Multiple Shooting, CEGAR-based Falsification for Hybrid Systems

Aditya Zutshi
University of Colorado, Boulder
aditya.zutshi@colorado.edu

Sriram Sankaranarayanan*
University of Colorado, Boulder
srirams@colorado.edu

Jyotirmoy V. Deshmukh
Toyota Technical Center
jyotirmoy.deshmukh@tema.toyota.com

James Kapinski
Toyota Technical Center
jim.kapinski@tema.toyota.com

ABSTRACT

In this paper, we present an approach for finding violations of safety properties of hybrid systems. Existing approaches search for complete system trajectories that begin from an initial state and reach some unsafe state. We present an approach that searches over *segmented trajectories*, consisting of a sequence of segments starting from *any* system state. Adjacent segments may have gaps, which our approach then seeks to narrow iteratively. We show that segmented trajectories are actually paths in the abstract state graph obtained by tiling the state space with cells. Instead of creating the prohibitively large abstract state graph explicitly, our approach implicitly performs a randomized search on it using a *scatter-and-simulate* technique. This involves repeated simulations, graph search to find likeliest abstract counterexamples, and iterative refinement of the abstract state graph. Finally, we demonstrate our technique on a number of case studies ranging from academic examples to models of industrial-scale control systems.

Categories and Subject Descriptors

I.6.4 [**Simulation and Modeling**]: Model Validation and Analysis; G.1.7 [**Numerical Analysis**]: Ordinary Differential Equations—*Boundary value problems*

Keywords

Hybrid Systems, Falsification, Simulation-Based Methods, Multiple Shooting, CEGAR.

1. INTRODUCTION

The problem of falsifying a safety property for a given system involves finding a counterexample that starts from the initial states of the system and reaches an unsafe (error) state. In this paper, we present a falsification technique for hybrid systems that relies on a search over *segmented trajectories*, which consist of segments of trajectories with gaps between the end of one segment and the start of the next segment (see Fig. 1). We show that segmented trajectories are, in fact, paths through a discrete abstraction of the hybrid system obtained by partitioning the state-space into cells. Such an abstraction is often prohibitively expensive, if not impossible, to construct *explicitly*. Therefore, our approach explores this abstraction *implicitly* through segmented trajectories. We present a counterexample-guided refinement scheme that uses abstract counterexamples to selectively refine this implicit abstraction. This yields a search over segmented trajectories with ever decreasing gaps between segments, potentially converging to a complete counterexample, as the number of iterations is increased.

Our approach has two distinguishing features: (a) we assume that the system under analysis is very likely to be *incorrect* for the property of interest, and perform a best effort search to find a violation. Such an assumption is complementary to *verification tools* such as theorem provers and static analyzers that implicitly assume a proof of correctness, and can be seen as a best-effort search for such a proof. (b) We operate under a *black-box* assumption, with limited knowledge of the system dynamics in each mode. This assumption is standard for simulation-based techniques for falsification [1, 6, 10, 23]. The black-box assumption allows our technique to operate on highly non-linear hybrid systems designed inside environments such as Simulink™/Stateflow™ [1]. We also assume that the system state is fully exposed to our technique through instrumentation.

Our approach constructs an abstraction incrementally through simulation. At each step, the system is simulated with some discrete time step, and each simulated point is "snapped" to an abstract cell. Similarly, each trajectory segment between two time points yields an edge. As a result, we obtain an abstract graph that approximates the behaviors of the underlying system. We show that each path in this graph is a segmented trajectory. Associating costs with these segmented trajectories as the distances between end points of segments, we search for minimal cost paths through the graph from initial to final cells using a standard shortest path algorithm. Next, we refine the abstraction by *implicitly* subdividing each cell into smaller ones. However, doing so makes the finite state abstraction a lot more complicated. To counter this, we focus our refinement on abstract counterexample paths to explore them further. Doing so, is shown to reduce the cost of the segmented trajectories. In the limit, as the algorithm goes through refinement steps, the costs of the segmented

*Zutshi and Sankaranarayanan were supported, in part, by the US National Science Foundation(NSF) under award numbers CNS-1319457 and CNS-0953941 and in part by Toyota Engineering and Manufacturing North America(TEMA). All opinions are those of the authors and not necessarily of the NSF or TEMA.

ESWEEK'14 October 12 - 17 2014, New Delhi, India

http://dx.doi.org/10.1145/2656045.2656061.

[1]Simulink and Stateflow are commercial model-based design tools from Mathworks Inc.

trajectories converge towards zero, and therefore under some assumptions about the underlying system, the segmented trajectories themselves converge towards a trajectory of the underlying system. In practice, our approach performs concrete simulations at each step to check if the abstract counterexamples do correspond to concrete simulation traces, and exits early upon success. Due to the black box assumptions, we note two key limitations: (a) our approach is limited to finding *robust* counterexamples, i.e., for every violation found by our approach, there is a small neighborhood around every state in the violation, such that initial conditions within this neighborhood also lead to a violation, and (b) our approach is randomized in nature, and performs a bounded number of simulations in each iteration. Finally, our approach uses a given system simulator as a black-box. If a guaranteed integrator such as VNODE is provided, our approach can also guarantee the validity of any counterexamples produced [21]. On the other hand, our approach can also work with numerical simulators that are commonly used by control designers to simulate their designs inside model-based design environments.

We have implemented our ideas in a prototype MATLAB^{TM} tool. The prototype is evaluated over a suite of benchmarks of varying sizes and complexities. We compare our approach against other related techniques including S-Taliro [1,3], a simulation-based falsification tool, and dReach, a symbolic approach that uses unrolling to find bounded depth counterexamples [14]. We find that our approach can find falsifications of properties in reasonable time and succeeds in many cases where other approaches fail.

1.1 Related Work

Falsification techniques Falsification techniques for hybrid systems can be classified into *single-shooting* (SS) or *multiple-shooting* (MS) approaches. SS approaches search over the space of *complete* trajectories, while MS approaches attempt to find trajectory segments starting from multiple initial conditions (and hence, possibly contain *gaps* between segments). Direct multiple-shooting (DMS) is a commonly used optimal control technique, where a control task over a finite time horizon is divided into segments and the control problem for each segment is solved with the additional requirement that the solutions should meet each other at the endpoints [7]. Our previous work [27] used a DMS approach to find unsafe trajectories, given the sequence of transitions. An NLP solver was used with gradient information gained by sensitivity analysis, to reduce the gaps between trajectory segments. This required knowledge of the system dynamics and could potentially fail when the search reaches a local, rather than global optimum. In this paper, we use randomized graph search and a CEGAR procedure to iteratively narrow trajectory gaps. Our current approach can work under black-box assumptions and avoids the pitfalls inherent in gradient descent search algorithms.

S-Taliro [1,3] and Breach [10] are single-shooting methods that use robustness-guided optimization-based search of the state-space to find initial states and inputs that cause violations of a given temporal logic property. For hybrid systems, S-Taliro requires a hybrid distance metric to perform efficiently, which requires detailed system knowledge [22]. Our approach is carefully designed to avoid system specific metrics. A direct comparison with S-Taliro is provided in Section 6.

Approaches based on Rapidly exploring Random Trees (RRT) [19] have been used to falsify non-autonomous systems [6,9,23]. These explore the search space by growing a random tree (backward or forward in time). As such, RRTs are a single-shooting approach and are performed on the concrete system. However, while RRTs have proved immensely successful in robotic path planning, their successful use in falsification of systems seems to be limited. The performance of RRTs relies on many factors including the local planner used, and the nature of the system under test. In our experience, the technique performs suboptimally on systems whose reachable set is smaller relative to the full state-space. Our approach, on the other hand, does not require a local planner and explores the reachable states on-the-fly. Interestingly, our approach can be easily extended to use RRT as the underlying search technique over the abstract state space.

Verification techniques Counterexample guided abstraction refinement (CEGAR)-based verification of hybrid systems involves symbolic techniques to build abstractions and model checking tools to explore the abstractions [2]. Ratschan and She present a symbolic approach over a rectangular tiling of the state-space similar to the one presented in this paper [24]. Their work uses a combination of grid-based state-space abstraction and refinement using constraint propagation to prune away unreachable states. Our approach explores a similar abstraction to *falsify* rather than prove. As a result, we do not insist on exploring the entire abstract state space. While this makes our approach unsuitable for proving properties, our evaluation shows the power of our approach for finding violations in a small amount of time for complex non-linear systems. A related approach combining interval constraint propagation with branch-and-bound is implemented in iSAT and dReal [13, 14, 16]. Our approach defines similar abstractions, but does not explicitly create them, rather exploring them through directed random simulations; however, in doing so, it loses the ability to find non-robust witnesses and proofs.

2. PRELIMINARIES

In this section, we present the basic model of dynamical system used in our work. We will focus on continuous and hybrid systems driven by external inputs.

