Chapter 4
Formal Techniques for Verification and Testing of Cyber-Physical Systems

Jyotirmoy V. Deshmukh and Sriram Sankaranarayanan

4.1 Introduction

Cyber-physical systems (CPS) involve the tight coupling of physical components such as electrical, mechanical, hydraulic, and biological with software systems that are primarily involved in tasks such as sensing, communication, control, and interfacing with human operators. Software components in CPS are often designed using the model-based development (MBD) paradigm [113]. The MBD process proceeds in many steps: (1) First, the designer specifies the *plant model*, i.e., the dynamical characteristics of the physical parts of the system using differential, logical, and algebraic equations. Examples of plant models include the rotational dynamics model of the camshaft in an automobile engine, the thermodynamic model of an internal combustion engine, kinematic and dynamic models for ground and air vehicles, and pharmacokinetic models of human physiology. (2) The next step is to design control software to regulate the behavior of the physical system. This step often involves the use of techniques from control theory to design embedded controllers, techniques from distributed systems to achieve communication and coordination, and more recently, techniques from artificial intelligence to allow learning and adaptation. (3) The final step is to define an environment model which encapsulates physical assumptions on the exogenous quantities that affect the system (such as atmospheric turbulence, driver behavior, or meal intake by a patient). The composition of these three types of models (plant, software, and environment) constitutes the overall *closed-loop* system.

J. V. Deshmukh (✉)
University of Southern California, Los Angeles, CA, USA
e-mail: jyotirmoy.deshmukh@usc.edu

S. Sankaranarayanan
University of Colorado, Boulder, CO, USA
e-mail: srirams@colorado.edu

M. A. Al Faruque, A. Canedo (eds.), *Design Automation of Cyber-Physical Systems*, https://doi.org/10.1007/978-3-030-13050-3_4

Typically, plant models in an MBD process are deterministic. Any uncertainty is encoded in the environment model as either a nondeterministic choice on inputs to the plant model (subject to an appropriate set of constraints) or a random choice on the inputs subject to an appropriate probability distribution. Though it is also possible to model certain phenomena such as manufacturing variations, uncertainties in physics-based modeling, and sensor/actuator noise using a *stochastic dynamical* plant model, industrial MBD frameworks rarely use stochastic models during the control design process. The controller models are typically deterministic, as they represent a software implementation. In this chapter, we focus on plant and controller models that are deterministic, and environment models that are nondeterministic (not stochastic[1]).

Mathematical models for CPS applications help us analyze the system in multiple ways: (1) models are simulated under various input conditions to predict how the system as a whole would behave. Often these input conditions may be hard and expensive to recreate in the physical world. For systems involving human operators, models serve as an important alternative to real physical tests that may be dangerous or even unethical; and (2) models can expose latent/hidden system variables that are hard to measure, and thus allow us to examine their presumed behavior. In Sect. 4.2, we summarize various kinds of mathematical models that are used in the CPS domain, and typical applications for each model type.

Next, we describe *behavioral specifications*. Note that many industrial settings use the term *requirements* to mean behavioral specifications. The term specification is instead used to designate *a specification model*—a high-level programmatic description of the embedded software code. Behavioral specifications go hand-in-hand with models and describe desirable properties of the system as a whole. The specifications can be high level ("end-to-end"), describing a desired property of the system as a whole (e.g., the car will not be physically damaged by the action of the adaptive cruise control subsystem) or at the modular level, focusing on an individual module of the system (e.g., when the input to the controller is within $[-2, 2]$, the output must be within $[-1, 1]$). In Sect. 4.2, we also discuss a formalism used for behavioral specifications of CPS models.

Given a mathematical model of the system M, and a behavioral specification φ, there are two main kinds of analysis problems that focus on ensuring correctness of the CPS design: *formal verification* and *falsification*. The main purpose of verification is to prove the absence of failures in a given CPS model, where a failure is defined as the violation of a given formal specification. Many verification procedures perform a best-effort search for a proof of system correctness, wherein a failure to find one may lead to an inconclusive result. On the other hand, test generation or *falsification* focuses on providing evidence of the *presence of failures*

[1]Allowing stochasticity in the plant or environment model necessitates treating the closed-loop CPS model as a stochastic dynamical system. The techniques for verification and testing of such systems are quite different. As we wish to focus on techniques that are closer to industrial use of MBD for CPS applications, we refer the reader to [36, 71] for excellent surveys.

in the form of counterexamples. Falsification procedures perform a best effort search for a counterexample to the property of interest, with a failure to find a counterexample leading to an inconclusive result.

We now formalize these problems. A typical abstraction for a mathematical model of a CPS, M, is as a *stateful* system that maps timed input behaviors (i.e., input signals) to output signals. A *signal* is defined as a function mapping a *time domain*—a finite or infinite subset of positive real numbers—to some value in a *signal domain*. For simplicity, we consider signal domains that are compact subsets of the real numbers. For ease of exposition, we assume that the time domain for the input and output signals is the same set $\mathbb{T}$, and the input and output signal domains are respectively $\mathscr{U}$ and $\mathscr{Y}$. Let the initial set of states for M be the set $\mathscr{X}_0$. Let $u \in \mathbb{T}^{\mathscr{U}}$ be an input signal and let $y \in \mathbb{T}^{\mathscr{Y}}$ be an output signal. Thus, M defines a function that maps a state $x_0 \in \mathscr{X}_0$, and an input signal u to an output signal y, i.e., $y = M(x_0, u)$. Finally, assume that we are given a specification φ, which maps every pair (u, y) to *true* or *false*.

Definition 4.1 (Verification) Given a model M, with initial states $\mathscr{X}_0$, a time domain $\mathbb{T}$, input domain $\mathscr{U}$ and output domain $\mathscr{Y}$, and a specification φ, the formal verification problem provides a proof that for all $x_0 \in \mathscr{X}_0$, and for all $u \in \mathbb{T}^{\mathscr{U}}$, if $y = M(x_0, u)$, then $\varphi(u, y)$ is *true*.

There are several techniques that have been proposed to solve the verification problem for CPS models. The most popular among these are *reachability analysis* techniques that are based on computing the set of states reachable (usually within a given finite-time horizon) from a given set of initial conditions and for a given set of input signals. In such techniques, a common assumption is that the system state is fully observable (i.e., the output signals are simply the state trajectories of the system). Further, the specification is typically provided as a set of *unsafe states* that should not be reached by the system. We discuss these techniques in Sect. 4.3.

The advantage of techniques based on reachability is that they are highly automatic; however, for systems with nonlinearities and switching behaviors, these techniques may suffer from imprecision. An alternative approach is to use manual insight to propose an *invariant* for the given CPS model. An invariant is a set that is guaranteed to contain the system behaviors for all time. The computational effort is then to automate the invariant generation process (as much as possible) and verify the validity of the system invariant. We discuss such techniques in Sect. 4.4.

In Sect. 4.5, we discuss various specification-driven falsification techniques for CPS models. A falsification problem attempts to provide a refutation to a verification question for a system. Formally, we define falsification as follows.

Definition 4.2 (Falsification) Given a model M, with initial states $\mathscr{X}_0$, a time domain $\mathbb{T}$, input domain $\mathscr{U}$ and output domain $\mathscr{Y}$, and a specification φ, the falsification problem provides a proof that there is some $x_0 \in \mathscr{X}_0$, and some $u \in \mathbb{T}^{\mathscr{U}}$, such that $y = M(x_0, u)$, and $\varphi(u, y)$ is *false*.

Falsification approaches are based on systematically searching for a counterexample to a specification. In Sect. 4.5, we present robustness-guided falsification approaches that use a robustness metric to map properties $\varphi(u, y)$ that provide *true*/*false* interpretation to signals to real-valued interpretations that measure how close a trace comes to satisfying or violating a property.

Finally, in Sect. 4.6, we highlight a significant challenge on the horizon for CPS applications that aspire to become autonomous or semi-autonomous. Developers for such applications are increasingly using AI-based software such as artificial (and deep) neural networks for various aspects such as perception, planning/decision-making, and control. We review some of the key challenges in this domain and summarize some of the recent work seeking to address these challenges.

4.1.1 Motivating Examples

In this section, we describe two motivating examples that illustrate the need for model-based design supported by formal design verification tools.

4.1.1.1 Autonomous Driving

There has been significant recent interest in the ability of vehicles to drive autonomously, i.e., without any intervention by a human driver [28, 77, 100]. The typical software stack for an autonomous vehicle consists of several components: (1) a perception component that processes data about the environment coming through sensors such as a Radar, forward-facing cameras, and LiDAR (light detection and ranging), (2) a decision/planning component that uses the environment models created by the perception component to plan the motion of the vehicle, and (3) a low-level control component that interfaces with the actuators of the vehicle to physically realize the motion plan determined by the planning component. There is ample scope for model-based design of the interfaces between each of these components. In particular, we consider one of the simplest problems for an autonomous vehicle, which is that of regulating its speed. This is based either on a desired speed determined by the high-level motion planner in accordance with the current weather conditions and speed-limit regulations, or based on the speed of the vehicle in front (whichever is lesser). The objective is twofold, if there is a lead car, then the ego car should always maintain a safe following distance from the lead car; otherwise, it should maintain a speed close to that suggested by the high-level motion plan.

An *adaptive cruise controller* (ACC) is a control scheme that seeks to automate the task of choosing the right acceleration for the ego vehicle so as to maintain its safety and performance objectives. Radar-based ACC systems have been implemented in several commercial cars, but continue to be of relevance in the autonomous-driving space, where the sensor inputs are not restricted to Radar. Furthermore, a typical autonomous vehicle has several subsystems that may try to control the longitudinal acceleration of the car (e.g., a controller that attempts to execute a lane-change maneuver, or a controller to execute an emergency stopping

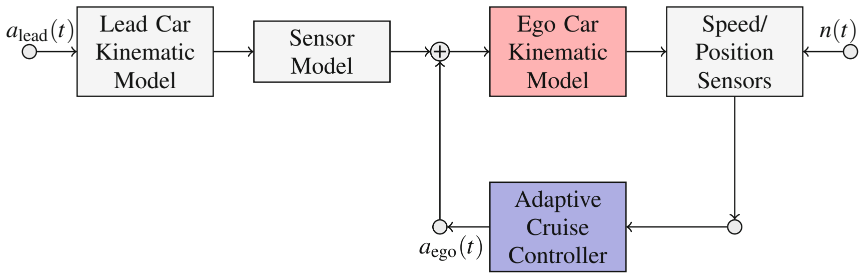

Fig. 4.1 Schematic diagram for an adaptive cruise control system

maneuver). In such cases, it is important that the ACC system is not designed in isolation, but is cognizant of other systems around it.

A schematic model of a typical ACC system is shown in Fig. 4.1. The typical model of the environment is to construct a kinematic model of the lead car (based on Newton's laws of motion), while assuming that the lead car can dynamically change its acceleration (denoted by $a_{\text{lead}}(t)$). The sensor model then captures the quantities in the lead car's motion that can be measured by the ego car. For example, for a Radar-based sensor, this would be the relative distance between the cars and the velocity of the lead car. The kinematic model of the ego car models the effect of the controller and environment inputs on the ego car's motion. We assume that the adaptive cruise controller estimates the ego car's motion through speed sensors (possibly coupled with an odometry-based position computation model). These sensors could have an associated measurement noise (modeled by $n(t)$). Finally, the ACC outputs a control signal (typically the ego car's acceleration, shown as $a_{\text{ego}}(t)$).

