

DRYVR: Data-driven verification and compositional reasoning for automotive systems*

Chuchu Fan, Bolun Qi, Sayan Mitra, and Mahesh Viswanathan

University of Illinois at Urbana-Champaign

{cfan10,bolunqi2,mitras,vmahesh}@illinois.edu

Abstract

We present the DRYVR framework for verifying hybrid control systems that are described by a combination of a black-box simulator for trajectories and a white-box transition graph specifying mode switches. The framework includes (a) a probabilistic algorithm for learning sensitivity of the continuous trajectories from simulation data, (b) a bounded reachability analysis algorithm that uses the learned sensitivity, and (c) reasoning techniques based on simulation relations and sequential composition, that enable verification of complex systems under long switching sequences, from the reachability analysis of a simpler system under shorter sequences. We demonstrate the utility of the framework by verifying a suite of automotive benchmarks that include powertrain control, automatic transmission, and several autonomous and ADAS features like automatic emergency braking, lane-merge, and auto-passing controllers.

1 Introduction

The starting point of existing hybrid system verification approaches is the availability of nice mathematical models describing the transitions and trajectories. This central conceit severely restricts the applicability of the resulting approaches. Real world control system “models” are typically a heterogeneous mix of simulation code, differential equations, block diagrams, and hand-crafted look-up tables. Extracting clean mathematical models from these descriptions is usually infeasible. At the same time, rapid developments in Advanced Driving Assist Systems (ADAS), autonomous vehicles, robotics, and drones now make the need for effective and sound verification algorithms stronger than ever before. The DRYVR framework presented in this paper aims to narrow the gap between sound and practical verification for control systems.

[SM: [\[Another seemingly related paper:http://dl.acm.org/citation.cfm?id=2382263\]](http://dl.acm.org/citation.cfm?id=2382263)]

*Appearing in Computer Aided Verification (CAV) 2017.

Model assumptions Consider an ADAS feature like automatic emergency braking system (AEB). The high-level logic deciding the timing of when and for how long the brakes are engaged after an obstacle is detected by sensors is implemented in a relatively clean piece of code and this logical module can be seen as a *white-box*. In contrast, the dynamics of vehicle itself, with hundreds of parameters, is more naturally viewed as a *black-box*. That is, it can be simulated or tested with different initial conditions and inputs, but it is nearly impossible to write down a nice mathematical model.

The empirical observation motivating this work is that many control systems, and especially automotive systems, share this combination of white and black boxes (see other examples in Sections 2.1, 2.5, and A.2). In this paper, we view hybrid systems as a combination of a white-box that specifies the mode switches and a black-box that can simulate the continuous evolution in each mode. Suppose the system has a set of modes \mathcal{L} and n continuous variables. The mode switches are defined by a *transition graph* G which is a directed acyclic graph (DAG) whose vertices and edges define the allowed mode switches and the switching times. The black-box is a set of trajectories \mathcal{TL} in \mathbb{R}^n for each mode in \mathcal{L} . We do not have a closed form description of \mathcal{TL} , but instead, we have a *simulator*, that can generate sampled data points on individual trajectories for a given initial state and mode. Combining a transition graph G , a set of trajectories \mathcal{TL} , and a set of initial states in \mathbb{R}^n , we obtain a hybrid system for which executions, reachability, and trace containment can be defined naturally.

We have studied a suite of automotive systems such as powertrain control [40], automatic transmission control [46], and ADAS features like automatic emergency braking (AEB), lane-change, and auto-passing, that are naturally represented in the above style. In verifying a lane change or merge controller, once the maneuver is activated, the mode transitions occur within certain time intervals. In testing a powertrain control system, the mode transitions are brought about by the driver and it is standard to describe typical driver classes using time-triggered signals. Similar observations hold in other examples.

Safety verification algorithm With black-box modules in our hybrid systems, we address the challenge of providing guaranteed verification. Our approach is based on the idea of simulation-driven reachability analysis [30, 22, 23]. For a given mode $\ell \in \mathcal{L}$, finitely many simulations of the trajectories of ℓ and a *discrepancy function* bounding the sensitivity of these trajectories, is used to over-approximate the reachable states. For the key step of computing discrepancy for modes that are now represented by black-boxes, we introduce a probabilistic algorithm that learns the parameters of exponential discrepancy functions from simulation data. The algorithm transforms the problem of learning the parameters of the discrepancy function to the problem of learning a linear separator for a set of points in \mathbb{R}^2 that are obtained from transforming the simulation data. A classical result in PAC learning, ensures that any such discrepancy function works with high probability for all trajectories. We performed dozens of experiments with a variety of black-box simulators and observed that 15-20 simulation traces typically give a discrepancy function that works for nearly 100% of all simulations. The reachability algorithm for the hybrid system proceeds along the vertices of the transition graph in a topologically sorted order and this gives a sound bounded time verification algorithm,

provided the learned discrepancy function is correct.

Reasoning White-box transition graphs in our modelling, identify the switching sequences under which the black-box modules are exercised. Complex systems have involved transition graphs that describe subtle sequences in which the black-box modules are executed. To enable the analysis of such systems, we identify reasoning principles that establish the safety of system under a complex transition graph based on its safety under a simpler transition graph. We define a notion of forward simulation between transition graphs that provides a sufficient condition of when one transition graph “subsumes” another — if G_1 is simulated by G_2 then the reachable states of a hybrid system under G_1 are contained in the reachable states of the system under G_2 . Thus the safety of the system under G_2 implies the safety under G_1 . Moreover, we give a simple polynomial time algorithm that can check if one transition graph is simulated by another.

Our transition graphs are acyclic with transitions having bounded switching times. Therefore, the executions of the systems we analyze are over a bounded time, and have a bounded number of mode switches. An important question to investigate is whether establishing the safety for bounded time, enables one can conclude the safety of the system for an arbitrarily long time and for arbitrarily many mode switches. With this in mind, we define a notion of sequential composition of transition graphs G_1 and G_2 , such that switching sequences allowed by the composed graph are the concatenation of the sequences allowed by G_1 with those allowed by G_2 . Then we prove a sufficient condition on a transition graph G such that safety of a system under G implies the safety of the system under arbitrarily many compositions of G with itself.

Automotive applications We have implemented these ideas to create the **Data-driven System for Verification and Reasoning (DRYVR)**. The tool is able to automatically verify or find counter-examples in a few minutes, for all the benchmark scenarios mentioned above. Reachability analysis combined with compositional reasoning, enabled us to infer safety of systems with respect to arbitrary transitions and duration.

Related work Most automated verification tools for hybrid systems rely on analyzing a white-box mathematical model of the systems. They include tools based decidability results [13, 37, 10, 3, 24, 32], semi-decision procedures that over-approximate the reachable set of states through symbolic computation [36, 48, 7, 45, 55, 33, 4, 9], using abstractions [1, 12, 11, 19, 54, 52, 16, 39, 51, 5, 6, 49, 53], and using approximate decision procedures for fragments of first-order logic [44]. More recently, there has been interest in developing simulation-based verification tools [41, 18, 17, 42, 2, 25, 15, 23]. Eventhough these are simulation based tools, they often rely on being to analyze a mathematical model of the system. The type of analysis that they rely on include instrumentation to extract a symbolic trace from a simulation [42], stochastic optimization to search for counter-examples [2, 25], and sensitivity analysis [18, 17, 15, 23]. Some of the simulation based techniques only work for systems with linear dynamics [34, 35].

2 Modeling/semantic framework

We introduce a powertrain control system from [40] as a running example to illustrate the elements of our hybrid system modeling framework.

2.1 Powertrain control system

This system (`Powertrn`) models a highly nonlinear engine control system. The relevant state variables of the model are intake manifold pressure (p), air-fuel ratio (λ), estimated manifold pressure (pe) and integrator state (i), and the overall system can be in one of four modes `startup`, `normal`, `powerup`, `sensorfail`. A Simulink[®] diagram describes the continuous evolution of the above variables. In this paper, we mainly work on the *Hybrid I/O Automaton Model* in the suite of powertrain control models. The Simulink[®] model consists of continuous variables describing the dynamics of the powertrain plant and sample-and-hold variables as the controller. One of the key requirements to verify is that the engine maintains the air-fuel ratio within a desired range in different modes for a given set of driver behaviors, and this requirement has implications on fuel economy and emissions. For testing purposes, the control system designers work with sets of driver profiles that essentially define families of switching signals across the different modes. Previous verification results on this problem have been reported in [21, 27] on a simplified version of the powertrain control model.

2.2 Transition graphs

We will use \mathcal{L} to denote a finite set of *modes* or locations of the system under consideration. The discrete behavior or mode transitions are specified by what we call a transition graph over \mathcal{L} .

Definition 2.1. A *transition graph* is a labeled, directed acyclic graph $G = \langle \mathcal{L}, \mathcal{V}, \mathcal{E}, vlab, elab \rangle$, where (a) \mathcal{L} is the set of vertex labels also called the set of *modes*, (b) \mathcal{V} the set of vertices, (c) $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, (d) $vlab : \mathcal{V} \rightarrow \mathcal{L}$ is a vertex labeling function that labels each vertex with a mode, and (e) $elab : \mathcal{E} \rightarrow \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}$ is an edge labeling function that labels each edge with a nonempty, closed, bounded interval defined by pair of non-negative reals.