Let $\mathcal{X} \subseteq \mathcal{L} \times \mathbb{R}^n$ be the (infinite) set of hybrid states of the system $\mathcal{S}$, where $\mathcal{L}$ is a finite set of discrete modes, and $\mathcal{U}$ be the set of input signals to $\mathcal{S}$ of the form $[0,T] \to \mathbb{R}^k$ for some given time horizon T. In this paper, we assume a family of relations $\xrightarrow{t,u} \subseteq \mathcal{X} \times \mathcal{X}$, parameterized by time $t \geq 0$ and input signal $u \in \mathcal{U}$ defined over time $[0,T]$. We use the notation $\mathbf{x} \xrightarrow{t,u} \mathbf{y}$ to denote $(\mathbf{x},\mathbf{y}) \in \xrightarrow{t,u}$. The relations $\xrightarrow{t,u}$ satisfy the following basic properties:

- Each state is reachable from itself in 0 time, $\forall u : \mathbf{x} \xrightarrow{0,u} \mathbf{x}$.
- *Forward determinism:* For each $\mathbf{x} \in \mathcal{X}$, $t \geq 0$, and input $u \in \mathcal{U}$, there is a unique $\mathbf{y} \in \mathcal{X}$ such that $\mathbf{x} \xrightarrow{t,u} \mathbf{y}$.
- *Causality:* For each $\mathbf{x} \in \mathcal{X}$, time $t \geq 0$, and for every inputs $u_1, u_2 \in \mathcal{U}$, if $u_1(s) = u_2(s)$ for $0 \leq s \leq t$ and $\mathbf{x} \xrightarrow{t,u_1} \mathbf{y}$ then $\mathbf{x} \xrightarrow{t,u_2} \mathbf{y}$.
- *Semi-group Property:* For each $\mathbf{x},\mathbf{y},\mathbf{z} \in \mathcal{X}$ and every $t_1, t_2 \geq 0$ and signals $u_1, u_2 \in \mathcal{U}$, if $\mathbf{x} \xrightarrow{t_1,u_1} \mathbf{y}$ and $\mathbf{y} \xrightarrow{t_2,u_2} \mathbf{z}$ then $\mathbf{x} \xrightarrow{t_1+t_2,u_1;u_2} \mathbf{z}$. Here we define $u_1;u_2$ as the composed signal $u(t)$ with $u(t) = u_1(t)$ for $t \in [0,t_1)$ and $u(t) = u_2(t-t_1)$ for $t \in [t_1, t_1+t_2)$.

We assume that the system $\mathcal{S}$ is equipped with a forward simulation function $\text{SIM}_{\mathcal{S}}$ that takes $\mathbf{x} \in \mathcal{X}$, $t \geq 0$ and $u \in \mathcal{U}$, and computes the unique successor state $\mathbf{y} = \text{SIM}_{\mathcal{S}}(\mathbf{x},u,t)$ such that $\mathbf{x} \xrightarrow{t,u} \mathbf{y}$. The subscript $\mathcal{S}$ will be dropped whenever the system $\mathcal{S}$ being referred to is clear from the context.

Black-Box Assumptions: For above properties of the $\xrightarrow{t,u}$ relation to hold, we require that $\mathcal{S}$ be deterministic. We also assume that $\mathcal{S}$ can

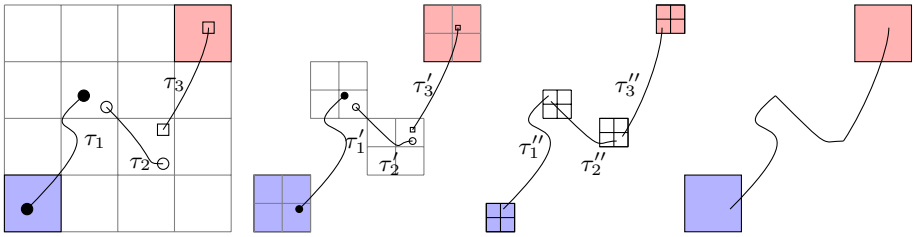


Figure 1: An illustration of our approach: (a) segmented trajectory reaching unsafe states (shaded red) starting from initial states (shaded blue), (b) refining an abstract counterexample and narrowing the inter-segment gap, (c) further narrowing the gap by refinement, and (d) a concrete trajectory with no gaps.

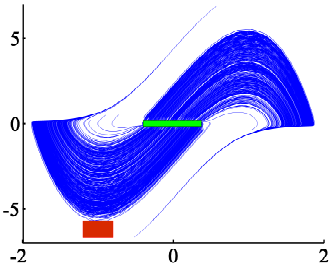


Figure 2: Van-der-Pol ODE trajectories with initial set $\mathcal{X}_0 : [-0.4, 0.4] \times [-0.4, 0.4]$ and unsafe set $[-1.2, -0.8] \times [-6.7, -5.7]$, shown in red.

be simulated from a given initial state using function $\text{SIM}_{\mathcal{S}}$. This requires existence and uniqueness of trajectories over a finite time horizon $[0, T]$, which can be guaranteed by Lipschitz continuity of the vector field in each hybrid mode and ruling away issues such as finite escape times [20]. From a practical standpoint, these assumptions do not pose significant restrictions for the type of problems that we wish to address. Finally, we assume full observability of the system state, ignore numerical errors due to simulation and *do not* assume that an analytic closed-form representation of the system dynamics is available.

Time-Bounded Reachability Problem: Given a system $\mathcal{S}$, an initial set $\mathcal{X}_0 \subseteq \mathcal{X}$, a set of unsafe states $\mathcal{X}_f \subseteq \mathcal{X}$, and a time bound $T > 0$, the *reachability problem* decides if there exists an initial state $\mathbf{x}_0 \in \mathcal{X}_0$, an unsafe state $\mathbf{y} \in \mathcal{X}_f$, time $t \in [0, T]$ and input signal $u \in \mathcal{U}$ such that $\mathbf{x}_0 \xrightarrow{t,u} \mathbf{y}$.

Verification techniques typically focus on proving that the set $\mathcal{X}_f$ is unreachable from the initial state $\mathcal{X}_0$ for the given time horizon T. Our approach here, focuses on *falsification*. Given a time-bounded reachability problem instance $\langle \mathcal{S}, \mathcal{X}_0, \mathcal{X}_f, T \rangle$, we seek a *concrete witness* of the form of $\mathbf{x}_0 \in \mathcal{X}_0$, $\mathbf{y} \in \mathcal{X}_f$, time $t \in [0, T]$ and an input signal $u \in \mathcal{U}$, such that $\mathbf{x}_0 \xrightarrow{t,u} \mathbf{y}$.

EXAMPLE 2.1. *Fig. 2 shows the trajectories of a van der Pol oscillator whose dynamics are defined by a system of ODEs over* $(x_1, x_2) \in \mathbb{R}^2$*:* $\frac{dx_1}{dt} = x_2$, $\frac{dx_2}{dt} = -5(x_1^2 - 1)x_2 - x_1$. *Consider the reachability of the set* $\mathcal{X}_f : [-1.2, -0.8] \times [-6.7, -5.7]$ *(shown as a red rectangle in Fig. 2) starting from an initial set* $\mathcal{X}_0 : [-0.4, 0.4] \times [-0.4, 0.4]$ *(shown as a green rectangle in the center). Attempting* 1000 *random simulations on the initial set yields precisely one trajectory that witnesses the violation.*

Metrics

Let $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ be a *metric*[2] over the state-space X of a system. Common metrics over continuous state spaces include the L_1, L_2, L_∞ metrics. Our approach assumes a suitable metric d that defines the distance between two points in the state space. Many such metrics are available for purely continuous state-spaces. However, for hybrid systems, the state-space requires us to define a metric d that compares two hybrid states $(\ell, \mathbf{x})$ and $(\ell', \mathbf{x}')$ belonging to different modes. The difficulties involved in designing a hybrid metric are discussed elsewhere by Nghiem et al. [22]. Our approach side-steps this difficulty. Let $\hat{d}$ be a metric defined over $\mathbb{R}^n$. Its *lifting* over a hybrid state-space $\mathcal{X} : \mathcal{L} \times \mathbb{R}^n$ is defined as:

$$d((\ell_1, \mathbf{x}_1), (\ell_2, \mathbf{x}_2)) = \begin{cases} \hat{d}(\mathbf{x}_1, \mathbf{x}_2), & \text{if } \ell_1 = \ell_2 \\ \infty, & \text{otherwise} \end{cases}$$

It is easy to verify that d is a metric, provided $\hat{d}$ is a metric and $\hat{d}(\mathbf{x}_1, \mathbf{x}_2)$ is finite for all pairs of continuous states $\mathbf{x}_1, \mathbf{x}_2$.