Recent work has focused on formal verification and correct-by-construction synthesis of ACC systems. In [102], the authors use quantified dynamic logic to verify the local lane control problem which uses an invariant-based theorem-proving approach. In [104], the authors use reachability analysis for proving safety of ACC systems. On the other hand, in [17, 114], the authors use correct-by-construction approaches using Lyapunov theory and control barrier certificates to automatically obtain safe implementations of ACC systems. While these studies have demonstrated the power of formal verification, more work can be done in formalizing behavioral specifications for an ACC system, and then applying the different techniques considered in this chapter to prove correctness of such a system.

4.1.1.2 Artificial Pancreas

Type-1 diabetes is characterized by the inability to regulate the blood glucose (BG) levels within an *euglycemic range* [70, 180] mg/dl in the human body due to the absence of insulin, a hormone that is responsible for reducing BG levels. The treatment is to externally replace the lost insulin. However, this insulin must

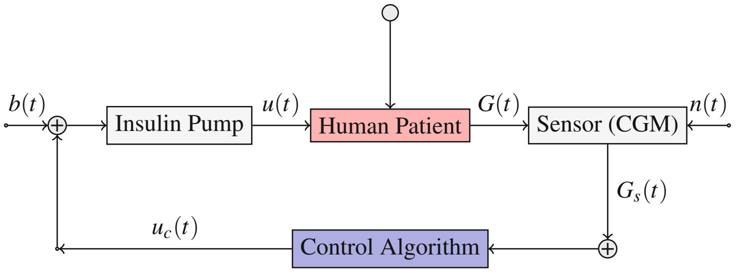

Fig. 4.2 Overview of the key components of an artificial pancreas control system. $b(t)$: external insulin, $u(t)$: insulin infused, $G(t)$: BG level, $n(t)$: measurement error, $G_s(t)$: sensed glucose level, $u_c(t)$: insulin infusion commanded

be delivered to compensate increases in blood glucose levels due to meals or endogenous glucose production by the liver. Too much insulin can expose the patient to the risk of hypoglycemia wherein the blood glucose levels fall below 70 mg/dl, whereas too little insulin causes high BG levels due to *hyperglycemia* wherein BG levels rise above 180 mg/dl leading to long-term damage to kidneys, eyes, heart, and the peripheral nerves. In order for insulin to be delivered, it is often infused subcutaneously through an insulin pump—a device that is programmed to deliver a constant low rate of insulin, known as *basal insulin*, or a larger *bolus* of insulin in advance of a meal or to treat high BG values [38, 138].

The artificial pancreas project seeks to partially or fully automate the delivery of insulin by combining a continuous glucose sensor which periodically senses BG levels subcutaneously, an insulin pump that delivers insulin, and a closed-loop control algorithm that uses inputs from the CGM and the user to control BG levels to a target value [48, 86, 143]. A schematic diagram is shown in Fig. 4.2.

Because of the severe risks posed by hypo- and hyperglycemia, AP devices are safety critical. They need to be used by patients 24/7/365 without expert supervision, though they are capable of serious harm to the patient. As a result, their design and implementation require careful consideration and thus form an ideal target for formal methods/automated reasoning approaches.

Notable attempts to verify medical devices include work on pacemakers and implantable cardiac defibrillators (ICDs). This started with physiological models of excitable cells in the heart [119], leading to approaches that employ these models to test closed-loop systems [88, 117].

Lee and collaborators studied a PID-based closed-loop system meant for intra-operative use in patients [39], using the dReal SMT solver [76] to prove safety for a range of parameters and controller gains. Other approaches to verifying artificial pancreas systems have relied on *falsification*, using temporal logic robustness [56, 68], and incorporated in tools such as S-TaLiRo [1, 111] and Breach [55]. Sankaranarayanan et al. have studied the use of falsification techniques for verifying closed-loop control systems for the AP [34]. Their initial work investigated a PID controller proposed by Steil et al. [140, 141] based on published descriptions of the control system available. Another recent study by Sankaranarayanan et al. [135] was performed to test a predictive pump shutoff controller designed by Cameron et al. [35] that has undergone outpatient clinical trials, recently [103]. Recently, Kushner et al. studied a personalized approach to analyzing controller parameters using data-driven models [96]. These studies have demonstrated the ability of formal approaches to verification and falsification to provide important behavioral specifications, combine a variety of models for every aspect of the artificial pancreas, and prove/falsify important properties.

4.2 Mathematical Models and Specifications

"*All models are wrong but some are useful*"–George E. Box [32].

Verifying properties of a system requires mathematical models and formal specifications. In this section, we briefly describe the varieties of mathematical models and specification formalisms that are used in cyber-physical systems (CPS). As mentioned earlier, CPS combine a variety of heterogeneous components, including physical (mechanical, electrical, chemical, and biological) systems, electronic (analog and digital circuits), and software components. Furthermore, they are subject to a wide variety of input stimuli from the environment that can range from disturbances such as wind to inputs from human operators. As a result, mathematical modeling is a key first step in order to provide a framework wherein we can define key properties of the system in a formal manner. A variety of mathematical models are employed in CPS, including ordinary and partial differential equation models for physical and biological components, automata-based models for digital electronic components, and software. Finally, stochastic models capture the behavior of disturbances such as the wind, noise, measurement errors, component failures, or mistakes made by human operators.

4.2.1 Mathematical Models

Table 4.1 lists some commonly employed mathematical models and the type of components that they are used to model. These models range from continuous-time models such as ODEs and SDEs to discrete time models such as finite and extended state machines. Each of these models have been well studied by communities of mathematicians, physicists, and engineers.

However, the challenge of CPS applications arises in the combination of multiple modeling paradigms within the same system. Due to this combination, the modeling

Table 4.1 Commonly employed mathematical models for various aspects of a CPS

Model type	Component type	Examples
Ordinary differential equation (ODE) [107]	Physical/analog	Vehicle body, engine speed, drug pharmacokinetics
Partial differential equation (PDE)	Physical continuum	Fluid flow, electromagnetic field, fabric, paper
Finite state automata [137]	Software/electronics	Switching logic, relays, digital circuits, software
Extended state automata	Software	Software controllers
Timed automata [12]	Real-time software	Schedulers, watchdog timers
Markov chains [115]	Disturbances/failures	Component failures, job arrivals
Stochastic differential equation (SDE) [116]	Disturbances	Wind disturbances, measurement noise

of CPS has focused on the combination of discrete-time models such as automata and continuous models such as ODEs to yield *hybrid dynamical systems* that are capable of continuous-time evolution in conjunction with discrete mode transitions.

4.2.1.1 Hybrid Systems

Hybrid systems model processes that combine the continuous evolution of state over time with discrete jumps that can instantaneously change the state as well as the future course of the dynamics. Such systems arise from a variety of sources: physical systems involving contact forces, biological systems, controlled systems with switched or periodically updated control action, and in general, software-driven control systems. The field of hybrid systems evolved historically from two complementary sources that included *computer scientists* studying languages and formalisms defined by the interaction of automata with physical process [9, 83]; *control theorists* extending previously well studied continuous models to include discrete switching actions [33, 139]. Labinaz et al. present an early survey that touches upon the historical development of hybrid systems [97].

The *hybrid automaton model* was proposed to provide a conceptual model for expressing hybrid systems [10]. Figure 4.3 shows an example of a hybrid automaton model expressing a temperature controller for a house that is heated by turning on/off a source of heated/cooled air. The automaton has two modes ON: representing the dynamics of the room temperature when the heater is turned on and OFF: representing the dynamics when the heat is turned off.

Definition 4.3 (Hybrid Automaton) Given a vector of system variables $\mathbf{x} \in X$, control inputs $\mathbf{u} \in U$, and disturbances $\mathbf{w} \in W$, a hybrid automaton $\mathscr{H} : \langle L, E, I, F, G, R \rangle$ consists of the following components:

1. A finite set of *modes* $L : \{\ell_1, \ldots, \ell_n\}$ and *transitions* that form edges between locations $E \subseteq L \times L$,

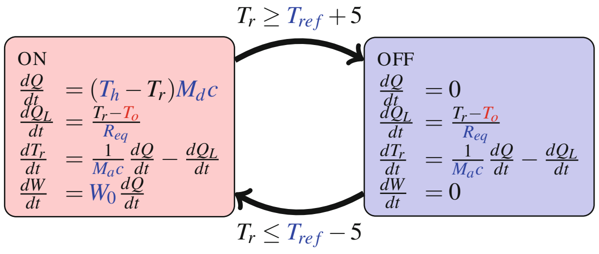
$T_r \geq T_{ref} + 5$
ON
$\frac{dQ}{dt} = (T_h - T_r) M_d c$
$\frac{dQ_L}{dt} = \frac{T_r - T_o}{R_{eq}}$
$\frac{dT_r}{dt} = \frac{1}{M_a c}\frac{dQ}{dt} - \frac{dQ_L}{dt}$
$\frac{dW}{dt} = W_0 \frac{dQ}{dt}$
OFF
$\frac{dQ}{dt} = 0$
$\frac{dQ_L}{dt} = \frac{T_r - T_o}{R_{eq}}$
$\frac{dT_r}{dt} = \frac{1}{M_a c}\frac{dQ}{dt} - \frac{dQ_L}{dt}$
$\frac{dW}{dt} = 0$
$T_r \leq T_{ref} - 5$

Fig. 4.3 Hybrid automaton model for the house heating demo example. The state variables include Q, the heat flowing in to the room, Q_L, the heat lost to the outside, T_r, the room temperature, and W, the total heating cost. The parameters are shown in blue and include M_a the mass of air inside the house, R_{eq} the "thermal resistance" equivalent of the house, M_d, the air flow rate through the heater, c the heat capacity of air at constant pressure, W_0 is cost per unit heat, and T_{ref} the desired reference temperature. The disturbance input is T_o the outside air temperature, shown in red

2. A map I that associates each location $\ell \in L$, a location invariant $I_\ell \subseteq X$,
3. A map F that associates each mode $\ell_i \in L$ with a vector field $F_l : X \times U \times W \mapsto (\mathscr{T} X)$ that forms the RHS of the ODE: $\frac{d\mathbf{x}}{dt} = F_{\ell_i}(\mathbf{x}, \mathbf{u}, \mathbf{w})$. The function F_ℓ is assumed to be Lipschitz continuous over $\mathbf{x}$ and continuous over the remaining inputs for all $\ell \in L$.
4. A guard map G that associates with a guard set $G_{(\ell_1,\ell_2)}$ with each transition, and
5. A reset map R that associates each transition with an update function $R_{(\ell_1,\ell_2)} : I_{\ell_1} \mapsto I_{\ell_2}$.

