Falsification of Hybrid Systems using Symbolic Reachability and Trajectory Splicing

Sergiy Bogomolov
Australian National University
Canberra, Australia

Goran Frehse
ENSTA ParisTech - U2IS
Palaiseau Cedex, France

Amit Gurung
Martin Luther Christian University
Shillong, India

Dongxu Li
Australian National University
Canberra, Australia

Georg Martius
Max Planck Institute for Intelligent Systems
Tübingen, Germany

Rajarshi Ray
National Institute of Technology Meghalaya
Shillong, India

ABSTRACT

The falsification of a hybrid system aims at finding trajectories that violate a given safety property. This is a challenging problem, and the practical applicability of current falsification algorithms still suffers from their high time complexity. In contrast to falsification, verification algorithms aim at providing guarantees that no such trajectories exist. Recent symbolic reachability techniques are capable of efficiently computing linear constraints that enclose all trajectories of the system with reasonable precision. In this paper, we leverage the power of symbolic reachability algorithms to improve the scalability of falsification techniques. Recent approaches to falsification reduce the problem to a nonlinear optimization problem. We propose to reduce the search space of the optimization problem by adding linear state constraints obtained with a reachability algorithm. We showcase the efficiency of our approach on a number of standard hybrid systems benchmarks demonstrating the performance increase in speed and number of falsifyable instances.

CCS CONCEPTS

• **General and reference → Verification**; • **Theory of computation → Timed and hybrid models**; • **Software and its engineering → Formal methods**;

KEYWORDS

trajectory splicing, falsification, reachability analysis, hybrid systems, safety verification, non-linear optimization

ACM Reference Format:
Sergiy Bogomolov, Goran Frehse, Amit Gurung, Dongxu Li, Georg Martius, and Rajarshi Ray. 2019. Falsification of Hybrid Systems using Symbolic Reachability and Trajectory Splicing . In *22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC '19), April 16–18, 2019, Montreal, QC, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3302504.3311813

HSCC '19, April 16–18, 2019, Montreal, QC, Canada

ACM ISBN 978-1-4503-6282-5/19/04…$15.00
https://doi.org/10.1145/3302504.3311813

1 INTRODUCTION

We consider the problem of finding bugs in dynamical systems with both continuously evolving variables as well as discrete states (modes), i.e., hybrid systems. This problem is known as *falsification* and different tools such as S-TaLiRo [3], Breach [15], and FalStar [14] are available. In this paper, we consider the special case of finding a trajectory that starts inside a given set of initial states and that leads to a given set of bad states. In addition to searching for bugs, our kind of falsification problem can arise naturally as part of a verification process. When one has failed to verify a system (proving that bad states are not reachable), it is often not clear whether the system actually violates the property or whether the verification tool was simply unable to show safety, e.g., due to overapproximations. Finding a concrete counterexample is then equivalent to our type of falsification problem. Such counterexamples are highly valuable to designers since they are essential for understanding bugs in the design or modeling mistakes that may have tripped up the verification process. In the present work, we propose to enhance existing falsification techniques by incorporating techniques from verification, and in particular reachability analysis.

One way to establish safety of the system is by a *reachability analysis* of the model wherein all reachable states of the automaton are computed. Computing the accurate set of reachable states for a hybrid system is in general intractable. Therefore, existing techniques produce a conservative over-approximation [1, 6, 10, 18, 29]. A commonly used over-approximation of the accurate reachable states of a dynamical system is known as a *flowpipe*. Verification of safety properties of a model is then implied from the emptiness of the intersection between the flowpipe and the specified unsafe set of states. Generally, one cannot resort to reachability analysis straightforwardly for a solution to falsification problems: a non-empty intersection of flowpipe and unsafe state set does not necessarily imply the violation of safety properties, essentially due to the over-approximation in a flowpipe.

Recently, a method for falsifying hybrid systems based on *trajectory splicing* has been proposed in [34]. The method starts with a candidate sequence of disjoint trajectory segments, each of which describes the evolution of the system in one discrete state. By minimizing the sum of the distance between each segment pair, the falsification task is solved upon identifying a segment sequence with a zero minimum cost.

We address two problems that may arise when applying the approach from [34] to complex hybrid systems: First, the trajectory splicing relies on a sequence of discrete states that contains a valid counterexample as input, which is assumed as known apriori. In fact, the sequence is usually manually picked by observing simulations of the system [34]. Such an approach can be hard to scale to large hybrid systems with complex behaviors. Also, simulations are inherently incomplete, in the sense that one may not always be able to observe a violating trajectory even if the system is unsafe. The second problem is that the approach initializes the nonlinear optimization problem using the bounding information from the system model, e.g. invariant constraints. Since the reachable set is usually a subset of the region defined by such explicitly attainable constraints in the model, the bounding information acquired this way can be rather coarse in practice. In particular on instances where such information is partially missing, the nonlinear optimization problem can explore an unnecessarily large search space and is also more likely to fall into local minima and cost more iterations to converge. This prevents the optimizer from a fast and stable convergence.

Contributions: In this paper, we propose to enhance the trajectory splicing approach with symbolic reachability analysis, which helps to solve the aforementioned issues.

(1) We use reachability analysis to automatically produce candidate sequences of discrete states. As opposed to [34], our approach is independent of expert knowledge of the system and provides a violating counterexample as soon as one can be found. This provides a fully automated falsification approach for hybrid systems.
(2) We exploit the constraints from the symbolic reachability analysis to efficiently prune the search space of the nonlinear optimization problem. This helps the optimizer and leads to a faster and more stable convergence. As a result, the falsification has a better chance to succeed.
(3) We propose a trajectory splicing-validation loop, which assures that the counterexample rigorously respects the invariant conditions.

We implemented the ideas presented in this paper in a state-of-art reachability analysis tool XSpeed [29] and compare with the baseline approach in [34]. We find that our approach can usually identify counterexample trajectories faster and more consistently. Particularly on longer trajectories, our approach is a clear winner. The relevant artifacts are publicly available online[1].

The rest of the paper is structured as follows. In Sect. 2, we briefly recall concepts of hybrid systems, reachability analysis, and falsification. Sect. 3 describes our main proposal to enhance falsification with symbolic reachability analysis and trajectory splicing, including how we take advantage of the reachability result to prune the search space. In Sect. 4, we present our practical evaluation on standard hybrid system models. We recall existing works in Sect. 5 and conclude the paper in Sect. 6 with discussions on possible future work.

[1] http://xspeed.nitmeghalaya.in/falsification.html

2 HYBRID AUTOMATA, REACHABILITY, AND FALSIFICATION

In this section, we formalize our notions of hybrid automata, reachability analysis, and falsification, based on established concepts from literature.

2.1 Hybrid Automata

Hybrid automata are a mathematical model of hybrid systems, i.e., systems exhibiting both continuous and discrete dynamics. A *state* of a hybrid automaton is a 2-tuple (ℓ, x), where ℓ is a location (discrete state, mode) and x is an n-dimension real vector representing the values of the continuous variables.