3. TILING-BASED ABSTRACTIONS OF HYBRID STATE-SPACES

In this section, we introduce the notion of segmented trajectories, and show that segmented trajectories are paths through an abstraction of the underlying system. We stress that our approach does not construct such an abstract state-space fully. Instead, we explore a subset of the abstract states in a randomized fashion using the scheme detailed in Sec. 4.

DEFINITION 3.1 (SEGMENTED TRAJECTORIES). *A* segmented trajectory σ *is a finite sequence* $\langle \tau_1, \ldots, \tau_k \rangle$ *of trajectory segments, wherein each trajectory segment* τ_i *is a system trajectory starting from a source state* $\mathbf{x}_i$ *(denoted* $\mathsf{src}(\tau_i)$*), evolving for time* t_i *under input* u_i *to a destination* $\mathbf{y}_i$ *(denoted* $\mathsf{dest}(\tau_i)$*), i.e.,* $\mathbf{x}_i \xrightarrow{t_i, u_i} \mathbf{y}_i$.

The *cost* of a trajectory segment w.r.t to a metric d is given by adding up the "gaps" between adjacent segments: $\text{COST}(\sigma) : \sum_{i=1}^{k-1} d(\mathsf{dest}(\tau_i), \mathsf{src}(\tau_{i+1}))$.

LEMMA 3.1. *A zero cost segmented trajectory is an actual system trajectory.*

The central idea in our approach is to search for a sequence of segmented trajectories of decreasing cost, that in the limit tends to an actual system trajectory.

[2]For $\mathbf{x}, \mathbf{y} \in \mathcal{X}$, a function $d(\mathbf{x}, \mathbf{y})$ is a metric iff: (a) $d(\mathbf{x}, \mathbf{y}) \geq 0$, (b) $d(\mathbf{x}, \mathbf{y}) = 0$ iff $\mathbf{x} = \mathbf{y}$, (c) $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$, and (d) $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$.

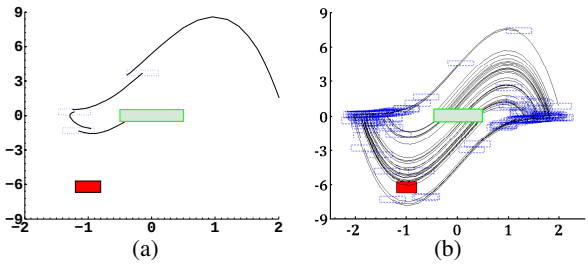


Figure 3: Van der Pol oscillator example: (a) single segmented trajectory. Green box denotes initial and red box denotes unsafe and (b) Randomly simulated segmented trajectories. Blue boxes highlight the gaps between segments.

EXAMPLE 3.1. *Fig. 3 shows segmented trajectories for the van der Pol system from Example 2.1. The Fig. 3(a) shows an example of a segmented trajectory, while Fig. 3(b) shows a collection of* 100 *segmented trajectories. Each segmented trajectory in this collection has two segments; the source of the first segment is picked from a random sampling of the initial state set, while the source of the second segment is chosen by adding a random vector to the destination of the first segment.*

Abstraction: To present falsification using search over the space of segmented trajectories as an instantiation of CEGAR, we first formally define an underlying abstraction of the continuous state-space. The abstraction is best described as an ϵ-tiling of the continuous state space $\mathbb{R}^n$ with a given $\epsilon > 0$.

DEFINITION 3.2 (ϵ-TILINGS). *Formally, a tiling $\mathcal{C} : \{C_1, \ldots, C_j, \ldots\}$ is a finite or countably infinite family of closed and bounded subsets C_j of $\mathcal{X}$ called* cells. *We require that the following conditions are met:*

1. *The union of cells must cover the entire state space.*
$$\bigcup_{C_i \in \mathcal{C}} C_i = \mathcal{X}$$
2. *Cell interiors must be pairwise disjoint.*
$$\mathsf{interior}(C_i) \cap \mathsf{interior}(C_j) = \emptyset, \text{ for } i \neq j$$
3. *For each cell $C_i \in \mathcal{C}$ there exists a* representative state $[\mathbf{x}]_i \in C_i$ *that is no farther than ϵ away from every state in the cell.*
$$\forall\, C_i \ \in \mathcal{C},\ \exists\, [\mathbf{x}]_i \in C_i,\ \forall\, \mathbf{x} \in C_i,\ d([\mathbf{x}]_i, \mathbf{x}) \leq \epsilon\,.$$

Fig. 4 shows a tiling of a subset of the state-space for the van der Pol system from Ex. 2.1, with $\epsilon = 0.5$. The meaning of the different colors assigned to the tiles will be explained subsequently.

Constructing a Tiling: We define a simple yet "canonical" construction to produce ϵ-tilings for any given $\epsilon > 0$, using an L_∞-norm[3] based distance metric. Informally, the proposed tiling uses hypercubes as tiles. Assume for convenience that $\epsilon = 10^{-k}$ for some $k > 0$. We first define an equivalence relation $\sim_k$ over $\mathbb{R}$ wherein $x \sim_k y$ iff the decimal representation of x, y agree to first k decimal places. We extend $\sim_k$ to relate vectors in $\mathbb{R}^n$ wherein $\mathbf{x} \sim_k \mathbf{y}$ iff every entry $x_i \sim_k y_i$. Similarly, for hybrid state spaces, we relate two states $(\ell_1, \mathbf{x}_1) \sim_k (\ell_2, \mathbf{x}_2)$ iff $\ell_1 = \ell_2$ and $\mathbf{x}_1 \sim_k \mathbf{x}_2$. It is easy to see that $\sim_k$ is an equivalence relation, i.e., it is reflexive, symmetric and transitive. Consider the tiling defined by the equivalence classes of $\sim_k$.

[3]For a vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, $||\mathbf{x}||_\infty$ is defined as $\max(|x_1|, \ldots, |x_n|)$.

DEFINITION 3.3 ($\sim_k$-DEFINED TILING). *The set of cells $\mathcal{C}$, where each $C_i \in \mathcal{C}$ is the closure of an equivalence class of $\sim_k$ defines an ϵ-tiling. All hybrid states $(\ell, \mathbf{x})$ in any given cell C_i have the same mode ℓ, and the decimal representations of the continuous states $\mathbf{x}$ matches for at least the first k decimal points. A* representative element $[\mathbf{x}]_i \in C_i$ *is defined as the hybrid state $(\ell, [\mathbf{x}]_i)$, where $[\mathbf{x}]_i$ is the state obtained by truncating the continuous state for any element C_i to the first k decimal digits.*

LEMMA 3.2. *Given $\epsilon = 10^{-k}$ for $k > 0$, the $\sim_k$-defined tiling $\mathcal{C}$ as defined in Def. 3.3 is an ϵ-tiling of $\mathcal{X}$.*

As noted earlier, explicitly constructing this tiling is impractical for most systems. The number of cells is infinite if $\mathcal{X}$ is unbounded. If $\mathcal{X}$ is bounded, then the number of cells is generally exponential in the dimensionality of $\mathcal{X}$ and dependent on the volume of $\mathcal{X}$. We reiterate that our technique does not seek to fully construct the tiling, or the abstraction resulting from it.

Abstraction: Given an ϵ-tiling, we can now define the abstract system through a standard existential abstraction over these tilings. We define a family of abstract timed reachability relations $\overset{t}{\rightsquigarrow}$ between cells, wherein, we denote $C_i \overset{t}{\rightsquigarrow} C_j$ iff

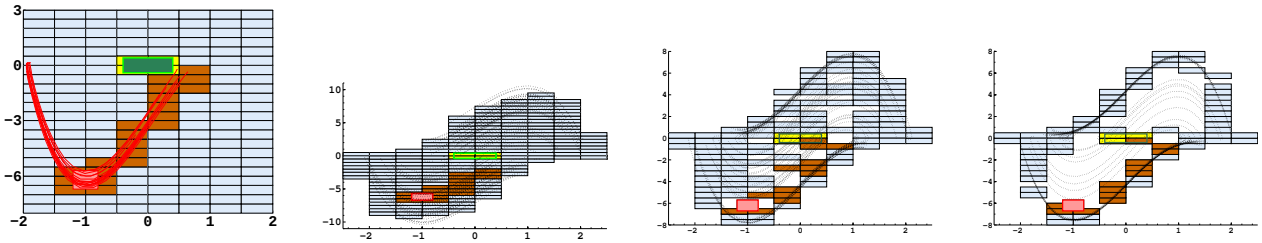
$$\exists\, \mathbf{x}_i \in C_i,\ \exists\, \mathbf{x}_j \in C_j,\ \exists\, u \in \mathcal{U},\ \mathbf{x}_i \xrightarrow{t,u} \mathbf{x}_j\,.$$

DEFINITION 3.4 (ABSTRACT STATE GRAPH). *Let $\Delta > 0$ be a fixed time step. The abstract state graph $\mathcal{H}(\Delta)$ for time step Δ is defined by the set of vertices $\mathcal{C}$, and edges (C_i, C_j) whenever $C_i \overset{\Delta}{\rightsquigarrow} C_j$. Let $\mathcal{C}_0$ denote the collection of initial cells in $\mathcal{C}$, i.e., cells C_i such that $C_i \cap X_0 \neq \emptyset$. Further, let $\mathcal{C}_u$ denote the set of unsafe cells, or cells C_j such that there is a state $\mathbf{x}_j \in C_j$ that reaches an unsafe state $\mathbf{y} \in X_f$ within time $0 \leq t_j < \Delta$. The abstraction $\mathcal{H}(\Delta)$ is given by $\left\langle \mathcal{C}, \overset{\Delta}{\rightsquigarrow}, \mathcal{C}_0, \mathcal{C}_u \right\rangle$.*

Besides the lack of scalability, there are other computational barriers to constructing such an abstract state graph. For instance, we would need a theorem prover to (1) determine if two cells C_i, C_j are connected, and (2) decide if a cell C_k is unsafe in the abstraction. Such tasks require us to perform symbolic reasoning about dynamics of nonlinear systems, which is often infeasible. Therefore, we resort to a simulation-based approach to explore the abstract graph $\mathcal{H}(\Delta)$.