Since G is a DAG, there is a nonempty subset $\mathcal{V}_{\text{init}} \subseteq \mathcal{V}$ of vertices with no incoming edges and a nonempty subset $\mathcal{V}_{\text{term}} \subseteq \mathcal{V}$ of vertices with no outgoing edges. We define the set of initial locations of G as $\mathcal{L}_{\text{init}} = \{\ell \mid \exists v \in \mathcal{V}_{\text{init}}, vlab(v) = \ell\}$. A (maximal) *path* of the graph G is a sequence $\pi = v_1, t_1, v_2, t_2, \dots, v_k$ such that, (a) $v_1 \in \mathcal{V}_{\text{init}}$, (b) $v_k \in \mathcal{V}_{\text{term}}$, and (c) for each (v_i, t_i, v_{i+1}) subsequence, there exists $(v_i, v_{i+1}) \in \mathcal{E}$, and $t_i \in elab((v_i, v_{i+1}))$. Paths_G is the set of all possible paths of G . For a given path $\pi = v_1, t_1, v_2, t_2, \dots, v_k$ its *trace*, denoted by $vlab(\pi)$, is the sequence $vlab(v_1), t_1, vlab(v_2), t_2, \dots, vlab(v_k)$. Since G is a DAG, a trace of G can visit the same mode finitely many times. Trace_G is the set of all traces of G .

An example transition graph for the Powertrain system of Section 2.1 is shown in Figure 1. The set of vertices $\mathcal{V} = \{0, \dots, 4\}$ and the $vlab$'s and $elab$'s appear adjacent to the vertices and edges.

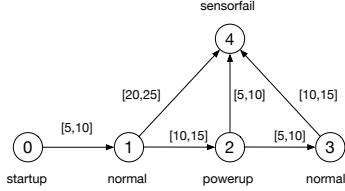


Figure 1: A sample transition graph for Powertrain system.

2.2.1 Trace containment

We will develop reasoning techniques based on reachability, abstraction, composition, and substitutivity. To this end, we will need to establish containment relations between the behaviors of systems. Here we define containment of transition graph traces. Consider transition graphs G_1, G_2 , with modes $\mathcal{L}_1, \mathcal{L}_2$, and a mode map $lmap : \mathcal{L}_1 \rightarrow \mathcal{L}_2$. For a trace $\sigma = \ell_1, t_1, \ell_2, t_2, \dots, \ell_k \in \text{Trace}_{G_1}$, simplifying notation, we denote by $lmap(\sigma)$ the sequence $lmap(\ell_1), t_1, lmap(\ell_2), t_2, \dots, lmap(\ell_k)$. We write $G_1 \preceq_{lmap} G_2$ iff for every trace $\sigma \in \text{Trace}_{G_1}$, there is a trace $\sigma' \in \text{Trace}_{G_2}$ such that $lmap(\sigma)$ is a prefix of σ' .

Definition 2.2. Given graphs G_1, G_2 and a mode map $lmap : \mathcal{L}_1 \rightarrow \mathcal{L}_2$, a relation $R \subseteq \mathcal{V}_1 \times \mathcal{V}_2$ is a *forward simulation relation from G_1 to G_2* iff

- (a) for each $v \in \mathcal{V}_{1\text{init}}$, there is $u \in \mathcal{V}_{2\text{init}}$ such that $(v, u) \in R$,
- (b) for every $(v, u) \in R$, $lmap(vlab_1(v)) = vlab_2(u)$, and
- (c) for every $(v, v') \in \mathcal{E}_1$ and $(v, u) \in R$, there exists a finite set u_1, \dots, u_k such that: (i) for each u_j , $(v, u_j) \in R$, and (ii) $elab_1((v, v')) \subseteq \cup_j elab_2((v, u_j))$.

Proposition 2.3. *If there exists a forward simulation relation from G_1 to G_2 with $lmap$ then $G_1 \preceq_{lmap} G_2$.*

2.2.2 Sequential composition of graphs

We will find it convenient to define the *sequential composition* of two transition graphs. Intuitively, the traces of the composition of G_1 and G_2 will be those that can be obtained by concatenating a trace of G_1 with a trace of G_2 . To keep the definitions and notations simple, we will assume (when taking sequential compositions) $|\mathcal{V}_{\text{init}}| = |\mathcal{V}_{\text{term}}| = 1$; this is true of the examples we analyze. It is easy to generalize to the case when this does not hold. Under this assumption, the unique vertex in $\mathcal{V}_{\text{init}}$ will be denoted as v_{init} and the unique vertex in $\mathcal{V}_{\text{term}}$ will be denoted as v_{term} .

Definition 2.4. Given graphs $G_1 = \langle \mathcal{L}, \mathcal{V}_1, \mathcal{E}_1, vlab_1, elab_1 \rangle$ and $G_2 = \langle \mathcal{L}, \mathcal{V}_2, \mathcal{E}_2, vlab_2, elab_2 \rangle$ such that $vlab_1(v_{1\text{term}}) = vlab_2(v_{2\text{term}})$, the *sequential composition* of G_1 and G_2 is the graph $G_1 \circ G_2 = \langle \mathcal{L}, \mathcal{V}, \mathcal{E}, vlab, elab \rangle$ where

- (a) $\mathcal{V} = (\mathcal{V}_1 \cup \mathcal{V}_2) \setminus \{v_{2\text{init}}\}$,
- (b) $\mathcal{E} = \mathcal{E}_1 \cup \{(v_{1\text{term}}, u) \mid (v_{2\text{init}}, u) \in \mathcal{E}_2\} \cup \{(v, u) \in \mathcal{E}_2 \mid v \neq v_{2\text{init}}\}$,
- (c) $vlab(v) = vlab_1(v)$ if $v \in \mathcal{V}_1$ and $vlab(v) = vlab_2(v)$ if $v \in \mathcal{V}_2$,
- (d) For edge $(v, u) \in \mathcal{E}$, $elab((v, u))$ equals (i) $elab_1((v, u))$, if $u \in \mathcal{V}_1$, (ii) $elab_2((v_{2\text{init}}, u))$, if $v = v_{1\text{term}}$, (iii) $elab_2((v, u))$, otherwise.

Given our definition of trace containment between graphs, we can prove a very simple property about sequential composition.

Proposition 2.5. *Let G_1 and G_2 be two graphs with modes \mathcal{L} that can be sequential composed. Then $G_1 \preceq_{\text{id}} G_1 \circ G_2$, where id is the identity map on \mathcal{L} .*

The proposition follows from the fact that every path of G_1 is a prefix of a path of $G_1 \circ G_2$. Later in Section 4.1 we see examples of sequential composition.

2.3 Trajectories

The evolution of the system’s continuous state variables is formally described by continuous functions of time called *trajectories*. Let n be the number of continuous variables in the underlying hybrid model. A *trajectory* for an n -dimensional system is a continuous function of the form $\tau : [0, T] \rightarrow \mathbb{R}^n$, where $T \geq 0$. The interval $[0, T]$ is called the *domain* of τ and is denoted by $\tau.\text{dom}$. The first state $\tau(0)$ is denoted by $\tau.\text{fstate}$, last state $\tau.\text{lstate} = \tau(T)$ and $\tau.\text{ltime} = T$. For a hybrid system with \mathcal{L} modes, each trajectory is labeled by a mode in \mathcal{L} . A *trajectory labeled by \mathcal{L}* is a pair $\langle \tau, \ell \rangle$ where τ is a trajectory and $\ell \in \mathcal{L}$.

A T_1 -*prefix* of $\langle \tau, \ell \rangle$, for any $T_1 \in \tau.\text{dom}$, is the labeled-trajectory $\langle \tau_1, \ell \rangle$ with $\tau_1 : [0, T_1] \rightarrow \mathbb{R}^n$, such that for all $t \in [0, T_1]$, $\tau_1(t) = \tau(t)$. A set of labeled-trajectories \mathcal{TL} is *prefix-closed* if for any $\langle \tau, \ell \rangle \in \mathcal{TL}$, any of its prefixes are also in \mathcal{TL} . A set \mathcal{TL} is *deterministic* if for any pair $\langle \tau_1, \ell_1 \rangle, \langle \tau_2, \ell_2 \rangle \in \mathcal{TL}$, if $\tau_1.\text{fstate} = \tau_2.\text{fstate}$ and $\ell_1 = \ell_2$ then one is a prefix of the other. A deterministic, prefix-closed set of labeled trajectories \mathcal{TL} describes the behavior of the continuous variables in modes \mathcal{L} . We denote by $\mathcal{TL}_{\text{init}, \ell} = \{\tau.\text{fstate} \mid \langle \tau, \ell \rangle \in \mathcal{TL}\}$, the set of initial states of trajectories in mode ℓ . Without loss generality we assume that $\mathcal{TL}_{\text{init}, \ell}$ is a connected compact subset of \mathbb{R}^n . We assume that trajectories are defined for unbounded time, that is, for each $\ell \in \mathcal{L}, T > 0$, and $x \in \mathcal{TL}_{\text{init}, \ell}$, there exists a $\langle \tau, \ell \rangle \in \mathcal{TL}$, with $\tau.\text{fstate} = x$ and $\tau.\text{ltime} = T$.

In control theory and hybrid systems literature, the trajectories are assumed to be generated from models like ordinary differential equations (ODEs) and differential algebraic equations (DAEs). This work avoids over-reliance on the models generating trajectories and closed-form expressions. Instead, DRYVR works with sampled data of $\tau(\cdot)$ generated from simulations or tests.

Definition 2.6. A *simulator* for a (deterministic and prefix-closed) set \mathcal{TL} of trajectories labeled by \mathcal{L} is a function (or a program) sim that takes as input a mode label $\ell \in \mathcal{L}$, an initial state $x_0 \in \mathcal{TL}_{\text{init}, \ell}$, and a finite sequence of time points t_1, \dots, t_k , and returns a sequence of states $\text{sim}(x_0, \ell, t_1), \dots, \text{sim}(x_0, \ell, t_k)$ such that there exists $\langle \tau, \ell \rangle \in \mathcal{TL}$ with $\tau.\text{fstate} = x_0$ and for each $i \in \{1, \dots, k\}$, $\text{sim}(x_0, \ell, t_i) = \tau(t_i)$.