The initial condition of a hybrid automaton is given by a location $\ell_0 \in L$ and an initial state $\mathbf{x}_0 \in I_{\ell_0}$. Let $\mathbf{u} : [0, T] \mapsto U$ be a control input signal and $\mathbf{w} : [0, T] \mapsto W$ be a disturbance input. The state of a hybrid automaton is given by a pair $(\ell, \mathbf{x})$ where $\ell \in L$ and $\mathbf{x}$ is a state belonging to the invariant set I_ℓ associated with the mode ℓ. The execution of a hybrid automaton over a time horizon T (can be finite or infinite $T = \infty$) is given by a sequence of *flows* and *jumps*:

- A flow $(\ell, \mathbf{x}, \tau) \rightsquigarrow (\ell, \mathbf{x}', \tau + \delta)$ for $\delta \geq 0$ is a solution to the ODE $\frac{d\mathbf{x}}{dt} = F_\ell(\mathbf{x}, \mathbf{u}, \mathbf{w})$ starting from the initial condition $t_0 = \tau, \mathbf{x}(\tau) = \mathbf{x}$ with $\mathbf{u}(\cdot)$ as the signal $\mathbf{u}(t)$ with $t \in [\tau, \tau + \delta)$ and likewise, $\mathbf{w}$ as the signal $\mathbf{w}(t)$ over $t \in [\tau, \tau + \delta)$. This trajectory is uniquely defined since F_ℓ is Lipschitz. Finally, $\mathbf{x}'$ is the state $\mathbf{x}(\tau + \delta)$.
- A jump $(\ell, \mathbf{x}, \tau) \rightarrow (\ell', \mathbf{x}', \tau)$ is an instantaneous transition from mode ℓ to ℓ' wherein $(\ell, \ell') \in E$, and $\mathbf{x} \in G_{(\ell,\ell')}$ must belong to the guard set of the transition. The state $\mathbf{x}' = R_{(\ell,\ell')}(\mathbf{x})$ is obtained by applying the reset map corresponding to the transition (ℓ, ℓ') to the state $\mathbf{x}$.

An execution trace of the hybrid automaton yields a hybrid time trajectory comprised of flows and jumps starting from the initial state $(\ell_0, \mathbf{x}_0)$ at time 0.

$$(\ell_0, \mathbf{x}_0, 0) \rightsquigarrow (\ell_1, \mathbf{x}_1, t_1) \rightarrow (\ell_1', \mathbf{x}_1', t_1) \rightsquigarrow (\ell_2, \mathbf{x}_2, t_2) \rightarrow \cdots$$

4.2.2 *Specifications*

In the formal methods literature, the term *specifications* is often used to describe the expected behavior of the overall system. Specifications can express properties defined over several behaviors of the system (e.g., the average energy consumption, mean time to failure, etc.), and can also express properties over individual system executions (e.g., the value of the overshoot is less than 10% of the reference value, the response time is at most 5 s, etc.). The first class of properties (that are defined over several system behaviors) are called *hyperproperties* [47]. The second class of properties are *trace* properties, i.e., given a (discrete or continuous) trace representing a system behavior, we can check the satisfaction or violation of such a property on this trace.

Types of Properties In hyperproperties, we can further make a distinction between *statistical* hyperproperties, i.e., properties that reason about statistical aspects of the system (such as average energy consumption, mean time to failure, etc.), and *relational* hyperproperties. There has been limited work on estimating statistical properties of CPS models [2], but not much work has been done to verify or falsify statistical hyperproperties. Relational hyperproperties are gaining popularity for expressing security and privacy properties such as information leakage, robust I/O behavior, noninterference, noninference, etc. [47, 112]. For example, consider a potential side-channel power attack: there exists a system behavior where for the input u the signal representing the magnitude of power (say y) that exceeds the value c for τ seconds, but for all other inputs u' *near* u, the corresponding y' is always below some value d s.t. $d < c$. There has not been much work on verification of relational hyperproperties for CPS models. Thus, as verification or testing for hyperproperties is a nascent field with limited results for narrow subproblems [29, 30, 53, 70]. Hence, we do not discuss this aspect in detail in this chapter, but instead focus on verification and falsification for trace properties.

4.2.2.1 Temporal Logics for Trace Properties

There are several possible ways in which trace-level properties can be expressed and checked. Many industrial practitioners often write custom programs in their preferred programming language to check a trace-level property. These programs are also known as *property monitors*. An *offline* monitor checks the satisfaction of a *finite-time* trace-level property by a given finite-time system execution after the execution has terminated. On the other hand, an *online* monitor continuously checks the satisfaction or violation of the property as the system runs. In an MBD framework, the same terminology applies to simulations of system behavior: offline monitoring requires the simulation to have terminated.

Having customized programs for property monitors can pose challenges in terms of interpretability and maintainability, and is prone to manual programming errors. An elegant alternative is to use a suitable logical formalism to describe the desired trace-level property. One such formalism is that of linear temporal logic (LTL). LTL was introduced in the late 1970s [123] to reason about the temporal behaviors of reactive systems, i.e., input-output systems with Boolean, discrete-time signals. CPS rarely have discrete-valued, discrete-time behaviors, as the physical components in a CPS have real-valued behaviors that evolve continuously in time. To reason about such systems, later, temporal logics such as timed propositional temporal logic (TPTL) [14], the duration calculus [37], and metric temporal logic (MTL) [94] were introduced to deal with dense-time system executions. These logics required first creating a set of atomic Boolean predicates over signals, and then introduced formulas that contained temporal operators that could be interpreted over dense time.

Signal Temporal Logic (STL) STL [106] was proposed in the context of analog and mixed-signal circuits as a specification language for expressing constraints on real-valued signals directly in the formula expressing the property of interest. Let $\mathbf{x}$ be an n-dimensional signal representing the system execution over some finite time, and for simplicity, let the codomain of this variable be $\mathbb{R}^n$. Without loss of generality, these predicates can be reduced to the form $\mu = f(\mathbf{x}) \sim c$, where f is a scalar-valued function from $\mathbb{R}^n$ to $\mathbb{R}$.

Temporal formulas are formed using temporal operators, "always" (denoted as $\mathbf{G}$), "eventually" (denoted as $\mathbf{F}$) and "until" (denoted as $\mathbf{U}$). Each temporal operator is indexed by intervals of the form (a, b), $(a, b]$, $[a, b)$, $[a, b]$, (a, ∞) or $[a, \infty)$ where each of a, b is a nonnegative real-valued constant. If I is an interval, then an STL formula is written using the following grammar:

$$\begin{array}{lll} \varphi := & \mathit{true} & \\ & \mid \ \mu & \text{atomicproposition} \\ & \mid \ \neg\varphi & \text{negation} \\ & \mid \ \varphi_1 \wedge \varphi_2 & \text{conjunction} \\ & \mid \ \varphi_1 \, \mathbf{U}_I \, \varphi_2 & \text{untiloperator} \end{array}$$

The always and eventually operators are defined as special cases of the until operator as follows: $\mathbf{G}_I\varphi \triangleq \neg\mathbf{F}_I\neg\varphi$, $\mathbf{F}_I\varphi \triangleq \mathit{true}\,\mathbf{U}_I\,\varphi$. When the interval I is omitted for the until operator, we take it as the default interval of $[0, +\infty)$. The semantics of STL formulas are defined informally through examples as follows.

Example 4.1 The signal $\mathbf{x}$ satisfies an atomic predicate $f(\mathbf{x}) > 10$ at time t (where $t \geq 0$) if the value of $f(\mathbf{x}(t))$ at time t is greater than 10.

The signal x satisfies $\varphi = \mathbf{G}_{[0,2)}\ (x > -1)$ if for all time $0 \leq t < 2$, $x(t) > -1$.

The signal x_1 satisfies $\varphi = \mathbf{F}_{[1,2)}\ x_1 > 0.4$ iff there exists time t such that $1 \leq t < 2$ and $x_1(t) > 0.4$.

The signal $\mathbf{x} = (x_1, x_2)$ over two-dimensional space satisfies the formula $\varphi = (x_1 > 10)\ \mathbf{U}_{[2.3,4.5]}\ (x_2 < 1)$ iff there is some time u where $2.3 \leq u \leq 4.5$ and $x_2(u) < 1$, and for all time v in $[2.3, u)$, $x_1(u)$ is greater than 10.

We formally define the semantics of STL as follows:

Definition 4.4 (STL Semantics) STL semantics are defined in terms of the *satisfaction* operator $\models$, for a given signal $\mathbf{x}$ at each time t as follows:

$$
\begin{array}{lll}
(\mathbf{x}, t) \models \mu & \iff & \mathbf{x}(t) \models \mu \\
(\mathbf{x}, t) \models \neg\varphi & \iff & (\mathbf{x}, t) \not\models \varphi \\
(\mathbf{x}, t) \models \varphi_1 \wedge \varphi_2 & \iff & (\mathbf{x}, t) \models \varphi_1 \text{ and } (\mathbf{x}, t) \models \varphi_2 \\
(\mathbf{x}, t) \models \mathbf{G}_{[a,b]}\varphi & \iff & \forall t' \in [t+a, t+b](\mathbf{x}, t') \models \varphi \\
(\mathbf{x}, t) \models \mathbf{F}_{[a,b]}\varphi & \iff & \exists t' \in [t+a, t+b](\mathbf{x}, t') \models \varphi \\
(\mathbf{x}, t) \models \varphi_1 \,\mathbf{U}_{[a,b]}\, \varphi_2 & \iff & \exists t' \in [t+a, t+b] \; s.t. \\
 & & \quad (\mathbf{x}, t') \models \varphi_2 \text{ and} \\
 & & \quad \forall t'' \in (t, t'), (\mathbf{x}, t'') \models \varphi_1
\end{array}
$$

Beyond STL Recently, there have been several efforts to consider alternatives to STL to address specific properties that may be cumbersome to express in STL, or inexpressible in STL. Timed regular expressions (TRE) first introduced in 2002 [21] allow expressing localized patterns in CPS behaviors. An efficient monitoring procedure has been proposed for TREs in [150], and an implementation of this procedure is available in the Montre tool [149]. Quantitative regular expressions (QREs) [13, 16] is yet another modeling and programming abstraction for specifying complex numerical queries over data streams. These have been used for analyzing complex behaviors such as arrhythmia in cardiac signals [4].

Finally, differential dynamic logic [120] is a logic for specifying and verifying correctness of hybrid systems. The language allows specifying hybrid systems operationally as hybrid programs and uses automated deduction-based theorem proving tools (such as KeyMaera and its extensions [74, 122]) to verifying program correctness. A key difference between deductive techniques and those that we consider in this chapter is that deductive techniques often require manual intervention in the form of lemmas and proof strategy selection when the automated theorem prover fails to prove program correctness. We omit such techniques from this chapter, and the interested reader can find an extensive treatment in [121].

4.3 Reachability Analysis

Reachability analysis asks whether a hybrid system starting from a set of initial states X_0 can reach any state in a given target set U. The problem is of fundamental importance to hybrid systems since the target set U often describes dangerous states which we wish to avoid reaching during an execution of the system.

Example 4.2 Consider the house heating system shown in Fig. 4.3. It is considered dangerous if the temperature of the house falls below 10 centigrade, while the system continues to be operational and the outside temperature behaves “reasonably”:

that is, it must be in the range $[-20, 50]$ °C and cannot increase/decrease more than 5 °C/h. Let us assume an initial state with $T_r = 27$. Is there a scenario in which the value of $T_r \leq 10$ is possibly under the constraints on the behavior of the outside temperature? Here, the target unsafe set is $U : \{(\ell, Q, Q_L, T_r, W) \mid T_r \leq 10\}$.

Another safety property asks whether it is possible for $T \leq T_{\text{ref}} - 5$ and simultaneously, the heater is in the "OFF" state. Here, the unsafe set is $V : \{(\ell, Q, Q_L, T_r, W) \mid T \leq T_{\text{ref}} - 5 \ \wedge\ \ell = \text{OFF}\}$.

Reachability analysis has been studied using a variety of approaches, and for various restrictions on the hybrid automaton model.

