Chapter 5
Data-Driven Safety Verification of Complex Cyber-Physical Systems

Chuchu Fan and Sayan Mitra

5.1 Introduction

Cyber-physical systems (CPS) are often safety critical and are expected to work in uncertain environments. Ensuring design correctness and safety of CPS has significant financial and legal implications. Existing design and test methodologies are inadequate for providing the needed level of safety assurances. For example, Koopman [55] argues how naïve test driving for reasonable catastrophic failure rates for a fleet of vehicles can grow to hundreds of billions of miles—a figure that is beyond the capabilities of even for large corporations. Formal verification, designed and deployed properly, can be the first line of defense against design bugs making their way into unsafe products [16].

A formal verification algorithm takes as input a cyber-physical system's (CPS) model and a requirement, and decides whether or not all the behaviors of the system meet the requirement. If the decision is "yes," the algorithm provides a supporting proof of this fact, which can then be used for certification, documentation, and for future testing, and maintenance. If the decision is "no," the algorithm produces a supporting counterexample or a "bug trace." This is a particular behavior of the systems resulting from specific initial states and inputs, which violates the requirement. For cyber-physical systems (CPS), the mathematical model may be a dynamical, switched, or a hybrid system, and the requirement may be a safety property, a stability property, or a temporal logic property.

Most instances of this model-based formulation of the verification problem for CPS are known to be undecidable [39, 67]. Significant progress has been made in the last decade and many powerful tools have been developed to solve approximate

C. Fan · S. Mitra (✉)
University of Illinois at Urbana-Champaign, ECE Department, Champaign, IL, USA
e-mail: cfan10@illinois.edu; mitras@illinois.edu

M. A. Al Faruque, A. Canedo (eds.), *Design Automation of Cyber-Physical Systems*, https://doi.org/10.1007/978-3-030-13050-3_5

versions of these problems for specific model classes [7, 15, 35, 36, 54]. Yet, these purely model-based techniques do not handle nonlinear and hybrid models that arise in practice. Real-world systems are often described by a heterogeneous mix of simulation code, differential equations, block diagrams, look-up tables, and machine learning modules, and it is unreasonable to even expect complete and precise models in the first place.

In the last 5 years, data-driven verification algorithms have gained momentum. Data-driven algorithms use executions (or numerical simulations) of the model in addition to statically analyzing the model itself. Thus, the verification algorithm can use powerful numerical simulators as a subroutine, which is particularly relevant for nonlinear models that do not permit a closed-form analytical solution. This opens the door to also verifying autonomous systems without complete and precise models.[1]

The basic principle of data-driven verification combines model-based *reachability analysis* with *sensitivity analysis* of the complex or unknown parts of the system. Sensitivity analysis algorithms give (probabilistic or worst-case) bounds on how much the states or outputs of a module change, with small changes in the input parameters. Under certain assumptions about the underlying system, we show that data-driven verification can indeed provide rigorous guarantees about system safety. An earlier sequence of papers culminating in [24] developed sensitivity analysis algorithms for nonlinear and hybrid systems with known models. These techniques are implemented in the C2E2 tool, which has been effectively used to verify an engine control system [46], a NASA-developed collision alerting protocol [63], and satellite controllers [24, 29]. For systems with unknown models, the deterministic sensitivity analysis algorithms have to be replaced with methods that only rely on execution data. In [32], we have shown how this problem can be cast as the well-known problem of learning a linear separator, and therefore, can be solved with probabilistic correctness guarantees. The resulting tool DRYVR was used to analyze several autonomous and ADAS-based[2] maneuvers [31, 32]. Other successful applications range across medical devices [40, 44], automotive [6, 22, 28, 47], air-traffic management [25], and energy systems [26]. A noteworthy related approach is *simulation-driven falsification*, which addresses the problem of finding bugs, but does not aim to prove their absence [1]. The search for bugs is formulated as an optimization problem, and since this typically works out to be a nonlinear and non-convex problem, stochastic optimization tools are employed to guide the search. The preeminent tool implementing this approach is S-taliro [5]; it has been effectively used to search for bugs in several practical applications [27, 65].

We present a broad and unified overview of data-driven verification with several case studies using both C2E2 and DRYVR . We classify the verification problems

[1]Autonomous systems sometimes also have incomplete requirements. The black-box approach described here does not address that problem.

[2]ADAS stands for advanced driving assistance systems such as adaptive cruise control, automatic emergency braking, etc.

regarding both the nature of the model and the requirement. First, in Sect. 5.2 we provide the necessary mathematical preliminaries; experienced readers can skip this. In Sect. 5.3, we set up the bounded verification problem and the related subproblem of sensitivity analysis. The existing techniques are described in the context of dynamical systems in Sect. 5.4, and extended for hybrid systems in Sect. 5.5. In Sect. 5.6, we discuss the black-box verification as in DRYVR. Two recent applications of data-driven verification are discussed in some detail, including a spacecraft rendezvous maneuver in Sect. 5.7.2 and an engine control challenge in Sect. 5.7.3. In Sect. 5.8, we conclude with a short summary of open problems and future research directions. Finally, in Sect. 5.9, we present pointers to additional works for further reading.

5.2 Mathematical Preliminaries

We will begin by defining the concepts and notations used throughout the chapter.

Matrix Norms For any matrix $A \in \mathbb{R}^{n \times n}$, A^T is its transpose; $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ are the maximum and minimum eigenvalues; a_{ij} denotes the element in the ith row and jth column. $\|A\|_1$, $\|A\|_2$, $\|A\|_\infty$, $\|A\|_F$ denote, respectively, the 1, 2, infinity, and the Frobenius norms of A. $|A|$ is the matrix obtained by taking the element-wise absolute value of matrix A.

Given a positive definite $n \times n$ real-valued matrix M, the *M-norm* of a vector $x \in \mathbb{R}^n$, $\|x\|_M = \sqrt{x^T M x}$ is the norm of x under the transformation M. Such M-norm will be used to represent reach sets of the system as ellipsoids. For any $M \succ 0$, there exists a nonsingular matrix $C \in \mathbb{R}^{n \times n}$, such that $M = C^T C$ and we write C as $M^{\frac{1}{2}}$. So, $\|x\|_M = \sqrt{x^T C^T C x} = \|Cx\|$. That is, $\|x\|_M$ is the 2-norm of the linearly transformed vector Cx. When $M = I$ is the identity matrix, $\|x\|_I$ coincidences with the 2-norm.

For sets $S_1, S_2 \subseteq \mathbb{R}^n$, $\texttt{hull}(S_1, S_2)$ is their convex hull. The hull of a set of $n \times n$ matrices is defined in the usual way, by considering each matrix as a vector in $\mathbb{R}^{n^2}$. The diameter of a compact set S is defined as $\texttt{Dia}(S) = \sup_{x,y \in S} \|x - y\|$. $E_{M,\delta}(x_0) = \{x \mid \|x - x_0\|_M \leq \delta\}$ represents an ellipsoid centered at $x_0 \in \mathbb{R}^n$, with *shape* M and *size* δ. The δ ball around x_0: $B_\delta(x) = \{x \mid ||x - x_0|| \leq \delta\}$ is a special case of $E_{M,\delta}(x_0)$ where M is the identity matrix I. A predicate over $\mathbb{R}^n$ is a computable function $\phi : \mathbb{R}^n \to \mathbb{B}$ that maps each state in $\mathbb{R}^n$ to either True or False.

Interval Matrices For a pair of matrices $B, C \in \mathbb{R}^{n \times n}$ with the property that: $b_{ij} \leq c_{ij}$ for all $1 \leq i, j \leq n$, we define the set of matrices $\texttt{Interval}([B, C]) \triangleq \{A \in \mathbb{R}^{n \times n} | b_{qij} \leq a_{ij} \leq c_{ij}, 1 \leq i, j \leq n\}$. Any such set of matrices is called an *interval matrix*. Interval matrices will be used to linearly over-approximate behaviors of nonlinear models. Two useful notions are the *center matrix* and the *range matrix*, defined, respectively, as $\texttt{CT}([B, C]) = (B + C)/2$ and $\texttt{RG}([B, C]) = (C - B)/2$. Then, $\texttt{Interval}([B, C])$ can also be written as $\texttt{Interval}([A_c -$

$A_r, A_c + A_r]$), where $A_c = \texttt{CT}([B, C])$, $A_r = \texttt{RG}([B, C])$. A *vertex matrix* of an interval matrix $\texttt{Interval}([B, C])$ is a matrix V whose every element is either b_{ij} or c_{ij}. Let $\texttt{VT}(\texttt{Interval}([B, C]))$ be the set of all the vertex matrices of the interval matrix $\texttt{Interval}([B, C])$. The cardinality of $\texttt{VT}(\texttt{Interval}([B, C]))$ with $B, C \in \mathbb{R}^{n\times n}$ is 2^{n^2}.

Dynamical Systems Let us denote the set of all the real-valued variables in the model as the set X. For this set of variables, the set of all values the variables can take, denoted as $val(X)$, is isomorphic to $\mathbb{R}^n$.

A continuous behavior of the system is modeled as a trajectory. A *trajectory* ξ is defined as a function $\xi : dom \to val(X)$ where dom is the time domain of evolution, and it is either $[0, T]$ for some $T > 0$, or it is $[0, \infty)$. The domain of ξ is referred as $\xi.dom$. The state of the system along the trajectory at time $t \in \tau.dom$ is $\xi(t)$. For a bounded trajectory with $\xi.dom = [0, T]$, the *duration* $\xi.dur = T$. For unbounded trajectories, $\xi.dur$ is defined as ∞. The *first state* $\xi(0)$ is denoted by τ.*fstate*, and for a bounded trajectory the *last state* $\xi.lstate = \xi(T)$ and $\xi.ltime = T$.

A T_1-*prefix* of ξ, for any $T_1 \in \xi.dom$, is the trajectory $\xi_1 : [0, T_1] \to \mathbb{R}^n$, such that for all $t \in [0, T_1]$, $\xi_1(t) = \xi(t)$. A set of trajectories $\mathscr{T}$ is *prefix-closed* if for any $\xi \in \mathscr{T}$, any of its prefix of ξ is also in $\mathscr{T}$. A set $\mathscr{T}$ is *deterministic* if for any pair $\xi_1(t), \xi_2(t) \in \mathscr{T}$, if $\xi_1(0) = \xi_2(0)$ then one is a prefix of the other. See, for example, [52] for detailed explanation of trajectories closed under prefix, suffix, and concatenation.

The continuous evolution of an n-dimensional *dynamical system* is given by an ordinary differential equation (ODE):

$$\dot{x} = f(x), \tag{5.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a locally Lipschitz and continuously differentiable function. A trajectory ξ is a *solution* of Eq. (5.1) if $\forall t \in \xi.dom$, $d\frac{\xi(t)}{dt} = f(\xi(t))$. The existence and uniqueness of solutions are guaranteed by the Lipschitz continuity of f. With an initial states and a time bound, an ODE defines a unique trajectory. Therefore, we abuse the notation and let $\xi(x_0, t)$ denote the solution $\xi(t)$ starting from $\xi(0) = x_0$. The *Jacobian* of f, $J_f : \mathbb{R}^n \to \mathbb{R}^{n\times n}$, is a matrix-valued function of all the first-order partial derivatives of f with respect to x, that is:

$$\left[J_f(x)\right]_{ij} = \frac{\partial f_i(x)}{\partial x_j}.$$

Example 5.1 The Moore–Greitzer model of a jet engine compression system is studied in [56] to understand and prevent two types of instabilities: rotating stall and surge. With a stabilizing feedback controller operating in the no-stall mode, it has the following dynamics:

$$\begin{cases} \dot{u} = -v - \frac{3}{2}u^2 - \frac{1}{2}u^3 \\ \dot{v} = 3u - v \end{cases}. \tag{5.2}$$

The Jacobian of the system is

$$J_f(x) = \begin{bmatrix} -3u - \frac{3}{2}u^2 & -1 \\ 3 & -1 \end{bmatrix}. \tag{5.3}$$

5.3 Overview of Data-Driven Verification

5.3.1 Simulations and Reachable States

Obtaining closed-form or analytical solutions for nonlinear ordinary differential equations (ODEs) is generally impossible; however, libraries such as VNODE-LP [62] and CAPD [11] use validated numerical integration to generate a sequence of evaluations of ξ with guaranteed error bounds. We define a *simulation* as a sequence of time-stamped hyper-rectangles that contain a solution of the system.

Definition 5.1 (Simulation) For any $x_0 \in \mathbb{R}$, $\tau > 0$, $\epsilon > 0$, $T > 0$, a (x_0, τ, ϵ, T)-*simulation* of the system described in Eq. (5.1) is a sequence of time-stamped sets $\{(R_i, t_i)_{i=0}^k\}$ satisfying the following:

1. $0 < t_i - t_{i-1} \leq \tau$, for each $i = 1, \ldots, k$, and $t_0 = 0$ and $t_k = T$; τ is called the *maximum sampling period*.
2. Each R_i is a hyper-rectangle in $\mathbb{R}^n$ with a diameter smaller than ϵ.
3. $\xi(x_0, t_i) \in R_i$, for each $i = 0, 1, \ldots, k$, and $\forall t \in (t_{i-1}, t_i)$, $\xi(x_0, t) \in \texttt{hull}(R_{i-1}, R_i)$, for $i = 1, \ldots, k$.

That is, at each time point t_i, the trajectory of the system $\xi(x_0, t_i)$ is contained in the hyper-rectangle R_i, and during the time intervals $t \in (t_{i-1}, t_i)$, the trajectory $\xi(x_0, t)$ is contained in the convex hull of R_{i-1} and R_i.