The trajectories of the Powertrn system are described by a Simulink[®] diagram. The diagram has several switch blocks and input signals that can be set appropriately to generate simulation data using the Simulink[®] ODE solver.

For simplicity, we assume that the simulations are perfect (as in the last equality of Definition 2.6). Formal guarantees of soundness of DRYVR are not compromised if we use *validated simulations* instead.

Trajectory containment Consider sets of trajectories, \mathcal{TL}_1 labeled by \mathcal{L}_1 and \mathcal{TL}_2 labeled by \mathcal{L}_2 , and a mode map $lmap : \mathcal{L}_1 \rightarrow \mathcal{L}_2$. For a labeled trajectory $\langle \tau, \ell \rangle \in \mathcal{TL}_1$, denote by $lmap(\langle \tau, \ell \rangle)$ the labeled-trajectory $\langle \tau, lmap(\ell) \rangle$. Write $\mathcal{TL}_1 \preceq_{lmap} \mathcal{TL}_2$ iff for every labeled trajectory $\langle \tau, \ell \rangle \in \mathcal{TL}_1$, $lmap(\langle \tau, \ell \rangle) \in \mathcal{TL}_2$.

2.4 Hybrid systems

Definition 2.7. An n -dimensional *hybrid system* \mathcal{H} is a 4-tuple $\langle \mathcal{L}, \Theta, G, \mathcal{TL} \rangle$, where (a) \mathcal{L} is a finite set of modes, (b) $\Theta \subseteq \mathbb{R}^n$ is a compact set of initial states, (c) $G = \langle \mathcal{L}, \mathcal{V}, \mathcal{E}, elab \rangle$ is a transition graph with set of modes \mathcal{L} , and (d) \mathcal{TL} is a set of deterministic, prefix-closed trajectories labeled by \mathcal{L} .

A *state* of the hybrid system \mathcal{H} is a point in $\mathbb{R}^n \times \mathcal{L}$. The set of initial states is $\Theta \times \mathcal{L}_{init}$. Semantics of \mathcal{H} is given in terms of executions which are sequences of trajectories consistent with the modes defined by the transition graph. An *execution* of \mathcal{H} is a sequence of labeled trajectories $\alpha = \langle \tau_1, \ell_1 \rangle \dots, \langle \tau_{k-1}, \ell_{k-1} \rangle, \ell_k$ in \mathcal{TL} , such that (a) $\tau_1.fstate \in \Theta$ and $\ell_1 \in \mathcal{L}_{init}$, (b) the sequence $\text{path}(\alpha)$ defined as $\ell_1, \tau_1.ltime, \ell_2, \dots, \ell_k$ is in Trace_G , and (c) for each consecutive trajectory, $\tau_{i+1}.fstate = \tau_i.lstate$. The set of all executions of \mathcal{H} is denoted by $\text{Execs}_{\mathcal{H}}$. The first and last states of an execution $\alpha = \langle \tau_1, \ell_1 \rangle \dots, \langle \tau_{k-1}, \ell_{k-1} \rangle, \ell_k$ are $\alpha.fstate = \tau_1.fstate$, $\alpha.lstate = \tau_{k-1}.lstate$, and $\alpha.fmode = \ell_1$ $\alpha.lmode = \ell_k$. A state $\langle x, \ell \rangle$ is *reachable* at time t and vertex v (of graph G) if there exists an execution $\alpha = \langle \tau_1, \ell_1 \rangle \dots, \langle \tau_{k-1}, \ell_{k-1} \rangle, \ell_k \in \text{Execs}_{\mathcal{H}}$, a path $\pi = v_1, t_1, \dots, v_k$ in Paths_G , $i \in \{1, \dots, k\}$, and $t' \in \tau_i.dom$ such that $vlab(\pi) = \text{path}(\alpha)$, $v = v_i$, $\ell = \ell_i$, $x = \tau_i(t')$, and $t = t' + \sum_{j=1}^{i-1} t_j$. The set of reachable states, reach tube, and states reachable at a vertex v are defined as follows.

$$\text{ReachTube}_{\mathcal{H}} = \{ \langle x, \ell, t \rangle \mid \text{for some } v, \langle x, \ell \rangle \text{ is reachable at time } t \text{ and vertex } v \}$$

$$\text{Reach}_{\mathcal{H}} = \{ \langle x, \ell \rangle \mid \text{for some } v, t, \langle x, \ell \rangle \text{ is reachable at time } t \text{ and vertex } v \}$$

$$\text{Reach}_{\mathcal{H}}^v = \{ \langle x, \ell \rangle \mid \text{for some } t, \langle x, \ell \rangle \text{ is reachable at time } t \text{ and vertex } v \}$$

Given a set of (unsafe) states $\mathcal{U} \subseteq \mathbb{R}^n \times \mathcal{L}$, the *bounded safety verification problem* is to decide whether $\text{Reach}_{\mathcal{H}} \cap \mathcal{U} = \emptyset$. In Section 3 we will present DRYVR's algorithm for solving this decision problem.

Remark 2.8. *Defining paths in a graph G to be maximal (i.e., end in a vertex in \mathcal{V}_{term}) coupled with the definition above for executions in \mathcal{H} , ensures that for a vertex v with outgoing edges in G , the execution must leave the mode $vlab(v)$ within time bounded by the largest time in the labels of outgoing edges from v .*

An instance of the bounded safety verification problem is defined by (a) the hybrid system for the Powertrn which itself is defined by the transition graph of Figure 1 and the trajectories defined by the Simulink[®] model, and (b) the unsafe set (\mathcal{U}_p): in `powerup` mode, $t > 4 \wedge \lambda \notin [12.4, 12.6]$, in `normal` mode, $t > 4 \wedge \lambda \notin [14.6, 14.8]$.

Containment between graphs and trajectories can be leveraged to conclude the containment of the set of reachable states of two hybrid systems.

Proposition 2.9. Consider a pair of hybrid systems $\mathcal{H}_i = \langle \mathcal{L}_i, \Theta_i, G_i, \mathcal{TL}_i \rangle$, $i \in \{1, 2\}$ and mode map $lmap : \mathcal{L}_1 \rightarrow \mathcal{L}_2$. If $\Theta_1 \subseteq \Theta_2$, $G_1 \preceq_{lmap} G_2$, and $\mathcal{TL}_1 \preceq_{lmap} \mathcal{TL}_2$, then $\text{Reach}_{\mathcal{H}_1} \subseteq \text{Reach}_{\mathcal{H}_2}$.

2.5 ADAS and Autonomous Vehicle Benchmarks

This is a suite of benchmarks we have created representing various common scenarios used for testing ADAS and Autonomous driving control systems. The hybrid system for a scenario is constructed by putting together several individual vehicles. The higher-level decisions (paths) followed by the vehicles are captured by transition graphs while the detailed dynamics of each vehicle comes from a black-box Simulink[®] simulator from Mathworks[®] [47].

Each vehicle has several continuous variables including the x, y -coordinates of the vehicle on the road, its velocity, heading, and steering angle. The vehicle can be controlled by two input signals, namely the throttle (acceleration or brake) and the steering speed. By choosing appropriate values for these input signals, we have defined the following modes for each vehicle — **cruise**: move forward at constant speed, **speedup**: constant acceleration, **brake**: constant (slow) deceleration, **em_brake**: constant (hard) deceleration. In addition, we have designed lane switching modes **ch_left** and **ch_right** in which the acceleration and steering are controlled in such a manner that the vehicle switches to its left (resp. right) lane in a certain amount of time.

For each vehicle, we mainly analyze four variables: absolute position (sx) and velocity (vx) orthogonal to the road direction (x -axis), and absolute position (sy) and velocity (vy) along the road direction (y -axis). The throttle and steering is captured using the four variables. We will use subscripts to distinguish between different vehicles. The following scenarios are constructed by defining appropriate sets of initial states and transitions graphs labeled by the modes of two or more vehicles. In all of these scenarios a primary safety requirement is that the vehicles maintain safe separation. See Appendix A.1 for more details on initial states and transition graphs of each scenario.

Merge: Vehicle A in the left lane is behind vehicle B in the right lane. A switches through modes **cruise**, **speedup**, **ch_right**, and **cruise** over specified intervals to merge behind B. Variants of this scenario involves B also switching to **speedup** or **brake**.

AutoPassing: Vehicle A starts behind B in the same lane, and goes through a sequence of modes to overtake B. If B switches to **speedup** before A enters **speedup** then A aborts and changes back to right lane.

Merge3: Same as AutoPassing with a third car C always ahead of B.

AEB: Vehicle A cruises behind B and B stops. A transits from **cruise** to **em_brake** possibly over several different time intervals as governed by different sensors and reaction times.

3 Invariant verification

A subproblem for invariant verification is to compute $\text{ReachTube}_{\mathcal{H}}$, or more specifically, the reachtubes for the set of trajectories \mathcal{TL} in a given mode, up to a

time bound. This is a difficult problem, even when \mathcal{TL} is generated by white-box models. The algorithms in [17, 22, 29] approximate reachtubes using simulations and sensitivity analysis of ODE models generating \mathcal{TL} . Here, we begin with a probabilistic method for estimating sensitivity from black-box simulators.