Definition 2.1 (Hybrid automaton). [2] A *hybrid automaton* (*HA*) is a tuple $(\mathcal{L}, X, Inv, Init, Flow, Trans)$ where:

- $\mathcal{L}$ is the set of locations of the hybrid automaton.
- $X = \{x_1, \ldots, x_n\}$ is the set of continuous variables.
- $Inv : \mathcal{L} \rightarrow 2^{\mathbb{R}^n}$ maps every location to a subset of $\mathbb{R}^n$, called the invariant. All trajectories must lie inside the invariant.
- $\mathcal{X}_0$ is a pair (ℓ_{init}, C_{init}) such that $\ell_{init} \in \mathcal{L}$ and $C_{init} \subseteq Inv(\ell_{init})$. It defines the set of initial states.
- *Flow* is a mapping of the locations to ODE equations of the form $\dot{x} = f(x)$, called the flow equation of the location. A flow equation defines the evolution of the system variables within a location.
- *Trans* is the set of (discrete) transitions of the automaton. A transition $\delta = (\ell, \mathcal{G}, \mathcal{M}, \ell')$ changes a state (ℓ, x) of the hybrid automaton to another state (ℓ', x') given that x lies in the guard set $\mathcal{G} \subseteq \mathbb{R}^n$. The linear map $\mathcal{M} : \mathbb{R}^n \rightarrow \mathbb{R}^n$ changes x to the new vector $x' = Mx + b$, where M is an $n \times n$ matrix and $b \in \mathbb{R}^n$. We denote the guard set and the linear map of a transition δ with $\mathcal{G}(\delta)$ and $\mathcal{M}(\delta)$.

We consider flows in the form of *linear differential equations*, i.e.

$$\dot{x} = Ax + u,$$

where $A \in \mathbb{R}^{n \times n}, x \in \mathbb{R}^n, u \in \mathbb{R}^n$, which implies that the closed-form solution of $flow(x, t)$ exists [29]:

$$flow(x, t) = e^{At}x + \Phi(A, t)u. \quad (1)$$

If A is invertible, $\Phi(A, t) = A^{-1}(e^{At} - I)$. Otherwise, it can be computed as a submatrix of the matrix exponential of a block matrix according to [18].

The behaviors of a hybrid automaton are formally described as runs, which are alternating sequences of time elapse, during which x evolves according to (1), and discrete transitions, which update x according to $x' = \mathcal{M}(x)$.

Definition 2.2 (Run). A *run* of a hybrid automaton is a sequence

$$(\ell_0, x_0) \xrightarrow{\tau_0} (\ell_0, y_0) \xrightarrow{\delta_0} (\ell_1, x_1) \xrightarrow{\tau_1} (\ell_1, y_1), \ldots, \xrightarrow{\delta_{N-1}} (\ell_N, x_N) \xrightarrow{\tau_N} (\ell_N, y_N)$$

such that for all $i = 0, \ldots, N$, (i) $(\ell_0, x_0) \in Init$; (ii) $\forall t \in [0, \tau_i]$, $flow_{\ell_i}(x_i, t) \in Inv(\ell_i)$; (iii) $flow_{\ell_i}(x_i, \tau_i) = y_i$; (iv) $y_i \in \mathcal{G}(\delta_i)$ and $x_{i+1} = \mathcal{M}(y_i)$. The times τ_i are called the dwell times of the system in the respective locations ℓ_i.

2.2 Symbolic Reachability Analysis

We give a brief summary of reachability analysis and describe the algorithm we use to obtain a symbolic abstraction of the reachable states.

Definition 2.3 (Reachability). A state (ℓ, x) is *reachable* if there is a run of the system such that $(\ell_0, x_0) \in \mathcal{X}_0$ and $(\ell_N, x_N) = (\ell, x)$.

Reachability analysis tools produce a conservative approximation of the reachable states of the automaton. Reachable states can be expressed as a union of *symbolic states*. A symbolic state s is a tuple (l, C) such that $l \in \mathcal{L}$ and $C \subseteq \text{Inv}(l)$. An *unsafe symbolic state set* $\mathcal{S}_B$ defines a set of error states of the automaton. If $\exists\, s_B \in \mathcal{S}_B$ such that s_B is reachable, then we say the automaton is *unsafe*. Since we are interested in finding short counterexamples, we assume that the reachability analysis is carried out as a breadth-first search up to a given bounded depth N.

Definition 2.4 (Symbolic Reachability and Exploration Graph). Given a *HA*, a set of unsafe symbolic states $\mathcal{S}_B$, and a search depth N, a breadth-first *symbolic reachability analysis* (abbr. *symbolic reachability*) produces an *exploration graph* (S, V), where S is a set of symbolic states, containing a root state s_0, and V is a set of edges $(s, s') \in S \times S$. Each edge (s, s') is associated with a transition, denoted as $\delta(s, s')$. We say that a run

$$(\ell_0, x_0) \xrightarrow{\tau_0} (\ell_0, y_0) \xrightarrow{\delta_0} (\ell_1, x_1) \xrightarrow{\tau_1} (\ell_1, y_1), \ldots, \xrightarrow{\tau_N} (\ell_N, y_N)$$

of *HA matches* a path

$$s_0, s_1, \ldots, s_N = (\ell_0, C_0), \ldots, (\ell_N, C_N)$$

in the exploration graph if $x_i \in C_i$ and $\delta_i = \delta(s_i, s_{i+1})$ for $i = 0, \ldots, N$. The graph is complete up to depth N is the sense that every run of length up to N has a matching path in the exploration graph. The converse is not necessarily true: A path in the exploration graph may have no matching run.

Our symbolic reachability analysis algorithm is a classical fixed-point computation over a symbolic state space based on support functions [20, 29]. The algorithm starts by initializing a queue of symbolic states with the set of initial states. For each symbolic state (ℓ, C), two image computations are carried out: first the continuous states reachable by time elapse via *Flow* are computed. This is called flowpipe approximation. Then for each outgoing transition of location ℓ, the symbolic state in the new location (also called a "transition image") is computed. All the obtained symbolic states are put back on the queue, unless they are contained in a previously visited symbolic state. We briefly describe those two image computations.

Flowpipe Approximation. We discretize time by a fixed time step Δt and overapproximate the reachable states with a union of convex sets $\Omega_0, \ldots, \Omega_{N-1}$, where $N = {}^{T}/_{\Delta t}$. The convex set Ω_i overapproximates the reachable states over a time interval $[i\Delta t, (i+1)\Delta t]$. Let $\mathcal{R}_{[0,T]}(C)$ denote the set of reachable states over a time horizon T. Let Ω_i be defined as

$$\Omega_0 = CH(C, e^{A\Delta t}\mathcal{X}_0 \oplus \Delta t u \oplus \alpha\mathcal{B}) \tag{2}$$

$$\Omega_{i+1} = e^{A\Delta t}\Omega_i \oplus \beta\mathcal{B} \tag{3}$$

where $CH(\cdot)$ is the convex hull operation; α, β are constants depending on $\mathcal{X}_0$, u and Δt; $\mathcal{B}$ is the unit ball in p-norm [26]. Then

$$\mathcal{R}_{[0,T]}(C) \subseteq \bigcup_{i=0}^{N-1} \Omega_i.$$

Transition Image. For each outgoing transition $(\ell, \mathcal{G}, \mathcal{M}, \ell')$, we compute the image as follows. First, we identify which Ω_i overlap with the guard $\mathcal{G}$, i.e.,

$$\Omega_i \cap \mathcal{G} \cap Inv(\ell) \neq \emptyset.$$

In case there are multiple sets intersecting with the guard, we adopt the box-template hull aggregation [18] to limit the number of symbolic states. For each of the sets that overlap with the guard, we compute the image

$$\Omega_i' = \mathcal{M}\big(\Omega_i \cap \mathcal{G} \cap Inv(\ell)\big) \cap Inv(\ell'). \tag{4}$$

The new symbolic state (ℓ', C') is added to the states of the exploration graph, along with the edge $\big((\ell, C), (\ell', C')\big)$ and the associated transition.

