set of initial states (from [6]).

Definition 4: A given simulation $\xi_H(q_0)$ is said to be *safe with respect to an unsafe set* U if and only if $\forall q_i = (Loc_i, x_i) \in \xi_H(q_0)$, $x_i \notin U$. Safety for bounded time simulations are defined similarly. We drop the subscript H from ξ_H when it is clear from the context.

Definition 5: A hybrid system H with initial set Θ, time bound T, and unsafe set U is said to be *safe with respect* *to its simulations* if all simulations starting from Θ for bounded time T are safe.

Definition 6: Given a hybrid system H with initial set Θ, time bound T, and unsafe set U, a counterexample ξ is said to be of *length* l if and only if $\exists X \triangleq \{x_i | x_i \in \xi \land x_i \in U\}$ such that $|X| \geq l$. A counterexample of maximum *length* is called the *longest counterexample*.

Notice that there need not be a unique counterexample of unique length. Therefore, any counterexample that has the maximum length can be considered a longest counterexample. For computing these counterexamples of interest, we use the simulation equivalent reachable set approach that is presented in [6], [17].

A. Superposition principle, Generalized Stars, and Simulation-equivalent Reachable Set

We now present three main aspects of the reachable set computation (from [6]). First is the superposition principle, second is the generalized star representation, and finally, the reachable set algorithm for a single mode and the simulation-equivalent reachable set that is returned by the computeSimEquivReach algorithm in [6].

Remark 1: 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 in a given location l 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 for a linear hybrid system. Before describing the algorithm for computing the reachable set, we introduce the data structure called a *generalized star* that is used to represent the reachable set of states.

Definition 7: A *generalized star* (or simply star) $\mathbb{S}$ is a tuple $\langle c, V, P\rangle$ where $c \in \mathbb{R}^n$ is called the *center*, $V = \{v_1, v_2, \ldots, v_n\}$ is a set of n vectors in $\mathbb{R}^n$ called the *basis vectors*, and $P : \mathbb{R}^n \to \{\top, \bot\}$ is a predicate.

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

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

Sometimes we will refer to both $\mathbb{S}$ and $[\![\mathbb{S}]\!]$ as $\mathbb{S}$. 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 [6], we consider predicates P which are conjunctions of linear constraints. This is primarily because linear programming is very efficient when compared to nonlinear arithmetic.

Reachable Set Computation For Linear Dynamical Systems Using Simulations: We briefly describe the algorithm for computing a simulation-equivalent reachable set for a single mode. This is primarily done to present some crucial observations which will later be used in the algorithms for generating counterexamples. Longer explanation and proofs for these observations and algorithms are available in prior work [6], [17].

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 $V = \{v_1, v_2, \ldots, v_n\}$, the algorithm performs a simulation starting from c (denoted as $\xi(c, 0)$), and $\forall 1 \leq j \leq n$, performs a simulation from $c + v_j$ (denoted as $\xi(c + v_j, 0)$). For a given time instance i, the reachable set denoted as $Reach_i(\Theta)$ is defined as $\langle c_i, V_i, P\rangle$ where $c_i = \xi(c, i)$ and $V_i = \langle v'_1, v'_2, \ldots, v'_m\rangle$ where $\forall 1 \leq j \leq m, v'_j = \xi(c+v_j, i)-\xi(c, i)$. An illustration of reachable set computation is shown in Fig. 1. *Notice that the predicate that defines the reachable set does not change.*

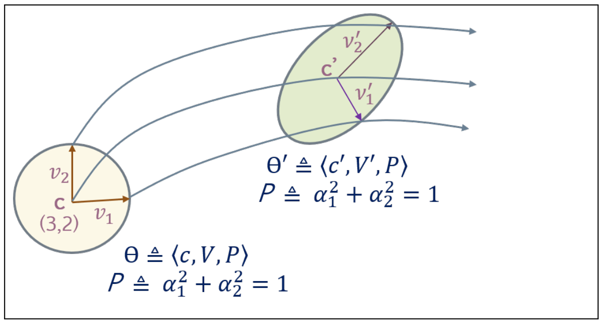
v'_2
c'
v'_1
$\Theta' \triangleq \langle c', V', P\rangle$
$P \triangleq \alpha_1^2 + \alpha_2^2 = 1$
v_2
c
(3,2)
v_1
$\Theta \triangleq \langle c, V, P\rangle$
$P \triangleq \alpha_1^2 + \alpha_2^2 = 1$

Fig. 1: Illustration of the reachable set using sample simulations and generalized star representation.