For a given initial set $\Theta \subseteq \mathbb{R}^n$, a state $x \in \mathbb{R}^n$ is said to be *reachable* if there exist a state $\theta \in \Theta$ and a time $t \geq 0$ such that $\xi(\theta, t) = x$. We denote by $\xi(\Theta, [t_1, t_2])$ the set of states that are reachable from Θ at any time $t \in [t_1, t_2]$. The set of reachable states at time t from initial set Θ is denoted by $\xi(\Theta, t)$. Given an n-dimensional dynamical system as in Eq. (5.1), a compact initial set $\Theta \subset \mathbb{R}^n$, an unsafe set $U \subseteq \mathbb{R}^n$, and a time bound $T > 0$, the *safety verification* problem (also called the bounded invariant verification) is to decide whether $\xi(\Theta, [0, T]) \cap U = \emptyset$. This problem is of fundamental importance as it captures many practical requirements.

Next, we define *reachtubes*, which are also sequences of time-stamped hyper-rectangles, but unlike simulations, they contain $\xi(\Theta, [0, T])$.

Definition 5.2 (Reachtube) For any $\Theta \subset \mathbb{R}^n$, $T > 0$, a (Θ, T)-*reachtube* is a sequence of time-stamped compact sets $\{(O_i, t_i)_{i=0}^k\}$, such that for each i in the sequence, $\xi(\Theta, [t_{i-1}, t_i]) \subseteq O_i$.

As we shall see in Sect. 5.3.3, computing precise reachtubes is sufficient for safety verification. Data-driven verification algorithms compute reachtubes from simulations using sensitivity analysis that we will discuss next.

5.3.2 Discrepancy Functions

Sensitivity of the solutions to changes in the initial states is formalized by discrepancy functions. Specifically, a discrepancy function bounds the distance between two neighboring trajectories as a function of the distance between their initial states and time [23, 30].

Definition 5.3 (Discrepancy Function) A continuous function $\beta : \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$ is a discrepancy function of (5.1) with initial set Θ if:

(1) for any pair of states $x_1, x_2 \in \Theta$, and any time $t \geq 0$,

$$\|\xi(x_1, t) - \xi(x_2, t)\| \leq \beta(\|x_1 - x_2\|, t), \textit{and}$$

(2) for any t,

$$\lim_{\|x_1 - x_2\| \rightarrow 0^+} \beta(\|x_1 - x_2\|, t) = 0.$$

In Definition 5.3, the norm can be any norm. We will make specific choices for designing algorithms. Consider the system (5.1), and suppose with $L > 0$ is the Lipschitz constant for $f(x)$. Then, it can be shown that $\beta(\|x_1 - x_2\|_2, t) = e^{Lt}\|x_1 - x_2\|_2$ is a discrepancy function (Proposition 1 in [21]). For Example 5.1, $L = 2$ is a Lipschitz constant, and therefore, $e^{2t}\|x_1 - x_2\|_2$ can be used as a discrepancy function for the jet engine system.

According to the definition of discrepancy function, for system (5.1), at any time t, the ball centered at $\xi(x_0, t)$ with radius $\beta(\delta, t)$ contains every solution of (5.1) starting from $B_\delta(x_0)$. Therefore, by bloating the simulation trajectories using the corresponding discrepancy function, we can obtain an over-approximation of the reachtube. We remark that this definition of discrepancy function is similar to the incremental lyapunov functions [4]; however, here we do not require that trajectories converge to each other.

5.3.3 Verification Algorithm

We are now ready to present the verification algorithm (Algorithm 1). The basic idea is simple and appeared in [18, 23] at different levels of generality. Recall, the goal is to have an algorithm that answers bounded safety queries correctly: given

Algorithm 1: Simulation-driven verification algorithm

```
   input: $\Theta, T, U, \epsilon_0, \tau_0$
1  $\delta \leftarrow$ Dia($\Theta$); $\epsilon \leftarrow \epsilon_0$; $\tau \leftarrow \tau_0$; $\mathsf{RT}_{all} \leftarrow \emptyset$;
2  $\mathsf{C} \leftarrow$ Cover($\Theta, \delta, \epsilon$);
3  while $C \neq \emptyset$ do
4      for $\langle \theta, \delta, \epsilon \rangle \in C$ do
5          $\psi = \{(R_i, t_i)_{i=0}^{k}\} \leftarrow$ Simulate($\theta, T, \epsilon, \tau$);
6          $\mathsf{RT} \leftarrow$ Bloat($\psi, \delta, \epsilon$);
7          if $RT \cap U = \emptyset$ then
8              $\mathsf{C} \leftarrow \mathsf{C} \backslash \{\langle \theta, \delta, \epsilon \rangle\}$;  $\mathsf{RT}_{all} \leftarrow \mathsf{RT}_{all} \cup \mathsf{RT}$ ;
9          else if $\exists j, R_j \subseteq U$ then
10             return $(U, \psi)$
11         else
12             $\mathsf{C} \leftarrow \mathsf{C} \cup$ Cover($B_\delta(\theta), \frac{\delta}{2}, \frac{\epsilon}{2}$)$\backslash \{\langle \theta, \delta, \epsilon \rangle\}$;
13             $\tau \leftarrow \frac{\tau}{2}$ ;
14 return (SAFE, $\mathsf{RT}_{all}$);
```

system (5.1), a compact initial set $\Theta \subset \mathbb{R}^n$, an unsafe set $U \subseteq \mathbb{R}^n$, and a time bound $T > 0$, it answers whether $\xi(\Theta, [0, T]) \cap U = \emptyset$. A verification algorithm is said to be *sound* if it answers the safety question correctly and it is said to be *complete* if it is guaranteed to terminate on any input. We know that for general nonlinear and hybrid models, the unbounded time verification problem is undecidable, that is, no algorithm exists that is both sound and complete. Even for the bounded time, version of this problem is known to be undecidable. Algorithm 1 is sound and is guaranteed to terminate under a mild assumption on the inputs.

If there exists some $\epsilon > 0$ such that $B_\epsilon(\xi(\Theta, [0, T])) \cap U = \emptyset$, we say the system is *robustly safe*. That is, all states in some envelope around the system behaviors are safe. If there exist some $\epsilon, x \in \Theta$, such that $B_\epsilon(\xi(x, t)) \subseteq U$ over some interval $[t_1, t_2], 0 \leq t_1 < t_2 \leq T$, we say the system is *robustly unsafe*. An algorithm is said to be *relatively complete* if it is guaranteed to terminate when the system is either robustly safe or robustly unsafe. Algorithm 1 is relatively complete. Another way of saying this is that Algorithm 1 is a semidecision procedure for robust safety verification.

The algorithm consists of the following three main steps: (1) Simulate the system from a finite set of states (θ) that are chosen from the compact initial set Θ. The union of a set of balls of diameter δ centered at each of the states should contain Θ. (2) Bloat the $\{(R_i, t_i)_{i=0}^{k}\}$ simulations using a discrepancy function such that the bloated sets are reachtubes from the initial covers. (3) Check each of these over-approximations, and decide if the system is safe or not. If such a decision cannot be made, then we should start from the beginning with balls with smaller diameter δ.

There are several functions referred to in Algorithm 1. Functions `Dia()` and `Simulate()` are defined to return the diameter of a set and a simulation result, respectively. The `Bloat()` function takes as the inputs the simulation ψ starting from θ, the size of the initial cover δ, and the simulation precision ϵ, and returns a reachtube that contains all the trajectories starting from the initial cover $B_\delta(\theta)$. This can be done by bloating the simulation using a discrepancy function as described in Sect. 5.4, which is an over-approximation of the distance between any neighboring trajectories starting from $B_\delta(\theta)$. Function `Cover()` returns a set of triples $\{\langle\theta, \delta, \epsilon\rangle\}$, where θs are sample states, the union of $B_\delta(\theta)$ covers Θ, and ϵ is the precision of simulation.

Initially, C contains a singleton $\langle\theta_0, \delta_0 = \texttt{Dia}(\Theta), \epsilon_0\rangle$, where $\Theta \subseteq B_{\delta_0}(\theta_0)$ and ϵ_0 is a small positive constant. For each triple $\langle\theta, \delta, \epsilon\rangle \in \mathsf{C}$, the **while**-loop from Line 3 checks the safety of the reachtube from $B_\delta(\theta)$, which is computed in Lines 5–6. ψ is a $(\theta, T, \epsilon, \tau)$-simulation, which is a sequence of time-stamped rectangles $\{(R_i, t_i)\}$ and is guaranteed to contain the trajectory $\xi(\theta, T)$ by Definition 5.1. Bloating the simulation result ψ by the discrepancy function to get RT, a $(B_\delta(\theta), T)$-reachtube, we have an over-approximation of $\xi(B_\delta(\theta), [0, T])$. The core function `Bloat()` will be discussed in detail next. If RT is disjoint from U, then the reachtube from $B_\delta(\theta)$ is safe and the corresponding triple can be safely removed from C. If for some j, R_j (one rectangle of the simulation) is completely contained in the unsafe set, then we can obtain a counterexample in the form of a trajectory that violates the safety property. Otherwise, the safety of $\xi(B_\delta(\theta), [0, T])$ is not determined, and a refinement of $B_\delta(\theta)$ needs to be made with smaller δ and smaller ϵ, τ.

Figure 5.1 gives a conceptual demonstration of Algorithm 1 running on the jet engine example (Example 5.1).

Theorem 5.1 *Algorithm 1 is sound. That is, if it returns SAFE, then indeed $\xi(\Theta, [0, T]) \cap U = \emptyset$; if it returns UNSAFE, then it also finds a counterexample, the simulation ψ which enters U. Algorithm 1 is also relatively complete. That is, for any robustly safe or unsafe system, it will terminate and decide either SAFE or UNSAFE.*

A crucial and challenging aspect of Algorithm 1 is choosing an appropriate discrepancy function with which to implement the `Bloat()` function. In the next section, we introduce algorithms that implement this function.

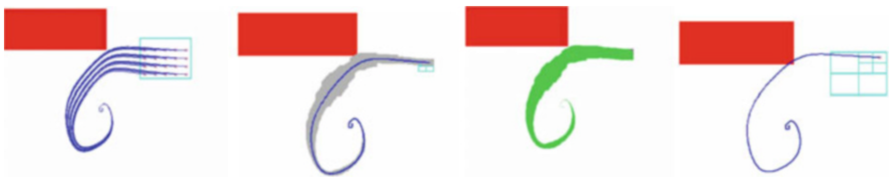


Fig. 5.1 Conceptual demonstration of verification algorithm. Red rectangle: unsafe set, cyan rectangle: cover of initial set K. Simulations (blue lines) cannot guarantee safety, but together with sensitivity analysis give reachsets (gray region) to prove safety (green region) or identify bug traces

5.4 Computing Discrepancy

In this section, we discuss several approaches for computing discrepancy functions of dynamical systems. We start with the simplest case of stable linear systems where Lyapunov equations can be used for computing discrepancy. Then, we move on to discuss nonlinear models and contraction metrics, and finally, we discuss locally optimal methods for general nonlinear systems.

5.4.1 Linear Models

For a linear time invariant (LTI) system $\dot{x} = Ax$, if the system is asymptotically stable we can find a discrepancy function by solving the Lyapunov equation:

Theorem 5.2 *For asymptotically stable linear system $\dot{x} = Ax$, given any positive definite matrix $Q \in \mathbb{R}^{n\times n}$, $\beta(\|x_1 - x_2\|_M, t) = e^{-\gamma t}\|x_1 - x_2\|_M$ is a discrepancy function, where $M \succ 0$ can be found by solving the Lyapunov equation $A^T M + MA + Q = 0$ and $\gamma = \frac{\lambda_{\min}(Q)}{2\lambda_{\max}(M)}$.*

Proof Fix any $x_1, x_2 \in \mathbb{R}^n$, and let $y(t) = \xi(x_1, t) - \xi(x_2, t)$, we have

$$\begin{aligned} d\frac{\|y(t)\|_M^2}{dt} &= \dot{y}^T(t)My(t) + y(t)M\dot{y}(t) = y^T(t)(A^T M + MA)y(t) \\ &= -y^T(t)Qy(t) \le -\lambda_{\min}(Q)y^T(t)y(t) \\ &\le -\tfrac{\lambda_{\min}(Q)}{\lambda_{\max}(M)}y^T(t)My(t) = -\tfrac{\lambda_{\min}(Q)}{\lambda_{\max}(M)}\|y(t)\|_M^2 \end{aligned}$$

By applying Grönwall's inequality, we obtain

$$\|y(t)\|_M \le e^{-\frac{\lambda_{\min}(Q)}{2\lambda_{\max}(M)}}\|y(0)\|_M. \tag{5.4}$$

5.4.2 Nonlinear Models: Optimization-Based Approaches

For nonlinear systems with trajectories that exponentially converge to each other, contraction metrics can be used as a certificate for this convergence [58]. Discrepancy functions can be computed from contraction metrics.