2.3 Counterexamples and Falsification

Symbolic reachability provides us with an exploration graph (S, V), from which we can extract a set of paths that lead from the initial to the bad states. We aim at identifying, which of these paths correspond to actual runs of the system.

Definition 2.5 (Abstract and concrete counterexamples). An *abstract counterexample* in an exploration graph (S, V) is a sequence

$$s_0, \delta_0, s_1, \delta_1, \ldots, \delta_{N-1}, s_N,$$

of symbolic states $s_i \in S$ and transitions $\delta_i = \delta(s_i, s_{i+1}) \in V$ such that $s_0 \subseteq Init$ and $s_N \cap \mathcal{S}_B \neq \emptyset$. We say that an abstract counterexample is also *concrete* if there is a matching run in *HA*. The length of a counterexample is $N + 1$, the number of locations visited.

In the process of checking whether an abstract counterexample is concrete, we will go through intermediate stages where the counterexample is concrete in the sense that there are corresponding solutions of the ODEs, but where invariants, guards and transition maps may not be satisfied. We formalize these intermediate counterexamples a follows.

Definition 2.6 (Relaxed counterexample). Given an abstract counterexample

$$(\ell_0, C_0), \delta_0, (\ell_1, C_1), \ldots, \delta_{N-1}, (\ell_N, C_N),$$

a corresponding *relaxed counterexample* is a sequence

$$(\ell_0, x_0) \xrightarrow{\tau_0} (\ell_0, y_0) \xrightarrow{\delta_0} (\ell_1, x_1) \ldots \xrightarrow{\tau_N} (\ell_N, y_N)$$

such that $\forall i \in [0, N]$ (1) $x_i, y_i \in C_i$ and (2) $flow_{\ell_i}(x_i, t) \in C_i$, $\forall t \in [0, \tau_i]$ with $flow_{\ell_i}(x_i, \tau_i) = y_i$. In other words, relaxed counterexamples are not required to satisfy invariants, guards, and transition maps of *HA*.

The above definitions allow us to define *falsification tasks* in hybrid systems, and it is our target in this paper.

Definition 2.7 (Falsification task). Given an HA and a set of unsafe symbolic states $\mathcal{S}_B$, a *falsification task* $(HA, \mathcal{S}_B)$ is to find a concretizable counterexample of HA.

The exploration graph produced by symbolic reachability defines a set of abstract counterexamples of given length: the set of paths in the graph that lead from the initial to the bad states. We can check each of these abstract counterexamples using falsification techniques for a given path. If no concrete counterexample is found, we can increase the search depth. If a user-defined depth is exceeded, we can report that no counterexamples have been found up to this depth. The precise mechanism for checking counterexamples is presented in the next section.

3 COMBINING FALSIFICATION AND SYMBOLIC REACHABILITY

In this section, we propose an approach to solving a falsification task using symbolic reachability and trajectory splicing. An overview of our approach is shown in Fig. 1. Given a falsification task, we first apply symbolic reachability analysis to find a set of abstract counterexamples. We use breadth-first-search (BFS) as the underlying search algorithm in the space of symbolic states, such that the number of discrete transitions is minimized. This helps to minimize the dimension of the non-linear optimization problem. We run symbolic reachability analysis up to a fixed number of discrete transitions. Each time an abstract counterexample is found, we formulate the falsification task as a nonlinear optimization problem using the idea of trajectory splicing, while taking into account the reachability analysis result to refine the bounding constraints.

If the nonlinear problem can be solved with a cost function close to zero, we acquire an instance of a relaxed counterexample. We then call the validation routine to check whether such a relaxed counterexample respects the invariant constraints or not. In case it doesn't, we refine the optimization problem to exclude such an invalid trajectory from the solution space. Depending on the setting, we terminate the process either after exhaustive attempts to splice all the abstract counterexamples or after finding a concrete trajectory.

In Sect. 3.1, we briefly recall the approach [34] to reduce a falsification task to a *trajectory splicing* problem. In Sect. 3.2, we explain the idea of using symbolic reachability analysis to improve the performance of solving a trajectory splicing problem. In Sect. 3.3, we describe how we formulate the trajectory splicing problem as a nonlinear optimization problem. In Sect. 3.4, we explain the necessity of validating a relaxed counterexample against invariant constraints and present the trajectory validation loop.

3.1 Concretizing Counterexamples Using Trajectory Splicing

A segmented trajectory defines a set of trajectories within the location sequence that is specified by an abstract counterexample.

Definition 3.1 (Segmented trajectory and trajectory segments). Given an abstract counterexample (ℓ_0, C_0), δ_0, (ℓ_1, C_1), ..., δ_{N-1}, (ℓ_N, C_N), a segmented trajectory P is defined to be a sequence of

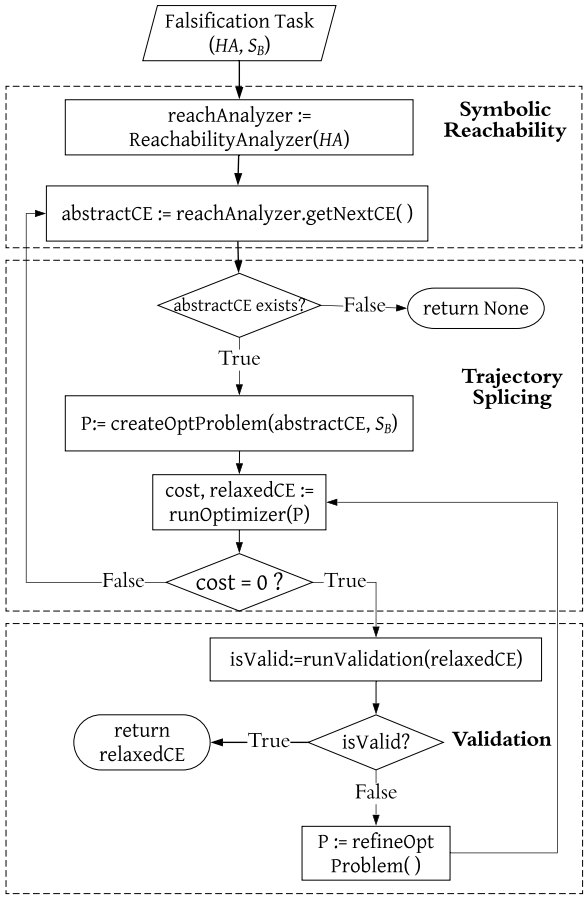


Figure 1: Proposed falsification scheme overview: Given an HA, `ReachabilityAnalyzer()` creates an instance of reachability analyzer. On each call of `getNextCE()`, the analyzer tries to find and returns an abstract counterexample (`abstractCE`). `createOptProblem()` creates an optimization problem given an abstract counterexample and a set of unsafe states. `runOptimizer()` runs an optimization routine given an optimization problem and returns a relaxed counterexample (`relaxedCE`) if cost equals to zero. `runValidation()` runs the trajectory validation routine and test whether the relaxed counterexample respects invariant conditions. `refineOptProblem()` refines the optimization problem taking into account the validation result.