Simulation-Equivalent Reachable Set for Hybrid Systems with Linear Dynamics: In [6], the algorithm presented in [17] has been extended to compute simulation equivalent reachable set for hybrid systems that accommodates for the invariants in each mode and the guard transitions for discrete mode jumps. For generating counterexamples, we use the fully de-aggregated version of the reachable set computation algorithm, denoted as computeSimEquivReach, where the stars in the reachable set are organized as a tree structure as illustrated in Fig. 2. With the initial set as its root, the elements of the tree represent the reachable set computed at discrete time instances. Each node (except the root) is either a continuous or a discrete successor of its parent.

Remark 2: Given a star $\mathbb{S}_i \triangleq \langle c_i, V_i, P_i\rangle$ in $ReachTree$ and its successor (either discrete or continuous) $\mathbb{S}_{i+1} \triangleq \langle c_{i+1}, V_{i+1}, P_{i+1}\rangle$, observe that one has to either perform intersection with the invariant or with the guards for obtaining the predicate P_{i+1} so that $P_{i+1} \subseteq P_i$. Hence, given a valuation of $\bar{\alpha}$ such that $P_{i+1}(\bar{\alpha}) = \top$, it is true that for all the stars that are the parents of P_{i+1}, the valuation of $\bar{\alpha}$ is contained in the predicate. Additionally, one can use this valuation of basis variables to generate the trace starting from the initial set Θ to P_{i+1}. We call the procedure that generates this execution $getExecution(\bar{\alpha}, ReachTree)$.

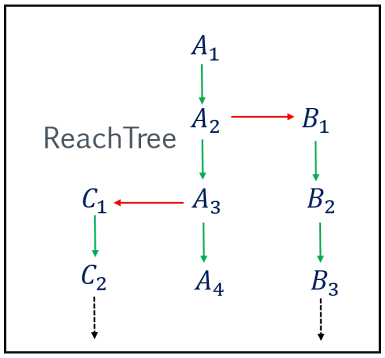

Fig. 2: *ReachTree* for a hybrid system with 3 locations - *A*, *B*, and *C*. Discrete transitions are shown in red, continuous transitions in green, and dashed transitions denote that there may be a transition. This snapshot of the *ReachTree* has 3 paths, $< A_1, A_2, A_3, A_4 >, < A_1, A_2, B_1, B_2, B_3, ... >$ and $< A_1, A_2, A_3, C_1, C_2, ... >$.

A side effect of the above observation is that all the trajectories that reach the star $\mathbb{S}_{i+1} \triangleq \langle c_{i+1}, V_{i+1}, P_{i+1} \rangle$ would originate from the subset of the initial set $\Theta' \triangleq \langle c_0, V_0, P_{i+1} \rangle$.

IV. Longest Counterexample

In this section, we formally state the longest counterexample problem and describe the underlying technique used for obtaining these counterexamples. For this purpose, we leverage the generalized star representation and the property of the reachable set that is provided in Remark 2.

A. Problem Statement

Definition 8: For a set $U \subseteq \mathbb{R}^n$, an indicator function

$$\mathbb{1}_U : Loc \times \mathbb{R}^n \to \{0, 1\}$$

is defined as

$$\mathbb{1}_U(q) = \begin{cases} 1, & \text{if } x \in U \\ 0, & \text{otherwise} \end{cases}$$

where $q = (loc, x)$ and $loc \in Loc$.

Problem Given the set of initial states Θ, the set of unsafe states U and its indicator function $\mathbb{1}_U$, compute

$$\arg\max_{q_0 \in \Theta} \sum_{t=0}^{T-1} \mathbb{1}_U(\xi(q_0, t))$$

where ξ is the system simulation from Definition 3. For a simulation, the optimization problem aims to maximize the number of overlaps with the unsafe set.

B. Constraint Propagation

We illustrate the problem of finding the longest counterexample through an illustration in Fig. 3. Consider five consecutive stars, $\mathbb{S}_1, \mathbb{S}_2, \mathbb{S}_3, \mathbb{S}_4$ and $\mathbb{S}_5$ in the reachable set have overlap with the unsafe set as shown. If one picks the state $e_1 \in \mathbb{S}_1$, then some of the *post* states of e_1, denoted as e_2 and e_5 do not lie in the unsafe set, while e_3 and e_4 lie in the unsafe set. Similarly, if one picks the state $l_1 \in \mathbb{S}_1$ as shown, then the *post* states l_2, l_4 and l_5 lie in the unsafe set but l_3 does not. Thus, the execution starting from l_1 provides a longer counterexample than the execution starting from e_1.