4.3.1 Decidability of Reachability

First, it is well known that the reachability problem is undecidable even for simple cases. For instance, in the absence of hybrid dynamics, reachability is undecidable for polynomial ODEs involving 3 or more state variables [82]. Furthermore, for linear dynamical systems, it is known that reachability of a single target state $\mathbf{y}$ from a single given initial state $\mathbf{x}_0$ is decidable. However, the reachability problem of a hyperplane target from a single state initial set (known as the Skolem–Pisot problem) is open. Recent result by Chonev connects the undecidability of this problem to a well-known and open number theoretic conjecture called the *Schaunel* conjecture [44]. Alur and Dill showed that the reachability problem is decidable for timed automata that can be seen as hybrid automata whose continuous variables are all *clocks* with dynamics $\frac{dT}{dt} = 1$. Furthermore, the guard conditions are restricted to comparing clocks with fixed constants, and resets are limited to setting clocks to fixed constant values. The result relies on an *untiming* construction through the region abstraction that produces a finite state automaton which is bisimulation equivalent to the original infinite state timed automaton [12]. However, Henzinger et al. show that if we allow "stopwatches," i.e., clocks that can be stopped by setting $\frac{dT}{dt} = 0$ in certain modes, even the presence of a single stopwatch in a timed automaton model renders the reachability problem undecidable [84]. The timed automaton model can be generalized to rectangular hybrid dynamics that allows the derivative of each variable x_i to lie within an interval $\frac{dx_i}{dt} \in [l_i^{(m)}, u_i^{(m)}]$ in each mode m. Henzinger et al. [84] show that the reachability problem is decidable for *initialized* rectangular hybrid automata that adds the following constraint: for every transition τ from mode m to $\hat{m}$, if the dynamics for $\frac{dx_i}{dt}$ changes going from m to $\hat{m}$, then the variable x_i must be reset to constant value by τ. However, failing this condition, the problem is undecidable, in general. Asarin et al. consider polyhedral hybrid systems that are defined by partitioning the state space into polyhedral regions defining modes and associating each polyhedral region with a mode m and a corresponding constant differential equation $\frac{dx_i}{dt} = c_i^{(m)}$ [23]. Transitions happen when the system moves from one polyhedral region to another in this model. While the reachability problem is decidable for 2D (planar) systems, it is undecidable for

Table 4.2 Summary of a few results establishing decidability/undecidability of reachability for hybrid systems

System	Outcome	Description
Timed automata [12]	Decidable	$\frac{dx_i}{dt} = 1$ for all x_i and all modes, guards $x_i\{\leq, \geq, =\}c$ and resets $x_i := c$
Stopwatch automata [84]	Undecidable	Timed automata + at least one stopwatch with $\frac{dx_i}{dt} = 0$ allowed in some modes
Initialized rectangular automata [83, 84]	Decidable	Rectangular dynamics $\frac{dx_i}{dt} \in [l_i, u_i]$ for each x_i and mode, guards + resets as in timed automata. Transition between different dynamics should reinitialize a variable
Polyhedral hybrid automata [23]	Undecidable	Decidable for 2 or fewer state variables
O-minimal hybrid automata [98]	Decidable	Automata whose guards, reset maps, and flows can be defined in an O-minimal logical theory

systems involving 3 or more variables. Table 4.2 summarizes some of the significant results on decidability/undecidability of reachability for various classes of hybrid systems.

Understanding the boundary between decidable and undecidable subclasses has been an active area of investigation with some open problems. However, early results showed that seemingly simple hybrid automata models can exhibit a high degree of complexity in terms of their behaviors. As a result, the focus has gradually shifted from finding new decidable classes to finding practical algorithms that can be useful to analyze models of interest to practitioners, even if the overall problem is known to be undecidable.

4.3.2 *Reachability Using Over-Approximations*

As discussed previously, the problem of deciding questions of reachability is undecidable. However, for many practical systems, the problem of reachability analysis can be resolved by computing over-approximations of the reachable set of states starting from the initial set X_0, or alternatively, by computing over-approximations of the backward reachable set starting from the unsafe/target set U. This is pictorially illustrated in Fig. 4.4. Over-approximations can be obtained for a *finite* time horizon if the value of T is finite, or an *infinite* time horizon if $T = \infty$. Naturally, infinite-time horizon approximations are more complicated and approached using deductive methods discussed in subsequent sections. The rest of this section focuses, for the most part, on finite-time horizon reachability analysis.

Let $S \subseteq X$ be a subset of states of a system and X_0 be the initial set. We say that S is a (forward) over-approximation for a time interval $[0, T)$ iff for any initial state $\mathbf{x}(0) \in X_0$, any state $\mathbf{x}(t)$ reachable from $\mathbf{x}(0)$ at time $t \in [0, T)$ belongs to S.

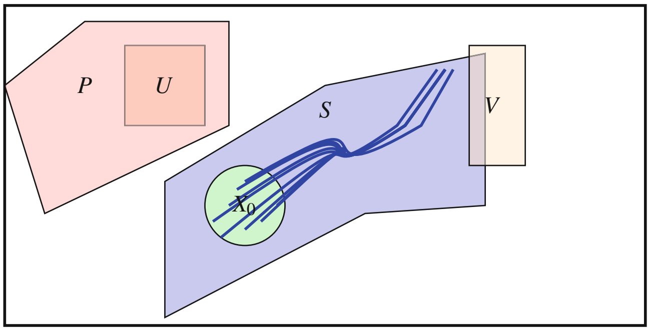

Fig. 4.4 S: Over-approximation of reachable set of states includes the initial condition X_0 and all states reached by trajectories starting from X_0. P: Backward-over approximation containing all states that can reach U. The set of states U is proven unreachable since $U \cap S$ is empty, or alternatively, However, the set of states V may or may not be reachable since $V \cap S$ is not empty

Using the forward over-approximation S, we may conclude that U is unreachable if $U \cap S = \emptyset$.

Alternatively, we can prove unreachability by computing a set of backward reachable states $P \subseteq X$ such that $U \subseteq P$ for the target set and every trajectory of the system $\mathbf{x}(\cdot)$ such that $\mathbf{x}(t) \in U$ at time $t \in [0, T)$ must satisfy $\mathbf{x}(0) \in P$. If $P \cap X_0 = \emptyset$, we may now conclude that no run of the system starting from X_0 may reach U within the given time horizon. Figure 4.4 illustrates how a backward reachable set can be used to prove unreachability, as well.

4.3.2.1 Approximate Reachability: Overview

We will now discuss how reachability analysis works at the high level, focusing first on computing over-approximations of forward reachable states starting from the initial state X_0 and an initial mode ℓ_0 of the hybrid system. The approach is based on *symbolic model checking*, wherein a set of reachable states is iteratively computed by repeatedly applying the *post-condition* operator to the initial set of states. The post-condition operator applied to a set of states S captures all the states reachable from S in a single "computational" step. Let $\mathsf{post}(S)$ denote the post-condition of S. Thus, we would normally compute

$$X_0 \cup \mathsf{post}(X_0) \cup \mathsf{post}^2(X_0) \cup \cdots .$$

However, there are three core problems with this approach:

1. Hybrid systems combine the continuous evolution of state variables with discrete transitions. There is no natural notion of a single discrete computational step.
2. The sets $\mathsf{post}^k(X_0)$ become increasingly complicated to represent in a computer, making the process prohibitively expensive.

3. The iteration does not terminate in finitely many steps for most systems, and therefore, the approach may not terminate.

The other alternative is to perform a *backward* iteration, starting from the unsafe set of states and iterating the weakest *pre-condition* operator. The precondition operator applied to a set of states S captures all those states that will reach S in one computational step. Let $\mathsf{pre}(S)$ denote the weakest precondition operator applied to a set S. Thus, we would normally compute

$$U \cup \mathsf{pre}(U) \cup \mathsf{pre}^2(U) \cup \cdots .$$

Once again, the same three problems we encountered for post-conditions arise for preconditions as well.

Reachability algorithms overcome the three key problems mentioned above through two important and closely intertwined ideas: (1) *abstraction* of the hybrid system by a simpler model; and (2) *abstract* (over-approximate) representation of sets of states by geometric primitives such as rectangles, polyhedra, ellipsoids, zonotopes, and Taylor models.

4.3.2.2 Abstractions

A system abstraction seeks to replace a given hybrid automaton $\mathscr{S}$ by another finite or infinite state system $\mathscr{T}$ over the same state space and set of modes as $\mathscr{S}$, such that *every trajectory of* $\mathscr{S}$ *is also a trajectory of* $\mathscr{T}$. In this case, we will write $\mathscr{S} \succeq \mathscr{T}$. Note, however, that $\mathscr{T}$ may have more trajectories that are not trajectories of $\mathscr{S}$. It is easy to show that any reachable state of $\mathscr{S}$ starting from a given state X_0 is also reachable in $\mathscr{T}$ (starting from a suitable superset of X_0).

Early approaches considered *finite state* abstractions that transform a given hybrid automaton into a finite state machine which simulates the original system, or in special cases, such as timed automata or initialized rectangular automata, exhibits a stronger connection through bisimulation relations [15, 24]. However, most systems of interest have been observed not to have finite bisimulation quotients. To circumvent this, Girard and Pappas consider the notion of an *approximate* bisimulation relation that is defined by means of a comparison metric between states of the two systems so that as the systems evolve in time starting from related initial states, the distance decays over time [79]. The notion of approximate bisimulation relations expands the class of systems for which we may find suitable finite state abstractions with some property preservation guarantees. Nevertheless, it remains the case that finding finite (approximate) bisimulation quotients is rare and seldom feasible for practical systems. Other approaches for finding finite abstractions have employed the use of predicate abstractions with counterexample guided refinements [11]. While the approach can perform well if the right set of predicates were to be provided, the problem of deriving such a set of predicates is often hard in practice. Furthermore, the refinement loop may often generate a large number of predicates making the finite state abstraction prohibitively expensive. More recently, hybridization approaches have investigated the abstraction of more complex dynamics such as nonlinear ODEs, linear hybrid systems by simpler dynamics such as rectangular automata [22, 50, 125, 132]. On one hand, these approaches can provide tradeoffs between the accuracy of the abstraction and its size. On the other hand, these approaches can also suffer from the curse of dimensionality since they rely on decomposing the state space into small compact regions in order to bound the error between the original system and the abstraction.

A related class of abstractions seeks to eliminate continuous dynamics by replacing the ODEs by relational models that relate a state $\mathbf{x}$ and a future reachable state $\mathbf{x}'$. Building such relational models can then allow off-the-shelf tools for model checking infinite state discrete systems to tackle the verification problem. The idea of relationalization, though implicit in earlier works such as Podelski and Wagner [124], was first formalized by Tiwari and Sankaranarayanan under the notion of relations that abstracted time away as well as relations that captured change in state over a fixed time step [136, 154]. Subsequent work studied various ways of constructing these relations that tracked time explicitly as "time-aware" relations [109]. Recently, Chen et al. explored the construction of these relations for nonlinear systems [42]. One of the key drawbacks for existing methods lies in the lack of approaches to refine these relations once they are constructed. A related issue lies in the tradeoff between constructing a coarse but simple relation versus a more complex and less conservative approximation. Approaches that can construct "multi-scale" relations that selectively refine interesting parts of the relation remain unexplored at the time of writing.