$N+1$ trajectory segments $\{\pi_0, \pi_1, \ldots, \pi_N\}$ [34]:

$$\begin{pmatrix} \pi_0: & (\ell_0, x_0) \xrightarrow{\tau_0} (\ell_0, y_0) \\ \pi_1: & (\ell_1, x_1) \xrightarrow{\tau_1} (\ell_1, y_1) \\ \vdots & \cdots \\ \pi_N: & (\ell_N, x_N) \xrightarrow{\tau_N} (\ell_N, y_N) \end{pmatrix} \tag{5}$$

such that (i) the location sequence is specified by the abstract counterexample, i.e. $(\ell_0, \ell_1, \ldots, \ell_N)$. (ii) $(\ell_0, x_0) \in Init$ and $(x_i, y_i) \in C_i$ (iii) $y_i \in \mathcal{G}(\delta_i)$, $\forall i \in [0, N-1]$, where $\mathcal{G}(\delta_i)$ is the guard set

of the transition δ_i and (iv) $flow_{\ell_i}(x_i, t) \in C_i$, $\forall t \in [0, \tau_i]$ with $flow_{\ell_i}(x_i, \tau_i) = y_i$, $\forall i, 0 \leq i \leq N$

Generally, a segmented trajectory may not depict a trajectory of the hybrid automaton since the segments can be disconnected with each other. However, if it is possible to splice the segmented trajectories, such a sequence indicates an actual trajectory of the system. Splicing can be formulated as a nonlinear optimization problem with the objective of minimizing the distance between the end-points of the trajectory segments. The overview of the optimization problem formulation using Definition 3.1 is as follows:

$$\underset{x_0,\ldots,x_N,\tau_0,\ldots,\tau_N}{\text{minimize}} \sum_{i=0}^{N-1} \text{COST}(\pi_i, \pi_{i+1}) \tag{6}$$

subject to

$$\text{COST}(\pi_i, \pi_{i+1}) = \text{dist}(\mathcal{M}(\delta_i)(y_i), x_{i+1}) \tag{7}$$

$$0 \leq \tau_i \leq T, \qquad \forall i : 0 \leq i \leq N-1 \tag{8}$$

$$x_i, y_i \in Inv(\ell_i), \qquad \forall i : 0 \leq i \leq N \tag{9}$$

$$x_0 \in C_{\text{Init}} \tag{10}$$

$$y_i \in \mathcal{G}(\delta_i), \qquad \forall i : 0 \leq i \leq N-1 \tag{11}$$

$$flow_{\ell_i}(x_i, t) \in Inv(\ell_i), \qquad \forall i : 0 \leq i \leq N, \ \forall t \in [0, \tau_i], \tag{12}$$

$$y_i = flow_{\ell_i}(x_i, \tau_i), \qquad \forall i : 0 \leq i \leq N \tag{13}$$

The function dist() is a mapping $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ which computes the Euclidean distance between two vectors. We remark that since we are considering $flow$ in the form of linear differential equations, one can compute y_i given x_i and τ_i according to Eq. 1. Therefore, x_i and τ_i are the only two sets of decision variables in this formulation.

When the cost is zero, the solution is a spliced segmented trajectories and therefore a concrete counterexample. No solution with zero cost implies the non-existence of a concrete counterexample for this abstract path.

3.2 Search Space Pruning Using Abstract Counterexample

According to Definition 2.4, a concrete trajectory segment in a location ℓ_i must be within C_i. Therefore, one can always take into account the constraint $C_i \subseteq Inv(\ell_i)$ to restrict the search space of the optimization problem (6). Since the reachable set is a subset of the invariant, it provides finer constraints. In addition, the invariant is quite often unbounded, in which case one can only use an infinite bound to initialize the optimization problem. In the following, we describe the additional constraints we deduce from the flowpipe. We start by introducing the following definition:

Definition 3.2 (Projection function). A projection function $\text{proj}_d(x)$ is a mapping from $\mathbb{R}^n \to \mathbb{R}$ that projects vector x to its d-th component, i.e. $\text{proj}_d(x) = e_d \cdot x$, where e_d is a unit vector with d-th component equal to 1 and all others equal to 0.

Note that projection gives an easy way to produce lower and upper bounds using linear programs. Given Definition 3.2 and an abstract counterexample (ℓ_0, C_0), δ_0, (ℓ_1, C_1), …, δ_{N-1}, (ℓ_N, C_N), we now formalize the constraints as follows.

Constraints on the starting point x_i. The starting point in location ℓ_i belongs to the set Ω_0, which over-approximates the set of reachable states over the time interval $[0, \Delta t]$. We add the constraint $x_i^{min} \leq x_i \leq x_i^{max}$, where

$$x_{i,d}^{min} = \min_{x \in \Omega_0} \{\text{proj}_d(x)\},\ d \in \mathbb{Z}, d \in [1, n] \tag{14}$$

$$x_{i,d}^{max} = \max_{x \in \Omega_0} \{\text{proj}_d(x)\},\ d \in \mathbb{Z}, d \in [1, n] \tag{15}$$

Constraints on the ending point y_i. (i) if ℓ_i is not the location of the last segment, the ending point for the trajectory segment in ℓ_i should satisfy the guard constraints. Additionally, we can deduce from the reachability analysis that the set of reachable states in l_i is a subset of C_i, i.e. $y_i \in C_i$. Therefore, the constraint we have is

$$y_i \in \mathcal{G}(\delta_i) \cap C_i, \ where\ i < N \tag{16}$$

(ii) if ℓ_i is the location of the last trajectory segment, the endpoint y_N should be inside the unsafe states $\mathcal{S}_B = (\ell_B, C_B)$. Following the same reasoning as in (16), we deduce $y_N \in C_N$. We add the constraint below:

$$y_N \in C_B \cap C_N \tag{17}$$

Constraints On dwell time. To obtain precise constraints over the dwell times τ_i, we add an extra variable t to the hybrid automaton, with flow $\dot{t} = 1$. Each discrete transition resets t to zero so that the time variable measures only the time elapsed in the current location of the automaton. Consequently, the flowpipe computed from the symbolic reachability analysis also provides reachability information on the dwell time in each mode. We can then obtain constraints on t, i.e., over the dwell times, in a similar way as on x_i and y_i. Since we know that y_i must satisfy the guard of the transition δ_i for a discrete transition to take place, we may take the projection of the time dimension from the reachable set C_i in a location ℓ_i and intersect the guard set of the transition δ_i, i.e. $\mathcal{G}(\delta_i)$ in order to obtain the bounds on the dwell time at location ℓ_i. We add the constraint $\tau_i^{\min} \leq \tau_i \leq \tau_i^{\max}$, where

$$\begin{aligned} \tau_i^{\min} &= \min_{x \in C_i \cap \mathcal{G}(\delta_i)} \{\text{proj}_{d'}(x)\} \\ \tau_i^{\max} &= \max_{x \in C_i \cap \mathcal{G}(\delta_i)} \{\text{proj}_{d'}(x)\}, \end{aligned} \tag{18}$$

where d' is the index for the time dimension. The projection of $C_i \cap \mathcal{G}(\delta_i)$ is always a subset of $[0, T]$ and ensures to prune the search space.