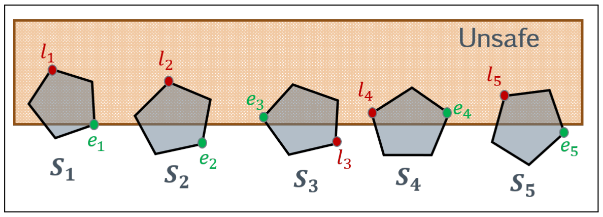

Fig. 3: Illustration of the longest counterexample.

The key insight behind the generation of longest counterexample is that one has to select the *appropriate* state such that its corresponding execution has the maximum number of overlaps with the unsafe set. In this instance, any state $x_1 \in \mathbb{S}_1 \cap U$, with its successors x_2, x_4, x_5 such that $x_2 \in \mathbb{S}_2 \cap U$, $x_4 \in \mathbb{S}_4 \cap U$ and $x_5 \in \mathbb{S}_5 \cap U$ may be an appropriate choice. For finding such a state, we perform constraint propagation (similar to the invariant constraint propagation in [6]). 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, $x_1 = c_1 + V \times \bar{\alpha} \in \mathbb{S}_1 \cap U$, $x_2 = c_2 + V_2 \times \bar{\alpha} \in \mathbb{S}_2 \cap U$, $x_4 = c_4 + V_4 \times \bar{\alpha} \in \mathbb{S}_4 \cap U$, and $x_5 = c_5 + V_5 \times \bar{\alpha} \in \mathbb{S}_5 \cap U$.

To extract these set of constraints, we convert the unsafe set U into the center and basis vectors of each of the stars $\mathbb{S}_1, \mathbb{S}_2, \mathbb{S}_4$ and $\mathbb{S}_5$. Thus, $\mathbb{S}_i \cap U = \langle c_i, V_i, P_i \wedge Q_i \rangle$. From Remark 2, 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 the maximum intersections of the unsafe set should originate from $\langle c_0, V_0, P_1 \wedge Q_1 \wedge P_2 \wedge Q_2 \wedge P_4 \wedge Q_4 \wedge P_5 \wedge Q_5 \rangle$. Further, if the set of constraints $P_1 \wedge Q_1 \wedge P_2 \wedge Q_2 \wedge P_4 \wedge Q_4 \wedge P_5 \wedge Q_5$ is satisfiable, then the corresponding trajectory corresponding to the basis variables that satisfy these constraints visits the unsafe set in maximum number of stars i.e., $\mathbb{S}_1, \mathbb{S}_2, \mathbb{S}_4$ and $\mathbb{S}_5$.

V. Computational Frameworks

A. MILP-based Framework

We formulate an MILP model for finding the longest counterexample as follows. For a given path Γ in the *ReachTree*, let Π be the set of stars $\mathbb{S}_i$ overlapping with the unsafe set U, and $\alpha \in \mathbb{R}^n$ be our basis variables such that $i = 1, \ldots, |\Pi|$ index the elements in Π. Additionally, let $C_i = Q_i \wedge P_i$ be the set of linear constraints that need to be satisfied in order for $\mathbb{S}_i$ to be overlapping with the unsafe set. We can write each of these constraints as $\left(a^{i,k}\right)^T \alpha \leq b$, for $i = 1, \ldots, |\Pi|$, and $k = 1, \ldots, |C_i|$. We therefore have:

$$\begin{aligned} &\max \sum_{i=1}^{|\Pi|} z_i \\ &\text{s.t. } \left(a^{i,k}\right)^T \alpha \leq b + M(1 - z_i), && i = 1, \ldots, |\Pi|, \\ & && k = 1, \ldots, |C_i|, \\ &z_i \in \{0, 1\}, && i = 1, \ldots, |\Pi|. \end{aligned}$$