EXAMPLE 3.2. *Fig. 4 is a partial depiction of $\mathcal{H}(\Delta)$ for the system from Ex. 2.1. The figure shows $\mathcal{C}_0$, $\mathcal{C}_u$, and a subset of cells reachable from the initial cells. Edges between cells are shown by showing a trajectory between states that belong to the cells.*

Abstract Paths and Segmented Trajectories Consider a path $\pi : C_0 \rightarrow C_1 \rightarrow \cdots \rightarrow C_N$ through the abstract graph $\mathcal{H}(\Delta)$. Corresponding to this path, we define a *witness segmented trajectory* $\sigma(\pi)$ with segments $\langle \tau_0, \tau_1, \ldots, \tau_{N-1} \rangle$, where $\mathsf{src}(\tau_i) \in C_i$, $\mathsf{dest}(\tau_i) \in C_{i+1}$ and $\mathsf{src}(\tau_i) \xrightarrow{\Delta,u} \mathsf{dest}(\tau_i)$. The existence of this witness $\sigma(\pi)$ follows immediately from the construction of $\mathcal{H}(\Delta)$. Furthermore, since the tiling $\mathcal{C}$ is an ϵ-tiling, we have that each inter-segment gap $d(\mathsf{dest}(\tau_i), \mathsf{src}(\tau_{i+1}))$ is at most 2ϵ. Therefore the overall cost is $\textsc{cost}(\sigma) \leq 2N\epsilon$. Finally, having fixed a time step Δ for each error trajectory and an overall time horizon T for the reachability property in question, we note that $N \leq \frac{T}{\Delta}$. Combining, these observations we obtain the following lemma.



(a) A tiling with $\epsilon = 0.5$ (b) $\Delta = 0.25$ (c) $\Delta = 0.75$ (d) $\Delta = 1.5$

Figure 4: (a) A tiling of the state-space. (b)-(d) Visualization of a subset of $\mathcal{H}(\Delta)$ showing cells reachable from the initial set of states for the van der Pol system. Yellow cells are initial (but not unsafe), cells shaded brown are unsafe and blue cells are neither initial nor known to be unsafe. The dotted trajectory segments represent the edges.

LEMMA 3.3. *For every path π through the abstract graph $\mathcal{H}(\Delta)$, there exists a witness segmented trajectory $\sigma(\pi)$, such that* $\text{COST}(\sigma) \leq 2N\epsilon \leq \frac{2T\epsilon}{\Delta}$.

Time Step Δ: As Δ increases, the relation $\overset{\Delta}{\rightsquigarrow}$ becomes more complex, involving more mode switches and effects of non-linear dynamics. At the limit, as $\Delta = T$, constructing the abstraction reduces to the (undecidable) reachability problem. At the other extreme, as $\Delta \to 0$, every cell is connected to its neighboring cell by virtue of sharing a common boundary. As a result, the abstraction becomes too conservative, and no longer reflects the dynamics of the underlying system. We can see the effect of varying Δ for the van der Pol system in Fig. 4.

4. EXPLORATION BY THE SCATTER-AND-SIMULATE APPROACH

In this section, we explore the abstract graph $\mathcal{H}(\Delta)$ by sampling a finite subgraph $G(\mathsf{V}, \mathsf{E})$. We call the approach *scatter-and-simulate* to reflect its two main steps: (1) uniformly sample a reachable cell to obtain initial states (2) perform repeated simulations from the obtained states for a fixed time step Δ. We assume that the space of input signals $\mathcal{U}$ is compact, and assume a suitable sampling distribution over $\mathcal{U}$.

The basic algorithm is shown in Figure 1. The two main parameters for the algorithm are the fixed time step Δ and a *scatter amount* $K > 0$. The search is initialized by sampling the initial region $\mathcal{X}_0$, and adding the sampled cells into a worklist (Line 1). For each cell C_j in the worklist, we (uniformly) sample K concrete states in the cell and K inputs from signals $\mathcal{U}$ (Lines 5,6). We then use a numerical simulator to obtain trajectories for Δ seconds (Line 8). We identify the cells corresponding to the end states of the trajectories and add them to workList and V, if previously unseen (Lines 11,12). We also add appropriate edges to E (Line 13). If the simulation starting from $\mathbf{x}_{i,j} \in C_j$ intersects the unsafe set $\mathcal{X}_f$, then we mark the entire cell C_j as unsafe in the abstraction (Line 14). We bound the size of the subgraph G that we wish to explore with the parameter L. We note that the size of the graph G thus obtained is restricted to L vertices and $O(L \times K)$ edges.

The data structure used to implement the workList affects the nature of the search. A guided state-space search A* using an admissible (under-approximating) heuristic (cf. [18]) is implemented by using a priority queue for workList. Alternatively, our approach can be modified using motion planning techniques such as RRTs to explore the discrete abstraction in a probabilistically complete fashion.

Algorithm 1: Scatter-and-Simulate search for exploring $\mathcal{H}(\Delta)$.

Input: Time-bounded Reachability problem $\langle \mathcal{S}, \mathcal{X}_0, \mathcal{X}_f, T \rangle$, Scatter amount K, Simulator function $\text{SIM}_{\mathcal{S}}$, size limit L

Output: Subgraph $G(\mathsf{V}, \mathsf{E})$, initial cells V_0 and unsafe cells V_u.

```
1  workList := sampleInitialSetAndCollectCells(X_0) ;
2  V := workList, V_0 := workList;
3  while workList ≠ ∅ and |V| < L do
4      C_j := pop(workList) ;
5      (x_{1,j}, ..., x_{K,j}) := sampleUniformly(C_j) ;
6      (u_1, ..., u_K) := sampleInputSignals(U) ;
7      for i ∈ [1, K] do
8          τ_{i,j} = SIM_S(x_{i,j}, u_i, Δ) ;
9          C_i := identifyCellFromConcreteState(dest(τ_{i,j})) ;
10         if C_i ∉ V then
11             workList := push(C_i, workList) ;
12             V := V ∪ {C_i} ;
13         E := E ∪ (C_j, C_i) ;
14         if dest(τ_{i,j}) ∈ X_f then V_u := V_u ∪ {C_j} ;
```

Scatter-And-Simulate on $\sim_k$-defined tilings: We consider Algorithm 1 on a $\sim_k$-defined tiling. The routine identifyCellFromConcreteState($\mathbf{x}$) (Line 9) identifies a cell C_i by obtaining the representative element $[\mathbf{x}]_i$ from $\mathbf{x}$ by truncating the values of the continuous state component of $\mathbf{x}$ to its first k decimal digits. The subroutine sampleUniformly(C_j) (Line 5) is implemented by adding a uniform random number in the range $[0, 10^{-k}]$ to each of the continuous state components of the representative element $[\mathbf{x}]_j$.

Fig. 4 depicts the scatter-and-simulate algorithm for the van der Pol system in Ex. 2.1, by limiting the number of vertices (L) to 200 and the scatter per cell (K) to 60.

Maintaining the Time Horizon: Since we are interested in exploring $\mathcal{S}$ up to the time horizon T, Algorithm 1 should be restricted to nodes that are reachable within $N = \lceil \frac{T}{\Delta} \rceil$ steps from an initial cell. Therefore, Algorithm 1 is modified to ensure that every node that is popped out of the workList has a shortest path length of at most N.

Identifying Likeliest Abstract Counterexamples: We now explain how we can use the graph G to help search for abstract counterexamples. These are paths in G of length at most $N = \lceil \frac{T}{\Delta} \rceil$ from initial cells V_0 to cells labelled unsafe V_u. In order to prioritize the likeliest counterexamples, we need a notion that associates costs

with paths. As each path π is associated with at least one witnessing segmented trajectory σ, we equate the cost of a path with the cost of the minimal cost witnessing segmented trajectory for π.