3.3 Finding Trajectories Using Nonlinear Optimization

Solving the optimization problem (6)–(13) cannot be done in a straightforward way because the cost function is nonlinear. Even for systems with linear $flow$ dynamics, the cost is nonlinear in the dwell times τ_i. One way to solve the problem efficiently is to convert most of the constraints into soft constraints, i.e., add them as penalty terms to the cost function (Lagrange multipliers) [8]. If all constraints are converted, then an unconstrained optimizer can be employed. Here, we keep some constraints hard and use a constrained nonlinear optimizer. It is reasonable to constrain

the initial point of each location to be in the invariant set and also keep the time limits as hard constraints. In contrast, the end-points of each location are nonlinearly coupled to the starting point and should be soft constraints to make the cost function smooth and differentiable. A constraint of the form $p \in \mathcal{P}$, where $\mathcal{P}$ is a polyhedron, becomes an additive part of the cost, which is 0 if $p \in \mathcal{P}$ and otherwise the sum of (squared) distances to the violated hyperplanes.

Definition 3.3. Given a polyhedron $\mathcal{P} = \{x \in \mathbb{R}^n \mid Wx \leq b\}$. W is an $m \times n$ matrix, where each row is normalized. We reflect the constraint $p \in \mathcal{P}$ in the optimization problem by adding an extra term computing the sum of (squared) distance to the violated hyperplanes.

$$\mathrm{dp}(p, \mathcal{P}) = \sum_{i=1}^{m} \max\left(0, \left(\sum_{j=1}^{n} W_{i,j} p_j\right) - b_i\right)^2 \tag{19}$$

The relaxed optimization problem is as follows:

$$\underset{x_0,\ldots,x_n,\tau_0,\ldots,\tau_n}{\text{minimize}} \sum_{i=0}^{n-1} \mathrm{CC}(\pi_i, \pi_{i+1}) + \mathrm{dp}(y_n, C_B) \tag{20}$$

subject to

$$\tau_i^{\min} \leq \tau_i \leq \tau_i^{\max}, \qquad \forall i : 0 \leq i \leq n-1 \tag{21}$$

$$x_0 \in C_0 \tag{22}$$

$$x_i \in Inv(\ell_i), \qquad \forall i : 0 \leq i \leq n \tag{23}$$

$$x_i^{min} \leq x_i \leq x_i^{max}, \qquad \forall i : 0 \leq i \leq n, \tag{24}$$

with

$$\mathrm{CC}(\pi_i, \pi_{i+1}) = \mathrm{dist}\big(\mathcal{M}(\delta_i)(y_i), x_{i+1}\big) + \mathrm{dp}(y_i, C_i) + \mathrm{dp}\big(y_i, \mathcal{G}(\delta_i)\big) \tag{25}$$

$$y_i = flow_{\ell_i}(x_i, \tau_i). \tag{26}$$

The only term that is missing from the new cost function is the invariant constraint (12). It is omitted because it would add infinite terms ($\forall t$). To ensure that this constraint is satisfied we add a validation routine, see Sect. 3.4 below.

The new cost function (20) has a particular structure that allows for a single optimization procedure. Note that all terms in the cost function are positive (≥ 0). If the constraints Eq.(21)–(24) are satisfied and (20) is zero then the original cost (6) is zero with all constraints satisfied. Thus, there is no trade-off between terms and the values of the Lagrange multipliers can be fixed to 1. Since the cost function is differentiable, we supply analytical gradients to the solver, see Appendix A.1. In practice, due to numerical calculations, the cost needs to be below a certain threshold in order to be considered successful. We observe that usually a cost below a threshold of 10^{-6} is sufficient to indicate a concrete counterexample.

3.4 Trajectory Validation

The particularly challenging constraint to encode in the optimization problem is the invariant constraint (12). The difficulty arises since $y_i(t) = flow_{\ell_i}(x_i, t)$ is a continuous function of t in the interval $[\tau_{min}, \tau_{max}]$. Therefore it is not possible to explicitly encode the containment check of $y_i(t)$ in the invariant of location ℓ_i. To address this problem, we propose to remove this constraint from the optimization problem and instead separately check for the possible violation of this constraint in the spliced trajectory. In this way, we delay the checking of this constraint to a later step that we call *trajectory validation*. During trajectory validation, (12) is explicitly checked at discrete time points t_k with the numerically computed $y_i(t_k)$. This checking is performed by computing the distance of the point $y_i(t_k)$ to the invariant set. A distance of zero implies that the trajectory point satisfies the invariant, whereas a non-zero distance implies a violation of the invariant. We consider only polyhedral invariants in this work. Our point to polyhedron distance function is shown in (19). In order to deal with numerical inaccuracy in the distance computation, a distance of less than 10^{-3} is considered to be zero. Note that the precision of the validation phase is limited to the number of discrete points in the trajectory checked for validity. Checking the trajectory samples for validity that are separated by a small time-step is desirable for precision but expensive because of the large number of distance computations. In our implementation, we keep the number of samples for validation as a parameter that can be tuned to meet the desired accuracy. We call this parameter the number of *validation steps*. The larger the number of validation steps, the better the accuracy is.

If a trajectory is found to be violating the invariant, we record all trajectory segments π_i and dwell times τ_i^* which violate the invariant ($\mathrm{dp}(y_i(\tau_i^*), Inv_i) > 0$). We call these the *refinement points* for the violating trajectory. The optimization problem is modified to avoid generating invalid trajectories at those points. This is done by adding the distance of $\mathrm{dp}(y_i(\tau_i^*), Inv_i)$ (for all refinement points) to the cost function. Our algorithm keeps adding the refinement points and restarting the solver until a valid trajectory is found.

4 EXPERIMENTS

We compare the performance of the trajectory splicing algorithm with additional constraints on dwell times and starting points of the trajectory segments (referred to as FC), and the approach in [34], which does not employ any extra constraints derived from a symbolic reachability analysis (referred to as WoFC).

4.1 Experiment Setup

Implementation and environment. We implement our ideas in the state-of-art reachability analysis tool XSpeed[29]. The reachability analysis is performed using a support-function-based algorithm as sketched in Sect. 2.2, with a *box-template* directional abstraction [**?**], and with a time-step of 0.01 for computing flowpipes.

The optimization problem for splicing is solved using the open-source library *Nlopt* [21], which provides an interface to a number of non-linear optimization routines in *C++*. In our experiments, we use a gradient-based optimizer MMA (Method of Moving Asymptotes) [31]. We fix two parameters of the optimizer, i.e. *error tolerance* and *max-iterations*. *Error tolerance* measures the accuracy of splicing. A splicing is accepted when the minimum of the cost function is less than or equal to the specified error tolerance, which we set as 10^{-6}. The number of iterations by the optimization routine is bounded by *max-iterations*, which we set as 20K.

All the experiments are performed on a machine with Intel Xeon CPU E5-1650 v4, 3.60GHz, 64 GB RAM.

Benchmark description. We run experiments to search counterexamples on different instances (19 in total) from four models. Each instance considers a unique initial and an unsafe set. These instances include Platooning [7], the oscillator [18], the Filtered oscillator [18] and the Navigation benchmarks [16]. The Platooning benchmark models the motion of a platoon of vehicles following a leader vehicle. The vehicles in the platoon communicate their respective position, velocity and acceleration with each other so that each vehicle can maintain a safe distance with the vehicle that is ahead in the platoon. The effect of the loss in communication is modeled as a "dropout" mode, which changes the dynamics of the automaton. The oscillator is an affine system with two variables x, y which oscillate between two equilibria. The Filtered oscillator models an oscillator system together with k-filters. The filters smoothen x to produce the output z. The Navigation benchmark models the motion of an object in a plane, partitioned into unit squares. The motion dynamics of the object varies in each square and changes instantaneously as the object crosses from one square to another in the plane. The details, e.g. dimensionality and the number of locations, of each instance can be found in all the tables in the following section.