The z_i variables indicate if all constraints in C_i are satisfied. Note that since this is a maximization problem, z_i will be 1 if it can, but the constraints prohibit it from being 1 if any one of constraints is not satisfied. Also note that this model requires the definition of an appropriate M, which in this instance can be set to

$$\max_{i=1,\ldots,|\Pi|} \max_{k=1,\ldots,|C_i|} \left\{ \sum_{j=1}^{n} \left| a_j^{i,k} \right| \right\},$$

noting that this can be refined if needed for each specific i and k. The formal procedure to find the longest counterexample using above MILP framework is provided in Algorithm 1.

```
input : Initial Set $\Theta$, the simulation equivalent
        reachable tree $ReachTree$ and unsafe set $U$
output: Trace $ce$ that spends longest time in $U$
1  $length_{max} \leftarrow -\infty$; $ce \leftarrow \bot$;
2  for each path $\Gamma$ in $ReachTree$ do
3      $\Pi \triangleq \{\mathbb{S}_i | \mathbb{S}_i \in \Gamma, \mathbb{S}_i \cap U \neq \emptyset\}$;
4      Introduce $|\Pi|$ decision variables $z_1, z_2 \ldots z_{|\Pi|}$;
5      $\mathcal{C}_\Pi \leftarrow \emptyset$;
6      Transform $U$ into $\langle c_i, V_i, Q_i \rangle$ where
        $\Pi[i] \triangleq \langle c_i, V_i, P_i \rangle$;
7      $\mathcal{C}_\Pi \leftarrow \bigwedge_{i=1}^{|\Pi|} \bigwedge_{c \in Q_i \wedge P_i} c + M(1 - z_i)$;
8      $length_\Pi \leftarrow max \sum_i z_i$ while $C_\Pi$ is feasible;
9      if $length_\Pi > length_{max}$ then
10         $length_{max} \leftarrow length_\Pi$;
11         $\bar{\alpha}_{len} \leftarrow feasible(\mathcal{C}_\Pi)$;
12     end
13 end
14 if $length_{max} \neq -\infty$ then
15     $ce \leftarrow getExecution(\bar{\alpha}_{len}, ReachTree)$;
16 end
17 return $ce$;
```

Algorithm 1: MILP-based Algorithm for computing the longest counterexample.

B. SMT-based Framework

An SMT solver primarily answers the decision problem of whether a given logical formula is *satisfiable* i.e., if there exists some assignment to variables included in the formula such that this assignment makes the logical formula evaluate to true. It either provides a satisfying assignment or declares that the formula is *unsatisfiable*. Certain SMT solvers also allow to encode linear optimization problems where the objective is to optimize a cost function while satisfying a given set of constraints. For this purpose, the solver provides the flexibility to specify constraints to be either soft or hard. A *hard* constraint is required to be asserted, whereas a *soft* constraint can be either satisfied or violated. Since a penalty is associated with the violation of soft constraint, the optimizer targets to minimize or maximize the overall penalty depending on the objective function. The approach to compute the longest counterexample using SMT is explained in Algorithm 2.

```
input : Initial Set $\Theta$, the simulation equivalent
        reachable tree $ReachTree$ and unsafe set $U$
output: Trace $ce$ that spends longest time in $U$
1  $length_{max} \leftarrow -\infty$; $ce \leftarrow \bot$;
2  for each path $\Gamma$ in $ReachTree$ do
3      $\Pi \triangleq \{\mathbb{S}_i | \mathbb{S}_i \in \Gamma, \mathbb{S}_i \cap U \neq \emptyset\}$;
4      Introduce $|\Pi|$ binary variables $b_1, b_2 \ldots b_{|\Pi|}$;
5      $\mathcal{C}_\Pi \leftarrow \triangle_{i=1}^{|\Pi|} b_i$;
6      Transform $U$ into $\langle c_i, V_i, Q_i \rangle$ where
        $\Pi[i] \triangleq \langle c_i, V_i, P_i \rangle$;
7      $\mathcal{C}_\Pi \leftarrow C_\Pi \bigwedge_{i=1}^{|\Pi|} (b_i == (Q_i \wedge P_i))$;
8      $length_\Pi \leftarrow Optimize_{SMT}(\mathcal{C}_\Pi)$ while $C_\Pi$ is
        feasible;
9      if $length_\Pi > length_{max}$ then
10         $length_{max} \leftarrow length_\Pi$;
11         $\bar{\alpha}_{len} \leftarrow feasible(\mathcal{C}_\Pi)$;
12     end
13 end
14 if $length_{max} \neq -\infty$ then
15     $ce \leftarrow getExecution(\bar{\alpha}_{len}, ReachTree)$;
16 end
17 return $ce$;
```

Algorithm 2: SMT-based Algorithm that computes the longest counterexample. $\triangle$ designates soft constraints.