Definition 5.4 (From [58]) A uniform metric $\mathtt{M} : \mathbb{R}^n \times \mathbb{R}^{\ge 0} \to \mathbb{R}^{n\times n}$ is called a contraction metric for (5.1) if $\exists \gamma \in \mathbb{R}^{\ge 0}$ such that:

$$J_f^T(x)\mathtt{M}(x,t) + \mathtt{M}(x,t)J_f(x) + \dot{\mathtt{M}}(x,t) + \gamma \mathtt{M}(x,t) \preceq 0.$$

Theorem 5.3 (Theorem 2 from [58]) *For system given by (5.1) that admits a contraction metric* $\mathtt{M}$*, the trajectories converge exponentially with time, i.e.,* $\exists k \geq 1, \gamma > 0$ *such that,* $\forall x_1, x_2 \in \mathbb{R}^n$*,* $y^T(t)y(t) \leq k y^T(0)y(0)e^{-\gamma t}$*, where* $y(t) = \xi(x_1, t) - \xi(x_2, t)$*.*

Proposition 5.1 (Proposition 5 from [21]) *For system given by (5.1) that admits a contraction metric* $\mathtt{M}$*,* $\beta(\|x_1 - x_2\|_2, t) = \sqrt{k}e^{-\frac{\gamma}{2}t}\|x_1 - x_2\|_2$ *is a discrepancy function, where* k, γ *are from Theorem 5.3.*

In [8], a technique for establishing exponential convergence among trajectories using sum of squares (SOS) optimization is proposed. Informally, it searches for a contraction metric that satisfies conditions given in Definition 5.4 as follows:

1. Select the degree of the polynomial d for contraction metric $\mathtt{M}(x)$. That is, all the terms in the contraction metric are fixed degree polynomial terms in the n real variables. For example, the general form of $\mathtt{M}(x)$ for a two-dimensional system with variables u and v is given as $\begin{bmatrix} \sum a_{ij}v^i u^j & \sum b_{ij}v^i u^j \\ \sum c_{ij}v^i u^j & \sum d_{ij}v^i u^j \end{bmatrix}$.
2. Calculate $R(x) = J_f^T(x)\mathtt{M}(x) + \mathtt{M}(x)J_f(x) + \dot{\mathtt{M}}(x)$ and enforce constraints on a_{ij}, b_{ij}, c_{ij} and d_{ij} such that $R(x)$ is symmetric.
3. Impose the restrictions that polynomials $y^T\mathtt{M}(x)y$ and $-y^T R(x)y$ are sum of squares polynomials and solve for the feasibility using SOS tools. If the solution exists, then the SOS solver will find values of coefficients of polynomials.
4. If the solution is feasible, compute the exponential rate of convergence by computing the value of γ such that:
$$J_f^T(x)\mathtt{M}(x) + \mathtt{M}(x)J_f(x) + \dot{\mathtt{M}}(x) + \gamma\mathtt{M}(x) \prec 0.$$
5. If SOS solver returns infeasible, then increase the degree of the polynomial terms in $\mathtt{M}$ and repeat.

For a given nonlinear ordinary differential equation (ODE), a contraction metric that is a sum of squares polynomial is not guaranteed to exist, and hence, the above procedure is not guaranteed to terminate.

5.4.3 *Nonlinear Models: Local Discrepancy*

The main obstacle to finding a (global) discrepancy function for general nonlinear systems is the difficulty to globally bound the convergence (or divergence) rates across all trajectories. By restricting the definition of discrepancy functions over carefully computed parts of the state space, we will gain two benefits. First, such local discrepancy functions will still be adequate to compute `Bloat` needed in Algorithm 1. Second, it will become possible to compute a *local* discrepancy function automatically from simulation traces.

We begin by observing that, over a compact set $S \subseteq \mathbb{R}^n$, the Jacobian J_f of the system described by Eq. (5.1) can be over-approximated by an interval matrix. Then, we establish that the distance between two trajectories in S satisfies a differential equation from a set of differential equations described using the interval matrix. By bounding the matrix measure of the interval matrix, we can get a discrepancy function.

Since we assume that the system is continuously differentiable, the Jacobian matrix is continuous, and therefore, over a compact set S, the elements of $J_f(x)$ are bounded. That is, there exists an interval matrix $\mathscr{A}$ such that $\forall x \in S$, $J_f(x) \in \mathscr{A}$. For interval matrix $\mathscr{A} = \texttt{Interval}(B, C)$, the bounds B and C can be obtained using interval arithmetic or an optimization toolbox by maximizing and minimizing the terms of J_f over S. (The set S can be chosen to be a coarse over-approximation of the reach set, obtained using the Lipschitz constant as in Sect. 5.4.) Once the bounds are obtained, we use the interval matrix that over-approximates the behavior of $J_f(x)$ over S to analyze the rate of convergence or divergence between trajectories:

Lemma 5.1 (Lemma 3.4 from [29]) *For system (5.1) with initial set Θ starting from time t_1, suppose $S \subseteq \mathbb{R}^n$ is a compact convex set, and $[t_1, t_2]$ is a time interval such that for any $\xi(\Theta, [t_1, t_2]) \subseteq S$. If there exists an interval matrix $\mathscr{A}$ such that $\forall x \in S$, $J_f(x) \in \mathscr{A}$, then for any $x_1, x_2 \in \Theta$, and for any $t \in [t_1, t_2]$, the distance $y(t) = \xi(x_2, t) - \xi(x_1, t)$ satisfies $\dot{y}(t) = A(t)y(t)$, for some $A(t) \in \mathscr{A}$.*

$\dot{y}(t) = A(t)y(t)$ used in Lemma 5.1 can be used to define a discrepancy function. Given any matrix $M \succ 0$, $\|y(t)\|_M^2 = y^T(t)My(t)$, and by differentiating $\|y(t)\|_M^2$, we have that for any fixed $t \in [t_1, t_2]$:

$$\frac{d\|y(t)\|_M^2}{dt} = \dot{y}^T(t)y(t) + y^T(t)\dot{y}(t) = y^T(t)(A(t)^T M + MA(t))y(t), \tag{5.5}$$

for some $A(t) \in \mathscr{A}$. We write $A(t)$ as A in the following for brevity. If there exists a $\hat{\gamma}$ such that $A^T M + MA \preceq \hat{\gamma} M, \forall A \in \mathscr{A}$, then (5.5) becomes $\frac{d\|y(t)\|_M^2}{dt} \leq \hat{\gamma}\|y(t)\|_M^2$. After applying Grönwall's inequality, we have

$$\|y(t)\|_M \leq \|y(t_1)\|_M e^{\frac{\hat{\gamma}}{2}(t-t_1)}, \forall t \in [t_1, t_2].$$

The above provides a discrepancy function: $\beta(\|x_1-x_2\|_M, t) = \|x_1-x_2\|_M e^{\frac{\hat{\gamma}}{2}(t-t_1)}$. This discrepancy function could result in more or less conservative reachtubes, depending on the selection of M and $\hat{\gamma}$. Ideally, we would like to identify the optimal M such that we can obtain the tightest bound $\hat{\gamma}$. This problem is formulated as follows:

$$\begin{aligned} &\min_{\hat{\gamma} \in \mathbb{R}, M \succ 0} \quad \hat{\gamma} \\ &\text{s.t} \quad A^T M + MA \preceq \hat{\gamma} M, \quad \forall A \in \mathscr{A}. \end{aligned} \tag{5.6}$$

Solving (5.6) to obtain the optimal $\hat{\gamma}$ for each time interval involves solving optimization problems with infinite numbers of constraints (imposed by the infinite set of matrices in $\mathscr{A}$). To overcome this problem, we introduce a strategy to transform (5.6) to an equivalent problem with finitely many constraints based on the vertex matrices.

Lemma 5.2 (Lemma 4.1 from [29]) *For system (5.1) with initial set Θ starting from time t_1, suppose $S \subseteq \mathbb{R}^n$ is a compact convex set, and $[t_1, t_2]$ is a time interval such that for any $x \in \Theta$, $t \in [t_1, t_2]$, $\xi(x, t) \in S$. Let M be a positive definite $n \times n$ matrix. If there exists an interval matrix $\mathscr{A}$ such that:*

(a) $\forall\, x \in S$, $J_f(x) \in \mathscr{A}$, and
(b) $\exists\, \hat{\gamma} \in \mathbb{R}$, $\forall\, A_i \in \mathtt{VT}(\mathscr{A})$, $A_i^T M + M A_i \preceq \hat{\gamma} M$,

then for any $x_1, x_2 \in \Theta$ and $t \in [t_1, t_2]$:

$$\|\xi(x_1, t) - \xi(x_2, t)\|_M \le e^{\frac{\hat{\gamma}}{2}(t-t_1)} \|x_1 - x_2\|_M.$$

Lemma 5.2 suggests the following bilinear optimization problem for finding discrepancy over compact subsets of the state space:

$$\min_{\hat{\gamma} \in \mathbb{R}, M \succ 0} \quad \hat{\gamma} \tag{5.7}$$
$$\text{s.t.} \quad \text{for each } A_i \in \mathtt{VT}(\mathscr{A}),\ A_i^T M + M A_i \preceq \hat{\gamma} M.$$

Letting $\hat{\gamma}_{\max}$ be the maximum of the eigenvalues of $A_i^T + A_i$ for all i, then $A_i^T + A_i \preceq \hat{\gamma}_{\max} I$ (i.e., $M = I$) holds for every A_i, so a feasible solution exists for (5.7). To obtain a minimal feasible solution for $\hat{\gamma}$, we choose a range of $\gamma \in [\gamma_{\min}, \gamma_{\max}]$, where $\gamma_{\min} < \gamma_{\max}$ and perform a line search of $\hat{\gamma}$ over $[\gamma_{\min}, \gamma_{\max}]$. Note that if $\hat{\gamma}$ is fixed, then (5.7) is a semidefinite program (SDP), and a feasible solution can be obtained by an SDP solver. As a result, we can solve (5.7) using a line search strategy, where an SDP is solved at each step.

This approach is computationally intensive due to the potential $O(2^{n^2})$ matrices in $\mathtt{VT}(\mathscr{A})$ that appear in the SDP (5.7). In [29], a second method is shown to avoid the exponential increase in the number of constraints in (5.7), at the expense of lower accuracy (i.e., increasing the conservativeness).

5.4.4 Algorithm to Compute Local Optimal Reach Set

Given an initial set $B_\delta(x)$ and time bound T, Lemma 5.2 provides discrepancy functions over compact subsets of the state space, and over a bounded time horizon. To compute the reach set of a nonlinear model from a set of initial states over a long time horizon $[0, T]$, we will divide the time interval $[0, T]$ into smaller intervals

$[0, t_1], \ldots, [t_{k-1}, t_k = T]$, and compute a piece-wise discrepancy function, where each piece is relevant for a smaller portion of the state space and time.

Consider two adjacent subintervals of $[0, T]$, $a = [t_1, t_2]$ and $b = [t_2, t_3]$. Let $E_{M_a, c_a(t_2)}(\xi(x_0, t_2))$ be an ellipsoid that contains $\xi(B_\delta(x), t_2)$, and suppose we are given a matrix M_b and we want to select a $c_b(t)$ such that $\xi(B_\delta(x), t_2) \subseteq E_{M_b, c_b(t_2)}(\xi(x_0, t_2))$. To over-approximate the reach set for the interval b, we require that $c_b(t_2)$ is chosen so that at the transition time t_2:

$$E_{M_a, c_a(t_2)}(\xi(x_0, t_2)) \subseteq E_{M_b, c_b(t_2)}(\xi(x_0, t_2)). \tag{5.8}$$

This is a standard SDP problem to compute the minimum value for $c_b(t_2)$ that ensures (5.8) (see, for example, [10]). This minimum value is used as $c_b(t_2)$ for computing the reachtube for time interval b.

Let Ea denote the ellipsoid $E_{M_a, c_a(t_2)}(\xi(x_0, t_2))$ and Eb denote the ellipsoid $E_{M_b, c}(\xi(x_0, t_2))$. The problem of minimizing $c_b(t_2)$, given M_a, M_b, $c_a(t_2)$, such that Eq. (5.8) holds, is the following optimization problem:

$$\begin{aligned} \min \quad & c \\ s.t. \quad & Eb \supseteq Ea. \end{aligned} \tag{5.9}$$

In what follows, let $c_b(t_2)$ be equal to a solution of the above. We can transfer problem (5.9) to the following *sum-of-squares* problem as the "S procedure" [57] to make it solvable by SDP solvers:

$$\begin{aligned} \min \quad & c \\ s.t. \quad & c - \|x - \xi(x_0, t_2)\|^2_{M_b} - \lambda \left(c_a(t_2) - \|x - \xi(x_0, t_2)\|^2_{M_a} \right) \geq 0, \lambda \geq 0. \end{aligned} \tag{5.10}$$

We present an algorithm to compute a $(B_\delta(x), T)$-reachtube for system (5.1) using the results from Lemmas 5.2. The inputs to Algorithm `Bloat` are as follows: (1) A simulation ψ of the trajectory $\xi(x, t)$, where $x = \xi(x, t_0)$ and $t_0 = 0$, represented as a sequence of points $\xi(x, t_0), \ldots, \xi(x, t_k)$ and a sequence of hyper-rectangles $Rec(t_{i-1}, t_i) \subseteq \mathbb{R}^n$. That is, for any $t \in [t_{i-1}, t_i]$, $\xi(x, t) \in Rec(t_{i-1}, t_i)$. (2) The Jacobian matrix $J_f(\cdot)$. (3) A Lipschitz constant L for the vector field (this can be replaced by a local Lipschitz constant for each time interval). (4) A matrix M_0 and constant c_0 such that $B_\delta(x) \subseteq E_{M_0, c_0}(x)$. The output is a $(B_\delta(x), T)$-Reachtube. We assume that the exact simulation of the solution $\xi(x, t)$ exists and can be represented as a sequence of points and hyper-rectangles for ease of exposition.

Algorithm `Bloat` uses Lemma 5.2 to update the coordinate transformation matrix M_i to ensure an optimal exponential rate γ_i of the discrepancy function in each time interval $[t_{i-1}, t_i]$. It will solve the optimization problem (5.7) in each time interval to get the local optimal rate, and solve the optimization problem (5.8) when it moves forward to the next time interval.