3.1 Discrepancy functions

Sensitivity of trajectories is formalized by the notion of discrepancy functions [22]. For a set \mathcal{TL} , a *discrepancy function* is a uniformly continuous function $\beta : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$, such that for any pair of identically labeled trajectories $\langle \tau_1, \ell \rangle, \langle \tau_2, \ell \rangle \in \mathcal{TL}$, and any $t \in \tau_1.\text{dom} \cap \tau_2.\text{dom}$: (a) β upper-bounds the distance between the trajectories, i.e.,

$$|\tau_1(t) - \tau_2(t)| \leq \beta(\tau_1.\text{fstate}, \tau_2.\text{fstate}, t), \quad (1)$$

and (b) β converges to 0 as the initial states converge, i.e., for any trajectory τ and $t \in \tau.\text{dom}$, if a sequence of trajectories $\tau_1, \dots, \tau_k, \dots$ has $\tau_k.\text{fstate} \rightarrow \tau.\text{fstate}$, then $\beta(\tau_k.\text{fstate}, \tau.\text{fstate}, t) \rightarrow 0$. In [22] it is shown how given a β , condition (a) can be used to over-approximate reachtubes from simulations, and condition (b) can be used to make these approximations arbitrarily precise. Techniques for computing β from ODE models are developed in [29, 28, 38], but these are not applicable here in absence of such models. Instead we present a simple method for discovering discrepancy functions that only uses simulations. Our method is based on classical results on PAC learning linear separators [43]. We recall these before applying them to find discrepancy functions.

3.1.1 Learning linear separators.

For $\Gamma \subseteq \mathbb{R} \times \mathbb{R}$, a *linear separator* is a pair $(a, b) \in \mathbb{R}^2$ such that

$$\forall (x, y) \in \Gamma. x \leq ay + b. \quad (2)$$

Let us fix a subset Γ that has a (unknown) linear separator (a_*, b_*) . Our goal is to discover some (a, b) that is a linear separator for Γ by sampling points in Γ ¹. The assumption is that elements of Γ can be drawn according to some (unknown) distribution \mathcal{D} . With respect to \mathcal{D} , the *error* of a pair (a, b) from satisfying Equation 2, is defined to be $\text{err}_{\mathcal{D}}(a, b) = \mathcal{D}(\{(x, y) \in \Gamma \mid x > ay + b\})$ where $\mathcal{D}(X)$ is the measure of set X under distribution \mathcal{D} . Thus, the error is the measure of points (w.r.t. \mathcal{D}) that (a, b) is not a linear separator for. There is a very simple (probabilistic) algorithm that finds a pair (a, b) that is a linear separator for a large fraction of points in Γ , as follows.

1. Draw k pairs $(x_1, y_1), \dots, (x_k, y_k)$ from Γ according to \mathcal{D} ; the value of k will be fixed later.
2. Find $(a, b) \in \mathbb{R}^2$ such that $x_i \leq ay_i + b$ for all $i \in \{1, \dots, k\}$.

Step 2 involves checking feasibility of a linear program, and so can be done quickly. This algorithm, with high probability, finds a linear separator for a large fraction of points.

¹We prefer to present the learning question in this form as opposed to one where we learn a Boolean concept because it is closer to the task at hand.

Proposition 3.1. *Let $\epsilon, \delta \in \mathbb{R}_+$. If $k \geq \frac{1}{\epsilon} \ln \frac{1}{\delta}$ then, with probability $\geq 1 - \delta$, the above algorithm finds (a, b) such that $\text{err}_{\mathcal{D}}(a, b) < \epsilon$.*

The proof follows from the PAC-learnability of concepts with low VC-dimension [43]. The detailed proof is provided in the Appendix A.3.

3.1.2 Learning discrepancy functions

Discrepancy functions will be computed from simulation data independently for each mode. Let us fix a mode $\ell \in \mathcal{L}$, and a domain $[0, T]$ for each trajectory. The discrepancy functions that we will learn from simulation data, will be of two different forms, and we discuss how these are obtained.

Global exponential discrepancy (GED) is a function of the form

$$\beta(x_1, x_2, t) = |x_1 - x_2| K e^{\gamma t}.$$

Here K and γ are constants. Thus, for any pair of trajectories τ_1 and τ_2 (for mode ℓ), we have

$$\forall t \in [0, T]. |\tau_1(t) - \tau_2(t)| \leq |\tau_1.\text{fstate} - \tau_2.\text{fstate}| K e^{\gamma t}.$$

Taking logs on both sides and rearranging terms, we have

$$\forall t. \ln \frac{|\tau_1(t) - \tau_2(t)|}{|\tau_1.\text{fstate} - \tau_2.\text{fstate}|} \leq \gamma t + \ln K.$$

It is easy to see that a global exponential discrepancy is nothing but a linear separator for the set Γ consisting of pairs $(\ln \frac{|\tau_1(t) - \tau_2(t)|}{|\tau_1.\text{fstate} - \tau_2.\text{fstate}|}, t)$ for all pairs of trajectories τ_1, τ_2 and time t . Using the sampling based algorithm described before, we could construct a GED for a mode $\ell \in \mathcal{L}$, where sampling from Γ reduces to using the simulator to generate traces from different states in $\mathcal{T}_{\text{init}, \ell}$. Proposition 3.1 guarantees the correctness, with high probability, for any separator discovered by the algorithm. However, for our reachability algorithm to not be too conservative, we need K and γ to be small. Thus, when solving the linear program in Step 2 of the algorithm, we search for a solution minimizing $\gamma T + \ln K$.

Piece-wise exponential discrepancy (PED). The second form of discrepancy functions we consider, depend on dividing up the time domain $[0, T]$ into smaller intervals, and finding a global exponential discrepancy for each interval. Let $0 = t_0, t_1, \dots, t_N = T$ be an increasing sequence of time points. Let $K, \gamma_1, \gamma_2, \dots, \gamma_N$ be such that for every pair of trajectories τ_1, τ_2 (of mode ℓ), for every $i \in \{1, \dots, N\}$, and $t \in [t_{i-1}, t_i]$, $|\tau_1(t) - \tau_2(t)| \leq |\tau_1(t_{i-1}) - \tau_2(t_{i-1})| K e^{\gamma_i t}$. Under such circumstances, the discrepancy function itself can be seen to be given as

$$\beta(x_1, x_2, t) = |x_1 - x_2| K e^{\sum_{j=1}^{i-1} \gamma_j (t_j - t_{j-1}) + \gamma_i (t - t_{i-1})} \quad \text{for } t \in [t_{i-1}, t_i].$$

If the time points $0 = t_0, t_1, \dots, t_N = T$ are fixed, then the constants $K, \gamma_1, \gamma_2, \dots, \gamma_N$ can be discovered using the learning approach described for GED; here, to

discover γ_i , we take Γ_i to be the pairs obtained by restricting the trajectories to be between times t_{i-1} and t_i . The sequence of time points t_i are also dynamically constructed by our algorithm based on the following approach. Our experience suggests that a value for γ that is ≥ 2 results in very conservative reach tube computation. Therefore, the time points t_i are constructed inductively to be as large as possible, while ensuring that $\gamma_i < 2$.

3.1.3 Experiments on Learning Discrepancy

We used the above algorithm to learn discrepancy functions for dozens of modes with complex, nonlinear trajectories. Our experiments suggest that around 10-20 simulation traces are adequate for computing both global and piece-wise discrepancy functions. For each mode we use a set S_{train} of simulation traces that start from independently drawn random initial states in $\mathcal{TL}_{\text{init},\ell}$ to learn a discrepancy function. Each trace may have 100-10000 time points, depending on the relevant time horizon and sample times. Then we draw another set S_{test} of 1000 simulations traces for validating the computed discrepancy. For every pair of trace in S_{test} and for every time point, we check whether the computed discrepancy satisfies Equation 1. We observe that for $|S_{\text{train}}| > 10$ the computed discrepancy function is correct for 96% of the points S_{test} in and for $|S_{\text{train}}| > 20$ it is correct for more than 99.9%, across all experiments.

3.2 Verification algorithm

In this section, we present algorithms to solve the bounded verification problem for hybrid systems using learned exponential discrepancy functions. We first introduce an algorithm *GraphReach* (Algorithm 1) which takes as input a hybrid system $\mathcal{H} = \langle \mathcal{L}, \Theta, G, \mathcal{TL} \rangle$ and returns a set of reachtubes—one for each vertex of G —such that their union over-approximates $\text{ReachTube}_{\mathcal{H}}$.

GraphReach maintains two data-structures: (a) RS accumulates pairs of the form $\langle RT, v \rangle$, where $v \in \mathcal{V}$ and RT is its corresponding reachtube; (b) $VerInit$ accumulates pairs of the form $\langle S, v \rangle$, where $v \in \mathcal{V}$ and $S \subset \mathbb{R}^n$ is the set of states from which the reachtube in v is to be computed. Each v could be in multiple such pairs in RS and $VerInit$. Initially, $RS = \emptyset$ and $VerInit = \{(\Theta, v_{\text{init}})\}$.

LearnDiscrepancy(S_{init}, d, ℓ) computes the discrepancy function for mode ℓ , from initial set S_{init} and upto time d using the algorithm of Section 3.1. *ReachComp*($S_{\text{init}}, d, \beta$) first generates finite simulation traces from S_{init} and then bloats the traces to compute a reachtube using the discrepancy function β . This step is similar to the algorithm for dynamical systems given in [22].

The *GraphReach* algorithm proceeds as follows: first, a topologically sorted array of the vertices of the DAG G is computed in *Order* (Line 1). The pointer *ptr* iterates over the *Order* and for each vertex *curv* the following is computed. The variable *dt* is set to the maximum transition time to other vertices from *curv* (Line 5). For each possible initial set S_{init} corresponding to *curv* in $VerInit$, the algorithm computes a discrepancy function (Line 7) and uses it to compute a reachtube from S_{init} up to time *dt* (Line 8). For each successor *nextv* of *curv*, the restriction of the computed reachtube RT to the corresponding transition time interval $\text{elab}((\text{curv}, \text{nextv}))$ is set as an initial set for *nextv* (Lines 11–12).