VI. Evaluation

For *ReachTree* computation, we use a Python-based verification tool `HyLAA` (in *de-aggregation* mode) which uses `scipy's odeint` for simulating the differential equations, `GLPK` for linear programming, and `numpy` for matrix operations. The measurements were performed on a system running Ubuntu 18.04 with an 2.20GHz Intel Core i7-8750H CPU with 12 cores and 32 GB RAM. We use `Z3Py` [11] as an SMT solver, and Gurobi Optimizer (called from `C++`) for solving MILPs [23].

The benchmarks for our study are taken from [1], [29] and [35]. As stated earlier, one can characterize various regions in the state space using specifications. We label that subspace as unsafe for our experiments. The designer can specify a region of interest in the same manner, and the longest execution obtained can be used to evaluate controllers with respect to that particular region/specification. Fig. 4 illustrates the experimental result for `Buck Converter`.

The evaluations on various benchmarks are provided in Table I. The counterexamples given are the valuation of basis variables for the longest counterexample. The *ReachTree* computed for a linear hybrid system can have multiple paths due to discrete transitions. Furthermore, each path may have multiple nodes overlapping with the unsafe set. As shown in the table, counterexample generation using SMT takes significantly more time than the time taken by MILP-based framework. Verification time of some benchmarks such as `Damped Oscillator` and `Ball String` is comparable to the SMT-based longest counterexample generation time.

Model	Dims, Modes	Actual Inter. Duration	LCE Duration		Counterexample		Verification	LCE Gen. Time	
			MILP	SMT	MILP	SMT	Time (sec)	MILP	SMT
Damped Oscillator 1	2, 1	[5 10] [34 44] [66 74]	[5 9] [35 44] [66 73]	[5 9] [35 44] [66 73]	[-5.28 0.764]	[-5.321 0.865]	0.44	0.04	0.51
Damped Oscillator 2	2, 1	[3 10] [29 49] [59 100]	[3 10] [30 49] [59 100]	[3 9] [29 49] [59 100]	[-5.0 0.398]	[-5.0 0.606]	0.59	0.04	0.55
Oscillating Particle	3, 1	[17 29] [51 58]	[18 21] [24 28] [52 56]	[18 21] [26 29] [52 57]	$x_1 = -0.0932$ $x_2 = 0.8193$ $x_3 = 0.9$	$x_1 = 0.0385$ $x_2 = 0.8877$ $x_3 = 0.9$	0.33	0.04	0.43
Vehicle Platoon 5	15, 1	[27 100]	[27 100]	[27 100]	$x_{11} = 0.96$ $x_i \in \{0.9, 1.1\}$	$x_i \in \{0.9, 1.1\}$	8.57	0.37	58
Vehicle Platoon 10	30, 1	[36 100]	[36 100]	[36 100]	$x_{23} = 1.029$ $x_i \in \{0.9, 1.1\}$	$x_{11} = 1.0$ $x_i \in \{0.9, 1.1\}$	31	0.91	360
Ball String	2, 2	**ext:** [12 13] **ext:** [15 20] **freefall:** [21 29]	[18 20] [21 28]	[12 13] [16 20] [21 24]	[-1.008 -0.15]	[-0.95 -0.15]	0.31	0.03	0.3
Two Tanks	2, 4	**loc3:** [16 28] **loc1:** [33 78]	[34 77]	[35 78]	[2.399 1.079]	[2.407 1.077]	19.13	0.50	6.51
Buck Converter	4, 6	**cl1:** [13 21] **op1:** [22 50] **cl2:** [51]	[13 21] [22 50] [51]	[13 21] [22 50] [51]	il = 1.0 vc = 0 t = 0, gt = 0	il = 0.6892 vc = 0 t= 0, gt = 0	0.66	0.04	0.60
Filtered Oscillator	34, 4	**loc3:** [3 5] **loc3:** [7 21] **loc4:** [26]	[5] [7 21] [26]	[5] [7 21] [26]	[0.2069 0.07 0...]	[0.205 0.07 0...]	37	2.14	49

TABLE I: **Longest Counterexample**. *Dims* is the number of system variables, *Modes* is the number of locations, *Longest Counterexample* is the valuation of basis variables for an initial state from which the execution overlaps with the unsafe set at maximum time steps. x_i represents all the variables whose values are not explicitly given. *Actual Inter. Duration* is the mode-wise ordered sequence of discrete time step intervals when the 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 which is exclusive of the counterexample generation time, *LCE Gen Time* is the time (in seconds) each framework takes to generate the longest counterexample.