Algorithm 2: Algorithm `Bloat`

```
   input    : $\psi$, $J_f(\cdot)$, $L$, $M_0$, $c_0$
   initially: RT $\leftarrow \emptyset$, $\gamma_0 \leftarrow -100$
1  $\delta_0 = \texttt{Dia}\left(E_{M_0,c_0}(x)\right)$ ;
2  for $i$ = $1{:}k$ do
3      $\Delta t \leftarrow t_i - t_{i-1}$ ;
4      $S \leftarrow B_{\delta_{i-1}e^{L\Delta t}}(Rec(t_{i-1}, t_i))$ ;
5      $\mathscr{A} \leftarrow \texttt{Interval}[B, C]$ such that $J_f(x) \in \texttt{Interval}[B, C], \forall x \in S$ ;
6      if $\forall V \in \texttt{VT}(\mathscr{A}) : V^T M_{i-1} + M_{i-1} V \preceq \gamma_{i-1} M_{i-1}$ then
7          $M_i \leftarrow M_{i-1}$; ;
8          $\gamma_i \leftarrow \arg\min_{\gamma \in \mathbb{R}} \forall V \in \texttt{VT}(\mathscr{A}) : V^T M_i + M_i V \preceq \gamma M_i$ ;
9          $c_{tmp} \leftarrow c_{i-1}$
10     else
11         compute $M_i, \gamma_i$ from Eq. (5.7) ;
12         compute minimum $c_{tmp}$ such that $E_{M_{i-1},c_{i-1}}(\xi(x, t_{i-1})) \subseteq E_{M_i,c_{tmp}}(\xi(x, t_{i-1}))$ ;
13     $c_i \leftarrow c_{tmp} e^{\gamma_i \Delta t}$ ;
14     $\delta_i \leftarrow \texttt{Dia}(E_{M_i,c_i}(\xi(x, t_i)))$ ;
15     $O_i \leftarrow B_{\delta'/2}(Rec(t_{i-1}, t_i))$ where $\delta' = \max\{dia\left(E_{M_i,c_{tmp}}(\xi(x, t_{i-1}))\right), \delta_i\}$ ;
16     RT $\leftarrow$ RT $\cup [O_i, t_i]$ ;
17 return RT ;
```

The algorithm proceeds as follows. The diameter of the ellipsoid containing the initial set $B_\delta(x)$ is computed as the initial set size (Line 1). At Line 4, $Rec(t_{i-1}, t_i)$, which contains the trajectory between $[t_{i-1}, t_i]$ is bloated by the factor $\delta_{i-1}e^{L\Delta t}$ which gives the set S that is guaranteed to contain $\xi(B_\delta(x), t)$ for every $t \in [t_{i-1}, t_i]$. Next, at Line 5, an interval matrix $\mathscr{A}$ containing $J_f(x)$, for each $x \in S$, is computed. The "if" condition in Line 6 determines whether the M_{i-1}, γ_{i-1} used in the previous iteration satisfy the conditions of Lemma 5.2 (γ_0 when $i = 1$, where γ_0 is an initial guess). This condition will avoid performing updates of the discrepancy function if it is unnecessary. If the condition is satisfied, then M_{i-1} is used again for the current iteration i (Lines 7–9) and γ_i will be computed as the smallest possible value such that Lemma 5.2 holds (Line 8) without updating the shape of the ellipsoid (i.e., $M_i = M_{i-1}$). In this case, the γ_i computed using M_{i-1} in the previous iteration $(i - 1)$ may not be ideal (minimum) for the current iteration (i), but we assume that it is acceptable. If M_{i-1} and γ_{i-1} do not satisfy the conditions of Lemma 5.2, that means the previous coordinate transformation can no longer ensure an accurate exponential converging or diverging rate between trajectories. Then, M_i and γ_i are recomputed at Line 11. For the vertex matrix constraints case, (5.7) is solved to update M_i and γ_i.

At Line 12, an SDP is solved to identify the smallest constant c_{tmp} for discrepancy function updating such that $E_{M_{i-1},c_{i-1}}(\xi(x, t_{i-1})) \subseteq E_{M_i,c_{tmp}}(\xi(x, t_{i-1}))$. At Line 13, we compute the updated ellipsoid size c_i such that $E_{M_i,c_i}(\xi(x, t_i))$ contains $\xi(B_\delta(x), t_i)$. At Line 14, the diameter of $E_{M_i,c_i}(\xi(x, t_i))$ is assigned to δ_i for next iteration. At Line 15, the set O_i is computed such that it contains the reach set during time interval $[t_{i-1}, t_i]$. Finally, at Line 16 RT is returned as an over-approximation of the reach set.

The next lemma states that the γ produced by Line 11 is a local optimal exponential converging or diverging rate between trajectories.

Lemma 5.3 (Lemma 5.1 from [29]) *In the ith iteration of Algorithm* `Bloat`*, suppose $\mathscr{A}$ is the approximation of the Jacobian over $[t_{i-1}, t_i]$ computed in Line 5. If E_{i-1} is the reach set at t_{i-1}, then for all M' and γ' such that $\xi(E_{i-1}, t_i) \subseteq E_{M',c'}(\xi(x, t_i))$ where c' is computed from γ' (Line 13), we have that the γ produced by Line 11 satisfies $\gamma \leq \gamma'$.*

Theorem 5.4 ensures soundness of the verification algorithm.

Theorem 5.4 (Theorem 5.2 from [29]) *For any (x, T)-simulation $\psi = \xi(x, t_0), \dots, \xi(x, t_k)$ and any constant $\delta \geq 0$, a call to* `Bloat`(ψ, δ) *returns a $(B_\delta(x), T)$-reachtube.*

Proof By Lemma 5.2, at any time $t \in [t_{i-1}, t_i]$, any other trajectory $\xi(x', t)$ starting from $x' \in E_{M_{i-1},c_{i-1}}(\xi(x, t_{i-1}))$ is guaranteed to satisfy

$$\|\xi(x, t) - \xi(x', t)\|_{M_i} \leq \|\xi(x, t_{i-1}) - x'\|_{M_i} e^{\frac{\gamma_i}{2}(t-t_{i-1})}. \tag{5.11}$$

Then, at time t_i, the reach set is guaranteed to be contained in the ellipsoid $E_{M_i,c_i}(\xi(x, t_i))$.

At Line 15, we want to compute the set O_i such that it contains the reach set during time interval $[t_{i-1}, t_i]$. According to Eq. (5.11), at any time $t \in [t_{i-1}, t_i]$, the reach set is guaranteed to be contained in the ellipsoid $E_{M_i,c(t)}(\xi(x, t))$, where $c(t) = c_{tmp} e^{\gamma_i(t-t_{i-1})}$. O_i should contain all the ellipsoids during time $[t_{i-1}, t_i]$. Therefore, it can be obtained by bloating the rectangle $Rec(t_{i-1}, t_i)$ using the largest ellipsoid's radius (half of the diameter). Since $e^{\gamma_i(t-t_{i-1})}$ is monotonic (increasing when $\gamma_i > 0$ or decreasing when $\gamma_i < 0$) with time, the largest ellipsoid during $[t_{i-1}, t_i]$ is either at t_{i-1} or at t_i. So, the largest diameter of the ellipsoids is $\max\{dia\left(E_{M_i,c_{tmp}}(\xi(x, t_{i-1}))\right), \delta_i\}$. Thus, at Line 15, O_i computed at Line 15 is an over-approximation of the reach set during time interval $[t_{i-1}, t_i]$.

When $i = 1$, because the initial ellipsoid $E_{M_0,c_0}(x)$ contains the initial set $B_\delta(x)$, we have that $E_{M_1,c_1}(\xi(x, t_1))$ defined at Line 14 contains $\xi(B_\delta(x), t_1)$. Also at Line 15, O_1 contains $\xi(B_\delta(x), [t_0, t_1])$. Repeating this reasoning for subsequent iterations, we have that $E_{M_i,c_i}(\xi(x, t_i))$ contains $\xi(B_\delta(x), t_i)$, and O_i contains $\xi(B_\delta(x), [t_{i-1}, t_i])$. Therefore, RT returned at Line 16 is a $(B_\delta(x), T)$-Reachtube.

Remark 5.1 It is straightforward to modify Algorithm 2 to accept validated simulations and the error bounds introduced. At Line 4 and Line 15, instead of bloating $Rec(t_{i-1}, t_i)$, we need to bloat `hull`$(\{R_{i-1}, R_i\})$, which is guaranteed to contain the solution $\xi(x, t)$, $\forall t \in [t_{i-1}, t_i]$. Also, at Line 12 and Line 14, when using the ellipsoid $E_{M_i,c_i}(\xi(x, t_i))$, we use $E_{M_i,c_i}(0) \oplus R_i$.

5.5 Hybrid System Verification

Hybrid systems are a natural and popular model for representing cyber-physical systems [3, 38, 51, 61]. One can view a hybrid system as a collection of ODEs—one for each *mode*—and a set of discrete transition rules for switching between the ODEs or modes. Thus, the continuous behavior of a hybrid system is described by differential equations, and discrete behavior is described by a set of transition rules that can be defined in terms of a labeled control graph, a program, or an automaton. In this section, we present extensions of the data-driven verification approach to fit hybrid models.

5.5.1 Hybrid Model

We will use L to denote a finite set of *modes*, *locations*, or discrete states. We will use a Euclidean space $X \subseteq \mathbb{R}^n$ for the continuous state. The combined *hybrid* state space is $\mathsf{L} \times X$. The discrete behavior or mode transitions will be specified a control graph over L with labels defining the guards and resets on X. A guard on X is predicate $G : X \to \mathbb{B}$, and reset function is a mapping $R : X \to X$.

Definition 5.5 Given a hybrid state space $\mathsf{L} \times X$, a control graph on $\mathsf{L} \times X$ is a labeled directed graph $G = \langle \mathsf{V}, \mathsf{E}, elab \rangle$, where:

1. $\mathsf{V} \subseteq \mathsf{L}$ is the set of vertices,
2. $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$ is the set of edges, and
3. $elab$ labels each edge $e \in \mathsf{E}$ with finitely many guards and reset maps on X.

The evolution of the system's continuous state variables is formally described by the continuous functions of initial states and time called trajectories (see Sect. 5.2). For a hybrid system with L modes, each trajectory is labeled by a mode in L. A *trajectory labeled by* L is a pair $\langle \xi(x_0, t), \ell \rangle$ where $\xi(x_0, t)$ is a trajectory starting from x_0, and $\ell \in \mathsf{L}$. A deterministic, prefix-closed set of labeled trajectories TL describes the behavior of the continuous variables in modes L.

In this section, we consider hybrid system with explicit continuous dynamics expressions. That is, the dynamical evolution of the hybrid system's continuous state variables in each mode is expressed by ODEs. Therefore, a hybrid system is formally defined as follows:

Definition 5.6 A hybrid system $\mathscr{H}$ is a tuple $\langle X, \mathsf{L}, \Theta, \mathsf{L}_{init}, G, \mathsf{TL} \rangle$, where:

1. $X \times \mathsf{L}$ is the hybrid state space,
2. $\Theta \times \mathsf{L}_{init} \subseteq X \times \mathsf{L}$ is a compact set of initial states,
3. $G = \langle \mathsf{V}, \mathsf{E}, elab \rangle$ is a control graph on $X \times \mathsf{L}$, and
4. TL is a set of deterministic, prefix-closed labeled trajectories. For each $\ell \in \mathsf{L}$, a set of trajectories TL_ℓ is specified by differential equations $f_\ell : \mathbb{R}^n \to \mathbb{R}^n$ and an invariant $\mathsf{I}_\ell \subseteq \mathbb{R}^n$, such that over any trajectory $\langle \xi, \ell \rangle \in \mathsf{TL}_\ell$, ξ evolves according to $d\frac{\xi}{dt} = f_\ell(\xi)$ at each time in the domain of ξ, and ξ satisfies the invariant I_ℓ.

Semantics of $\mathcal{H}$ is given in terms of executions which are sequences of trajectories consistent with the modes defined by the control graph. An execution of $\mathcal{H}$ starting from $x_0 \in \Theta$ and $\ell_{init} \in \mathsf{L}_{init}$ is a sequence of labeled trajectories $exec(x_0, \ell_{init}) = \langle \xi_{\ell_1}, \ell_1 \rangle, \cdots, \langle \xi_{\ell_k}, \ell_k \rangle$ such that:

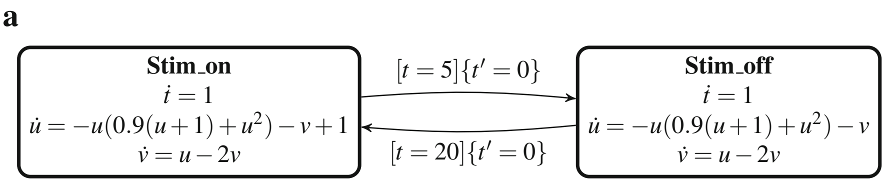
1. $\xi_{\ell_1}.fstate = x_0 \in \Theta$ and $\ell_1 = \ell_{init} \in \mathsf{L}_{init}$,
2. $\sum_{j=1}^{k} \xi_{\ell_j}.dur = T$,
3. $\ell_1, \cdots, \ell_k$ follow the control graph G. That is, for each $i > 1$, there is an edge $e \in \mathsf{E} : v_{i-1} \rightarrow v_i$ with the edge label $elab = [Guard_e]\{Reset_e\}$, such that v_{i-1} corresponds to the mode ℓ_{i-1} and v_i corresponds to the mode ℓ_i, $\xi_{\ell_{i-1}}.lstate$ satisfies the guard: $Guard_e(\xi_{\ell_{i-1}}.lstate) =$ True, and $\xi_{\ell_i}.fstate$ satisfies the reset map: $Reset_e(\xi_{\ell_i}.fstate) =$ True.

The set of all executions of $\mathcal{H}$ is denoted by $\texttt{Execs}_{\mathcal{H}}$. A state $\langle x, \ell \rangle$ is *reachable* at vertex ℓ (of graph G) if there exists an execution $\langle \xi_{\ell_1}, \ell_1 \rangle, \ldots, \langle \xi_{\ell_k}, \ell_k \rangle \in \texttt{Execs}_{\mathcal{H}}$, $i \in \{1, \ldots k\}$, and $t' \in \xi_i.dom$ such that $\ell = \ell_i$, $x = \xi_{\ell_i}(t')$. The set of reachable states is defined as:

$$\xi(\mathcal{H}, T) = \{\langle x, \ell \rangle \mid \text{for some } \ell,\ \langle x, \ell \rangle \text{isreachableatvertex} \ell\}.$$

Given a set of (unsafe) states $U \subseteq X \times \mathsf{L}$, the bounded safety verification problem is to decide whether $\xi(\mathcal{H}, T) \cap U = \emptyset$.