4.3.2.3 Flowpipe Computation

Flowpipe computation approaches rely primarily on computing reachable sets by approximating the time trajectories of the system rather than abstracting the system itself. A large variety of flowpipe computation approaches have been proposed in the literature, and many proposed techniques are supported by tools for experimental validation. Some of these tools are specialized to linear hybrid systems, while others tackle a larger class of nonlinear systems. Most flowpipe construction methods are instances of the forward reachability computation using the post-condition operator presented previously. However, in order to extend this scheme to hybrid systems, it is important to consider four important aspects (illustrated in Fig. 4.5).

1. A systematic way to represent sets of continuous states. Since not all sets are representable inside a computer, common representations include intervals, convex polyhedra, ellipsoids, zonotopes, support vectors, and Taylor models.
2. Once a representation is chosen, we need to compute sets of reachable states for given nonlinear dynamics inside a mode. This operation has been variously termed "time elapse," "continuous post-condition," or "continuous image computation" in the literature.

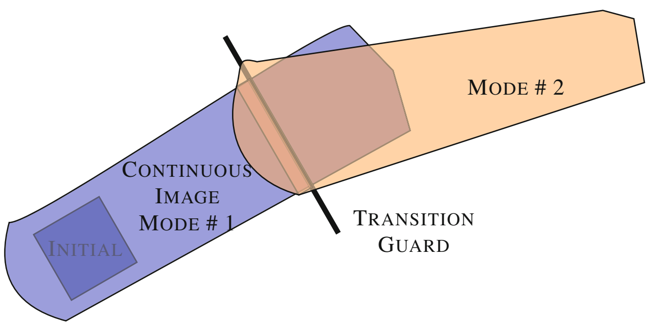

Fig. 4.5 Illustration of basic primitives for flowpipe computation. Starting from initial set in mode 1, we perform a continuous image computation for a given time horizon. Next, we compute states from which a transition to a different mode (mode 2) is possible. From these states, we compute the reachable states for mode 2 shown in orange

3. The effect of a discrete transition must be computed. This operation is called "discrete post" or "discrete image" computation.
4. Finally, the primitives mentioned above must be integrated into a model checking scheme that employs them in order to compute the reachable set estimation for the system as a whole. To this end, operations such as subsumption checks, aggregation, simplification, and extrapolation are often used.

The fundamental scheme of performing forward reachability using a combination of continuous and discrete image computation with specialized operators has been carried out through a variety of approaches, which are summarized in Table 4.3.

Table 4.3 presents an overview of selected approaches-based classified in terms of the representations used for sets, and the type of models handled by the approach. As we note in the table, there has been significant recent work in scaling up the reachability analysis of linear ODEs to millions of variables [25], linear hybrid systems to many hundreds of variables [73], and nonlinear systems up to a few tens of variables (assuming nonchaotic and nonstiff ODEs) [7, 41]. Furthermore, a variety of recent tricks including decomposition of a monolithic model into smaller submodels that can be exploited by the reachability analysis [43]. However, significant variability in performance is seen across models. Furthermore, many of the approaches have numerous tunable parameters that need to be carefully adjusted for each model to obtain optimal performance. Another important drawback lies in the lack of support for richer models of hybrid systems that can incorporate features such as lookup tables, gain scheduling, predictive models, and learning-enabled loops involving neural networks. Supporting these features remains the subject of ongoing research at the time of writing this survey.

Table 4.3 Reachability analysis approaches using flowpipe construction at a glance

Reference, *Representation*, and Dynamics	Remarks
Krogh et al. [45, 46], *Polyhedra*, NLHybrid	Precise flowpipes for linear systems. Uses numerical optimization for nonlinear systems. Builds abstract finite state model for checking
Dang et al. [31, 49], *Orthogonal Polyhedra*, LHybrid	Introduced face lifting algorithm for computing reachable sets
Kurzhanski and Varaiya [95], *Ellipsoids*, LODE	Uses ellipsoidal calculus and introduced the idea of support vectors. Handling of discrete transitions requires approximations due to ellipsoid–hyperplane intersections
Mitchell and Tomlin [108], Level Sets, *NLHybrid*	Uses Hamilton–Jacobi PDEs solved using state-space discretization. Solves viability problems (computation of control and reachability)
Girard [78], *Zonotopes*, LHybrid	Efficient image computation for continuous dynamics. Handling of discrete transitions remains problematic similar to ellipsoids. Available as part of Spaceex tool
Frehse et al. [73], *Support Functions*, LHybrid	Efficient image computation and handling of discrete transitions. Implemented in tool SpaceEx
Berz and Makino [27, 105], *Taylor Models*, NLODE	No handling of discrete transitions. Introduced higher-order interval methods for guaranteed ODE integration
Chen et al. [40, 41], *Taylor Models*, NLHybrid	Extends techniques from Berz et al. with handling of discrete transitions
Althoff et al. [7, 8], *Multiple*, NLHybrid	Combination of multiple set valued representations including nonlinear zonotopes, matrix zonotopes, and Taylor models for nonlinear hybrid systems reachability analysis
Bak and Duggirala [25], *Polyhedron*, LODE	Using simulations to implicitly compute reachable sets and resolve safety properties. Shown to scale beyond hundreds of thousands of state variables

NLHybrid: Nonlinear hybrid, NLODE: Nonlinear ODEs with continuous RHS, LODE: Linear ODEs, LHybrid: Linear hybrid

4.3.2.4 Constraint Solvers and Reachability

Another approach relies on using constraint solvers for estimating reachable sets that can be used to prove properties of interest. This approach essentially integrates many of the ideas summarized thus far naturally into a constraint-solving framework. The approach has been termed the SAT-modulo ODE approach, originated in the work by Ratschan and She [129, 130], Ratschan et al. [72], incorporated into tools such as HySAT [85]. More recently, the approach was formalized by Gao et al. into *delta-decision* procedures for proving properties of hybrid systems [76]. The key idea is to provide procedures that can either conclude that a system does not satisfy a property or that the system under a bounded perturbation violates a perturbed property, under a well-defined perturbation model. A similar idea is presented independently by Ratschan wherein termination of the reachability analysis is guaranteed under the condition of robust safety wherein a bounded perturbation of the system continues to satisfy the safety property in question [130].

4.3.2.5 Simulation-Guided Reachability Analysis

A significantly different approach for estimating reachable states relies on using simulations coupled with user-provided annotations. The main idea is to obtain a simulation trajectory and to bloat the trajectory in such a way that for each initial state included in the bloated trajectory, the trajectory beginning at this initial state is also included in the bloated trajectory. Such a bloated trajectory is also known as a *reach tube*. The first idea to compute reach tubes was by exploiting the sensitivity of the numerical solutions of an ODE to perturbations in its initial conditions [57]. A similar idea was also explored in [91] for continuous dynamical systems with inputs. Recent advances in simulation-based reachability have shown promise in being able to handle models with industrial-scale complexity [62, 63, 69]. These techniques rely on a user-provided annotation in the form of a discrepancy function. Essentially, a discrepancy function provides a mechanism of bounding the distance between adjacent trajectories as a function of the distance between the initial states for the trajectories. Thus, with a reasonably tight discrepancy function, an over-approximation of the reachable state space can be obtained by performing a (potentially) small number of simulations.

4.4 Techniques Based on Safety Invariants

Techniques based on reachability are highly automated and have shown remarkable progress. However, when faced with highly nonlinear plant models, and especially in the presence of discrete switching, these techniques can suffer from loss of precision.

A different approach is offered by semi-automated techniques based on *invariants*. The simplest definition of an invariant is that it is a set such that starting from an element of this set, the time evolution of the system trajectories remains within this set at all times. Typically, we consider the forward time evolution of the system trajectories (i.e., time increases along a trajectory), and thus focus on *forward invariants*. Given a set of *safe* states S, an *invariant* set $\mathscr{I}$ is called a *safety invariant*, if $\mathscr{I} \cap \overline{S} = \emptyset$. Various kinds of invariants have been proposed in the literature to help automate proofs of safety. The prime challenge in invariant-based verification is that it is typically very difficult to find invariants in an automated fashion, and may require human insight.

A key body of work in invariant-based verification is with the use of the KeyMaera and KeyMaeraX theorem proving tools [74, 122]. These tools allow a user to systematically construct the proof of safety of a hybrid system (modeled as a hybrid program). The user has the choice of introducing various kinds of invariants to automate safety proofs. An important class considered is that of *differential invariants* [120]. These are invariants that allow proving the properties of a differential equation without having to solve the equation itself. See [120] for a comprehensive survey. There are certain specializations of invariant-based reasoning that we discuss now.

Control Envelopes. Arechiga et al. [20] present the problem of safety verification for embedded control systems. Here, given a model of the continuous dynamics of a plant, the technique postulates the computation of an *envelope-invariant* pair. The technique assumes that the plant dynamics are given by an ODE of the form:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}),$$

where $\mathbf{x}$ is the state of the plant, and $\mathbf{u}$ is the control input from some set U. We assume that we are given an *invariant* set N (a subset of the plant state space X). We then compute a *control envelope* E that is a function from X to $\mathscr{P}(U)$.[2] The pair (N, E) satisfy the property that for all times t, for any given state $\mathbf{x}(t) \in N$, if the input provided by the controller $\mathbf{u}(t)$ is in the set $E(\mathbf{x})$, then for all $t' > t$, $\mathbf{x}(t') \in N$. Further, if the intersection of N and the set of unsafe states is empty, this gives us a proof of the safety of the closed-loop control system. They also provide specific examples of control envelope-invariant pairs, but does not provide a procedure to compute such pairs for general systems. Computing such control envelopes remains an interesting problem that has attracted recent interest due to applications to runtime monitoring.

Barrier Certificates. A barrier certificate is a type of a safety certificate. Let X be the state space of a system specified by the ODE $\dot{\mathbf{x}} = f(\mathbf{x})$, let I be the set of initial states for the system, and let S be the set of safe states for the system. Then, a barrier certificate is defined as a differentiable function B, which has the following properties:

1. $\forall \mathbf{x} \in I : B(\mathbf{x}) \leq 0$,
2. $\forall \mathbf{x} \in \overline{S} : B(\mathbf{x}) > 0$,
3. $\forall \mathbf{x} \in X : (B(\mathbf{x}) = 0) \implies \frac{\partial B}{\partial \mathbf{x}} \cdot f(\mathbf{x}) < 0$

The intuitive idea is that the set $B(\mathbf{x}) = 0$ serves as a barrier preventing the trajectories of the system that originate in the set I from reaching the set $\overline{S}$. As B is a continuous and differentiable function, every trajectory that starts at a point where B is negative must pass a point where B is zero before reaching a point where B is positive. However, because the Lie derivative of B along the manifold where $B(\mathbf{x}) = 0$ is negative, at each point, the system dynamics forces the B function from not increasing. Barrier certificates were first proposed in [126, 127], and a procedure

[2]For a set X, let $\mathscr{P}(X)$ denote its power set.

based on Sum-of-Squares programming was proposed for finding barrier certificates for systems with polynomial dynamics. These techniques were extended for systems with certain nonpolynomial dynamics [81, 118]. However, the problem of finding barrier certificates for general nonlinear systems remains open.

Simulation-Guided Search for Invariants. Though invariant-based techniques show a lot of promise to prove safety of systems with highly nonlinear and hybrid dynamics, finding the required invariants remains a hard problem. In [145, 146], the authors suggested a simulation-guided technique to estimate the region-of-attraction (ROA) for a given dynamical system. The main idea in this work was to convert a set of bilinear matrix inequalities encountered in estimating the ROA (which are computationally expensive to solve) into linear matrix inequalities, which are computationally less expensive.