For a given edge e in G, let $TS(e)$ be a set of trajectory segments defined as the set $\{\tau \mid \mathsf{src}(\tau) \in \mathsf{src}(e) \wedge \mathsf{dest}(\tau) \in \mathsf{dest}(e)\}$, i.e., the collection of all trajectory segments beginning and ending in the source and destination cells of e respectively. Two edges $e_1, e_2 \in \mathsf{E}$ are said to be adjacent iff $\mathsf{dest}(e_1) = \mathsf{src}(e_2)$. Let $\tau_i \in TS(e_1)$ and $\tau_j \in TS(e_2)$, we then define a weight $W(e_1, e_2)$ for edges e_1, e_2 as follows:

$$W(e_1, e_2) = \begin{cases} \min(\mathsf{dest}(\tau_i), \mathsf{src}(\tau_j)) & \text{if } e_1, e_2 \text{ are adjacent} \\ \infty & \text{otherwise.} \end{cases} \quad (1)$$

The cost of path $\pi : e_1 e_2 \ldots e_m$ in G is then defined as $\textsc{cost}(\pi) = \sum_{j=1}^{m-1} W(e_j, e_{j+1})$; this quantity is finite as each edge e_i in π is adjacent to e_{i+1}.

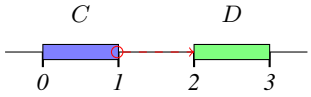
Even though the cost is defined over pairs of edges rather than edge weights, a simple modification of Dijkstra's shortest path algorithm can be used to compute shortest paths from an initial node to an unsafe node. If no such path exists, our approach re-executes the scatter-and-simulate step (Algorithm 1) with increased values for the size of the graph L and the scatter amount K.

4.1 Analysis of Scatter and Simulate

Scatter-and-simulate (Algorithm 1) explores the abstract state graph $\mathcal{H}(\Delta)$ in a randomized fashion. In doing so, it suffers from two drawbacks w.r.t. discovering violations: (a) it may miss an edge and (b) it may fail to label a node as unsafe.

For a cell C_j in the ϵ-tiling, let $N(C_j)$ represent its neighbors in $\mathcal{H}(\Delta)$. Algorithm 1 samples a subset of the neighbors from $\mathcal{H}(\Delta)$. Unfortunately, not every neighbor in $N(C_j)$ will be explored by scatter and simulate, even when system has simple constant dynamics as in Example 4.1.

EXAMPLE 4.1. *Consider a one dimensional autonomous system with dynamics $\dot{x} = 1$ and a cell C defined by the interval $x \in [0, 1]$. Let us fix $\Delta = 1$. Consider the cell D defined by the interval $[2, 3]$. We note that $C \overset{1}{\rightsquigarrow} D$, since $(x = 1) \overset{1}{\rightarrow} (x = 2)$.*



If we sampled the cell C uniformly at random in Algorithm 1, the probability of a single trajectory segment reaching D is zero. This is because, $x = 1$ is the single state in C that can reach D in one time step. Sampling this single state has probability 0.

Consider the function defined by $\textsc{sim}_{\mathcal{S}}(\mathbf{x}_0, u, \Delta)$ for a fixed Δ; let us denote it by Φ. Consider the probability that the edge $e : C \overset{\Delta}{\rightsquigarrow} D$ in $\mathcal{H}(\Delta)$ can be discovered by sampling C, and denote it by $\mathbb{P}(C, D)$. The probability is well-defined if we assume that Φ is measurable when restricted to cell C. The value of $\mathbb{P}(C, D)$ is a function of the distributions chosen to sample states and inputs. For the purpose of this discussion, let us fix a uniform distribution for sampling the "scatter" states, and also fix the set of inputs to a finite set, and consider a uniform distribution over this set.

DEFINITION 4.1 (ROBUST EDGES). *An edge $e : (C, D) \in \mathcal{H}(\Delta)$ is robust iff its probability $\mathbb{P}(C, D) > 0$. A path π is robust if each of the edges in the path is robust.*

The following shows that if a robust path exists from an initial to an unsafe cell in $\mathcal{H}(\Delta)$, then Algorithm 1 will discover it in the limit with probability 1. As a technicality, we assume that the workList data structure is *fair* (e.g., a FIFO queue), i.e., every cell placed in it is eventually processed. The result that follows directly from the definition of robust edges, and from the fundamental laws of probability:

LEMMA 4.1. *Let C be a cell reachable from some initial cell C_0 through a robust path π of length at most $\lceil \frac{T}{\Delta} \rceil$. As $L \to \infty, K \to \infty$, the cell C will belong to the graph G explored by Algorithm 1 with probability* 1.

A limitation of our approach is the lack of probabilistic guarantees on its ability to discover non-robust edges in $\mathcal{H}(\Delta)$. However, even for a fixed scatter amount K for each cell, we are not guaranteed to discover all the robust outgoing edges in $N(C_j)$, for a given cell C_j. The choice of value K is thus a crucial tradeoff between scalability and effectiveness. A simple technique is to fix two thresholds p and c to find a value K that guarantees that edges $e : C \overset{\Delta}{\rightsquigarrow} D$ with $\mathbb{P}(C, D) \geq p$ are discovered with probability at least c. Assume that Algorithm 1 draws K independent and identically distributed samples for the states and inputs.

LEMMA 4.2. *If $K \geq \frac{\log(1-c)}{\log(1-p)}$ then an edge $e : C \overset{\Delta}{\rightsquigarrow} D$ with $\Pr(C, D) \geq p$ is discovered with probability at least c.*

PROOF. Consider a sequence of K simulations and let us assume that none of them allowed us to discover the edge e. The probability of this happening is at most $(1 - p)^K$. We therefore wish to ensure that this event happens with probability at most $(1 - c)$.

$$(1 - p)^K \leq (1 - c)$$

Therefore, we obtain $K \geq \frac{\log(1-c)}{\log(1-p)}$. □

For instance, drawing $K = 59$ or more samples per cell will guarantee that any single outgoing edge with probability 0.05 or more is found more than 95% of the time by our sampling process.

In summary, Algo. 1 finds robust edges in $\mathcal{H}(\Delta)$, while non-robust edges are discovered *almost never* (with probability 0). Lemma 4.1 guarantees if the workList in Algorithm 1 is *fair*, as $K, L \to \infty$, all robust edges in $\mathcal{H}(\Delta)$ are discovered *almost surely* (with probability 1). As a result, all robust paths are also discovered *almost surely*. Next, Lemma 4.2 provides bounds on the number of samples per cell K to achieve guarantees on the probabilities of missing an edge.

5. REFINING ABSTRACT COUNTEREXAMPLES

In the previous section, we outlined an approach to explore a subgraph of the abstract graph corresponding to an ϵ-tiling, using segmented trajectories. Further, we also discussed a notion of using weights on pairs of adjacent edges in this subgraph to obtain a set of prioritized abstract counterexamples with path cost less than a given budget. We now present a way to refine the abstraction induced by an ϵ-tiling $\mathcal{C}$ by defining the notion of a refined tiling. The basic idea of refinement is to construct a δ-tiling $\mathcal{D}$ where $\delta < \epsilon$, and the cells in $\mathcal{D}$ subdivide those in $\mathcal{C}$. We then wish to check if the abstract counterexamples corresponding to the ϵ-tiling continue to be counterexamples in the δ-tiling. We first formalize the notion of a refined tiling.

DEFINITION 5.1 (REFINED TILING). *Let $\mathcal{C}$ be an ϵ-tiling of $\mathcal{X}$. A refinement $\mathcal{D}$ of $\mathcal{C}$ is a tiling of $\mathcal{X}$ that satisfies the following:*

Algorithm 2: Multiple Shooting CEGAR Algorithm.

Input: Problem $\langle \mathcal{S}, \mathcal{X}_0, \mathcal{X}_f, T \rangle$, Scatter amount K, Simulator $\text{SIM}_{\mathcal{S}}$, size limit L, Max Refinement steps P, ϵ_0: start tiling width, and shrink : scale factor for ϵ.

Output: VIOLATION or FAIL.

1 $\epsilon := \epsilon_0$, $G := \emptyset$, PATHS $:= \emptyset$;
2 **for** $i \in [1, P]$ **do**
3 G := scatterAndSimulate($\mathcal{S}, \mathcal{X}_0, \mathcal{X}_f, T, \epsilon, L, K$, Paths);
4 $\Pi :=$ findAbstractCounterExamples(G) ;
5 **if** $\Pi = \emptyset$ **then** return FAIL ;
6 **if** *checkIfConcretizable*(Π) **then** return VIOLATION ;
7 PATHS $:= \Pi, \epsilon := \frac{\epsilon}{\text{shrink}}$;
8 return FAIL ;

1. *$\mathcal{D}$ is a δ-tiling for some $\delta < \epsilon$.*

2. *For every $D_j \in \mathcal{D}$, there is a unique cell $C_j \in \mathcal{C}$ such that $D_j \subseteq C_j$. D_j is called a* sub-cell *of C_j.*

3. *For every $D_j \in \mathcal{D}$ and $C_k \in \mathcal{C}$ if $\mathsf{interior}(D_j) \cap \mathsf{interior}(C_k) \neq \emptyset$ then $D_j \subseteq C_k$.*

For an ϵ-tiling, and its corresponding refined δ-tiling, let us denote their respective abstract state graphs (as defined in Def. 3.4) by $\mathcal{H}_\epsilon(\Delta)$ and $\mathcal{H}_\delta(\Delta)$.