4.2 Results

In our evaluation, we empirically evaluate the *coverage* of our approach and demonstrate that it finds a *larger number* of concrete counterexamples in most of the cases. Note that by adding extra constraints into the optimization problem, we can reduce its search space. At the same time, the structure of the optimization problem might get more complicated due to introduced non-linear terms. Therefore, it is important to investigate whether the embedding of additional constraints in fact pays off. In this setting, we first run the reachability analysis. While exploring the symbolic state space, the reachability algorithm collects all the found abstract counterexamples. After this preprocessing phase has been finished, we iterate over all the abstract counterexamples and try to concretize each of them. We report the total number of concrete counterexamples found. Here, a higher number of found concrete counterexamples clearly indicates a better coverage.

Table 1 shows the results of the approaches FC and WoFC with respect to the coverage. We observe the following:

- By employing additional constraints, the optimization engine succeeds in finding at least one concrete counterexample for all the 19 instances. In the absence of these constraints, the optimizer only succeeds on 7 instances.
- On instances solved by both approaches (seven in total), FC on average finds a concrete counterexample 2-4 times faster than WoFC on 5 instances: Nav (4), Nav (5), Nav (6), Nav (8) and Platooning (1).
- On instances Nav (1) and Nav (9), WoFC found more concrete counterexamples using less time than FC.

Thus, we conclude that extra constraints improve the robustness of the concretization algorithm.

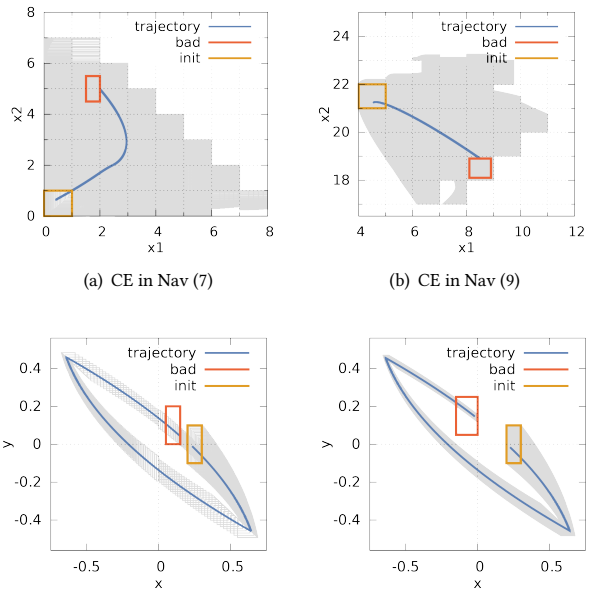
Figure 2 shows the generated concrete counterexamples of different lengths for two instances of the navigation benchmark, as well as the instances of oscillator and filtered oscillator benchmarks.

Table 1: Results of the FC and WoFC approaches. #AbsCE is the number of abstract counterexample from the reachability analysis. This number can be large due to the over-approximations in the flowpipe computation. The results of the benchmarks marked (*) are obtained by using a time-step of 0.1 during flowpipe computation. #ConCE is the number of concrete counterexamples found by each approach. The best performer is highlighted in bold.

Benchmark	Dim.	#Loc	#AbsCE	#ConCE	
				FC	WoFC
Nav (1)	5	9	2	1	**2**
Nav (2)	5	25	5	**2**	-
Nav (3)	5	25	275	**1**	-
Nav (4)*	5	225	969	**6**	2
Nav (5)*	5	225	414	**2**	1
Nav (6)*	5	400	19	**4**	1
Nav (7)*	5	400	101	**9**	-
Nav (8)	5	625	10	**3**	1
Nav (9)	5	625	29	3	**4**
Nav (10)	5	625	2	**2**	-
Nav (11)	5	625	2	**2**	-
Nav (12)	5	625	2	**2**	-
Platooning (1)	11	2	1	**1**	**1**
Platooning (2)	11	2	2	**1**	-
Oscillator (1)	3	4	3	**3**	-
Oscillator (2)	3	4	1	**1**	-
Oscillator (3)	3	4	2	**2**	-
Filtered Osc. (1)	11	4	6	**2**	-
Filtered Osc. (2)	11	4	2	**1**	-

4.2.1 Efficiency. In this setting, we demonstrate the efficiency of our approach on solving a given falsification task. The falsification setting corresponds to the overall scheme depicted in Fig. 1, where we interleave the reachability analysis and optimization calls. Once an abstract counterexample is found by the reachability analysis, we try to concretize and validate it. The process terminates as soon as it finds a concrete counterexample. We record the accumulative reachability analysis (RA Time) time, accumulative optimization time (Opt. Time) and sum them up as the time to solve the falsification task (Fal. Time). Smaller falsification time is in favour. Table 2 shows the result for the falsification setting. We observe that FC results in faster falsification in comparison to WoFC, in 15 out of the 19 instances. Also, WoFC is not successful in finding any falsifying trajectory in 12 of the instances where FC succeeds. We note that in 4 of the instances, WoFC results in faster falsification than FC. This shows that having additional constraints in the optimization problem is not always beneficial in terms of performance of a falsification task.

We remark that we do not make a direct comparison with the tool S-TaLiRo [3] as it would boil down to the comparison between the optimization engines in Matlab and C++. Instead, we concentrate on showing the influence of additional constraints from the reachability analysis on the performance of an optimization engine.



(a) CE in Nav (7) (b) CE in Nav (9)

(c) CE in Osc. (2) (d) CE in F-Osc.(1)

Figure 2: (a) - (b): Counterexamples of length 9 and 8 for Nav (7) and Nav (9) benchmark instances, respectively. (c): A counterexample of the length 5 on Oscillator (2). (d): A counterexample of the length 4 on Filtered Oscillator (1). The yellow box shows the set of initial states. The red box shows the set of unsafe states. The gray region shows the flowpipe. The blue curve shows the concrete counterexample.

The experiments provide evidence that the optimization engine indeed benefits from extra constraints.

5 RELATED WORK

Our work is closely related to optimal control, RRTs and task planning.

Optimal control. Studies on falsification from the perspective of optimal control can be broadly classified into single-shooting and multiple-shooting approaches. A single-shooting approach searches directly for a full counterexample, while a multiple-shooting search for piecewise but possibly disjoint trajectory segments. In this regard, the basis of our work can be seen as a *direct multiple-shooting* approach [9] in the domain of hybrid systems. Recent works in optimal control [23, 33] propose to combine simulations and (heuristic-guided) graph search to identify the most promising abstract counterexample. One common problem these approaches have is that they usually provide a weak (if any) guarantee on finding a counterexample due to the incompleteness of simulations and the quality of the heuristic. Our work distinguishes from them by enhancing the falsification process with symbolic reachability analysis, which provides a guarantee on the finding of an abstract counterexample.

RRTs. Rapidly-growing random tree (RRT) is a technique widely used in robotics and motion planning [5, 24, 25, 32]. These methods grow a tree-like structure to explore the continuous space until a target state is reached. RRTs have variations depending on the direction of the search (bi-directional RRTs [4, 22]) and whether the search is heuristic-informed [19]. However, the steering of the growth of RRTs in the hybrid scenario remains challenging as the classical directional information becomes less useful because of the presence of discrete transitions.