The invariant verification algorithm *VerifySafety* decides safety of \mathcal{H} with respect to a given unsafe set \mathcal{U} and uses *GraphReach*. The detailed pseu-

Algorithm 1: $GraphReach(\mathcal{H})$ computes bounded time reachtubes for each vertex of the transition G of hybrid system \mathcal{H} .

```

1  $RS \leftarrow \emptyset$ ;  $VerInit \leftarrow \{(\Theta, v_{init})\}$ ;  $Order \leftarrow TopSort(G)$ ;
2 for  $ptr = 0 : len(Order) - 1$  do
3    $curv \leftarrow Order[ptr]$ ;
4    $\ell \leftarrow vlab(curv)$ ;
5    $dt \leftarrow \max\{t' \in \mathbb{R}_{\geq 0} \mid \exists vs \in \mathcal{V}, (curv, vs) \in \mathcal{E}, (t, t') \leftarrow elab((curv, vs))\}$ ;
6   for  $S_{init} \in \{S \mid \langle S, curv \rangle \in VerInit\}$  do
7      $\beta \leftarrow LearnDiscrepancy(S_{init}, dt, \ell)$ ;
8      $RT \leftarrow ReachComp(S_{init}, dt, \beta)$ ;
9      $RS \leftarrow RS \cup \langle RT, curv \rangle$ ;
10    for  $nextv \in curv.succ$  do
11       $(t, t') \leftarrow elab((curv, nextv))$ ;
12       $VerInit \leftarrow VerInit \cup \langle Restr(RT, (t, t')), nextv \rangle$ ;
13 return  $RS$ ;

```

decode appears in Appendix A.4. This algorithm proceeds in a way similar to the simulation-based verification algorithms for dynamical and hybrid systems [22, 30]. Given initial set Θ and transition graph G of \mathcal{H} , this algorithm partitions Θ into several subsets, and then for each subset S it checks whether the computed over-approximate reachtube RS from S intersects with \mathcal{U} : (a) If RS is disjoint, the system is safe starting from S ; (b) if certain part of a reachtube RT is contained in \mathcal{U} , the system is declared as unsafe and RT with the the corresponding path of the graph are returned as counter-example witnesses; (c) if neither of the above conditions hold, then the algorithm performs refinement to get a more precise over-approximation of RS . Several refinement strategies are implemented in DRYVR to accomplish the last step. Broadly, these strategies rely on splitting the initial set S into smaller sets (this gives tighter discrepancy in the subsequent vertices) and splitting the edge labels of G into smaller intervals (this gives smaller initial sets in the vertices).

The above description focuses on invariant properties, but the algorithm and our implementation in DRYVR can verify a useful class of temporal properties. These are properties in which the time constraints only refer to the time since the last mode transition. For example, for the `Powertrn` benchmark the tool verifies requirements like “after 4s in `normal` mode, the air-fuel ratio should be contained in [14.6, 14.8] and after 4s in `powerup` it should be in [12.4, 12.6]”.

Correctness Given a correct discrepancy function for each mode, we can prove the soundness and relative completeness of Algorithm 2. This analysis closely follows the proof of Theorem 19 and Theorem 21 in [20]. Combining this with the probabilistic correctness of the $LearnDiscrepancy$, we obtain the following probabilistic soundness guarantee.

Theorem 3.2. *If the β ’s returned by $LearnDiscrepancy$ are always discrepancy functions for corresponding modes, then $VerifySafety(\mathcal{H}, \mathcal{U})$ (Algorithm 2) is sound. That is, if it outputs “SAFE”, then \mathcal{H} is safe with respect to \mathcal{U} and if it outputs “UNSAFE” then there exists an execution of \mathcal{H} that enters \mathcal{U} .*

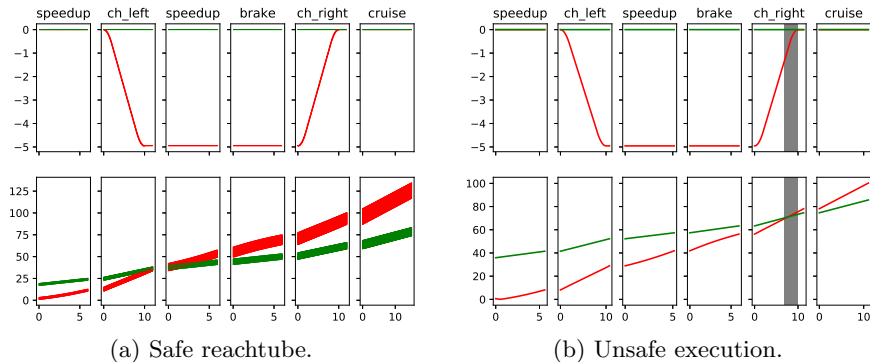


Figure 2: **AutoPassing** verification. Vehicle A’s (red) modes are shown above each subplot. Vehicle B (green) is in cruise. Top: sx_A, sx_B . Bottom: sy_A, sy_B .

3.3 Experiments on Safety Verification

The algorithms have been implemented in DRYVR and have been used to automatically verify the benchmarks from Section 2 and an Automatic Transmission System (Appendix A.2). The transition graph, the initial set, and unsafe set are given in a text file. DRYVR uses simulators for modes, and outputs either “Safe” or “Unsafe”. Reachtubes or counter-examples computed during the analysis are also stored in text files.

The implementation is in Python using the MatLab’s Python API for accessing the Simulink[®] simulators. Py-GLPK [31] is used to find the parameters of discrepancy functions; either global (GED) or piece-wise (PED) discrepancy can be selected by the user. Z3 [14] is used for reachtube operations.

Figure 2 shows example plots of computed safe reachtubes and counter-examples for a simplified **AutoPassing** in which vehicle B stays in the cruise always. As before, vehicle A goes through a sequence of modes to overtake B. Initially, for both $i \in \{A, B\}$, $sx_i = vx_i = 0$ and $vy_i = 1$, i.e., both are cruising at constant speed at the center of the right lane; initial positions along the lane are $sy_A \in [0, 2], sy_B \in [15, 17]$. Figure 2a shows the lateral positions (sx_A in red and sx_B in green, in the top subplot), and the positions along the lane (sy_A in red and sy_B in green, in the bottom plot). Vehicle A moves to left lane (sx decreases) and then back to the right, while B remains in the right lane, as A overtakes B (bottom plot). The unsafe set ($|sx_A - sx_B| < 2 \ \& \ |sy_A - sy_B| < 2$) is proved to be disjoint from computed reachtube. With a different initial set, $sy_B \in [30, 40]$, DRYVR finds counter-example (Figure 2b).

Table 1 summarizes some of the verification results obtained using DRYVR. ATS is an automatic transmission control system (see Appendix A.2 for more details). These experiments were performed on a laptop with Intel Core i7-6600U CPU and 16 GB RAM. The initial range of only the salient continuous variables are shown in the table. The unsafe sets are discussed with the model description. For example \mathcal{U}_c means two vehicles are too close. For all the benchmarks, the algorithm terminated in a few minutes which includes the time to simulate, learn discrepancy, generate reachtubes, check the safety of the reachtube, over all refinements.

For the results presented in Table 1, we used GED. The reachtube gener-

Model	TH	Initial set	\mathcal{U}	Ref	Safe	Runtime
Powertrn (5 vers, 6 edges)	80	$\lambda \in [14.6, 14.8]$	\mathcal{U}_p	2	✓	217.4s
AutoPassing (12 vers, 13 edges)	50	$sy_A \in [-1, 1] \quad sy_B \in [14, 16]$	\mathcal{U}_c	4	✓	208.4s
	50	$sy_A \in [-1, 1] \quad sy_B \in [4, 6.5]$	\mathcal{U}_c	5	✗	152.5s
Merge (7 vers, 7 edges)	50	$sx_A \in [-5, 5] \quad sy_B \in [-2, 2]$	\mathcal{U}_c	0	✓	55.0s
	50	$sx_A \in [-5, 5] \quad sy_B \in [2, 10]$	\mathcal{U}_c	-	✗	38.7s
Merge3 (6 vers, 5 edges)	50	$sy_A \in [-3, 3] \quad sy_B \in [14, 23]$ $sy_C \in [36, 45]$	\mathcal{U}_c	4	✓	197.6s
	50	$sy_A \in [-3, 3] \quad sy_B \in [14, 15]$ $sy_C \in [16, 20]$	\mathcal{U}_c	-	✗	21.3s
ATS (4 vers, 3 edges)	50	$Erpm \in [900, 1000]$	\mathcal{U}_t	2	✓	109.2s

Table 1: Safety verification results. Numbers below benchmark names: # vertices and edges of G , TH: duration of shortest path in G , Ref: # refinements performed; Runtime: overall running time.

ated by PED for **Powertrn** is more precise, but for the rest, the reachtubes and the verification times using both GED and PED were comparable. In addition to the *VerifySafety* algorithm, DRYVR also looks for counter-examples by quickly generating random executions of the hybrid system. If any of these executions is found to be unsafe, DRYVR will return “Unsafe” without starting the *VerifySafety* algorithm.