Algorithm 2 summarizes the overall approach using scatter-and-simulate with repeated refinements. The approach performs up to P refinement steps. At each step, a slight modification of scatter-and-simulate algorithm (Line 3) is applied. The resulting graph G is then analyzed to find all abstract counterexamples (Line 4). The cost of the counterexamples can be used to select the most promising ones. If no counterexamples exist, we declare failure (and possibly restart). Otherwise, Line 6 checks if any of the abstract paths are concretizable. This step samples initial states from the initial cells for the abstract counterexamples. For each sample, it performs a complete simulation and checks if the concrete simulation results in a violation. If so, the algorithm terminates with a real violation witnessed by a concrete trajectory. Line 7 computes a set of abstract counterexample paths (PATHS) to guide the exploration of the refinement in the next step. Finally, the value of ϵ is scaled down for the refinement. We note that the scatter-and-simulate algorithm is modified by providing it a set of cells PATHS (Line 3). The set of abstract paths serves to restrict the search for counterexamples in the refinement. We consider two strategies for this restriction.

Likely Initial Cells: In this strategy, we use the initial cells of counterexample paths in PATHS. Here, we replace the RHS of Line 1 of Algorithm 1 by a random sampling of such initial cells.

Refining Abstract Counterexample Paths: In this strategy, we use all cells that belong to some path in PATHS. We explore $\mathcal{H}_\delta(\Delta)$ using Algorithm 1, while restricting our attention to the cells in PATHS. Therefore, Algorithm 1 is modified at line 9 and line 11 to ensure that a cell is added only if it is a sub cell of a cell in the set PATHS.

In practice, we observe that both approaches yield similar results. However, the likely initial cells approach is the simplest to implement.

EXAMPLE 5.1. *Fig. 5 shows a series of refinements for the van der Pol system from Example 2.1 with decreasing values of ϵ. In each step, we obtain paths in the resulting abstract state graphs, leading from some initial to an unsafe cell. The corresponding segmented trajectories also have costs that decrease in direct proportion to ϵ.*

5.1 Asymptotic Analysis of Refinement

Using the above mentioned strategies to explore the state graph based on a δ-tiling restricts the exploration to regions associated with previously explored abstract states in the ϵ-tiling. As a result, each concrete state obtained by sampling is now clustered within a smaller cell. Let the tiling granularity in the i^{th} refinement step be ϵ_i, and let a witnessing segmented trajectory corresponding to an abstract counterexample path π in $H_{\epsilon_j}(\Delta)$ (if one exists) be denoted σ_j. Then, recalling the result from Lemma 3.3, $\text{COST}(\sigma_j) \leq \frac{2T\epsilon_j}{\Delta}$.

Consider a series of refinement steps, wherein the i^{th} step involves an ϵ_i-tiling $\mathcal{D}_i$, such that $\epsilon_1 > \epsilon_2 > \cdots$ is a strictly decreasing sequence converging to 0 in the limit. Assume that at each refinement step, we obtain an abstract counterexample π_i from an initial cell of $\mathcal{H}_{\epsilon_i}(\Delta)$, the graph at the i^{th} step to an unsafe cell. The costs of the witnessing segmented trajectories converge to zero as well. This could lead us to believe that, in the limit, our approach is guaranteed to find a violation. We show that this is not true, in general, for hybrid systems.

EXAMPLE 5.2. *In this example, we show that convergent limits of segmented trajectory may not be a trajectory For simplicity, we consider a system $\mathcal{S}$ with no inputs and two discrete modes ℓ_0 and ℓ_1. Let the dynamics in mode ℓ_0 be the ODEs $\frac{dx}{dt} = 1$, $\frac{dx}{dt} = 2$, and in ℓ_1 be the ODEs $\frac{dx}{dt} = \frac{dy}{dt} = 0$. The system transitions from mode ℓ_0 to ℓ_1 upon hitting the guard set $x + y = 4$. The initial set is taken to be $\mathcal{X}_0 : \{\ell_0\} \times [0, 0.5] \times [0, 0.5]$ and the unsafe set is taken to be $\mathcal{X}_f : \{\ell_0\} \times [2, 3] \times [4, 5]$. We note that no behavior can reach $\mathcal{X}_f$, as any trajectory in mode ℓ_0 will hit the switching surface $x + y = 4$, and remain there in mode ℓ_1. Now consider a sequence of segmented trajectories σ_ϵ, each with two segments of the form: $(0.5 - \epsilon, 0.5 - \epsilon) \xrightarrow{1} (1.5 - \epsilon, 2.5 - \epsilon)$, $(1.5 + \epsilon, 2.5 + \epsilon) \xrightarrow{1} (2.5 + \epsilon, 4.5 + \epsilon)$.*

$\text{COST}(\sigma_\epsilon)$ is 2ϵ with the L_∞ norm. However, as $\epsilon \to 0$, we expect the segments to converge onto the zero cost "trajectory" $(0.5, 0.5) \xrightarrow{1} (1.5, 2.5) \xrightarrow{1} (2.5, 4.5)$. But the limit is not a valid trajectory as the system switches to the mode ℓ_1 at $(\ell_0, 1.5, 2.5)$ and never changes state again.

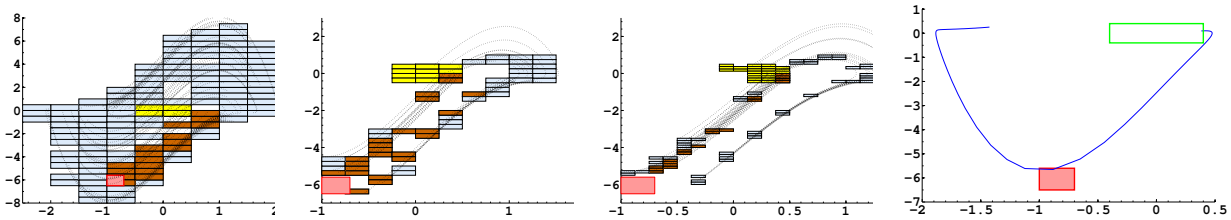
The counterexample above illustrates "non-robust" behavior that hybrid systems can exhibit. The example relies on the system switching at a precisely defined surface that our segmented trajectories happen to converge to asymptotically. While certainly possible, this behavior is pathological; real-life hybrid systems often exhibit robust behavior such as continuous dependence on initial condition over a small neighborhood. Under the following conditions that guarantee robust counterexamples, a zero cost segmented trajectory in the limit is, in fact, an actual system trajectory:

(1) Each cell in a counterexample $\pi^{(j)}$ in the j^{th} refinement is a sub-cell of a cell in a counterexample π in the i^{th} step for $i \leq j$.

(2) There is an i such that for all cells in counterexample paths at refinement steps $j > i$, the flow map defined by $\Phi(\mathbf{x}_0, u)$ (i.e., $\text{SIM}_{\mathcal{S}}(\mathbf{x}_0, u, \Delta)$ for a fixed Δ) is continuous over $\mathbf{x}_0$ for each $u \in \mathcal{U}$.

Let us consider a sequence of segmented trajectories $\pi^{(1)}, \cdots$, that satisfy conditions (1) and (2) above.

THEOREM 5.1. *The segmented trajectories $\pi^{(j)}$ converge uniformly to a zero cost segmented trajectory π^* that is an actual violation of the underlying system.*



(a) $\epsilon = \frac{1}{4}$ (b) $\epsilon = \frac{1}{8}$ (c) $\epsilon = \frac{1}{16}$ (d) Error Trajectory

Figure 5: A series of refinements for the van der Pol system with decreasing values of ϵ. A counterexample path exists in each refinement step, yielding a concrete violation.

We note that condition (2) is not very stringent. It is satisfied by continuous systems with Lipschitz continuous ODEs, and in our experience by most hybrid system models that can be numerically simulated in tools such as SIMULINK/STATEFLOW(TM).

6. EXPERIMENTAL EVALUATION

We have implemented [4] Algorithm 1 and the refinement heuristics described in Sec. 5 in MATLAB™ , and tested our falsifier on a number of systems. These range from small but complex systems with challenging falsification problems, to representative examples of complex industrial systems.

6.1 Implementation Details

This section highlights the various aspects of our implementation.