Task planning. Task planning targets at finding a feasible *plan*, which is a sequence of actions, brings the system from the *initial state* to a state which satisfies a set of *goal constraints*. Falsification translates to task planning by viewing the constraints defining the bad set as the goals. In recent years, the planning community has made progress towards hybrid domains with both continuous and discrete evolutions [17]. The cutting-edge hybrid domain planners [13, 27, 28, 30] usually employ time discretization. Among them, UPMurphi [27] and DiNo [28] also adopt a refinement loop by detecting spurious system behaviors in a validation routine. Upon identifying a spurious behavior, they re-run the planner with a smaller granularity discretization, in which case their search space easily blows up. We avoid this problem by taking advantage of gradient information and non-linear optimization routines. It would also be interesting to investigate the feasibility of our idea in the planning domain considering recent works on the translation between planning domains and hybrid system models [11, 12].

6 CONCLUSIONS

We propose a falsification technique for hybrid systems, combining symbolic reachability analysis and non-linear trajectory optimization using the splicing approach. In this way, we present the first algorithm for a fully automated falsification process that scales to large problem sizes. Our method is implemented in the XSpeed software package [29] and will be made publicly available. The experimental evaluation shows that our approach can produce concrete counterexamples in an efficient and robust manner. We remark that the principle of adding reachability constraints is also applicable to nonlinear systems which we leave for future work.

ACKNOWLEDGMENTS

This work was partially supported by DST-SERB, GoI under Project No. YSS/2014/000623, by the Air Force Office of Scientific Research under award number FA2386-17-1-4065 and by the ARC project DP140104219 (Robust AI Planning for Hybrid Systems). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force.

REFERENCES

[1] Matthias Althoff, Olaf Stursberg, and Martin Buss. 2010. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear analysis: hybrid systems* 4, 2 (2010), 233–249.
[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. 1995. The algorithmic analysis of hybrid systems. *Theoretical Computer Science* 138, 1 (1995), 3–34.
[3] Yashwanth Singh Rahul Annapureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. 2011. S-TaLiRo: A Tool for Temporal Logic Falsification for

Table 2: Result for the falsification setting. RA Time: the accumulative time for the reachability analysis for FC and WoFC. Opt. Time: the accumulative time for the optimization calls for FC and WoFC. Fal. Time: the falsification time, which is the sum of reachability analysis time and optimization time. #Att.: the number of abstract counterexamples explored to find the first concrete counterexample; The results of the benchmarks marked (*) are obtained by using a time-step of 0.1 during flowpipe computation.

			FC Time (secs)			WoFC Time (secs)			#Att.	
Benchmark	Dim.	#Locs	RA	Opt.	Fal.	RA	Opt.	Fal.	FC	WoFC
Nav (1)	5	9	10.76	43.79	54.55	7.31	0.74	**8.05**	2	1
Nav (2)	5	25	25.67	67.00	**92.67**	-	-	-	2	-
Nav (3)	5	25	4488.46	32163.70	**36652.20**	-	-	-	231	-
Nav (4)*	5	225	5.67	247.53	253.20	5.56	203.48	**209.04**	5	5
Nav (5)*	5	225	217.17	5262.36	5479.53	60.39	1083.67	**1144.06**	45	11
Nav (6)*	5	400	7.91	205.13	**213.04**	24.55	1763.22	1787.77	3	15
Nav (7)*	5	400	64.20	1495.34	**1559.54**	-	-	-	12	-
Nav (8)	5	625	7.82	351.14	**358.96**	9.24	596.68	605.92	5	7
Nav (9)	5	625	91.77	1636.79	1728.56	94.53	1377.20	**1471.73**	11	9
Nav (10)	5	625	12.01	29.35	**41.36**	-	-	-	1	-
Nav (11)	5	625	16.41	127.32	**143.73**	-	-	-	1	-
Nav (12)	5	625	22.84	374.72	**397.56**	-	-	-	1	-
Platoon (1)	11	625	0.90	0.68	**1.58**	0.88	0.72	1.60	1	1
Platoon (2)	11	625	2.01	718.16	**720.17**	-	-	-	2	-
Oscillator (1)	3	4	2.74	0.95	**3.69**	-	-	-	1	-
Oscillator (2)	3	4	3.50	1.29	**4.79**	-	-	-	1	-
Oscillator (3)	3	4	2.63	1.00	**3.63**	-	-	-	1	-
Filtered Osc. (1)	11	4	12.77	1823.68	**1836.45**	-	-	-	4	-
Filtered Osc. (2)	11	4	12.89	673.77	**686.66**	-	-	-	2	-

Hybrid Systems. In *Tools and algorithms for the construction and analysis of systems (LNCS)*, Vol. 6605. Springer, 254–257. https://sites.google.com/a/asu.edu/s-taliro/.

[4] C. Wouter Bac, Tim Roorda, Roi Reshef, Sigal Berman, Jochen Hemming, and Eldert J. van Henten. 2016. Analysis of a motion planning problem for sweet-pepper harvesting in a dense obstacle environment. *Biosystems Engineering* 146 (2016), 85 – 97. Special Issue: Advances in Robotic Agriculture for Crops.

[5] Stanley Bak, Sergiy Bogomolov, Thomas A. Henzinger, and Aviral Kumar. 2017. Challenges and Tool Implementation of Hybrid Rapidly-Exploring Random Trees. In *International Workshop on Numerical Software Verification*. Springer.

[6] Stanley Bak and Parasara Sridhar Duggirala. 2017. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*. ACM, 173–178.

[7] I. Ben Makhlouf and S. Kowalewski. 2015. Networked Cooperative Platoon of Vehicles for Testing Methods and Verification Tools. In *Proc. of ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*. 37–42.

[8] Dimitri P Bertsekas. 1999. *Nonlinear programming*. Athena scientific Belmont.

[9] H.G. Bock and K.J. Plitt. 1984. A Multiple Shooting Algorithm for Direct Solution of Optimal Control Problems*. *IFAC Proceedings Volumes* 17, 2 (1984), 1603 – 1608. 9th IFAC World Congress: A Bridge Between Control Science and Technology, Budapest, Hungary, 2-6 July 1984.

[10] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Frédéric Viry, Andreas Podelski, and Christian Schilling. 2018. Reach set approximation through decomposition with low-dimensional sets and high-dimensional matrices. In *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*. ACM, 41–50.

[11] Sergiy Bogomolov, Daniele Magazzeni, Stefano Minopoli, and Martin Wehrle. 2015. PDDL+ Planning with Hybrid Automata: Foundations of Translating Must Behavior.. In *ICAPS*. 42–46.

[12] Sergiy Bogomolov, Daniele Magazzeni, Andreas Podelski, and Martin Wehrle. 2014. Planning as Model Checking in Hybrid Domains.. In *AAAI*. 2228–2234.

[13] Amanda Jane Coles, Andrew I Coles, Maria Fox, and Derek Long. 2012. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research* 44 (2012), 1–96.

[14] Adel Dokhanchi, Shakiba Yaghoubi, Bardh Hoxha, Georgios Fainekos, Gidon Ernst, Zhenya Zhang, Paolo Arcaini, Ichiro Hasuo, and Sean Sedwards. 2018. ARCH-COMP18 Category Report: Results on the Falsification Benchmarks. In *ARCH18. 5th International Workshop on Applied Verification of Continuous and Hybrid Systems (EPiC Series in Computing)*, Goran Frehse and Matthias Althoff (Eds.), Vol. 54. EasyChair, 104–109. FalStar : https://github.com/ERATOMMSD/.