In [90], the authors propose a technique to iteratively compute an invariant using simulations, based on the idea of estimating *Lyapunov functions*. Given a system of the form $\dot{\mathbf{x}} = f(\mathbf{x})$, where $f(\mathbf{0}) = 0$, a Lyapunov function $V(\mathbf{x})$ is a function that is positive everywhere except when $\mathbf{x} = \mathbf{0}$, its Lie derivative $\frac{\partial V}{\partial \mathbf{x}}$ is negative everywhere except at $\mathbf{0}$, and at $\mathbf{x} = \mathbf{0}$, both the value of V and its Lie derivative is 0. A Lyapunov function is a tool that can be used to prove stability of a system to the point $\mathbf{x} = \mathbf{0}$. Furthermore, any level set of the Lyapunov function, i.e., $L(\mathbf{x}) = \{\mathbf{x} | V(\mathbf{x}) = \ell\}$ is an invariant for the system. The iterative procedure in the technique proposed in [90] is as follows: (1) the technique fixes the form of a candidate Lyapunov function as some polynomial $P(\mathbf{c}, \mathbf{x})$, where $\mathbf{c}$ is a vector of coefficients of the polynomial function, (2) it uses a set of discrete-time trajectories of the system from a given set of initial states, and uses these to impose constraints on $\mathbf{c}$, (3) it solves the constraints using an appropriate solver to obtain a candidate Lyapunov function, (4) it searches for counterexample for the candidate using an SMT solver, and (5) if a counterexample exists, it is added to the set of initial conditions used in step 2, and the method repeats; else, it terminates with an answer.

The key step is in the formulation of constraints in step 2. For a fixed polynomial form with unknown coefficients, imposing positivity of V at each point in a system trajectory results in a linear constraint. Suppose we are given two points in a discrete-time trajectory of the system (say $\mathbf{x}_n = \mathbf{x}(t_n)$, and $\mathbf{x}_{n+1} = \mathbf{x}(t_{n+1})$), where $t_{n+1} > t_n$. Then, a sufficient condition for the negativity of the Lie derivative is to impose that $V(\mathbf{x}_n) - V(\mathbf{x}_{n+1}) > 0$. Note that this is again a linear constraint in the coefficients of V as $\mathbf{x}_n$ and $\mathbf{x}_{n+1}$ are known. Thus, solving the constraints in step 3 can be done using a standard linear-programming solver.

Step 4 also merits a remark. A candidate Lyapunov function (or by extension a candidate invariant that is the level set of the candidate Lyapunov function) obtained in Step 4 satisfies the required conditions for being a valid Lyapunov function (resp., invariant) on the selected set of system trajectories, but there is no guarantee that these conditions are met globally in the state space. Thus, the method uses a satisfiability modulo theories (SMT) solver that is equipped to reason about satisfiability of arbitrary nonlinear queries; δ-sat solver dReal is such a solver [76]. It returns an answer **unsat** if the query is unsatisfiable, otherwise returns a interval of width δ in the state space where the query may be satisfiable. As checking validity of a condition is equivalent to checking the satisfiability of its negation, an **unsat** answer from dReal helps us establish the conditions required for a given set to be an invariant.

4.5 Falsification Techniques

In this section, we will review techniques to perform *requirement-driven* test generation of CPS models. There are several automated test generation procedures and heuristics that attempt to tackle this problem by viewing it as a special case of software testing. Commercial tools such as the Simulink Design Verifier™ (SLDV) toolbox from the Mathworks [75, 99], the Reactis® [131] tool, and the TestWeaver tool from QTronic [89] are notable for their adoption within industrial MBD practice.

The Reactis Tester tool evaluates open-loop controller models with a patented technique to generate test inputs using a combination of random and targeted methods. The targeted phase of the tool uses data structures to store intermediate states, and constraint-solving algorithms to search for previously uncovered coverage targets. SLDV uses techniques based on SAT modulo theories (SMT) in conjunction with the Prover tool to automatically generate test inputs to maximize coverage criteria. SLDV is intended for open-loop (discrete-time) controller models, as it cannot process closed-loop (hybrid) models.

The TestWeaver tool does test generation with the goal to maximize state coverage of the underlying system (where coverage is defined in a specific fashion). The test generation algorithm itself is based on proprietary heuristics. The tool relies on the user to quantize the inputs to the model-under-test, discretize the time domain, and also to manually identify system variables that are most sensitive to the inputs. This user intervention may require an understanding of the system dynamics and engineering intuition to use the tool effectively.

With the exception of certain features in TestWeaver, the above tools are primarily focused on testing the controller models for CPS systems, while unable to effectively reason about the plant/environment model. Furthermore, the properties that these tools check are typically hand-coded by the user and tend to be simpler static properties (such as the bounds on a signal value over a specified time interval).

We now discuss falsification techniques that overcome some of the shortcomings of the existing commercial techniques in various ways:

1. They allow specifications expressed in formal specification languages such as those based on signal temporal logic (STL). This allows complex temporal properties over continuous-valued, continuous-time signal to be seamlessly specified.
2. They can effectively search both continuous and hybrid state spaces that arise from closed-loop models.

3. They can be augmented with metrics to measure coverage of continuous and hybrid state spaces.
4. They can combine search for bugs in the software controller with corner case behaviors in the continuous plant model.

In this chapter, we discuss two main classes of such techniques. The first class of techniques allows falsifying closed-loop specifications of temporal behavior with the help of black-box optimization tools. The second class of techniques combines a novel exploration of plant model behaviors with a technique inspired by multiple shooting methods found in numerical ODE solving with symbolic execution techniques for analyzing controller code.

4.5.1 Falsifying Temporal Specifications Using Optimization

A key technology that enables falsification techniques is *quantitative satisfaction semantics* for real-time temporal logics. Robust satisfaction semantics were proposed for metric temporal logic by Fainekos and Pappas in their seminal paper [68], while quantitative semantics for STL were proposed by Donzé and Maler [58], which we now explain.

Quantitative Semantics of STL. For a formula φ in a given logical formalism and a signal trace $\mathbf{x}$, Boolean satisfaction semantics for the logic provide a *true*/*false* answer for whether $\mathbf{x}$ satisfies φ. Quantitative semantics extend this notion to *robust satisfaction*, i.e., they define a *robust satisfaction degree* (abbreviated as *robustness*) of φ by $\mathbf{x}$. The intuition is that if the robustness value is a positive number, then $\mathbf{x}$ satisfies φ; if it is negative, it does not satisfy φ, and the magnitude of the robustness degree indicates how *strongly* φ is satisfied (or violated).

We provide the formal robustness semantics for STL below in terms of a function ρ that maps a given trace $\mathbf{x}$, a formula φ, and a time t to a real number. This function for a predicate of the form $f(\mathbf{x}) > 0$ at time t is simply the value of $f(\mathbf{x})$ at time t, i.e., $\rho(f(\mathbf{x}) > 0, \mathbf{x}, t) = f(\mathbf{x}(t))$. Then, ρ is defined inductively for every STL formula using the following rules:

$$\rho(\neg\varphi, \mathbf{x}, t) = -\rho(\varphi, \mathbf{x}) \tag{4.1}$$

$$\rho(\varphi_1 \wedge \varphi_2, \mathbf{x}, t) = \min(\rho(\varphi_1, \mathbf{x}, t), \rho(\varphi_2, \mathbf{x}, t)) \tag{4.2}$$

$$\rho(\mathbf{F}_I\varphi, \mathbf{x}, t) = \sup_{t' \in t+I} \rho(\varphi, \mathbf{x}, t') \tag{4.3}$$

$$\rho(\mathbf{G}_I\varphi, \mathbf{x}, t) = \inf_{t' \in t+I} \rho(\varphi, \mathbf{x}, t') \tag{4.4}$$

$$\rho(\varphi_1 \mathbf{U}_I \varphi_2, \mathbf{x}, t) = \sup_{t' \in t+I} \left(\min \left(\rho(\varphi_2, \mathbf{x}, t'), \inf_{t'' \in [t, t')} \rho(\varphi_1, \mathbf{x}, t'') \right) \right) \tag{4.5}$$

By convention, the robustness of φ by $\mathbf{x}$ is then simply $\rho(\varphi, \mathbf{x}, 0)$. If we omit the time argument, the implicit assumption is that we are computing the robustness at time 0, i.e., $\rho(\varphi, \mathbf{x}) = \rho(\varphi, \mathbf{x}, 0)$.

We recall that a closed-loop model M can be viewed as a function mapping finite-time input signals $\mathbf{u}$ (defined over time $[0, T]$) to output signals $\mathbf{y}$. For simplicity, we assume that the specification φ is an appropriate STL formula over output signals. Then, the falsification problem can be restated as a search for an input signal $\mathbf{u}$ such that $\rho(\varphi, \mathbf{y}) < 0$. The central idea in most falsification tools is to solve this problem by solving the following optimization problem:

$$\mathbf{u}^* = \underset{\mathbf{u} \text{ s.t. } \mathbf{y}=M(\mathbf{u})}{\arg\min} \; \rho(\varphi, \mathbf{y}) \tag{4.6}$$

If we find a $\mathbf{u}^*$ such that $\rho(\varphi, M(\mathbf{u}^*)) < 0$, then we have effectively found a violation of the specification, or successfully falsified the model. While the above setup seems straightforward, there are several caveats.

Input Signal Parameterization. The first is that optimizing over a dense-time input signal is an infinite-dimensional optimization problem. A common approach is to make the search space finite by assuming a *finite parameterization* of the input signal space. For example, one of the approaches adopted by tools such as S-TaLiRo [19] and Breach [55] is to introduce n uniformly spaced discrete time points $t_0, \ldots, t_{n-1}$ along the time axis, also known as *control points*. Here, $t_0 = 0$, and $t_{n-1} = T$. Then, the input signal $\mathbf{u}$ is defined in terms of $(\mathbf{u}_0, \ldots, \mathbf{u}_{n-1})$, as follows: for all $i \in [0, n-1]$: $\mathbf{u}(t) = \mathbf{u}_i$ if $\frac{i}{n-1}T \leq t < \frac{i+1}{n-1}T$. In simpler terms, the signal $\mathbf{u}(t)$ is obtained by *constant interpolation* over values $(\mathbf{u}_0, \ldots, \mathbf{u}_{n-1})$ equally spaced in time. This notion can be generalized by introducing variably spaced time points, and user-defined interpolation functions (such as piecewise linear, splines, etc.).

Another approach is to define a finite grid over the input signal space, i.e., in addition to discretization of the time axis, we also quantize the value axis of the signal. The input signal is ultimately constructed using interpolation over points over this finite grid. (See Fig. 4.6a for an illustration.) Such a grid can then be refined iteratively by the optimization algorithm. This is the approach explored in [52].