System description: The input format can either be an explicit hybrid automaton, or a user-defined function $\text{SIM}_{\mathcal{S}}$. Though we use ODE solvers and event detection mechanisms provided by MATLAB™ in the former case to simulate the system, it is still treated as a black box system during falsification.

Parameters: Our implementation allows a user to customize various system specific parameters within Algorithm 1, such as the initial tiling granularity defined by ϵ, the time step Δ, and the overall time horizon T. A maximum number of switches can also be specified to avoid trajectories with Zeno behavior.

Scaling: Thus far, our definition of tiling subdivides the space uniformly for all variables. However, in practice, different system variables have different ranges and therefore, our implementation treats ϵ as a vector of size n (number of continuous states), with each element defining the tiling size for that particular dimension.

Refinement: As stated in Sec. 4, in every refinement iteration, G is found by sampling $\mathcal{H}(\Delta)$. The finite representation of G is limited by various budgets on the number of samples and the number of vertices. Our technique can fail in two ways: (a) failing to find a abstract counterexample path or (b) going beyond maximum number of refinement steps without finding a concrete violation. Upon such a failure, our implementation restarts the search from the first iteration. The *Likely Initial Cells* approach was used for refining the abstraction and to obtain the presented results.

Success: The implementation can either succeed or fail because of a timeout. Upon success we return all found concrete counterexample traces, each represented by its initial condition, discrete transition sequence and sequence of inputs. This enables easy reproduction of traces using $\text{SIM}_{\mathcal{S}}$.

Being a generic framework, several aspects of the scatter-and-simulate are configurable. For instance, the sampling distribution of the cells can be varied. Moreover, when exploring the state space, adjacent cells to reachable cells can also be added, thus ensuring a more thorough search. In addition to using forward simulations beginning from the initial set, backward simulations originating from the unsafe cells can be used. This might not always be possible or tractable, specially when multiple backward solutions exist or the property is defined only on a subset of the state variables, respectively. Several guiding heuristics for the exploration of the abstraction in a given iteration can be used. Our implementation selects uniformly random inputs and samples in a cell, but techniques like RRT can be used to explore the state space more uniformly.

[4] https://csel.cs.colorado.edu/~zutshi/

6.2 Case Studies

We evaluated our method on a number of examples ranging from academic examples of nonlinear and hybrid dynamical systems to representative models of industrial control systems.

Academic examples The first set of benchmarks includes well known examples of complex nonlinear systems such as the Van Der Pol oscillator [25], the Brusselator, and the Lorenz reactor [25]. The second set of benchmarks contains hybrid dynamical systems: a 4 state model of a bouncing ball [27], a variation (B.Ball+S.H.M.) with the ball bouncing on a simple harmonically oscillating platform, and a constrained pendulum [26]. As such, these systems have no formal specification of an unsafe region. We use random testing to synthesize challenging falsification goals by identifying "hard-to-reach" regions.

The third set of benchmarks consists of a modified instance of navigation benchmark (NAV-30) [27], derived from [11]. These benchmarks describe a particle moving through a 2D grid, with different affine motion dynamics for each cell. The falsification problem is to determine if certain cells are reachable, given the initial position and velocities of the particle.

Automotive Control System Examples

Idle Speed Controller: When a car is stationary, but the engine is running, it is required for the engine speed to be maintained at a certain reference value in the presence of external disturbances such as torque demand (τ_ℓ) (e.g., due to the A/C system) and the clutch being operated. We use a simplified affine hybrid automaton model of a closed-loop system containing a model of the engine and the powertrain, with a single control input: the throttle plate angle. The controller uses Proportional + Integral (PI) control to maintain the engine speed to 800 ± 35rpm. The aim of the falsifier is to find a simulation trace in the closed-loop model which exceeds this range for time-varying values of τ_ℓ $(\in [0, 5]Nm)$.

Diesel engine airpath control: This benchmark contains a simplified version of the system presented in [17]. It is a piecewise affine representation of a diesel engine model and a model-predictive controller regulating the intake manifold pressure in the engine. The state space of the closed-loop system consists of 69 polytopes, each representing a certain region of operation of the plant and a corresponding control law. The falsifier tries to find a system trajectory that leads

to an increase of intake manifold pressure beyond a reference value.

Glucose-Insulin Models (G-I) This benchmark is based on work by Fisher [12] and describes an *artificial pancreas* controller to maintain safe levels of plasma blood glucose (pbg) in type I diabetes patients. The insulin-glucose dynamics are modeled using the Bergman minimal model [4]. The plant model uses an exponential glucose absorption sub-model. Using data of 18 different subjects [5] we search for scenario of hypoglycemia (i.e., $pbg < 3.6$ mmol/L). We incorporate an uncertainty interval for each of the patient parameters in our model. We only report the result for case (patient ID:1) where we found a violation.

dReal Benchmarks We also try our approach on dReal's deterministic benchmarks [5], the quadcopter and cardiac benchmarks. For the quadcopter (14 continuous states), we try to find initial conditions which can lead to instability. This is naively detected when a state exceeds desired bounds. In the cardiac example, we try to detect the critical condition of atrial fibrillation. Note that we have modified the properties slightly to make them harder to detect by random testing.

6.3 Results

To obtain results, we focused on properties which are hard to falsify using random testing. Testing vectors were generated by (uniform) randomly sampling 100,000 states in the initial set, and simulating for the given time horizon and noting the number of violations, which are reported along with time taken in Table 1. This provides a qualitative reference (of difficulty of finding a violation) as well as a quantitative comparison against sampling based tools.

The comparison with S-Taliro was done by observing the successful runs and noting the mean time taken for 10 runs. A run of S-Taliro/scatter-and-simulate is successful if a concrete violation is found within $1hr$, and is mentioned in the *Succ. Runs* column of Table 1. Both S-Taliro and scatter-and-simulate use restarts and a run can comprise of several restarts. Clearly, our tool is able to find a falsification in every run, whereas S-Taliro fails in some instances. Though, the times taken by S-Taliro and our tool are listed, they are hard to compare because our implementation is parallelized and uses multiple threads (4), which gives it a substantial performance boost (verified by a 2X speedup for the bouncing ball benchmark). This is in contrast with S-Taliro's single threaded implementation. Finally, our performance seems comparable, if not superior, to S-Taliro. This may be attributed to S-Taliro's use of global search on robustness function, which can be overly dependant on a good distance metric. In comparison, we are relatively insensitive to distance metrics.

We also compare with dReal/dReach. dReal is a SMT solver and symbolically analyses a reachability problem. It either returns an *abstract counterexample* in the form of a δ-satisfiability trace or proves unreachability (unsat). The latter capability is a clear advantage over our simulation based approach, which will be unsuccessful when no violations exists. Hence, we compare our results with dReal only when a violation is known to exist. We ran dReal on our benchmark suite, but timed out in $1hr$ on *all of them, except* the cardiac benchmark. The cardiac benchmark was completed by dReal in $33s$ with a precision of 0.001, and in $2.5min$ with precision of 0.0001.

Even though dReal can work with complex and highly non-linear benchmarks, as evident from the results presented by Gao et al. [14], the falsification instances considered in their work do not pose challenges for an approach based on numerical simulations. For instance, for the deterministic dReal benchmarks (*bouncing ball, stabilized quadcopter, cardiac*), we found that numerical simulations with uniform random sampling are almost always successful in finding violations. Hence, we adjusted the properties for these models to pose more of a challenge to simulation-based falsification tools.

Overall, we note that our technique can successfully find falsifications where 10^5 random simulations are unable to find any. Our approach scales successfully to some systems with as many as 14 state variables and in some cases systems with hundreds of modes. In general, our tool succeeds because we use

- relatively inexpensive simulations which can be easily computed in parallel, and
- multiple shooting, which is an efficient way to break down the complexity of the problem, specially in the presence of highly non-linear trajectories.

A comparison with the HTG [9] was also attempted, but did not succeed due to the restricted semantics of resets in the tool.

[5] `http://dreal.cs.cmu.edu/#!dreach.md`
We will add functionality to handle benchmarks with non-determinism in the future.

7. CONCLUSION

In conclusion, we presented and implemented a generic framework for falsification of safety properties, which relaxes the search over trajectories to trajectory segments to efficiently explore the state space. We justified this relaxation by introducing tiling-based abstraction of the hybrid state space. We also included a set of benchmarks to emphasize the practicality of our approach.

8. REFERENCES

[1] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivancic, and A. Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *Trans. on Embedded Computing Systems (TECS)*, 12:95–, 2013.

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

[3] Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. *Proc. TACAS*, pages 254–257, 2011.

[4] R. N. Bergman. Toward physiological understanding of glucose tolerance: minimal-model approach. *Diabetes*, 38(12):1512–1527, 1989.