[15] Alexandre Donzé. 2010. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification*. Springer, 167–170. Breach : https://github.com/decyphir/breach.

[16] Ansgar Fehnker and Franjo Ivancic. 2004. Benchmarks for Hybrid Systems Verification. In *HSCC (Lecture Notes in Computer Science)*, Rajeev Alur and George J. Pappas (Eds.), Vol. 2993. Springer, 326–341.

[17] Maria Fox and Derek Long. 2002. PDDL+: Modeling continuous time dependent effects. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*, Vol. 4. 34.

[18] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV) (LNCS)*, Shaz Qadeer Ganesh Gopalakrishnan (Ed.). Springer. http://spaceex.imag.fr.

[19] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. 2014. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2997–3004.

[20] Colas Le Guernic and Antoine Girard. 2009. Reachability Analysis of Hybrid Systems Using Support Functions. In *CAV (Lecture Notes in Computer Science)*, Ahmed Bouajjani and Oded Maler (Eds.), Vol. 5643. Springer, 540–554.

[21] Steven G. Johnson. 2018. The NLopt nonlinear-optimization package. http://ab-initio.mit.edu/nlopt.

[22] James J. Kuffner and Steven M. LaValle. 2000. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *ICRA*.

[23] Jan Kuřátko and Stefan Ratschan. 2014. Combined global and local search for the falsification of hybrid systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 146–160.

[24] Steven M LaValle. 1998. Rapidly-exploring random trees: A new tool for path planning. (1998).

[25] Steven M LaValle and James J Kuffner Jr. 2001. Randomized kinodynamic planning. *The international journal of robotics research* 20, 5 (2001), 378–400.

[26] Colas Le Guernic and Antoine Girard. 2010. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems* 4, 2 (2010), 250–262.
[27] Giuseppe Della Penna, Daniele Magazzeni, Fabio Mercorio, and Benedetto Intrigila. 2009. UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *ICAPS*.
[28] Wiktor Mateusz Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio. 2016. Heuristic Planning for PDDL+ Domains.
[29] Rajarshi Ray, Amit Gurung, Binayak Das, Ezio Bartocci, Sergiy Bogomolov, and Radu Grosu. 2015. XSpeed: Accelerating Reachability Analysis on Multi-core Processors. In *Hardware and Software: Verification and Testing - 11th International Haifa Verification Conference, HVC 2015, Proceedings (Lecture Notes in Computer Science)*, Nir Piterman (Ed.), Vol. 9434. Springer, 3–18. http://xspeed.nitmeghalaya.in.
[30] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramirez. 2016. Interval-Based Relaxation for General Numeric Planning.. In *ECAI*. 655–663.
[31] Krister Svanberg. 1987. The method of moving asymptotes – a new method for structural optimization. *International journal for numerical methods in engineering* 24, 2 (1987), 359–373.
[32] Gu Ye and Ron Alterovitz. 2017. Guided motion planning. In *Robotics research*. Springer, 291–307.
[33] Aditya Zutshi, Jyotirmoy V Deshmukh, Sriram Sankaranarayanan, and James Kapinski. 2014. Multiple shooting, cegar-based falsification for hybrid systems. In *Proceedings of the 14th International Conference on Embedded Software*. ACM, 5.
[34] Aditya Zutshi, Sriram Sankaranarayanan, Jyotirmoy V. Deshmukh, and James Kapinski. 2013. A trajectory splicing approach to concretizing counterexamples for hybrid systems. In *Proceedings of the 52nd IEEE Conference on Decision and Control, CDC 2013, December 10-13, 2013*. 3918–3925.

A APPENDIX

A.1 Derivatives of cost function terms

We provide the gradient of the terms in Eq. (20) with respect to x_i first. We denote $z_i = \mathcal{M}(\delta_i)(y_i)$.

(1) distance between end-points:

$$\begin{aligned}\frac{\partial\left(\sum_{j=0}^{N-1}\text{dist}(z_j, x_{j+1})\right)}{\partial x_i} &= \left[\frac{\partial\left(\text{dist}(z_{i-1}, x_i)\right)}{\partial x_i}\right]_{i>0} + \frac{\partial(\text{dist}\,(z_i, x_{i+1}))}{\partial x_i} \\ &= \left[\frac{\partial(x_i - z_{i-1})^2}{\partial x_i}\right]_{i>0} + \frac{\partial(x_{i+1} - z_i)^2}{\partial x_i} \\ &= 2\big([(x_i - z_{i-1})]_{i>0} \\ &\quad - \mathcal{M}(\delta_i)(e^{A\tau_i})(x_{i+1} - z_i)\big) \end{aligned} \tag{27}$$

(2) point-polytope distance (Eq.(19)):

$$\frac{\partial \text{dp}(p, \mathcal{P})}{\partial p} = \sum_{i=1}^{m} W_i \max\left(0, (W_i \cdot p) - b_i\right) \tag{28}$$

$$\frac{\partial y_i}{\partial x_i} = \frac{\partial(e^{A\tau_i}x_i + \Phi(A, \tau_i)u)}{\partial x_i} = e^{A\tau_i} \tag{29}$$

One can easily compute $\frac{\partial \text{dp}(y_i, C_i)}{\partial x_i}$ and $\frac{\partial \text{dp}(y_i, \mathcal{G}(\delta_i))}{\partial x_i}$ using Eq. (28), (29) and the chain rule.

We now provide the gradient of the terms in Eq. (20) with respect to τ_i.

(1) distance between end-points:

$$\begin{aligned}\frac{\partial\left(\sum_{j=0}^{N-1}\text{dist}(z_j, x_{j+1})\right)}{\partial \tau_i} &= \frac{\partial \text{dist}(z_i, x_{i+1})}{\partial \tau_i} \\ &= 2(z_i - x_{i+1}) \cdot \frac{\partial z_i}{\partial \tau_i} \end{aligned} \tag{30}$$

Now we compute $\frac{\partial z_i}{\partial \tau_i}$. For brevity, we use M, b to denote the transformation matrix and bias vector of $\mathcal{M}(\delta_i)$.

$$\begin{aligned}\frac{\partial z_i}{\partial \tau_i} &= \frac{\partial \mathcal{M}(\delta_i)(y_i)}{\partial \tau_i} \\ &= \frac{\partial(My_i + b)}{\partial y_i} \cdot \frac{\partial y_i}{\partial \tau_i} \\ &= M \cdot \frac{\partial y_i}{\partial \tau_i} \end{aligned} \tag{31}$$

$$\begin{aligned}\frac{\partial y_i}{\partial \tau_i} &= \frac{\partial(e^{A\tau_i}x_i + \Phi(A, \tau_i)u)}{\partial \tau_i} \\ &= Ae^{A\tau_i}x_i + e^{A\tau_i}u \end{aligned} \tag{32}$$

(2) point-polytope distance (Eq. 19):
One can easily compute $\frac{\partial \text{dp}(y_i, C_i)}{\partial \tau_i}$ and $\frac{\partial \text{dp}(y_i, \mathcal{G}(\delta_i))}{\partial \tau_i}$ using Eq. (28), (32) and the chain rule.