4 Reasoning Principles for Trace Containment

For a fixed unsafe set \mathcal{U} and two hybrid systems \mathcal{H}_1 and \mathcal{H}_2 , proving $\text{Reach}_{\mathcal{H}_1} \subseteq \text{Reach}_{\mathcal{H}_2}$ and the safety of \mathcal{H}_2 , allows us to conclude the safety of \mathcal{H}_1 . Proposition 2.9 establishes that proving containment of traces, trajectories, and initial sets of two hybrid systems, ensures the containment of their respective reach sets. These two observations together give us a method of concluding the safety of one system, from the safety of another, provided we can check trace containment of two graphs, and trajectory containment of two trajectory sets. In our examples, the set of modes \mathcal{L} and the set of trajectories \mathcal{TL} is often the same between the hybrid systems we care about. So in this section present different reasoning principles to check trace containment between two graphs.

Semantically, a transition graph G can be viewed as one-clock timed automaton, i.e., one can construct a timed automaton T with one-clock variable such that the timed traces of T are exactly the traces of G . This observation, coupled with the fact that checking the timed language containment of one-clock timed automata [50] is decidable, allows one to conclude that checking if $G_1 \preceq_{lmap} G_2$ is decidable. However the algorithm in [50] has non-elementary complexity. Our next observation establishes that forward simulation between graphs can be checked in polynomial time. Combined with Proposition 2.3, this gives a simple sufficient condition for trace containment that can be efficiently checked.

Proposition 4.1. *Given graphs G_1 and G_2 , and mode map $lmap$, checking if there is a forward simulation from G_1 to G_2 is in polynomial time.*

The proof follows from the algorithm for checking timed simulations between timed automata [8] and the correspondence between one-clock timed automata. Detailed proof is provided in the Appendix A.5.

Executions of hybrid systems are for bounded time, and bounded number of mode switches. This is because our transition graphs are acyclic and the labels on edges are bounded intervals. Sequential composition of graphs allows one to consider switching sequences that are longer and of a longer duration. We now present observations that will allow us to conclude the safety of a hybrid system with long switching sequences based on the safety of the system under short switching sequences. To do this we begin by observing simple properties about sequential composition of graphs. In what follows, all hybrid systems we consider will be over a fixed set of modes \mathcal{L} and trajectory set \mathcal{TL} . Also id will be identity function on \mathcal{L} . Our first observation is that trace containment is consistent with sequential composition.

Proposition 4.2. *Let $G_i, G'_i, i \in \{1, 2\}$, be four transition graphs over \mathcal{L} such that $G_1 \circ G_2$ and $G'_1 \circ G'_2$ are defined, and $G_i \preceq_{\text{id}} G'_i$ for $i \in \{1, 2\}$. Then $G_1 \circ G_2 \preceq_{\text{id}} G'_1 \circ G'_2$.*

Next we observe that sequential composition of graphs satisfies the “semi-group property”.

Proposition 4.3. *Let G_1, G_2 be graphs over \mathcal{L} for which $G_1 \circ G_2$ is defined. Let $v_{1\text{term}}$ be the unique terminal vertex of G_1 . Consider the following hybrid systems: $\mathcal{H} = \langle \mathcal{L}, \Theta, G_1 \circ G_2, \mathcal{TL} \rangle$, $\mathcal{H}_1 = \langle \mathcal{L}, \Theta, G_1, \mathcal{TL} \rangle$, and $\mathcal{H}_2 = \langle \mathcal{L}, \text{Reach}_{\mathcal{H}_1}^{v_{1\text{term}}}, G_2, \mathcal{TL} \rangle$. Then $\text{Reach}_{\mathcal{H}} = \text{Reach}_{\mathcal{H}_1} \cup \text{Reach}_{\mathcal{H}_2}$.*

Consider a graph G such that $G \circ G$ is defined. Let \mathcal{H} be the hybrid system with transition graph G , and \mathcal{H}' be the hybrid system with transition graph $G \circ G$; the modes, trajectories, and initial set for \mathcal{H} and \mathcal{H}' are the same. Now by Proposition 2.5 and 2.9, we can conclude that $\text{Reach}_{\mathcal{H}} \subseteq \text{Reach}_{\mathcal{H}'}$. Our main result of this section is that under some conditions, the converse also holds. This is useful because it allows us to conclude the safety of \mathcal{H}' from the safety of \mathcal{H} . In other words, we can conclude the safety of a hybrid system for long, possibly unbounded, switching sequences (namely \mathcal{H}') from the safety of the system under short switching sequences (namely \mathcal{H}).

Theorem 4.4. *Suppose G is such that $G \circ G$ is defined. Let v_{term} be the unique terminal vertex of G . For natural number $i \geq 1$, define $\mathcal{H}_i = \langle \mathcal{L}, \Theta, G^i, \mathcal{TL} \rangle$, where G^i is the i -fold sequential composition of G with itself. In particular, $\mathcal{H}_1 = \langle \mathcal{L}, \Theta, G, \mathcal{TL} \rangle$. If $\text{Reach}_{\mathcal{H}_1}^{v_{\text{term}}} \subseteq \Theta$ then for all i , $\text{Reach}_{\mathcal{H}_i} \subseteq \text{Reach}_{\mathcal{H}_1}$.*

Proof. Let $\Theta_1 = \text{Reach}_{\mathcal{H}_1}^{v_{\text{term}}}$. From the condition in the theorem, we know that $\Theta_1 \subseteq \Theta$. Let us define $\mathcal{H}'_i = \langle \mathcal{L}, \Theta_1, G^i, \mathcal{TL} \rangle$. Observe that from Proposition 2.9, we have $\text{Reach}_{\mathcal{H}'_i} \subseteq \text{Reach}_{\mathcal{H}_i}$.

The theorem is proved by induction on i . The base case (for $i = 1$) trivially holds. For the induction step, assume that $\text{Reach}_{\mathcal{H}_i} \subseteq \text{Reach}_{\mathcal{H}_1}$. Since \circ is associative, using Proposition 4.3 and the induction hypothesis, we have $\text{Reach}_{\mathcal{H}_{i+1}} = \text{Reach}_{\mathcal{H}_1} \cup \text{Reach}_{\mathcal{H}'_i} \subseteq \text{Reach}_{\mathcal{H}_1} \cup \text{Reach}_{\mathcal{H}_i} = \text{Reach}_{\mathcal{H}_1}$. \square

Theorem 4.4 allows one to determine the set of reachable states of a set of modes \mathcal{L} with respect to graph G^i , provided G satisfies the conditions in the statement. This observation can be generalized. If a graph G_2 satisfies conditions similar to those in Theorem 4.4, then using Proposition 4.3, we can conclude that the reachable set with respect to graph $G_1 \circ G_2^i \circ G_3$ is contained in the reachable set with respect to graph $G_1 \circ G_2 \circ G_3$. The formal statement of this observation and its proof is skipped in the interest of space, but we will use it in our experiments.

4.1 Experiments on Trace Containment Reasoning

Graph simulation Consider the AEB system of Section 2.5 with the scenario where Vehicle B is stopped ahead of vehicle A, and A transits from `cruise` to `em_brake` to avoid colliding with B. In the actual system (G_2 of Figure 3 in Appendix A.6), two different sensor systems trigger the obstacle detection and emergency braking at time intervals $[1, 2]$ and $[2.5, 3.5]$ and take the system from vertex 0 (`cruise`) to two different vertices labeled with `em_brake`. To illustrate trace containment reasoning, consider a simpler graph G_1 that allows a single transition of A from `cruise` to `em_brake` over the interval bigger $[0.5, 4.5]$. Using Proposition 2.9 and checking that graph $G_2 \preceq_{\text{id}} G_1$, it follows that verifying the safety of AEB with G_1 is adequate to infer the safety with G_2 . Figure 3c shows that the safe reachtubes returned by the algorithm for G_1 in red, indeed contain the reachtubes for G_2 (in blue and gray).

Sequential composition We revisit the Powertrn example of Section 2.1. The initial set Θ and unsafe set are the same as in Table 1. Let G_A be the graph $(v_0, \text{startup}) \xrightarrow{[5,10]} (v_1, \text{normal}) \xrightarrow{[10,15]} (v_2, \text{powerup})$, and G_B be the graph $(v_0, \text{powerup}) \xrightarrow{[5,10]} (v_1, \text{normal}) \xrightarrow{[10,15]} (v_2, \text{powerup})$. The graph $G_1 = (v_0, \text{startup}) \xrightarrow{[5,10]} (v_1, \text{normal}) \xrightarrow{[10,15]} (v_2, \text{powerup}) \xrightarrow{[5,10]} (v_3, \text{normal}) \xrightarrow{[10,15]} (v_4, \text{powerup})$, can be expressed as the composition $G_1 = G_A \circ G_B$. Consider the two hybrid systems $\mathcal{H}_i = \langle \mathcal{L}, \Theta_i, G_i, \mathcal{T}\mathcal{L} \rangle$, $i \in \{A, B\}$ with $\Theta_A = \Theta$ and $\Theta_B = \text{Reach}_{\mathcal{H}_A}^{v_2}$. DRYVR’s estimate of Θ_B had λ in the range from 14.68 to 14.71. The reachset $\text{Reach}_{\mathcal{H}_B}^{v_2}$ computed by DRYVR had λ from 14.69 to 14.70. The remaining variables also were observed to satisfy the containment condition. Therefore, $\text{Reach}_{\mathcal{H}_B}^{v_2} \subseteq \Theta_B$. Consider the two hybrid systems $\mathcal{H}_i = \langle \mathcal{L}, \Theta, G_i, \mathcal{T}\mathcal{L} \rangle$, $i \in \{1, 2\}$, where G_1 is (defined above) $G_A \circ G_B$, and $G_2 = G_A \circ G_B \circ G_B \circ G_B$. Using Theorem 4.4 it suffices to analyze \mathcal{H}_1 to verify \mathcal{H}_2 . \mathcal{H}_1 was been proved to be safe by DRYVR without any refinement. As a sanity check, we also verified the safety of \mathcal{H}_2 . DRYVR proved \mathcal{H}_2 safe without any refinement as well.