[5] R. N. Bergman, L. S. Phillips, and C. Cobelli. Physiologic evaluation of factors controlling glucose tolerance in man: measurement of insulin sensitivity and beta-cell glucose sensitivity from the response to intravenous glucose. *Journal of Clinical Investigation*, 68(6):1456, 1981.

[6] A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. *Proc. of HSCC*, pages 451–471, 2004.

[7] H. G. Bock and K.-J. Plitt. A multiple shooting algorithm for direct solution of optimal control problems. 1983.

[8] A. Casagrande, A. Balluchi, L. Benvenuti, A. Policriti, T. Villa, and A. Sangiovanni-Vincentelli. A new algorithm for reachability analysis of hybrid automata. Technical report, Citeseer.

[9] T. Dang and T. Nahhal. Coverage-guided test generation for continuous and hybrid systems. *Formal Methods in System Design*, 34(2):183–213, 2009.

[10] A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Proc. CAV*, pages 167–170, 2010.

Table 1: **All times are in minutes**. Mean falsification times (T_{avg}) were computed on only the successful runs out of **10** total runs. Unsuccessful runs indicate a **timeout** ($\geq$ **1hr**). Columns at the left summarize the benchmarks, with the number of continuous states(**S**), inputs(**I**), parameters(**P**) and discrete Modes. Random simulations (**100,000**) and scatter-and-simulate use **4 threads**, while S-Taliro used a single thread. All the computations were done on Ubuntu 12.04, running on a 4 core Intel i7-2820QM CPU @2.30GHz with 8GB RAM.

Benchmark	S	I	P	Modes	Prop.	Random Testing		S-Taliro		Scatter-Sim	
						Num. Vio.	T (mins.)	Succ. Runs	T_{avg} (mins.)	Succ. Runs	T_{avg} (mins.)
Van der Pol [25]	2	0	0	-	P1	**0**	10.2	10	0.9	10	0.4
					P2	21	41.9	10	11.8	10	2.9
					P3	19	26.4	10	2.5	10	1.6
					P4	11	42.5	**8**	18.4	10	6.6
Lorenz [25]	4	0	0	-	P	36	132	**3**	20.6	10	3.1
Brusselator	2	0	0	-	P	429	22	10	1.1	10	0.2
B.Ball [27]	4	0	0	1	P1	3	20	**1**	11.0	10	6.5
B.Ball + S.H.M.	5	0	0	1	P1	13k	202	10	0.5	10	0.4
					P2	3k	202	10	1.4	10	0.4
					P3	378	202	10	6.3	10	0.4
Pendulum [26]	3	0	1	2	P1	**0**	6.6	10	4.7	10	4.7
					P2	**0**	25	10	5.2	10	1.2
Nav.30 [11, 27]	4	0	0	625	P	1	200	**3**	24.9	10	5.0
					Q	7	200	**1**	48.1	10	5.5
					R	1	200	10	23.2	10	17.5
					S	108	495	**3**	33.2	10	20.1
Idle Speed [8]	9	2	0	4	P	70	262	**2**	7.3	10	7.6
MPC [17]	3	0	0	69	P	1	5.6	10	0.5	10	0.6
G-I [4, 5, 12, 27]	4	0	2	2	P	1	30	10	5.0	10	4.0
Quadcopter	14	0	0	-	P	40 [6]	21.8	**8**	10.9	10	5.9
Cardiac [14, 15]	5	0	0	4	P	7	12.6	10	4.0	10	0.4

[11] A. Fehnker and F. Ivanĉić. Benchmarks for hybrid systems verification. In *Proc. of HSCC*, volume 2993, pages 326–341, 2004.

[12] M. E. Fisher. A semiclosed-loop algorithm for the control of blood glucose levels in diabetics. *Biomedical Engineering, IEEE Transactions on*, 38(1):57–61, 1991.

[13] M. Fränzle, C. Herde, S. Ratschan, T. Schubert, and T. Teige. Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. *JSAT—Journal on Satisfiability, Boolean Modeling and Computation, Special Issue on SAT/CP Integration*, 1:209–236, 2007.

[14] S. Gao, S. Kong, and E. M. Clarke. Satisfiability modulo odes. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 105–112, 2013.

[15] R. Grosu, G. Batt, F. H. Fenton, J. Glimm, C. Le Guernic, S. A. Smolka, and E. Bartocci. From cardiac cells to genetic regulatory networks. In *Computer Aided Verification*, pages 396–411. Springer, 2011.

[16] C. Herde, A. Eggers, and T. Franzle, M. Teige. Analysis of hybrid systems using HySAT. In *Third International Conference on Systems, 2008. ICONS 08.*, pages 13–18. IEEE, 2008.

[17] M. Huang, H. Nakada, S. Polavarapu, R. Choroszucha, K. Butts, and I. Kolmanovsky. Towards combining nonlinear and predictive control of diesel engines. In *American Control Conference, 2013. Proceedings of the 2004*, pages 2852–2859. IEEE, 2013.

[18] S. Kupferschmid, J. Hoffmann, H. Dierks, and G. Behrmann. Adapting an AI planning heuristic for directed model checking. In *Proc. of SPIN*, pages 35–52, 2006.

[19] S. M. LaValle. Rapidly-exploring random trees a new tool for path planning. Technical Report TR 98-11, Computer Science Dept., Iowa State University, Ames, Iowa, 1998.

[20] J. D. Meiss. *Differential Dynamical Systems*. SIAM publishers, 2007.

[21] N. S. Nedialkov, K. R. Jackson, and G. F. Corliss. Validated solutions of initial value problems for ordinary differential equations. *Applied Mathematics and Computation*, 105(1):21–68, 1999.

[22] T. Nghiem, S. Sankaranarayanan, G. E. Fainekos, F. Ivančić, A. Gupta, and G. J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Hybrid Systems: Computation and Control*, pages 211–220. ACM Press, 2010.

[23] E. Plaku, L. Kavraki, and M. Vardi. Falsification of LTL safety properties in hybrid systems. *Proc. TACAS*, pages 368–382, 2009.

[24] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *HSCC*, volume 3414, pages 573–589, 2005.

[25] S. H. Strogatz. *Nonlinear Dynamics And Chaos*. Perseus Books Group, 1 edition, 1994.

[26] A. J. van der Schaft and J. M. Schumacher. *An introduction to hybrid dynamical systems*, volume 251. Springer London, 2000.

[27] A. Zutshi, S. Sankaranarayanan, J. V. Deshmukh, and J. Kapinski. A trajectory splicing approach to concretizing counterexamples for hybrid systems. In *IEEE Conf. on Decision and Control (CDC)*. IEEE Press, 2013.

[6] out of 1,000 simulations

APPENDIX

A. LEMMAS

LEMMA A.1. *A zero cost trajectory segment σ is an actual trajectory of the system.*

PROOF. As $d(\mathbf{x}, \mathbf{y}) \geq 0$ for all $\mathbf{x}, \mathbf{y}$, $\text{COST}(\sigma) = 0$ implies that for each pair $\mathsf{dest}(\tau_i), \mathsf{src}(\tau_{i+1})$, the distance is 0. Further, $d(\mathsf{dest}(\tau_i), \mathsf{src}(\tau_{i+1})) = 0$ iff $\mathsf{dest}(\tau_i) = \mathsf{src}(\tau_{i+1})$. Thus, σ is an actual system trajectory. □

LEMMA A.2. *Given $\epsilon = 10^{-k}$ for $k > 0$, the $\sim_k$-defined tiling $\mathcal{C}$ as defined in Def. 3.3 is an ϵ-tiling of $\mathcal{X}$.*

PROOF. By definition, $\bigcup_{C_i \in \mathcal{C}} C_i = \mathcal{X}$. and any state $(\ell, \mathbf{x}) \in C_i$ satisfies the inequality $||\mathbf{x} - [\mathbf{x}]_i||_\infty \leq 10^{-k} \leq \epsilon$ (satisfying the first and third conditions of Def. 3.2). Let C_i, C_j be two different cells, we can prove that their interiors are disjoint by contradiction. Suppose there is a state $(\ell, \mathbf{x})$ that belongs to the interiors of both C_i and C_j. Thus, the first k decimal digits of $\mathbf{x}$ would have to match both $[\mathbf{x}]_i$ and $[\mathbf{x}]_j$ (which by definition are distinct), leading to a contradiction. □

LEMMA A.3. *Let C be a cell reachable from some initial cell C_0 through a robust path π of length at most $\lceil \frac{T}{\Delta} \rceil$. As $L \to \infty, K \to \infty$, the cell C belongs to the graph G explored by Algorithm 1 with probability* 1.

PROOF. Algorithm 1 will explore each node along π starting with C_0 with some probability $p > 0$. Therefore, as $K \to \infty$, for each node C' in the graph G, every robust edge outgoing from C' is discovered with probability 1. Since $L \to \infty$ and the workList data structure is "fair", we note that every discovered node is eventually processed and therefore every node along π is eventually discovered. □