Nonconvex Search Space. Most optimization tools critically rely on the optimization problem being defined over a convex space, which enables gradient descent-like optimization methods. Further, such approaches may also require the exact analytic gradient to be available. The optimization problem set up in Eq. (4.6) almost never has such nice properties. First, the method M can be an arbitrary hybrid dynamical system with a high-dimensional state space. Further, the cost function ρ is itself not a smooth function of its input. Thus, most falsification tools rely on *black-box* optimization techniques such as the derivative-free Nelder–Mead technique used in Breach [55], heuristic search techniques such as genetic algorithms [19], Ant Colony optimization [18], the Cross Entropy method [134], or stochastic gradient descent combined with discrete Tabu search [52]. A common theme in these

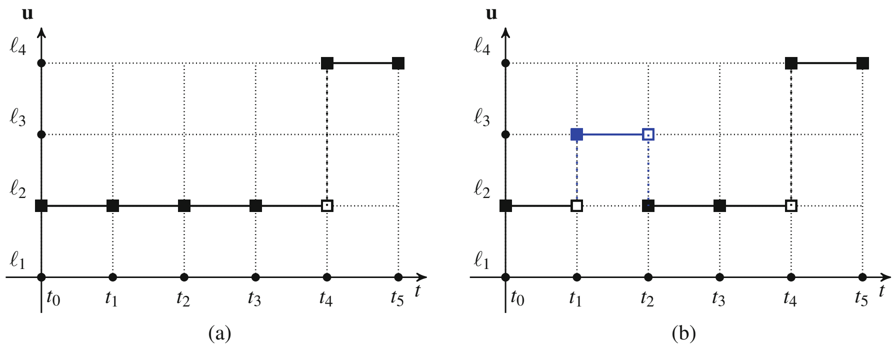

Fig. 4.6 (**a**) Example of using a finite grid to approximate an input signal. The input signal $\mathbf{u}(t)$ is obtained by constant interpolation over the sequence $(\ell_2, \ell_2, \ell_2, \ell_2, \ell_4, \ell_4)$ over the time domain $(t_0, t_1, t_2, t_3, t_4, t_5)$. (**b**) Example of a grid neighbor of the input signal shown in (**a**)

methods is to evaluate the cost function, i.e., the robustness value for a heuristically sequence of points in the input space, and generally choose input points with lower costs. The exact heuristics of how the sequence of inputs is chosen depends on the specific algorithm in question. For example, in Fig. 4.6b, we show how an input signal corresponding to the grid neighbor of the input signal in Fig. 4.6a is chosen for cost function evaluation.

Recently, given the immense success of machine learning techniques in learning and approximating black-box functions, there have been efforts to apply such methods to the falsification problem [6, 51, 92].

Yet another class of methods focuses on simultaneously trying to maximize coverage of the hybrid state space and find a violation of the property of interest. The technique in [59] iteratively computes the input signal incrementally using the rapidly exploring random trees (RRT) algorithm used for motion planning. The RRT algorithm is tuned to pick goal states that maximize a weighted combination of the (incremental) robustness of the output signal, and a coverage metric over the continuous state space of the closed-loop model. In [5], the authors combine a coverage metric on the input signal space with a machine learning technique to classify already covered regions in the input space. In [54], the authors define a hybrid distance metric to obtain coverage over discrete mode switches in the closed-loop model.

4.5.2 Falsification Using Trajectory Splicing

Thus far, the approaches to falsification are *single shooting* approaches that search over a single trajectory starting from some initial condition that falsifies the specification. An alternative approach is to use *multiple shooting*, wherein the approach

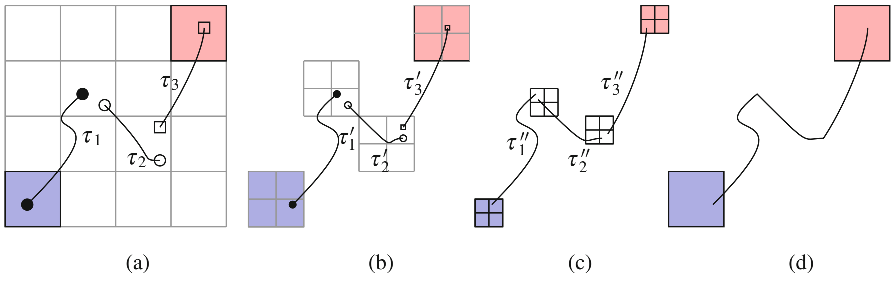

Fig. 4.7 An illustration of the trajectory splicing approach: (**a**) segmented trajectory reaching unsafe states (shaded red) starting from initial states (shaded blue), (**b**) refining an abstract counterexample and narrowing the inter-segment gap, (**c**) further narrowing the gap by refinement, and (**d**) a concrete trajectory with no gaps

splices a collection of trace segments that take us from one state to another in the state space. An approximate trajectory takes a sequence of such trace segments with possible gaps between the ith trace segment and the $(i+1)$th segment. The approach then iteratively narrows the gap through a suitable optimization procedure, leading from an initial sequence of segments to a trajectory of the system obtained when the gaps are reduced to zero. The trajectory splicing approach using local gradient descent was first proposed by Zutshi et al., inspired in turn by collocation-based approaches to integrating systems of differential equations and similar multiple shooting approaches to optimal control (see [152]). Subsequently, this was extended to a larger class of systems using graph-based search and iterative refinement [153]. See Fig. 4.7 for an illustration of the iterative refinement procedure used in the tool S3CAM that performs trajectory splicing for arbitrary hybrid systems.

Trajectory splicing is essentially a state-space exploration technique for hybrid systems. Recall that in many CPS applications, the closed-loop system model is often expressed as a hybrid or continuous plant model composed with a discrete software controller. It is possible to enhance the efficacy of splicing-based falsification techniques by combining trajectory segments explored in the plant's state space by symbolic execution of the controller. This approach was explored in [151], and uses symbolic path exploration tools based on SMT solvers. The scalability of this technique is currently limited by that of existing SMT solvers.

4.6 Challenge Problem: Verification of AI-Based Systems

AI-based systems, especially those based on artificial neural networks (ANNs) and by extension, deep neural networks (DNNs) have gained increasing prominence in CPS applications where they support perception tasks from rich image, LIDAR, and other sensor data [80], and the design of control using ideas such as reinforcement learning [142]. However, a key drawback of neural networks lies in the inability of humans to understand their operation and the well-publicized instances of incorrect operation that can potentially endanger life [110]. How do we verify systems governed by deep neural networks? Currently, the problem of verifying CPS applications that use ANNs/DNNs has received increasing attention from researchers and two independent streams have emerged.

Testing for Perception Components. The first set of techniques focuses on testing deep neural networks used for perception tasks. One approach lies in reasoning about properties of the perception tasks such as recognizing features in images reliably. The main challenges in this area include the hard challenge of writing behavioral specifications for perception tasks that involve feature rich input sources such as images, videos, and LIDAR data streams. Another challenge lies in the sheer size of the network in terms of the number of neurons and the depth of the network, which makes existing verification tools hard to apply directly. *Adversarial test generation* is a popular paradigm which has spawned a number of research papers, focused on identifying mild perturbations to images that result in failed object recognition. Typical approaches use gradient search over the network, or a mixed integer linear programming problem to analyze the robustness of classification tasks to a set of changes to pixels in the images [133, 144]. Another related direction of research is framed as a search for "adversarial" inputs that expose problems with the current network. A linear programming-based approach for finding adversarial inputs is presented by Bastani et al. [26]. A related approach for finding adversarial inputs using SMT solvers that relies on a layer-by-layer analysis is presented by Huang et al. [87]. Currently, falsification-based approaches have proven advantageous for these tasks given the sheer size and complexity of the neural networks involved. Yet, the number of simulations needed, and time taken for each simulations remain astronomically high. Currently, it is important to derive approaches that can significantly reduce both these bottlenecks for falsification.

Dreossi et al. present an approach that uses falsification to test neural network-based perception systems used in autonomous driving by manually generating scenes with known ground truth data [60, 61]. A more elaborate end-to-end approach has been proposed by Abbas et al. using falsification tools to drive the process of testing various scenarios and popular gaming engines to recreate the driving scenarios in order to provide visual inputs to the cameras [3], or the use of robotic simulators to create visual inputs to the perception algorithm (in concert with a closed-loop vehicle dynamics model and a controller) [147].

Testing/Verification of AI-Based Control. The second stream of work considers the safe learning of control laws that take the form of neural network, starting from high level behavioral end-to-end specifications. Here, verification approaches have reported more initial successes due to the much smaller size of neural networks involved in these tasks, when compared to perception tasks. A fundamental primitive that arises in such verification involves the propagation of interval uncertainties over a neural network. Recently, there has been a surge of interest in this problem starting from an approach that linearizes the nonlinear activation function [128], the Reluplex solver by Katz et al. that modifies the Simplex approach to handle piecewise linear constraints posed by the nonlinear rectified linear units [93], an approach using a reduction to mixed integer solvers [101], a combination of local and global search [65], and an integration of convexification with conflict clauses driven by a SAT solver [67]. Whereas these works have considered the neural network in isolation, recent work by Dutta et al. has focused on integrating the learning and verification in a systematic manner using both plant and controller models [64, 66]. The work in [148] uses a closed-loop model of a plant and a neural network-based controller (trained using reinforcement learning) and obtains a barrier certificate for the system. The technique relies on using simulations to find an appropriate barrier certificate and uses the interval constraint propagation-based SMT solver dReal [76] to provide the ultimate proof of safety.

While most of the above approaches have initiated the work of tackling the hard problem of verifying AI-based systems, there is more work to be done. Scaling current approaches to real-world DNNs is a significant challenge, as is the challenge of expressing verification goals for such algorithms in a clean mathematical formalism.

4.7 Conclusion

In this chapter, we reviewed some of the main topics in the formal verification and falsification of cyber-physical systems. The key challenge for such systems is the coupling of the continuous-time behaviors of a physical component with discrete-time control software in the presence of an uncertain environment. Such systems can be mathematically modeled as *hybrid dynamical systems*. Proving safety of such systems over a bounded time horizon can be addressed by solving the *reachability analysis* problem for such hybrid systems, which involves over-approximating the set of behaviors of the system, and proving that this set does not include the unsafe behaviors. An alternate approach is to use *falsification* techniques that seek to find incorrect system behaviors through systematic search procedures. A key assumption for verification or falsification is the ability to express safe behaviors of a system in a formal specification language. We review signal temporal logic, which is a formal logic capable of expressing several interesting properties for CPS applications. We conclude the chapter with a challenge problem that will test the limits of existing verification and falsification techniques.

Acknowledgements We dedicate this chapter to the memory of Dr. Oded Maler, a great friend and collaborator, who shaped our knowledge and perspectives on this vast topic through numerous insightful discussions over the years. The authors also acknowledge contributions from numerous collaborators with special thanks to Xin Chen, Georgios Fainekos, James Kapinski, Nikos Aréchiga, Xiaoqing Jin, and Aditya Zutshi.

This work was funded in part by the US National Science Foundation (NSF) under award numbers CAREER 0953941, CNS 1319457, CPS 1446900, SHF 1527075, CPS 1646556, CCF 1837131, and the Air Force Research Laboratory (AFRL). All opinions expressed are those of the authors and not necessarily of the US NSF or AFRL.