Example 5.2 A hybrid system that models the behavior of a cardiac pacemaker system is given in Fig. 5.2a. The hybrid system has two modes, namely, Stim_on and Stim_off. The continuous variables u and v model the voltage and the current on the tissue membrane and the timer t measures the time spent in each location.



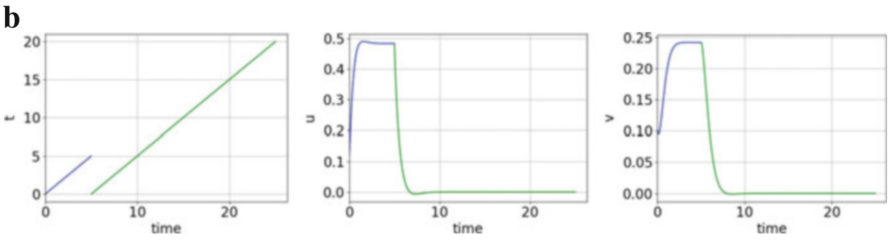


Fig. 5.2 **(a)** Hybrid system model of a cardiac cell with a pacemaker. **(b)** Sample execution of the cardiac cell-pacemaker system from the initial state $[0, 0.1, 0.1]$. Blue and green trajectories correspond to the Stim_on and Sim_off modes, respectively

The system stays in $\mathsf{Stim\_on}$ location when the pacemaker gives a stimulus to the cell and is in $\mathsf{Stim\_off}$ when the stimulus is absent. The discrete transition from $\mathsf{Stim\_on}$ to $\mathsf{Stim\_off}$ is enabled when $t = 5$; and t is reset to 0 after a transition; u and v are left unchanged. Transition from $\mathsf{Stim\_off}$ to $\mathsf{Stim\_on}$ is enabled when $t = 20$; and both these transitions are urgent. Thus, the pacemaker gives a stimulus every 25 time units for a duration of 5 time units. The behavior of the continuous variables t, u, v within a time period is given in Fig. 5.2b.

5.5.2 *Hybrid System Verification Algorithm*

We outline the hybrid extension of Algorithm 2 now presented as Algorithm 3. Algorithm 2 computes the set of reachable states for a given continuous system as described in Eq. (5.1) for a given time interval. Therefore, one can essentially apply this algorithm for each of the relevant modes of a hybrid system. For simplicity, let us assume that all the mode invariants and transition guards to be convex polyhedra, and that all the reset mappings are linear functions. Without loss of generality, we assume that there is only one mode ℓ_{init} in the set of initial locations L_{init}. Algorithm 3 performs the following three steps iteratively until the time horizon for verification:

1. For the given mode ℓ and a given initial set Θ, the algorithm first simulates from the center of Θ, computes the Jacobian of the continuous dynamics in mode ℓ, and then computes the reachable set RT_ℓ for that mode from Θ for the bounded remaining time specified using Algorithm 2.
2. The reachable set is pruned by removing all the states that violate the mode invariant.
3. The reachable set is checked to satisfy any guards for discrete transitions, and if so, the initial states for the next mode are computed by applying the reset map of the states that satisfy the guard predicate. As the reachable set of states for a hybrid system at a given time might belong to two different modes, we track the discrete transitions using a queue of tuples $\langle \Theta_{next}, \ell_{next}, t_{left} \rangle$, where ℓ_{next} is the next location that needs to be checked, Θ_{next} is the initial set that corresponds to the location ℓ_{next}, and t_{left} is remaining time we need to compute the reachable set in ℓ_{next}.

Algorithm 3 computes the reachable set for a hybrid system. The main loop that performs the three key steps iteratively happens from Line 2 to Line 9. Line 2 simulates from the center state of Θ. Then at Line 3, we compute an ellipsoid $E_{M_0,c_0}(\texttt{center}(\Theta))$ to contain the initial set Θ as an ellipsoidal initial set is required by Algorithm 2. Line 4 computes the Jacobian matrix of f_ℓ, continuous dynamics in mode ℓ. With these elements, at Line 5, we can use the `Bloat` function as Algorithm 2 to get the reachable set of states from Θ for the corresponding

Algorithm 3: Algorithm `HybridReachtube`

```
input    : Hybrid System $\mathscr{H} = \langle X \cup \{\ell\}, \Theta, \ell_{init}, T, G, \mathsf{TL} \rangle$, Time bound $T$, Lipschitz
           constants $\{L_\ell\}_{\ell \in \mathsf{L}}$, Parameters for validated simulation $\epsilon, \tau$.
initially: $\mathsf{Q} \leftarrow \langle \Theta, \ell_{init}, T \rangle$, $RT_{hybrid} \leftarrow \emptyset$
1  for each $\langle \Theta, \ell, t_{left} \rangle \in Q$ do
2      $\psi = \{(R_i, t_i)\}_{i=0}^{k} \leftarrow$ Simulate(center($\Theta$), $t_{left}, \epsilon, \tau$) ;
3      Compute $M_0, c_0$ such that $\Theta \subseteq E_{M_0,c_0}$(center($\Theta$)) ;
4      $J_{f_\ell}(x) \leftarrow$ Jacobian matrix of $f_\ell$ in mode $\ell$ ;
5      $\mathsf{RT}_\ell \leftarrow$ Bloat($\psi, J_{f_\ell}(x), L_\ell, M_0, c_0$) ;
6      $\mathsf{RT}_\ell \leftarrow \mathsf{RT}_\ell \cap \mathsf{I}_\ell$ ;
7      $\{\langle \Theta_{next}, \ell_{next}, t_{left} \rangle\} \leftarrow$ discreteTransitions($\mathsf{RT}_\ell$) ;
8      $RT_{hybrid} \leftarrow RT_{hybrid} \cup \mathsf{RT}_\ell$ ;
9      $\mathsf{Q}$.append($\{\langle \Theta_{next}, \ell_{next}, t_{left} \rangle\}$) ;
10 return $RT_{hybrid}$ ;
```

mode ℓ. Line 6 checks the invariant for the reachable set and line 7 computes the states reached Θ_{next} and the remained time t_{left} to be checked after discrete transitions.

C2E2

Algorithms 1–3 are the core procedures implemented in the verification tool *Compute Execute Check Engine*(C2E2) developed at University of Illinois [24, 33]. C2E2 is a software tool for simulating and verifying hybrid automata models. Hybrid models and the requirements have to be specified in an xml format. The tool parses the xml model to generate C++ libraries for numerical simulations and computes other relevant quantities like the Jacobians of the different modes. Using the data-driven verification algorithms, C2E2 can automatically check bounded time invariant properties of nonlinear hybrid automata. The tool also supports compositional modeling, a graphical user interface for model editing, and plotting. C2E2 has been used for modeling and analyzing robots, autonomous cars, and medical devices. Some of these applications are discussed in Sect. 5.7.

Example 5.3 (Example 5.2 Continued) Figure 5.3 shows the reachtubes of the continuous variables u and v of the cardiac cell-pacemaker system computed using the verification tool C2E2.

5.6 Verification of Models with Black-Box Components

In hybrid system models, we have discussed thus far the evolution of the continuous state variables that is explicitly described by differential equations and trajectories. In real-world control systems, “models” are typically a heterogeneous mix of simulation code, differential equations, block diagrams, and hand-crafted look-up tables. Extracting clean mathematical models (e.g., ODEs) from these descriptions

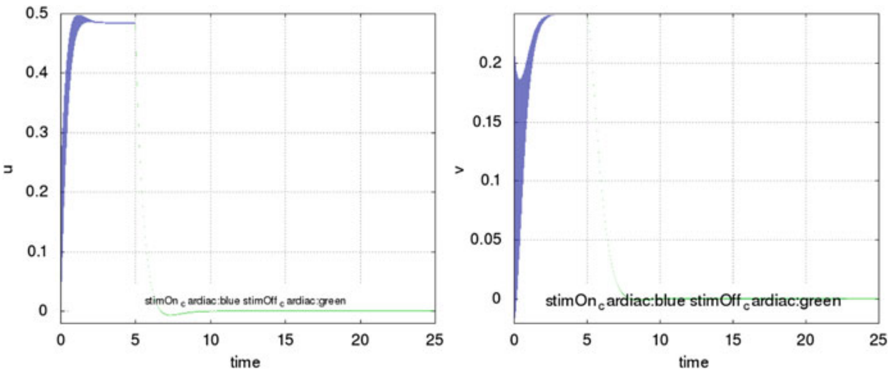


Fig. 5.3 Reachtubes of the cardiac cell-pacemaker system produced by C2E2 with initial set $t \in [0, 0]$, $u, v \in [0, 0.2]$. Left: u vs time. Right: v vs time. Blue and green regions correspond to the Stim_on and Stim_off modes, respectively

is usually infeasible. The high-level logic deciding the transitions of when and for how long the system stays in each mode is usually implemented in a relatively clean piece of code and this logical module can be seen as the control graph as in Definition 5.5. In contrast, the dynamics of physical plant, 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. This unavailability of explicit "white-box" models is a major roadblock for formal techniques becoming practical for CPS. In this section, we address this problem in the context of data-driven verification. We will view hybrid systems as a combination of a "white-box" control graph that specifies the mode switches and a "black-box" that can simulate the continuous evolution in each mode.

5.6.1 A Hybrid Formalism Accommodating Black-Boxes

Suppose the hybrid system has a set of modes L and continuous state space X as in Definition 5.6. The mode switches are defined by a control graph over L and X, as in Definition 5.5. The black-box generates a set of trajectories TL in X for each mode in L. We denote by $\mathsf{TL}_{init,\ell} = \{\xi.fstate \mid \langle \xi, \ell \rangle \in \mathsf{TL}\}$, the set of initial states of trajectories in mode ℓ. Without loss of generality, we assume that $\mathsf{TL}_{init,\ell}$ is a connected, compact subset of X.

Instead of a closed-form description of TL as in Definition 5.6, we have a *simulator* that can generate sampled data points on individual trajectories. We will develop techniques that avoid over-reliance on the models generating the trajectories and instead, work with sampled data of $\xi(\cdot)$ generated from the simulators. Of course, in order to obtain safety guarantees we will need to make assumptions about the underlying system generating the data.

Definition 5.7 A simulator for a (deterministic and prefix-closed) set TL of trajectories labeled by L is a function (or a program) $\textsc{sim}$ that takes as input a mode label $\ell \in \mathsf{L}$, an initial state $x_0 \in \mathsf{TL}_{init,\ell}$, and a finite sequence of time points $t_1, \ldots, t_k$, and returns a sequence of states $\textsc{sim}(x_0, \ell, t_1), \ldots, \textsc{sim}(x_0, \ell, t_k)$ such that there exists $\langle \xi, \ell \rangle \in \mathscr{T}$ with $\xi.\mathit{fstate} = x_0$ and for each $i \in \{1, \ldots, k\}$, $\textsc{sim}(x_0, \ell, t_i) = \xi(t_i)$.

For simplicity, we assume that the simulations are perfect (as in the last equality of Definition 5.7). Formal guarantees of soundness are not compromised if we use validated simulations instead. Our new definition of a hybrid system, therefore, is analogous to Definition 5.6 except that TL is a set of deterministic trajectories labeled by L that can be simulated but does not necessarily come from any known differential equations. Executions and reachable states are defined analogously to those in Sect. 5.5.1.

5.6.2 Learning Discrepancy from Simulations

The key subroutine needed for computing the reachable states with Algorithm 1 has to compute a discrepancy function which upper bounds the distance between trajectories. Owing to the absence of ODE models, the `Bloat` function of Algorithm 2 is useless. We will use a probabilistic algorithm for estimating the discrepancy from the data generated by black-box simulators [32].

Recall that a discrepancy function is a continuous function $\beta : \mathbb{R}^n \times \mathbb{R}^{\geq 0} \rightarrow \mathbb{R}^{\geq 0}$, such that for any pair of identically labeled trajectories $\langle \xi_1, \ell \rangle, \langle \xi_2, \ell \rangle \in \mathsf{TL}$, and any $t \in \xi_1.\mathit{dom} \cap \xi_2.\mathit{dom}$: (a) β upper bounds the distance between the trajectories, that is:

$$\|\xi_1(t) - \xi_2(t)\| \leq \beta(\|\xi_1.\mathit{fstate} - \xi_2.\mathit{fstate}\|, t), \tag{5.12}$$

and (b) β converges to 0 as the initial states converge, i.e., for any trajectory ξ and $t \in \xi.\mathit{dom}$, if a sequence of trajectories $\xi_1, \ldots, \xi_k, \ldots$ has $\xi_k.\mathit{fstate} \rightarrow \xi.\mathit{fstate}$, then $\beta(\|\xi_k.\mathit{fstate} - \xi.\mathit{fstate}\|, t) \rightarrow 0$. We present a simple method for discovering discrepancy functions that only uses simulations. Our method is based on a classical result in PAC learning theory [53]. We revisit this result before applying it to finding discrepancy functions.

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:

$$\forall (x, y) \in \Gamma.\ x \leq ay + b. \tag{5.13}$$

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.[3] The assumption is that elements of Γ can be drawn according to some (unknown) distribution $\mathscr{D}$. With respect to $\mathscr{D}$, the *error* of a pair (a, b) from satisfying Eq. (5.13) is defined to be $\mathsf{err}_{\mathscr{D}}(a, b) = \mathscr{D}(\{(x, y) \in \Gamma \mid x > ay + b\})$ where $\mathscr{D}(X)$ is the measure of set X under distribution $\mathscr{D}$. Thus, the error is the measure of points (w.r.t. $\mathscr{D}$) 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 $(x_1, y_1), \ldots (x_k, y_k)$ from Γ according to $\mathscr{D}$; 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 $i \in \{1, \ldots k\}$.

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.