5 Conclusions

The work presented in this paper takes an alternative view that complete mathematical models of hybrid systems are unavailable. Instead, the available system description combines a black-box simulator and a white-box transition graph. Starting from this point of view, we have developed the semantic framework,

a probabilistic verification algorithm, and results on simulation relations and sequential composition for reasoning about complex hybrid systems over long switching sequences. Through modeling and analysis of a number of automotive control systems using implementations of the proposed approach, we hope to have demonstrated their promise. One direction for further exploration in this vein, is consider more general timed and hybrid automata models of the white-box, and develop the necessary algorithms and the reasoning techniques.

References

- [1] Rajeev Alur, Thao Dang, and Franjo Ivančić. Counter-example guided predicate abstraction of hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 208–223. Springer, 2003.
- [2] Yashwant Annapureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2011.
- [3] Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 365–370. Springer, 2002.
- [4] Andrea Balluchi, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Tiziano Villa, and Alberto L Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Proceedings of the International Symposium on Mathematical Theory of Networks and Systems*. Citeseer, 2006.
- [5] Sergiy Bogomolov, Alexandre Donze, Goran Frehse, Radu Grosu, Taylor T. Johnson, Hamed Ladan, Andreas Podelski, and Martin Wehrle. Guided search for hybrid systems based on coarse-grained space abstractions. *International Journal on Software Tools for Technology Transfer*, 2014.
- [6] Sergiy Bogomolov, Goran Frehse, Marius Greitschus, Radu Grosu, Corina S. Pasareanu, Andreas Podelski, and Thomas Strump. Assume-guarantee abstraction refinement meets hybrid systems. In *10th International Haifa Verification Conference*, pages 116–131, 2014.
- [7] Oleg Botchkarev and Stavros Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *International Workshop on Hybrid Systems: Computation and Control*, pages 73–88. Springer, 2000.
- [8] Kārlis Čerāns. Decidability of bisimulation equivalences for parallel timer processes. In *International Conference on Computer Aided Verification*, pages 302–315. Springer, 1992.
- [9] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification*, pages 258–263, 2013.
- [10] Alongkrit Chutinan and Bruce H Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *International workshop on hybrid systems: computation and control*, pages 76–90. Springer, 1999.

- [11] Edmund Clarke, Ansgar Fehnker, Zhi Han, Bruce Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *International journal of foundations of computer science*, 14(04):583–604, 2003.
- [12] Edmund Clarke, Ansgar Fehnker, Zhi Han, Bruce Krogh, Olaf Stursberg, and Michael Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 192–207. Springer, 2003.
- [13] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool kronos. In *Hybrid Systems III*, pages 208–219. Springer, 1996.
- [14] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
- [15] Yi Deng, Akshay Rajhans, and A. Agung Julius. Strong: A trajectory-based verification toolbox for hybrid systems. In *International Conference on Quantitative Evaluation of SysTems*, pages 165–168, 2013.
- [16] Henning Dierks, Sebastian Kupferschmid, and Kim G Larsen. Automatic abstraction refinement for timed automata. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 114–129. Springer, 2007.
- [17] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification*, pages 167–170. Springer, 2010.
- [18] Alexandre Donzé and Oded Maler. Systematic simulation using sensitivity analysis. In *International Workshop on Hybrid Systems: Computation and Control*, pages 174–189. Springer, 2007.
- [19] Laurent Doyen, Thomas A. Henzinger, and Jean-Francois Raskin. Automatic rectangular refinement of affine hybrid systems. In *International Conference on Formal Modelling and Analysis of Timed Systems, vol. 3829 in LNCS*, pages 144–161, 2005.
- [20] Parasara Sridhar Duggirala. *Dynamic Analysis of Cyber-Physical Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 2015.
- [21] Parasara Sridhar Duggirala, Chuchu Fan, Sayan Mitra, and Mahesh Viswanathan. Meeting a powertrain verification challenge. In *In the Proceedings of International Conference on Computer Aided Verification (CAV 2015)*, volume 9206 of *LNCS*, pages 536–543, San Francisco, 2015. Springer.
- [22] Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. Verification of annotated models from executions. In *Proceedings of International Conference on Embedded Software (EMSOFT 2013)*, pages 1–10, Montreal, QC, Canada, September 2013. ACM SIGBED, IEEE.

- [23] Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. C2e2: A verification tool for stateflow models. In *Proceedings of 21st International Conference Tools and Algorithms for the Construction and Analysis of Systems 2015, London, UK, April 11-18, 2015.*, volume 9035 of *Lecture Notes in Computer Science*, pages 68–82. Springer, 2015.
- [24] Bruno Dutertre and Maria Sorea. Timed systems in sal. Technical report, Computer Science Laboratory, 2004.
- [25] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410:4262–4291, 2009.
- [26] Georgios E Fainekos, Sriram Sankaranarayanan, Koichi Ueda, and Hakan Yazarel. Verification of automotive control applications using s-taliro. In *American Control Conference (ACC), 2012*, pages 3567–3572. IEEE, 2012.
- [27] Chuchu Fan, Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. Progress on powertrain verification challenge with C2E2. In *Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH 2015)*, 2015.
- [28] Chuchu Fan, James Kapinski, Xiaoqing Jin, and Sayan Mitra. Locally optimal reach set over-approximation for nonlinear systems. In *Proceedings of the 13th ACM-SIGBED International Conference on Embedded Software (EMSOFT)*, EMSOFT '16, pages 6:1–6:10, New York, NY, USA, 2016. ACM.
- [29] Chuchu Fan and Sayan Mitra. Bounded verification with on-the-fly discrepancy computation. In *13th Intl. Symposium on Automated Technology for Verification and Analysis (ATVA 2015), Shanghai, China.*, volume 9364 of *LNCS*, pages 446–463, 2015.
- [30] Chuchu Fan, Bolun Qi, Sayan Mitra, Mahesh Viswanathan, and Parasara Sridhar Duggirala. Automatic reachability analysis for nonlinear hybrid models with c2e2. In *International Conference on Computer Aided Verification*, pages 531–538. Springer, 2016.
- [31] Thomas Finley. Python package PyGLPK. <http://tfinley.net/software/pyglpk/>.
- [32] Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *International workshop on hybrid systems: computation and control*, pages 258–273. Springer, 2005.
- [33] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. Spaceex: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 379–395. Springer, 2011.

- [34] Antoine Girard and George J Pappas. Verification using simulation. In *International Workshop on Hybrid Systems: Computation and Control*, pages 272–286. Springer, 2006.
- [35] Antoine Girard, Giordano Pola, and Paulo Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Transactions on Automatic Control*, 55(1):116–126, 2010.
- [36] Mark R Greenstreet and Ian Mitchell. Reachability analysis using polygonal projections. In *International Workshop on Hybrid Systems: Computation and Control*, pages 103–116. Springer, 1999.
- [37] Thomas A Henzinger and Pei-Hsin Ho. Hytech: The cornell hybrid technology tool. In *International Hybrid Systems Workshop*, pages 265–293. Springer, 1994.
- [38] Zhenqi Huang, Chuchu Fan, Alexandru Mereacre, Sayan Mitra, and Marta Kwiatkowska. Invariant verification of nonlinear hybrid automata networks of cardiac cells. In *Computer Aided Verification (CAV 2014)*, 2014.
- [39] Sumit Kumar Jha, Bruce H. Krogh, James E. Weimer, and Edmund M. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In *Proceedings of the International Conference on Hybrid Systems: Control and Computation*, pages 287–300, 2007.
- [40] Xiaoqing Jin, Jyotirmoy V Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. Powertrain control verification benchmark. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 253–262. ACM, 2014.
- [41] A. Agung Julius, Georgios E. Fainekos, Madhukar Anand, Insup Lee, and George J. Pappas. Robust test generation and coverage for hybrid systems. In *Proceedings of the International Conference on Hybrid Systems: Control and Computation*, pages 329–342, 2007.
- [42] Aditya Kanade, Rajeev Alur, Franjo Ivančić, S. Ramesh, Sriram Sankaranarayanan, and K. C. Shashidhar. Generating and analyzing symbolic traces of simulink/stateflow models. In *Proceedings of the International Conference on Computer-Aided Verification*, pages 430–445, 2009.
- [43] Michael J Kearns and Umesh Virkumar Vazirani. *An introduction to computational learning theory*. MIT press, 1994.
- [44] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dreach: δ -reachability analysis for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 200–205. Springer, 2015.
- [45] Alexander B Kurzhanski and Pravin Varaiya. Ellipsoidal techniques for reachability analysis: internal approximation. *Systems & control letters*, 41(3):201–211, 2000.

- [46] Mathworks. Modeling an Automatic Transmission and Controller. <http://www.mathworks.com/videos/modeling-an-automatic-transmission-and-controller-68823.html>.
- [47] Mathworks. Simple 2D Kinematic Vehicle Steering Model and Animation. <https://www.mathworks.com/matlabcentral/fileexchange/54852-simple-2d-kinematic-vehicle-steering-model-and-animation?requestedDomain=www.mathworks.com>.
- [48] Ian Mitchell and Claire J Tomlin. Level set methods for computation in hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 310–323. Springer, 2000.
- [49] Johanna Nellen, Erika Ábrahám, and Benedikt Wolters. A CEGAR tool for the reachability analysis of PLC-controlled plants using hybrid automata. In *Formalisms for Reuse and Systems Integration*, volume 346, pages 55–78. Springer, 2015.
- [50] Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science*, pages 54–63. IEEE, 2004.
- [51] Pavithra Prabhakar, Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. Hybrid automata-based cegar for rectangular hybrid systems. *Formal Methods in System Design*, 46(2):105–134, 2015.
- [52] Stefan Ratschan and Zhikun She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. *ACM Transactions in Embedded Computing Systems*, 6(1), 2007.
- [53] Nima Roohi, Pavithra Prabhakar, and Mahesh Viswanathan. Hybridization based cegar for hybrid automata with affine dynamics. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 752–769. Springer, 2016.
- [54] Marc Segelken. Abstraction and counterexample-guided construction of ω -automata for model checking of step-discrete linear hybrid models. In *International Conference on Computer Aided Verification*, pages 433–448. Springer, 2007.
- [55] Norihiko Shishido and Claire J Tomlin. Ellipsoidal approximations of reachable sets for linear games. In *Proceedings of the 39th IEEE Conference on Decision and Control*, volume 1, pages 999–1004. IEEE, 2000.