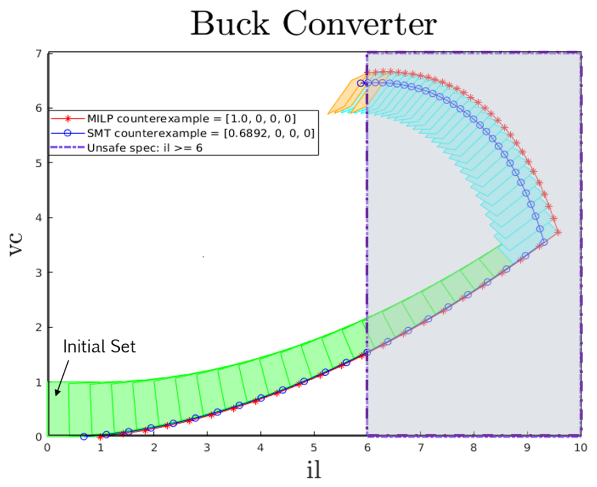

Fig. 4: Illustration of the longest counterexample in Buck Converter. The original benchmark has 2 locations but our model has 6 locations so as to incorporate transition resets. The figure shows the reachable set computed for locations - **closed1, open1,** and **closed2**.

VII. Discussion and Conclusion

As MILP turns out to be the better alternative for generating the longest counterexample, an obvious question arises whether this framework is suitable in all cases. An issue with an MILP-based approach is numerical stability. The model requires the definition of M, which is larger than any value. Initial testing, even with a state-of-the-art commercial solver, led to a numerically unstable solution, returning a trajectory that isn't really a counterexample. Through iterating with `Z3` we were able to identify a suitable choice of M, but this can lead to an incorrect solution if sufficient care is not taken. On the other hand, `Z3` is slow but it doesn't suffer with the problem of numerical instability.

We also notice that the MILP-based approach and SMT-based approach return different counterexamples. This is compatible with our definition as longest counterexample need not be unique. In fact, the overlap with the unsafe set in the counterexamples returned by these two approaches can differ (as shown in the Table I). Further, although the length of the longest counterexample is fixed, the counterexamples as well as their intersection intervals may differ across both frameworks. For instance, the longest counterexample length for `Ball String` is 11, and its respective duration generated with MILP and SMT is [18 20][21 28] and [12 13][16 20][21 24], respectively.

Conclusion: We have illustrated the longest counterexample for the safety specification of linear hybrid systems, and provided SMT- and MILP-based frameworks to generate these counterexamples. We have evaluated these techniques on multiple benchmarks. Although relatively slow, the SMT-based formalism provides guarantees on the solution obtained; the MILP-based solution is much faster but it may be numerically unstable. For the future work, we plan to explore BDD-optimization techniques [8] [7] to overcome some limitations of MILP, and also, to make use of these counterexamples in controller synthesis.

Acknowledgements: We thank Muqsit Azeem (TCS Research, India) and Aditya A. Shrotri (Rice University) for providing valuable inputs in SMT encoding. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-19-1-0288 and National Science Foundation (NSF) under grant numbers CNS 1739936, 1935724. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force or National Science Foundation.

References

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

[2] Houssam Abbas and Georgios E. Fainekos. Linear hybrid system falsification through local search. In *Automated Technology for Verification and Analysis*, 2011.

[3] R. Alur, T. Dang, and F. Ivančić. Counter-example guided predicate abstraction of hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems*. 2003.

[4] Yashwanth Annpureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, 2011.

[5] Stanley Bak and Parasara Sridhar Duggirala. HyLAA: A tool for computing simulation-equivalent reachability for linear systems. In *Hybrid Systems Computation and Control*, 2017.

[6] Stanley Bak and Parasara Sridhar Duggirala. Rigorous simulation-based analysis of linear hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2017.

[7] David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John Hooker. *Decision Diagrams for Optimization*. Springer Publishing Company, Incorporated, 2016.