Proposition 5.2 (Proposition 4 from [32]) *Let $\epsilon, \delta \in \mathbb{R}^{\geq 0}$. If $k \geq \frac{1}{\epsilon} \ln \frac{1}{\delta}$, then, with probability $\geq 1 - \delta$, the above algorithm finds (a, b) such that $\mathsf{err}_{\mathscr{D}}(a, b) < \epsilon$.*

Proof The result follows from the PAC learnability of concepts with low VC dimension [53]. 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 $\mathscr{D}(B) > \epsilon$. 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.

5.6.3 Discrepancy Functions as Linear Separators

Using the above result, we will compute discrepancy functions from simulation data, independently for each mode. Let us fix a mode $\ell \in \mathsf{L}$, and a domain $[0, T]$ for each trajectory. The special type of discrepancy functions that we will learn from simulation data are called *global exponential discrepancy (GED)* and have the special form:

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

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

Here, K and γ are constants. Thus, for any pair of trajectories ξ_1 and ξ_2 (for mode ℓ), we have

$$\forall t \in [0, T].\ \|\xi_1(t) - \xi_2(t)\| \leq \|\xi_1.\mathit{fstate} - \xi_2.\mathit{fstate}\| K e^{\gamma t}.$$

Taking logs on both sides and rearranging terms, we have

$$\forall t.\ \ln \frac{\|\xi_1(t) - \xi_2(t)\|}{\|\xi_1.\mathit{fstate} - \xi_2.\mathit{fstate}\|} \leq \gamma t + \ln K.$$

It is easy to see that a global exponential discrepancy is nothing but a linear separator for the set Γ consisting of pairs $\left(\ln \frac{\|\xi_1(t)-\xi_2(t)\|}{\|\xi_1.\mathit{fstate}-\xi_2.\mathit{fstate}\|}, t\right)$ for all pairs of trajectories ξ_1, ξ_2 and time t. Using the sampling-based algorithm described before, we could construct a GED for a mode $\ell \in \mathsf{L}$, where sampling from Γ reduces to using the simulator to generate traces from different states in $\mathsf{TL}_{init,\ell}$. Proposition 5.2 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 $\gamma T + \ln K$.

Learned Discrepancy and Guarantees in Practice
In theory, there is some probability that the learned discrepancy function β is incorrect. That is, some pair of executions $\xi, \xi' \in \mathsf{TL}$ of the system, starting from the same initial state Θ, diverges more than the bound given by the computed β. However, experiments in [32] on dozens of modes with complex, nonlinear trajectories suggest that this almost never happens. In the reported experiments, for each mode a set S_{train} of simulation traces that start from independently drawn random initial states in $\mathsf{TL}_{init,\ell}$ are used to learn a discrepancy function. Each trace has between 100–10, 000 data points, depending on the relevant time horizon and sample times. Then, another set S_{test} of 1, 000 simulations traces are drawn for validating the computed discrepancy. For every pair of trace in S_{test} and for every time point, it is checked whether the computed discrepancy satisfies Eq. (5.12). It is observed that for $|S_{\mathsf{train}}| > 10$ the computed discrepancy function is correct for 96% of the points S_{test} in and for $|S_{\mathsf{train}}| > 20$ it is correct for more than 99.9%, across all experiments.

DryVR
Replacing the `Bloat` function in Algorithm 3 with a subroutine for learning discrepancy, we can obtain a complete verification algorithm for black-box hybrid models. This is the core of the approach implemented in the open-source DRYVR verification tool [32]. The tool supports other forms of discrepancy functions (for example, piece-wise exponential and polynomial) that can also be learned from simulation data with the same type of guarantees. DryVR has been effectively employed to analyze space-craft control systems and maneuvers involving multiple autonomous and semiautonomous vehicles (see Sect. 5.7 for some examples).

5.7 Verification Case Studies

Data-driven verification algorithms have been implemented in a number of software tools such as Breach [19], C2E2[4] [33], and DryVR[5] [32]. These tools have been effective in verifying challenging benchmark applications from the automotive, aerospace, energy, and medical devices domain. In the following, we discuss three applications that were beyond the capabilities of automatic verification tools until recently, and help paint a picture of the rapid developments in this area over the last 5 years.

5.7.1 Automatic Braking and Forward Collision Avoidance System

Growth of autonomy and advanced driver assist (ADAS) features in cars has led to significant pressures for assuring system-level safety at design time. The broad topic of safety certification for such systems is currently a big open problem. While this topic touches multiple technical challenges in several disciplines that are beyond the scope of our discussion (for example, human-autonomy interactions, traffic modeling, and testing for different weather conditions), formal verification, and in particular data-driven verification can play an effective role for creating safety assurance cases needed for certification with standards like the ISO2626 [64]. Here, we summarize a comprehensive case study from [31] which looks at the most common type of rear-end crashes involving automatic emergency braking (AEB) and forward collision avoidance systems.

Each scenario for safety verification is constructed by composing several hybrid automaton models—one for each vehicle or road agent. Each vehicle has several continuous variables including the x, y-coordinates of the vehicle on the road, its velocity, heading, and steering angle. The detailed dynamics of each vehicle comes from a black-box simulator (for example, written in Python or MatLab). The higher-level decisions about the modes (for example, for "cruising," "speeding," "merging left," etc.) followed by the vehicles are captured by control graphs. In more detail, a vehicle can be controlled by two input signals, namely the throttle (acceleration or brake) and the steering. By choosing appropriate values for these input signals, the modes are defined— cruise: move forward at constant speed, speedup: constant acceleration, brake: constant (slow) deceleration, and em_brake: constant (hard) deceleration. The switching rules (guards) between the modes is defined by "driver models." For example, one such rule might state that if the distance between the ego vehicle and its leading car drops below a threshold S_{safe}, then the ego vehicle

[4]C2E2 available from:http://publish.illinois.edu/c2e2-tool/.

[5]DryVR available from:https://gitlab.engr.illinois.edu/dryvrgroup/dryvrtool.

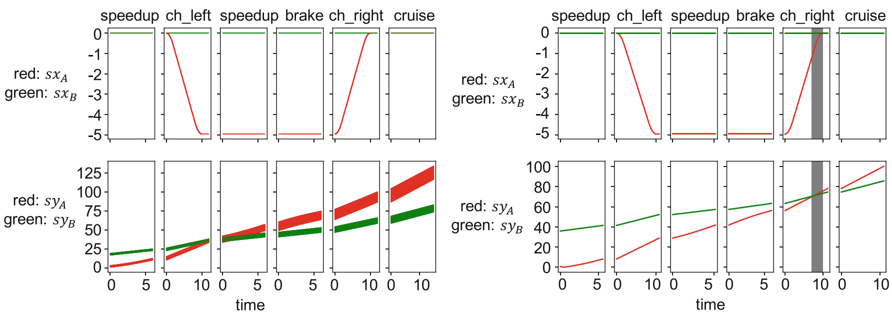


Fig. 5.4 Verification of the vehicles overtake scenario. Left: safe reachtube. Right: unsafe execution. Vehicle A's (red) modes are shown above each subplot. Vehicle B (green) is in cruise. Top: sx_A, sx_B vs time. Bottom: sy_A, sy_B vs time

switches to brake after a delay of T_{react}, where T_{react} is a parameter corresponding to driver's reactions time. Typical values of these parameters were obtained from previously available driving data. The composed hybrid automaton graph is then presented to DRYVR as the input model.

Consider a scenario with Vehicle A behind B in the same lane starting with the same speed, and A wanting to overtake B. A will switch to the left lane after it approaches B, and then switch back to the right lane once it is ahead of B. In some cases, A may fail to get ahead of B, in which case it times out and returns back in the right lane behind B. The safety requirement is that the vehicles maintain safe separation. Figure 5.4 (*left*) shows a version of this scenario that is verified to be safe by DRYVR. The plots show the reachtube over-approximations computed by DRYVR. Vehicle B stays in the cruise always but Vehicle A goes through a sequence of modes speedup, change_left, speedup, brake, and change_right, cruise to overtake B. Figure 5.4 left top shows the projection of reachtubes on lateral positions (sx_A in red and sx_B in green) subplot, and the bottom plot shows the positions along the lane (sy_A in red and sy_B in green, in the bottom plot). 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 $sy_A \in [0, 2]$, $sy_B \in [15, 17]$. As time advances, 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). With a different initial set, $sy_B \in [30, 40]$, DRYVR finds counterexample demonstrating unsafe behavior of the system (Fig. 5.4 (right)). In both of these instances, the running time for verification is of the order of minutes.

In [31], hundreds of scenarios are analyzed for 2 and 3 vehicles, with different ranges of initial velocities of the cars, different reaction times (T_{react}), and different braking profiles. DRYVR proves certain scenarios to be safe and for others it computes the severity of accidents based on the worst-case relative velocity of collisions. In [31], it is shown how these verification results can be aggregated with information about the distribution of model parameters (T_{react}, S_{safe}, etc.), to assess the system-level risk, which in turn is essential for determining automotive safety integrity levels (ASIL) for standards like the ISO26262 . In summary, this case study demonstrated that data-driven verification can be effective in analyzing relevant vehicle autonomy scenarios involving complex composition of hybrid automata and black-box simulators.

5.7.2 Autonomous Spacecraft Rendezvous

The extreme cost of failures and the infeasibility of terrestrial testing have made formal methods singularly attractive for space systems. Reachability-based automatic safety verification for satellite control systems was first studied in [48]. At the time of that study, hybrid verification tools were available only for linear hybrid systems, which have restricted applicability because many satellite control problems involve nonlinear orbital dynamics and nonlinear constraints. Here, we present a case study based on the ARPOD problem introduced in [43]. ARPOD stands for autonomous rendezvous proximity operations and docking. It captures an overarching mission needed to assemble a new space station that has been launched in separate modules. Our discussion here is based on the results presented in [12, 14].

A generic ARPOD scenario involves a passive module or a *target* (launched separately into orbit) and a *chaser* spacecraft that must transport the passive module to an on-orbit assembly location. The chaser maintains a relative bearing measurement to the target, but initially it is too far to use its range sensors. Once range measurements become available, the chaser gets more accurate relative positioning data and it can stage itself to dock with the target. Docking must happen with a specific angle of approach and closing velocity, in order to avoid collision and to ensure that the docking mechanisms on each spacecraft will mate.

For simplicity, here we discuss the planar (or 2-dimensional) version of the model. The variables of the hybrid model include position (relative to the target) x, y (in meters), time t (in minutes), and horizontal and vertical velocity v_x, v_y. The modes of the hybrid automaton capture four phases of the docking maneuver. Each phase is defined by a separation distance $\rho = \sqrt{x^2 + y^2}$ between the chaser and target spacecraft, closing this distance from up to 10 km down to 0, and then performing a maneuver once the satellites are docked. As seen in Fig. 5.5 (left), the chaser spacecraft begins in Phase 1 while the separation distance ρ is not available but only has angular of approach $\theta = \mathtt{atan}(\frac{y}{x})$ available, and the system is unobservable. While ρ gets small enough, the mission moves into Phase 2, where the chaser spacecraft now has a ranging measurement to the chaser spacecraft and must position itself for the Phase 3 docking. After the chaser moves such that $\rho \leq 100$, the docking phase, Phase 3 is initiated and additional docking port constraints are active. Once the spacecraft dock (i.e., $\rho = 0$), both spacecraft move into Phase 4, where the joint assembly must move to the relocation position.

The chaser must adhere to different sets of constraints in each discrete mode. In [13], a switched linear quadratic regulator (LQR) is designed to meet these constraints while maintaining liveness in navigating toward the target spacecraft.

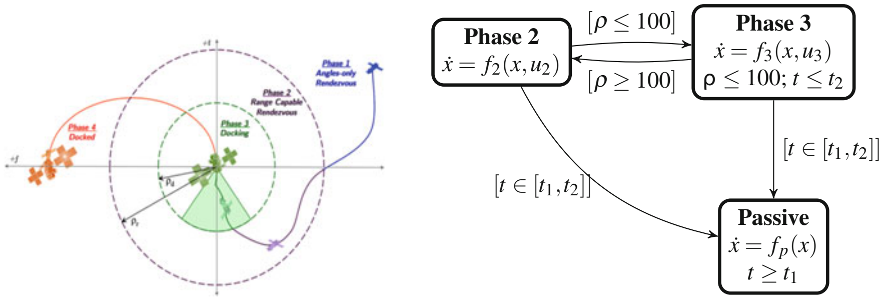


Fig. 5.5 Left: description of the overall mission phases (not to scale). Right: hybrid system model of the autonomous spacecraft rendezvous mission

Figure 5.5 (right) gives the hybrid system model of interest. In addition to the existing mode, the model also has a Passive mode in which the chaser has the thrusters shut down. The system may nondeterministically transition to the Passive mode as a result of a failure or loss of power. The nonlinear dynamic equations describing the motion of the chaser spacecraft relative to the target is given by:

$$\begin{cases} \dot{x} &= v_x \\ \dot{y} &= v_y \\ \dot{v_x} &= n^2 x + 2n v_y + \frac{\mu}{r^2} - \frac{\mu}{r_c^3}(r + x) + \frac{u_x}{m_c} \\ \dot{v_y} &= n^2 y - 2n v_x - \frac{\mu}{r_c^3} y + \frac{u_y}{m_c}. \end{cases}$$

The parameters are $\mu = 3.986 \times 10^{14} \times 60^2$ [m^3 / min^2], $r = 42164 \times 10^3$ [m], $m_c =$ 500 [kg], $n = \sqrt{\frac{\mu}{r^3}}$, and $r_c = \sqrt{(r + x)^2 + y^2}$. The linear feedback controllers for the different modes are defined as $[u_x, u_y]^T = K_1 \underline{x}$ for mode Phase 2, and $[u_x, u_y]^T = K_2 \underline{x}$ for mode Phase 2, where $\underline{x} = [x, y, v_x, v_y]^T$ is the vector of system states. The feedback matrices K_i were determined with an LQR approach applied to the linearized system dynamics, where the detailed number can be found at [13]. In mode Passive, the system is uncontrolled $[u_x, u_y]^T = [0, 0]^T$. The spacecraft starts from the initial set $x \in [-925, -875]$ [m], $y \in [-425, -375]$ [m], $v_x = 0$ [m/min] and $v_y = 0$ [m/min]. For the considered time horizon of $t \in [0, 200]$ [min], the following specifications have to be satisfied:

- **Line-of-sight:** In mode Phase 3, the spacecraft has to stay inside line-of-sight cone:

$$\{[x, y]^T \mid (x \ge -100) \wedge (y \ge x \tan(30°)) \wedge (-y \ge x \tan(30°))\}.$$

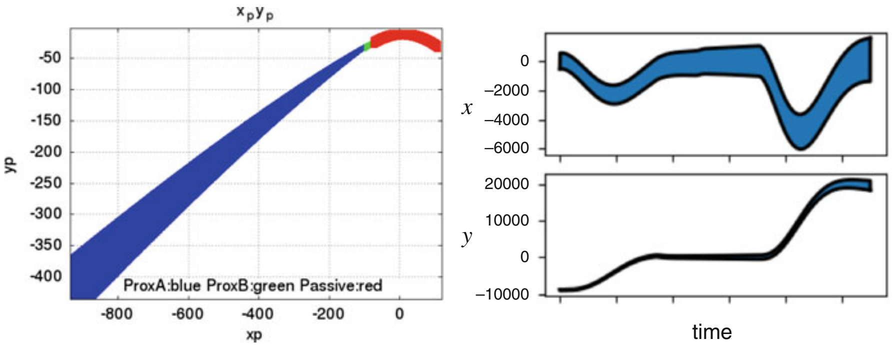


Fig. 5.6 Left: reachtube of x (x-axis) vs y (y-axis) produced by C2E2. Right: reachtube of x vs time (above) and y vs time (below) produced by DRYVR

- **Collision avoidance:** In mode Passive, the spacecraft has to avoid a collision with the target, which is modeled as a box B with 0.2 m edge length and the center placed at the origin.
- **Velocity constraint:** In mode Phase 3, the absolute velocity has to stay below 3.3 [m/min]:

$$\sqrt{v_x^2 + v_y^2} \leq 3.3 \text{ [m/min]}.$$

C2E2 was used to prove that the autonomous rendezvous system with the LQR controller satisfying the above requirements. Figure 5.6 (left) shows the reachtube of x (x-axis) vs y (y-axis) produced by C2E2. A different control strategy for ARPOD was proposed in [60] which characterizes the family of individual controllers and the required properties they should induce for the closed-loop system to solve the problem within each phase, then use a supervisor that robustly coordinates the individual controllers. Using these controlled subsystems as a black-box, we have been able to check the safety of the overall system using DRYVR. Figure 5.6 (right) shows the reachtube of x and y produced by DRYVR.

5.7.3 Powertrain Control System

The demand of greater fuel efficiency and lower emissions constantly challenges automotive companies to improve control software in the powertrain systems. Recently, a suite of benchmarks were published in [45] to introduce realistic, industrial scale models to the formal verification community. The suite consists of three Simulink® models with increasing levels of complexity and sophistication. These models capture the behavior of chemical reactions in internal combustion engines, and hybrid models are deemed suitable for capturing the discrete transitions of control software and the continuous parameters in these models. At a high level, the models take inputs from a driver (throttle angle) and the environment (sensor failures), and define the dynamics of the engine. The key controlled quantity is the air-to-fuel ratio which in turn influences the emissions, the fuel efficiency, and torque generated.

The most complicated model (Model 1) in the suite captures all the interactions taking place in a physical process and faithfully models the control software. It contains several hierarchical components in Simulink® with look-up tables, and delay differential equations. Model 1 is simplified to a model with periodic inputs to ordinary differential equations using several heuristics (Model 2), which as per the authors, exhibit similar behavior of Model 1. Then, Model 2 is further simplified to a hybrid system with only polynomial ODEs (Model 3). At the time of publication of [45], these models were beyond the reach of the then available verification tools, but within a year the simplified models were verified using C2E2 [22], and subsequently, the more complex models were handled by DryVR in [32].

In more detail, Model 2 and 3 have four variables: intake manifold pressure p, air-fuel ratio λ, intake manifold pressure estimate p_e, and integrator state i, and four modes: Start_up, Normal, Power, and Sensor_fail. The hybrid model also receives an input signal θ_{in} (throttle angle) as the user input. The required safety specification of powertrain control systems was given in [45] as a number of Signal Temporal Logic properties. Here, we only illustrate one primary result for each model, with the simple unsafe set U: in Power mode, $t > 4 \vee \lambda \notin [12.4, 12.6]$, in Normal mode, $t > 4 \vee \lambda \notin [14.6, 14.8]$. We refer readers to [22, 32] for more comprehensive studies involving other scenarios and requirements.

Figure 5.7 (left) shows the hybrid model of the powertrain control system Model 2. The physical plant dynamics are modeled using continuous variables $x_p = [p, \lambda]$,

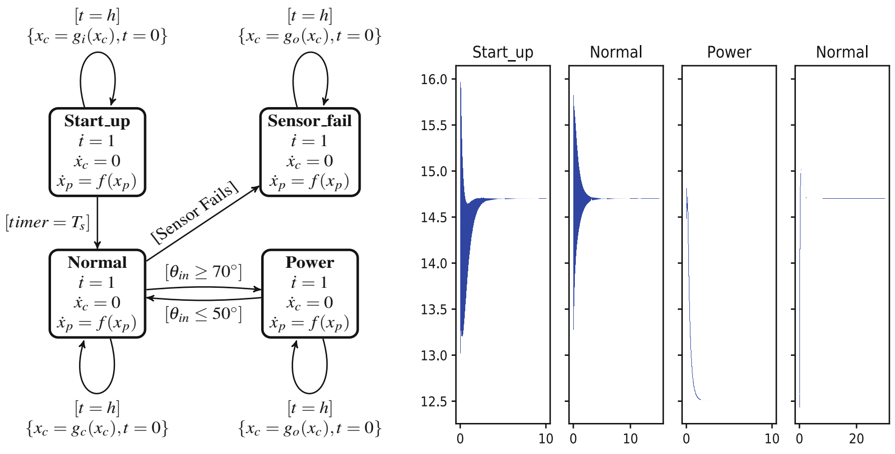


Fig. 5.7 Left: hybrid system model of the powertrain control system Model 2. Right: reachtube for λ vs time of Model 2 produced by DRYVR

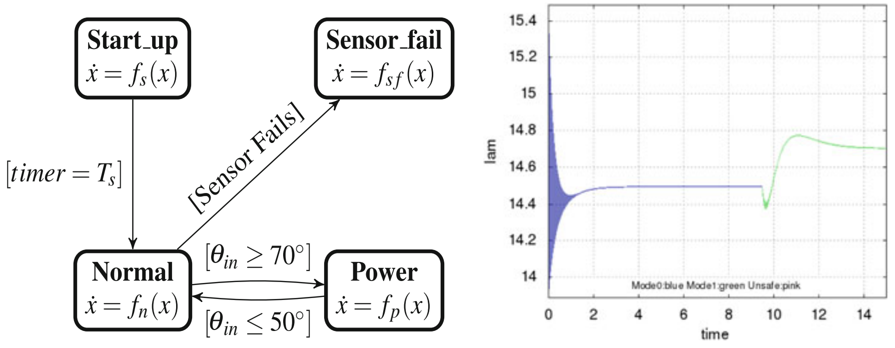


Fig. 5.8 Left: hybrid system model of the powertrain control system Model 3. Right: reachtube for λ vs time of Model 3 produced by C2E2; blue and green regions correspond to the Start_up and Normal modes, respectively

which evolve according to a nonlinear ODE $\dot{x}_p = f(x_p)$. The controller variables $x_c = [p_e, i]$ are, instead, updated periodically every h time units by the reset functions $g_i(x_c)$, $g_o(x_c)$, $g_c(x_c)$ in different modes. We treat the entire system as a black-box simulator with the four given variables and four modes. With the initial set $p \in [0.6115, 0.6315]$, $\lambda \in [14.6, 14.8]$, $p_e \in [0.5555, 0.5755]$, $i \in [0, 0.01]$, DryVR is able to prove that the system satisfies the safety requirements as stated above. Figure 5.7 (right) shows a safe reachtube of the Air/Fuel variable λ computed using DryVR going through the sequence of modes Start_up, Normal, Power, and Normal.

Model 2 got further simplified such that all four variables are continuous and follow a set of polynomial differential equations in Model 3 (see [45] for detailed ODEs). This model can be handled by C2E2. Figure 5.8 (left) shows the hybrid model (Model 3), and Fig. 5.8 (right) gives a safe reachtube of λ from the same initial set as above.

Both the spacecraft rendezvous and the powertrain control applications can be verified by either C2E2 or DryVR within a couple of minutes. These two case studies show that for hybrid systems with complex nonlinear ODEs, C2E2 can take the verification challenge, and when it is difficult to get a complete mathematical model of the system, DryVR can address the problem by treating the dynamics in each mode as a black-box.

5.8 Conclusions

Data-driven verification has shown promise in a range of real-world problems. The key to its success is the powerful amalgamation of the speed of numerical simulations with the guarantees coming from sensitivity analysis.

Nevertheless, these are the early days of exploration of these ideas; and the current approaches have several limitations: First, we observe that the tools produce better results when the system is stable. This is because the proposed methods can usually find a tighter discrepancy function for stable systems, which in turn decreases the number of refinements needed to conclude safety or find a counterexample. For unstable systems, the over-approximation of reachable sets can get very conservative, and therefore, the algorithm may not terminate in a reasonable amount of time. Second, our proposed algorithm mainly looks at safety requirements, although the computed reachtubes can be used to check for much more general specifications such as linear temporal logic. Usability of the tools remains to be improved if they are to be adopted commercially. Modeling, inter-operation with simulators, editing properties, and analyzing verification results—all of this has to become user-friendly. Finally, as usual, scalability remains a challenge. The dimension of the state space of the biggest examples the current tools have handled within a reasonable amount of time (around 2 h) is 12 for nonlinear systems and 350 for linear systems. High dimensionality will not only increase the difficulty of computing discrepancy functions but also introduce a huge number of refinements as the number of initial covers needed to cover the initial set in data-driven verification will increase exponentially.

Other important directions that call for further investigation are broadly *compositional techniques* for handling networked and distributed CPS. Examples of such systems are abundant in automotive control systems, power networks, and embedded medical devices. The naïve approach to consider such systems is to compute the cross-product of all components. However, in this way, the resulting hybrid system will become inevitably complicated with huge dimensionality and a tremendous amount of mode switches. Methods to make the analysis scalable for networked CPS with large-scale components will become a necessity. As an early step towards this direction, the notion of input-to-state discrepancy was introduced in [41, 42], and has been used to conduct a compositional sensitivity analysis of closed networked dynamical and hybrid systems [40]. The learning-based discrepancy function approach can be seen as learning an envelope which safely contains the possibly trajectories of the system. It is worth to explore more interesting learning models for identifying the dynamics of the black-box systems. There has been a methodology with a long history for building mathematical models of dynamic systems using the system's input and output behaviors called system identification. However, methods for identifying and verifying systems with guarantees remain to be developed.

5.9 Further Reading

Many new works on verification of CPS got published every year. The major conferences in this area include but not limit to International Conference on Hybrid Systems: Computation and Control (HSCC), International Conference on Computer

Aided Verification (CAV), and Applied Verification for Continuous and Hybrid Systems (ARCH).

Recently, verification tools such as Flow* [15], NLTOOLBOX [17], iSAT [34], dReach [54], and CORA [2] have demonstrated the feasibility of verifying nonlinear dynamic and hybrid models. These tools are still limited in terms of the complexity of the models and the type of external inputs they can handle, and they require quite often manual tuning of algorithmic parameters. Some of these tools' approaches for reach set estimation operate directly on the vector field involving higher-order Taylor expansions [15, 54]. However, this method suffers from complexity that increases exponentially with both the dimension of the system and the order of the model.

Several approaches have been proposed to obtain proofs about (bounded time) invariant or safety properties from simulations [20, 37]. A technique that is very close to discrepancy functions is called *sensitivity matrix*, a matrix that captures the sensitivity of the system to its initial condition x_0. This is then used to give an upper bound on the distance between two system trajectories. In [49], the authors provided sound simulation-driven methods to over-approximate the distance between trajectories, but these methods are mainly limited to affine and polynomial systems. For general nonlinear models, this approach may not be sound, as higher-order error terms are ignored when computing this upper bound.

The idea of computing the reach sets from trajectories is similar to the notions of incremental Lyapunov function [4]. In this work, we do not require systems to be incrementally stable. Similar ideas have also been considered for control synthesis in [68]. The work closest to this paper involves reachability analysis using matrix measures [59], where the authors use the fact that the matrix measure of the Jacobian matrix can bound the distance between neighboring trajectories [9, 66]. Unlike the approach in this paper which automatically computes the bounds on matrix measures, the technique there relies on user-provided closed-form matrix measure functions, which are in general difficult to compute.

Although data-driven verification is a young field, the literature in this area is growing and interesting results are published every year. For an alternative view of this topic from the modeling, testing, and verification of embedded control system perspective, we refer the interested readers to [50].

References

1. Abbas, H., & Fainekos, G. E. (2011). Linear hybrid system falsification through local search. In *Proceedings of the 9th International Symposium on Automated Technology for Verification and Analysis (ATVA 2011)*, Taipei, Taiwan, October 11–14, 2011 (pp. 503–510). https://doi.org/10.1007/978-3-642-24372-1_39.
2. Althoff, M., & Grebenyuk, D. (2016). Implementation of interval arithmetic in CORA 2016. In *ARCH Workshop* (pp. 91–105). Manchester: EasyChair.