A Appendix

A.1 ADAS and Autonomous Vehicle Benchmarks

We provide more details for the different scenarios used for testing ADAS and Autonomous driving control systems.

Recall, that each vehicle model in Simulink[®] has several continuous variable including the x, y -coordinates of the vehicle on the road, its velocity, heading, steering angle, etc. The vehicle can be controlled by two input signals, namely the throttle (acceleration or brake) and the steering speed. By choosing appropriate values of these input signals, we have defined the following modes for each vehicle (a) **cruise**: move forward at constant speed, **speedup**: constant acceleration, **brake**: constant (slow) deceleration, **em.brake**: constant (hard) We have designed lane switching modes **ch.left** and **ch.right** in which the acceleration and steering are controller in such a manner that the vehicle switches to its left (resp. right) lane in a certain amount of time.

For each vehicle, we mainly analysis four variables: absolute position (sx) and velocity vx orthogonal to the road direction (x -axis), and absolute position (sy) and velocity vy along the road direction (x -axis). The throttle and steering information can be expressed using the four variables. We will use subscripts to distinguish between different vehicles. The following scenarios are constructed by defining appropriate sets of initial states and transitions graphs labeled by the modes of two or more vehicles.

MergeBehind: Initial condition: vehicle A is in left and vehicle B is in the right lane; initial positions and speeds are in some range; A is in **cruise** mode, and B is in **cruise** or **speedup**. Transition graph: Vehicle A goes through the mode sequence **speedup**, **ch.right**, **cruise** with specified intervals of time to transit from mode to another mode. Requirement: A merges behind B within a time bound and maintains at least a given safe separation.

MergeAhead: Initial condition: same as **MergeBehind** with except that B is in **cruise** or **brake** mode. Transition graph: same structure as **MergeBehind** with different timing parameters. Requirement: A merges ahead of B and maintains at least a given safe separation.

AutoPassing: Initial condition: vehicle A behind B in the same lane, with A in **speedup** and B in **cruise**; initial positions and speeds are in some range. Transition graph: A goes through the mode sequence **ch.left**, **speedup**, **brake**, and **ch.right**, **cruise** with specified time intervals in each mode to complete the overtake maneuver. If B switches to **speedup** before A enters **speedup** then A aborts and changes back to right lane. If B switches to **brake** before A enters **ch.left**, then A should adjust the time to switch to **ch.left** to avoid collision. Requirement: vehicle A overtakes B while maintaining minimal safe separation.

AEB: (Emergency brakes) Initial condition: vehicle A behind B in the same lane with A in **cruise**, B is stopped (in **cruise** mode with velocity 0). Initial positions and speeds are in some range; Transition graph: A transits from **cruise** to **em.brake** over a given interval of time or several disjoint intervals

of time. Requirement: vehicle A stops behind B and maintains at least a given safe separation.

MergeBetween: Initial condition: vehicle A, B, C are all in the same lane, with A behind B, B behind C, and in the `cruise` mode, initial positions and speeds are in some range. Transition graph: A goes through the mode sequence `ch_left`, `speedup`, `brake`, and `ch_right`, `cruise` with specified time intervals in each mode to overtake B. C transits from `cruise` to `speedup` then transits back to `cruise`, so C is always ahead of A. Requirement: vehicle A merges between B and C and any two vehicles maintain at least a given safe separation.

A.2 Automatic transmission control

We provide some details about the Automatic transmission control benchmark that we have modeled as a hybrid system that combine white-box and black-box components and we have verified using DRYVR’s safety verification algorithm.

This is a slightly modified version of the Automatic Transmission model provided by Mathworks[®] as a Simulink[®] demo [46]. It is a model of an automatic transmission controller that exhibits both continuous and discrete behavior. The model has been previously used by S-talro [26] for falsifying certain requirements. We are not aware of any verification results for this system.

For our experiments, we made some minor modifications to the Simulink[®] model to create the hybrid system ATS. This allows us to simulate the vehicle from any one of the four modes, namely, `gear1`, `gear2`, `gear3` and `gear4`. Although the system has many variables, we are primarily interested in the car Speed (v), engine RPM (Erpm), impeller torque (T_i), output torque (T_o), and transmission RPM (Trpm), and therefore, use simulations that record these. Transition graph of ATS encodes transition sequences and intervals for shifting from `gear1` through to `gear4`. Requirement of interest is that the engine RPM is less than a specified maximum value, which in turn is important for limiting the thermal and mechanical stresses on the cylinders and camshafts. Typical unsafe set \mathcal{U}_t could be $\text{Erpm} > 4000$.

A.3 Proof for Proposition 3.1

Proposition 3.1. Let $\epsilon, \delta \in \mathbb{R}_+$. If $k \geq \frac{1}{\epsilon} \ln \frac{1}{\delta}$ then, with probability $\geq 1 - \delta$, the above algorithm finds (a, b) such that $\text{err}_{\mathcal{D}}(a, b) < \epsilon$.

Proof. The result follows from the PAC-learnability of concepts with low VC-dimension [43]. However, since the proof is very simple in this case, we reproduce it here for completeness. Let k be as in the statement of the proposition, and suppose the pair (a, b) identified by the algorithm has error $> \epsilon$. We will bound the probability of this happening.

Let $B = \{(x, y) \mid x > ay + b\}$. We know that $\mathcal{D}(B) > \epsilon$. The algorithm chose (a, b) only because no element from B was sampled in Step 1. The probability that this happens is $\leq (1 - \epsilon)^k$. Observing that $(1 - s) \leq e^{-s}$ for any s , we get $(1 - \epsilon)^k \leq e^{-\epsilon k} \leq e^{-\ln \frac{1}{\delta}} = \delta$. This gives us the desired result. \square

A.4 Safety Verification Algorithm

The safety verification algorithm is shown in 2. It proceeds along the line of the simulation-based verification algorithms presented in [22, 29, 23].

Algorithm 2: $VerifySafety(\mathcal{H}, \mathcal{U})$ verifies safety of hybrid system \mathcal{H} with respect to unsafe set \mathcal{U} .

```

initially:  $\mathcal{I}.push(Partition(\Theta))$ 
1 while  $\mathcal{I} \neq \emptyset$  do
2    $S \leftarrow \mathcal{I}.pop()$ ;
3    $RS \leftarrow GraphReach(\mathcal{H})$  ;
4   if  $RS \cap \mathcal{U} = \emptyset$  then
5     | continue;
6   else if  $\exists(x, l, t) \in RS$  s.t.  $\langle RT, v \rangle \in RS$  and  $(x, l, t) \subseteq \mathcal{U}$  then
7     | return UNSAFE,  $\langle RT, v \rangle$ 
8   else
9     |  $\mathcal{I}.push(Partition(S))$  ;
10    | Or,  $G \leftarrow RefineGraph(G)$  ;
11 return SAFE

```

A.5 Proof for Proposition 4.1

Proposition 4.1. Given graphs G_1 and G_2 , and mode map $lmap$, checking if there is a forward simulation from G_1 to G_2 is in polynomial time.

Proof. The result can be seen to follow from the algorithm for checking timed simulations between timed automata [8] and the correspondence between one-clock timed automata; the fact that the automata have only one clock ensures that the region construction is poly-sized as opposed to exponential-sized. However, in the special case of transition graphs there is a more direct algorithm which does not involve region construction that we describe here.

Observe that if $\{R_i\}_{i \in I}$ is a family of forward simulations between G_1 and G_2 then $\cup_{i \in I} R_i$ is also a forward simulation. Thus, like classical simulations, there is a unique largest forward simulation between two graphs that is the greatest fixpoint of a functional on relations over states of the transition graph. Therefore, starting from the relation $\mathcal{V}_1 \times \mathcal{V}_2$, one can progressively remove pairs (v, u) such that v is not simulated by u , until a fixpoint is reached. Moreover, in this case, since G_1 is a DAG, one can guarantee that the fixpoint will be reached in $|\mathcal{V}_1|$ iterations. \square

A.6 Graphs and Reachtubes for Graph Simulation

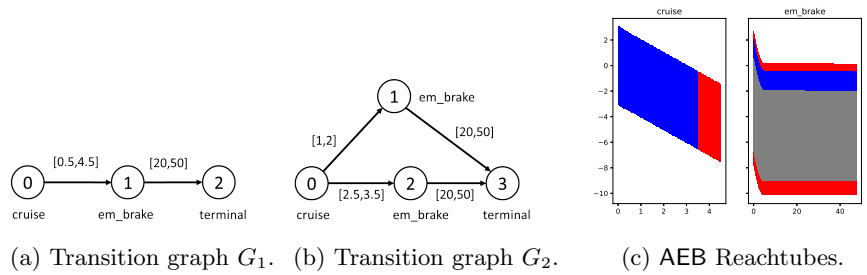


Figure 3: Graphs and reachtubes for the Automatic Emergency Braking AEB system.