[8] David Bergman, Willem-Jan van Hoeve, and John N. Hooker. Manipulating mdd relaxations for combinatorial optimization. In Tobias Achterberg and J. Christopher Beck, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2011.

[9] Michael S Branicky. Multiple lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on automatic control*, 1998.

[10] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *International Conference on Computer Aided Verification*, 2000.

[11] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[12] Jyotirmoy V. Deshmukh, Georgios E. Fainekos, James Kapinski, Sriram Sankaranarayanan, Aditya Zutshi, and Xiaoqing Jin. Beyond single shooting: Iterative approaches to falsification. In *American Control Conference, ACC*, 2015.

[13] H. Dierks, S. Kupferschmid, and K.G. Larsen. Automatic Abstraction Refinement for Timed Automata. In *Proceedings of the International Conference on Formal Modelling and Analysis of Timed Systems*, 2007.

[14] Alexandre Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification*, 2010.

[15] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems*, 2010.

[16] Parasara S. Duggirala and Sayan Mitra. Abstraction-refinement for stability. In *Proceedings of 2nd IEEE/ACM International Conference on Cyber-physical systems (ICCPS)*, 2011.

[17] Parasara S. Duggirala and Mahesh Viswanathan. Parsimonious, simulation based verification of linear systems. In *International Conference on Computer Aided Verification*, 2016.

[18] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 2009.

[19] A. Fehnker, E.M. Clarke, S. Jha, and B. Krogh. Refining Abstractions of Hybrid Systems using Counterexample Fragments. In *Proceedings of the International Conference on Hybrid Systems Computation and Control*, 2005.

[20] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, 2011.

[21] Shromona Ghosh, Dorsa Sadigh, Pierluigi Nuzzo, Vasumathi Raman, Alexandre Donzé, Alberto L Sangiovanni-Vincentelli, S Shankar Sastry, and Sanjit A Seshia. Diagnosis and repair for synthesis from signal temporal logic specifications. In *Hybrid Systems: Computation and Control*, 2016.

[22] Manish Goyal and Parasara Sridhar Duggirala. On generating a variety of unsafe counterexamples for linear dynamical systems. In *Proc. 6th IFAC Conference on Analysis and Design of Hybrid Systems*, 2018.

[23] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018.

[24] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 1990.

[25] D. Liberzon and A Stephen Morse. Basic problems in stability and design of switched systems. *IEEE Control systems*, 1999.

[26] Oded Maler, Dejan Nickovic, and Amir Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of computer science*. 2008.

[27] Kumpati S Narendra and Jeyendran Balakrishnan. A common lyapunov function for stable lti systems with commuting a-matrices. *IEEE Transactions on automatic control*, 1994.

[28] Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancić, Aarti Gupta, and George J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Hybrid Systems: Computation and Control*, 2010.

[29] Luan Nguyen and Taylor Johnson. Benchmark: Dc-to-dc switched-mode power converters (buck converters, boost converters, and buck-boost converters). 2014.

[30] Pavithra Prabhakar, Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. Hybrid automata-based cegar for rectangular hybrid systems. In *Verification, Model Checking, and Abstract Interpretation*, 2013.

[31] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M. Murray, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Model predictive control with signal temporal logic specifications. In *IEEE Conference on Decision and Control, CDC*, 2014.

[32] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M Murray, and Sanjit A Seshia. Reactive synthesis from signal temporal logic specifications. In *18th International Conference on Hybrid Systems: Computation and Control*, pages 239–248. ACM, 2015.

[33] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *Hybrid Systems: Computation and Control*. 2005.

[34] Sriram Sankaranarayanan and Ashish Tiwari. Relational abstractions for continuous and hybrid systems. In *International Conference on Computer Aided Verification*, 2011.

[35] Christoffer Sloth and Rafael Wisniewski. Verification of continuous dynamical systems by timed automata. *Formal Methods in System Design*, 2011.

[36] Armando Solar-Lezama. *Program synthesis by sketching*. University of California Berkeley, 2008.

[37] Kazuo Tanaka, Tsuyoshi Hori, and Hua O Wang. A multiple lyapunov function approach to stabilization of fuzzy control systems. *IEEE Transactions on fuzzy systems*, 2003.

[38] Aditya Zutshi, Jyotirmoy V Deshmukh, Sriram Sankaranarayanan, and James Kapinski. Multiple shooting, cegar-based falsification for hybrid systems. In *International Conference on Embedded Software*, 2014.