3. Alur, R., Courcoubetis, C., Henzinger, T. A., & Ho, P. H. (1993). Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R. L. Grossman, A. Nerode, A. P. Ravn, & H. Rischel (Eds.), *Hybrid systems. Lecture notes in computer science* (Vol. 736, pp. 209–229). Berlin: Springer.
4. Angeli, D. (2002). A Lyapunov approach to incremental stability properties. *IEEE Transactions on Automatic Control, 47*(3), 410–421.
5. Annapureddy, Y., Liu, C., Fainekos, G., & Sankaranarayanan, S. (2011). S-TaLiRo: a tool for temporal logic falsification for hybrid systems. In *TACAS*. Berlin: Springer.
6. Aréchiga, N., Kapinski, J., Deshmukh, J. V., Platzer, A., & Krogh, B. (2015). Numerically-aided deductive safety proof for a powertrain control system. *Electronic Notes in Theoretical Computer Science, 317*, 19–25.
7. Asarin, E., Bournez, O., Dang, T., & Maler, O. (2000). Approximate reachability analysis of piecewise-linear dynamical systems. In B. Krogh & N. Lynch (Eds.), *Hybrid systems: computation and control. Lecture notes in computer science* (Vol. 1790, pp. 20–31). Berlin: Springer.
8. Aylward, E.M., Parrilo, P.A., & Slotine, J. -J. E. (2008). Stability and robustness analysis of nonlinear systems via contraction metrics and SOS programming. *Automatica, 44*(8), 2163–2170.
9. Boichenko, V.A., & Leonov, G.A. (1998). Lyapunov's direct method in estimates of topological entropy. *Journal of Mathematical Sciences, 91*(6), 3370–3379.
10. Boyd, S., El Ghaoui, L., Feron, E., & Balakrishnan, V. (1994). *Linear matrix inequalities in system and control theory. Studies in applied mathematics* (Vol. 15). Philadelphia, PA: SIAM.
11. CAPD. (2002). Computer assisted proofs in dynamics.
12. Chan, N., & Mitra, S. (2017). Verified hybrid LQ control for autonomous spacecraft rendezvous. In *56th IEEE Annual Conference on Decision and Control, CDC 2017, Melbourne, December 12–15, 2017* (pp. 1427–1432). Piscataway: IEEE.
13. Chan, N., & Mitra, S. (2017) Verified hybrid LQ control for autonomous spacecraft rendezvous. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)* (pp. 1427–1432). Piscataway: IEEE.
14. Chan, N., & Mitra, S. (2017). Verifying safety of an autonomous spacecraft rendezvous mission. In *ARCH17. 4th International Workshop on Applied Verification of Continuous and Hybrid Systems, Collocated with Cyber-Physical Systems Week (CPSWeek), Pittsburgh, PA, April 17, 2017* (pp. 20–32).
15. Chen, X., Ábrahám, E., & Sankaranarayanan, S. (2013). Flow*: an analyzer for non-linear hybrid systems. In *CAV* (pp. 258–263). Berlin: Springer.
16. Cook, B. (2018). Formal reasoning about the security of amazon web services. In *Computer Aided Verification—30th International Conference, CAV 2018, held as part of the Federated Logic Conference, FloC 2018, Oxford, July 14–17, 2018, Proceedings, Part I* (pp. 38–47). New York: Springer International Publishing.
17. Dang, T., Le Guernic, C., & Maler, O. (2009). Computing reachable states for nonlinear biological models. In *CMSB. Lecture notes in computer science* (Vol. 5688, pp. 126–141). Berlin: Springer.
18. Donzé, A. (2010). Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *CAV* (pp. 167–170). Berlin: Springer.
19. Donzé, A. (2010). Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification. CAV 2010. Lecture Notes in Computer Science* (Vol. 6174). Berlin: Springer.
20. Donzé, A., & Maler, O. (2007). Systematic simulation using sensitivity analysis. In *HSCC* (pp. 174–189). Berlin: Springer.
21. Duggirala, P. S. (2015). *Dynamic Analysis of Cyber-Physical Systems*. PhD thesis. Champaign: University of Illinois at Urbana-Champaign.
22. Duggirala, P. S., Fan, C., Mitra, S., & Viswanathan, M. (2015). Meeting a powertrain verification challenge. In *Computer Aided Verification* (pp. 536–543). Berlin: Springer.

23. Duggirala, P. S., Mitra, S., & Viswanathan, M. (2013). Verification of annotated models from executions. In *EMSOFT* (pp. 26:1–26:10). Piscataway: IEEE Press.
24. Duggirala, P. S., Mitra, S., Viswanathan, M., & Potok, M. (2015). C2E2: A verification tool for stateflow models. In *TACAS* (pp. 68–82). Berlin: Springer.
25. Duggirala, P. S., Wang, L., Mitra, S., Viswanathan, M., & Muñoz, C. (2014). Temporal precedence checking for switched models and its application to a parallel landing protocol. In *Formal methods* (pp. 215–229). Cham: Springer.
26. El-Guindy, A., Han, D., & Althoff, M. (2016) Formal analysis of drum-boiler units to maximize the load-following capabilities of power plants. *IEEE Transactions on Power Systems* (99), 1–12.
27. Fainekos, G. E. (2015). Automotive control design bug-finding with the s-taliro tool. In *American Control Conference, ACC 2015, Chicago, IL, July 1–3, 2015* (p. 4096). Piscataway: IEEE.
28. Fainekos, G. E., Sankaranarayanan, S., Ueda, K., & Yazarel, H. (2012) Verification of automotive control applications using S-TaLiRo. In *American Control Conference (ACC), 2012* (pp. 3567–3572). Citeseer. Piscataway: IEEE.
29. Fan, C., Kapinski, J., Jin, X., & Mitra, S. (2016). Locally optimal reach set over-approximation for nonlinear systems. In *EMSOFT* (pp. 6:1–6:10). New York: ACM.
30. Fan, C., & Mitra, S. (2015). Bounded verification with on-the-fly discrepancy computation. In *ATVA* (pp. 446–463). Berlin: Springer.
31. Fan, C., Qi, B., & Mitra, S. (2018). Data-driven formal reasoning and their applications in safety analysis of vehicle autonomy features. *IEEE Design & Test, 35*(3), 31–38.
32. Fan, C., Qi, B., Mitra, S., Viswanathan, M. (2017). Dryvr: data-driven verification and compositional reasoning for automotive systems. In *Computer Aided Verification, CAV 2017* (pp. 441–461). Heidelberg: Springer International Publishing
33. Fan, C., Qi, B., Mitra, S., Viswanathan, M., & Duggirala, P. S. (2016). Automatic reachability analysis for nonlinear hybrid models with C2E2. In *Computer Aided Verification–28th International Conference, CAV 2016, Toronto, ON, July 17–23, 2016, Proceedings, Part I* (pp. 531–538). Cham: Springer.
34. Fränzle, M., Herde, C., Teige, T., Ratschan, S., & Schubert, T. (2007). Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. *JSAT, 1*(3–4), 209–236.
35. Frehse, G. (2005). Phaver: algorithmic verification of hybrid systems past hytech. In M. Morari & L.Thiele (Eds.), *HSCC* (Vol. 3414, pp. 258–273) *Lecture notes in computer science* . Berlin: Springer.
36. Frehse, G., Guernic, C. L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T, & Maler, O. (2011). SpaceEx: scalable verification of hybrid systems. In S. Qadeer & G. Gopalakrishnan (Eds.), *CAV. Lecture Notes in Computer Science*. Berlin: Springer.
37. Girard, A., Pola, G., & Tabuada, P. (2010). Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Transactions on Automatic Control, 55*(1), 116–126.
38. Henzinger, T. A. (1996). The theory of hybrid automata. In *11th Annual IEEE Symposium on Logic in Computer Science* (pp. 278–292). Washington: IEEE Computer Society.
39. Henzinger, T. A., Kopke, P. W., Puri, A., & Varaiya, P. (1998). What's decidable about hybrid automata? *Journal of Computer and System Sciences, 57*, 94–124.
40. Huang, Z., Fan, C., Mereacre, A., Mitra, S., & Kwiatkowska, M. Z. (2014). Invariant verification of nonlinear hybrid automata networks of cardiac cells. In *CAV* (pp. 373–390). Berlin: Springer.
41. Huang, Z., Fan, C., & Mitra, S. (2017). Bounded invariant verification for time-delayed nonlinear networked dynamical systems. *Nonlinear Analysis: Hybrid Systems, 23*, 211–229.
42. Huang, Z., & Mitra, S. (2014). Proofs from simulations and modular annotations. In *HSCC, Berlin, Germany*. New York: ACM press.

43. Jewison, C., & Erwin, R. S. (2016). A spacecraft benchmark problem for hybrid control and estimation. In *2016 IEEE 55th Conference on Decision and Control (CDC)* (pp. 3300–3305). Piscataway: IEEE.
44. Jiang, Z., Pajic, M., Moarref, S., Alur, R., & Mangharam, R. (2012). Modeling and verification of a dual chamber implantable pacemaker. In *TACAS* (pp. 188–203). Berlin: Springer.
45. Jin, X., Deshmukh, J. V., Kapinski, J., Ueda, K., & Butts, K. (2014). Powertrain control verification benchmark. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control, HSCC '14* (pp. 253–262). New York, NY: ACM.
46. Jin, X., Deshmukh, J. V., Kapinski, J., Ueda, K., & Butts, K. R. (2014). Powertrain control verification benchmark. In *17th International Conference on Hybrid Systems: Computation and Control (Part of CPS Week), HSCC'14, Berlin, April 15–17, 2014* (pp. 253–262). New York: ACM.
47. Jin, X., Donzé, A., Deshmukh, J. V., & Seshia, S. A. (2015). Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 34*(11), 1704–1717.
48. Johnson, T. T., Green, J., Mitra, S., Dudley, R., & Erwin, R. S. (2012). Satellite rendezvous and conjunction avoidance: case studies in verification of nonlinear hybrid systems. In *FM 2012: Formal Methods—18th International Symposium, Paris, France, August 27–31, 2012. Proceedings* (pp. 252–266). Berlin: Springer.
49. Julius, A. A., & Pappas, G. J. (2009). Trajectory based verification using local finite-time invariance. In *HSCC* (pp. 223–236). Berlin: Springer.
50. Kapinski, J., Deshmukh, J. V., Jin, X., Ito, H., & Butts, K. (2016). Simulation-based approaches for verification of embedded control systems: an overview of traditional and advanced modeling, testing, and verification techniques. *IEEE Control Systems, 36*(6), 45–64.
51. Kaynar, D. K., Lynch, N., Segala, R., & Vaandrager, F. (2005). *The theory of timed I/O automata*. Synthesis Lectures on Computer Science. Morgan Claypool, November. Also available as Technical Report MIT-LCS-TR-917.
52. Kaynar, D. K., Lynch, N., Segala, R., & Vaandrager, F. (2010). The theory of timed I/O automata. *Synthesis Lectures on Distributed Computing Theory, 1*(1), 1–137.
53. Kearns, M. J., & Vazirani, U. V. (1994) *An introduction to computational learning theory*. Cambridge: MIT press.
54. Kong, S., Gao, S., Chen, W., & Clarke, E. (2015) dReach: δ-reachability analysis for hybrid systems. In *TACAS* (pp. 200–205). Berlin: Springer.
55. Koopman, P., & Wagner, M. (2016) Challenges in autonomous vehicle testing and validation. *SAE International Journal of Transportation Safety, 4*(2016-01-0128), 15–24.
56. Krstic, M., Kokotovic, P. V., & Kanellakopoulos, I. (1995). *Nonlinear and adaptive control design* (1st ed.). New York, NY: Wiley.
57. Liberzon, D. (2012). *Switching in systems and control*. Berlin: Springer Science & Business Media.
58. Lohmiller, W., & Slotine, J. -J. E. (1998) On contraction analysis for non-linear systems. *Automatica, 34*(6), 683–696.
59. Maidens, J., & Arcak, M. (2015). Reachability analysis of nonlinear systems using matrix measures. *IEEE Transactions on Automatic Control, 60*(1), 265–270.
60. Malladi, B. P., Sanfelice, R. G., Butcher, E., & Wang, J. (2016). Robust hybrid supervisory control for rendezvous and docking of a spacecraft. In *2016 IEEE 55th Conference on Decision and Control (CDC)* (pp. 3325–3330). Piscataway: IEEE.
61. Mitra, S. (September 2007). *A Verification Framework for Hybrid Systems*. PhD thesis. Cambridge, MA: Massachusetts Institute of Technology, 02139.
62. Nedialkov, N. (2006). VNODE-LP: validated solutions for initial value problem for ODEs. Technical report. Hamilton: McMaster University.
63. Perry, R. B., Madden, M. M., Torres-Pomales, W., & Butler, R. W. (2013). *The simplified aircraft-based paired approach with the ALAS alerting algorithm*. Technical Report NASA/TM-2013-217804. Hampton: NASA, Langley Research Center.

64. Road vehicles—Functional safety. (November 2011). Standard, International Organization for Standardization (ISO), Geneva, Switzerland.
65. Sankaranarayanan, S., Kumar, S. A., Cameron, F., Bequette, B. W., Fainekos, G., & Maahs, D. M. (March 2017) Model-based falsification of an artificial pancreas control system. *SIGBED Review, 14*(2), 24–33.
66. Sontag, E. D. (2010). Contractive systems with inputs. In *Perspectives in mathematical system theory, control, and signal processing* (pp. 217–228). Berlin: Springer.
67. Vladimerou, V., Prabhakar, P., Viswanathan, M., & Dullerud, G. E. (2008). Stormed hybrid systems. In *ICALP (2). Lecture Notes in Computer Science* (Vol. 5126, pp. 136–147). Berlin: Springer.
68. Zamani, M., Pola, G., Mazo, M., & Tabuada, P. (2012). Symbolic models for nonlinear control systems without stability assumptions. *IEEE Transactions on Automatic Control, 57*(7), 1804–1809.