References

1. Abbas, H., Fainekos, G., Sankaranarayanan, S., Ivancic, F., & Gupta, A. (2013). Probabilistic temporal logic falsification of cyber-physical systems. *ACM Transactions on Embedded Computing Systems, 12*, 95.
2. Abbas, H., Hoxha, B., Fainekos, G., & Ueda, K. (2014). Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems. In *2014 IEEE 4th Annual International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)* (pp. 1–6). Piscataway: IEEE.
3. Abbas, H., O'Kelly, M., Rodionova, A., & Mangharam, R. (2017). Safe at any speed: A simulation-based test harness for autonomous vehicles. In *7th Workshop on Design, Modeling and Evaluation of Cyber Physical Systems (CyPhy'17)*.
4. Abbas, H., Rodionova, A., Bartocci, E., Smolka, S. A., & Grosu, R. (2017). Quantitative regular expressions for arrhythmia detection algorithms. In *Proceedings of the International Conference on Computational Methods in Systems Biology* (pp. 23–39). Berlin: Springer.
5. Adimoolam, A., Dang, T., Donzé, A., Kapinski, J., & Jin, X. (2017). Classification and coverage-based falsification for embedded control systems. In *International Conference on Computer Aided Verification* (pp. 483–503). Berlin: Springer.
6. Akazaki, T., Liu, S., Yamagata, Y., Duan, Y., & Hao, J. (2018). Falsification of cyber-physical systems using deep reinforcement learning. arXiv preprint arXiv:1805.00200.
7. Althoff, M. (2015). An introduction to CORA 2015. In *Proceedings of the Workshop on Applied Verification for Continuous and Hybrid Systems* (pp. 120–151).
8. Althoff, M., & Grebenyuk, D. (2016). Implementation of interval arithmetic in CORA 2016. In *Proceedings of the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems* (pp. 91–105).
9. Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T. A., Ho, P. H., Nicollin, X., et al. (1995). The algorithmic analysis of hybrid systems. *Theoretical Computer Science, 138*(1), 3–34.
10. 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 *Workshop on International Hybrid Systems* (pp. 209–229). Berlin: Springer.
11. Alur, R., Dang, T., & Ivančić, F. (2003). Counter-example guided predicate abstraction of hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science* (Vol. 2619, pp. 208–223). Berlin: Springer
12. Alur, R., & Dill, D.L. (1994). A theory of timed automata. *Theoretical Computer Science, 126*(2), 183–235.
13. Alur, R., Fisman, D., & Raghothaman, M. (2016). Regular programming for quantitative properties of data streams. In *Proceedings of the European Symposium on Programming Languages and Systems* (pp. 15–40). Berlin: Springer.
14. Alur, R., & Henzinger, T. A. (1989). A really temporal logic. In *Proceedings of the Symposium on Foundations of Computer Science* (pp. 164–169).
15. Alur, R., Henzinger, T. A., Lafferriere, G., & Pappas, G.J. (2000). Discrete abstractions of hybrid systems. *Proceedings of the IEEE, 88*(7), 971–984.
16. Alur, R., Mamouras, K., & Ulus, D. (2017). Derivatives of quantitative regular expressions. In *Models, algorithms, logics and tools* (pp. 75–95). Cham: Springer.
17. Ames, A. D., Grizzle, J. W., & Tabuada, P. (2014). Control barrier function based quadratic programs with application to adaptive cruise control. In *2014 IEEE 53rd Annual Conference on Decision and Control (CDC)* (pp. 6271–6278). Piscataway: IEEE.
18. Annapureddy, Y. S. R., & Fainekos, G. E. (2010). Ant colonies for temporal logic falsification of hybrid systems. In *Proceedings of the 36th Annual Conference of IEEE Industrial Electronics* (pp. 91–96). Piscataway: IEEE.

19. Annpureddy, Y., Liu, C., Fainekos, G. E., & Sankaranarayanan, S. (2011). S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems* (pp. 254–257). Berlin: Springer.
20. Aréchiga, N., & Krogh, B. (2014). Using verified control envelopes for safe controller design. In *2014 American Control Conference (ACC)* (pp. 2918–2923). Piscataway: IEEE.
21. Asarin, E., Caspi, P., & Maler, O. (2002). Timed regular expressions. *Journal of the ACM, 49*(2), 172–206.
22. Asarin, E., Dang, T., & Girard, A. (2007). Hybridization methods for the analysis of nonlinear systems. *Acta Informatica, 43*(7), 451–476.
23. Asarin, E., Maler, O., & Pnueli, A. (1995). Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science, 138*, 35–65.
24. Baier, C., & Katoen, J. P. (2008). *Principles of model checking*. Cambridge, MA: MIT Press.
25. Bak, S., & Duggirala, P. S. (2017). HyLAA: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control* (pp. 173–178). New York: ACM.
26. Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A., & Criminisi, A. (2016). Measuring neural net robustness with constraints. In *Advances in Neural Information Processing Systems* (pp. 2613–2621).
27. Berz, M. (1999). *Modern map methods in particle beam physics*. Advances in Imaging and Electron Physics (Vol. 108). London: Academic.
28. Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., et al. (2016) End to end learning for self-driving cars. arXiv preprint arXiv:1604.07316.
29. Bonakdarpour, B., & Finkbeiner, B. (2016). Runtime verification for HyperLTL. In *International Conference on Runtime Verification* (pp. 41–45). Cham: Springer.
30. Bonakdarpour, B., Sanchez, C., & Schneider, G. (2018). Monitoring hyperproperties by combining static analysis and runtime verification. In *International Symposium on Leveraging Applications of Formal Methods* (pp. 8–27). Berlin: Springer.
31. Bournez, O., Maler, O., & Pnueli, A. (1999). Orthogonal polyhedra: Representation and computation. In *Hybrid systems: Computation and control. Lecture Notes in Computer Science* (Vol. 1569, pp. 46–60). Berlin: Springer.
32. Box, G. E. P. (1979). Robustness in the strategy of scientific model building. In *Robustness in Statistics* (pp. 201–236). London: Academic.
33. Brockett, R. (1993). Hybrid models for motion control systems. In *Essays on control: Perspectives in the theory and its applications* (pp. 29 –53). Boston: Birkhäuser.
34. Cameron, F., Fainekos, G., Maahs, D. M., & Sankaranarayanan, S. (2015). Towards a verified artificial pancreas: Challenges and solutions for runtime verification. In *Proceedings of Runtime Verification (RV'15). Lecture Notes in Computer Science* (Vol. 9333, pp. 3–17). Cham: Springer.
35. Cameron, F., Wilson, D. M., Buckingham, B. A., Arzumanyan, H., Clinton, P., Chase, H. P., et al. (2012). Inpatient studies of a Kalman-filter-based predictive pump shutoff algorithm. *Journal of Diabetes Science and Technology, 6*(5), 1142–1147.
36. Cassandras, C. G., & Lygeros, J. (2006). *Stochastic hybrid systems*. Boca Raton: CRC Press.
37. Chaochen, Z., Hoare, C. A. R., & Ravn, A. P. (1991). A calculus of durations. *Information Processing Letters, 40*(5), 269–276.
38. Chee, F., & Fernando, T. (2007). *Closed-loop control of blood glucose*. Berlin: Springer.
39. Chen, S., O'Kelly, M., Weimer, J., Sokolsky, O., & Lee, I. (2015). An intraoperative glucose control benchmark for formal verification. In *5th IFAC conference on Analysis and Design of Hybrid Systems (ADHS)* (2015)
40. Chen, X., Ábrahám, E., & Sankaranarayanan, S. (2012). Taylor model flowpipe construction for non-linear hybrid systems. In *Proceedings of the 2012 IEEE 33rd Real-Time Systems Symposium (RTSS'12)* (pp. 183–192). Piscataway: IEEE.
41. Chen, X., Ábrahám, E., & Sankaranarayanan, S. (2013). Flow*: An analyzer for non-linear hybrid systems. In *International Conference on Computer Aided Verification. Lecture Notes in Computer Science* (Vol. 8044, pp. 258–263). Berlin: Springer.

42. Chen, X., Mover, S., & Sankaranarayanan, S. (2017). Compositional relational abstraction for nonlinear systems. *ACM Transactions on Embedded Computing Systems, 16*(5s), 187.
43. Chen, X., & Sankaranarayanan, S. (2016). Decomposed reachability analysis for nonlinear systems. In *2016 IEEE Real-Time Systems Symposium (RTSS)* (pp. 13–24). Piscataway: IEEE.
44. Chonev, V., Ouaknine, J., & Worrell, J. (2016). On the Skolem problem for continuous linear dynamical systems. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016). Leibniz International Proceedings in Informatics* (Vol. 55, pp. 100:1–100:13). Wadern: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
45. Chutinan, A., & Krogh, B. (1998). Computing polyhedral approximations to flow pipes for dynamic systems. In *Proceedings of the 37th IEEE Conference on Decision and Control*. Piscataway: IEEE.
46. Chutinan, A., & Krogh, B. H. (2003). Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control, 48*(1), 64–75. https://doi.org/10.1109/TAC.2002.806655
47. Clarkson, M. R., & Schneider, F. B. (2010). Hyperproperties. *Journal of Computer Security, 18*(6), 1157–1210.
48. Cobelli, C., Man, C. D., Sparacino, G., Magni, L., Nicolao, G. D., & Kovatchev, B. P. (2009). Diabetes: Models, signals and control (methodological review). *IEEE Reviews in Biomedical Engineering, 2*, 54–95.
49. Dang, T., & Maler, O. (1998). Reachability via face lifting. In *Hybrid Systems: Computation and Control. Lecture Notes in Computer Science* (Vol. 1386, pp. 96–109). Berlin: Springer
50. Dang, T., Maler, O., & Testylier, R. (2010). Accurate hybridization of nonlinear systems. In *Hybrid Systems: Computation and Control (HSCC '10)* (pp. 11–20). New York: ACM.
51. Deshmukh, J., Horvat, M., Jin, X., Majumdar, R., & Prabhu, V. S. (2017). Testing cyber-physical systems through Bayesian optimization. *ACM Transactions on Embedded Computing Systems, 16*(5s), 170.
52. Deshmukh, J., Jin, X., Kapinski, J., & Maler, O. (2015). Stochastic local earch for falsification of hybrid ystems. In *International Symposium on Automated Technology for Verification and Analysis* (pp. 500–517). Berlin: Springer.
53. Dimitrova, R., Finkbeiner, B., Kovács, M., Rabe, M. N., & Seidl, H. (2012). Model checking information flow in reactive systems. In *International Workshop on Verification, Model Checking, and Abstract Interpretation* (pp. 169–185). Berlin: Springer.
54. Dokhanchi, A., Zutshi, A., Srinivas, R. T., Sankaranarayanan, S., & Fainekos, G. E. (2015). Requirements driven falsification with coverage metrics. In *2015 International Conference on Embedded Software (EMSOFT'15)* (pp. 31–40). Piscataway: IEEE.
55. Donzé, A. (2010). Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *International Conference on Computer Aided Verification* (pp. 167–170). Berlin: Springer.
56. Donzé, A., Ferrère, T., & Maler, O. (2013). Efficient robust monitoring for STL. In *Computer Aided Verification* (pp. 264–279). Berlin: Springer.
57. Donzé, A., & Maler, O. (2007). Systematic simulation using sensitivity analysis. In *International Workshop on Hybrid Systems: Computation and Control* (pp. 174–189). Berlin: Springer.
58. Donzé, A., & Maler, O. (2010). Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems* (pp. 92–106). Berlin: Springer.
59. Dreossi, T., Dang, T., Donzé, A., Kapinski, J., Deshmukh, J., & Jin, X. (2015). Efficient guiding strategies for testing of temporal properties of hybrid systems. In *NASA Formal Methods Symposium* (pp. 127–142). Berlin: Springer.
60. Dreossi, T., Donzé, A., & Seshia, S. A. (2017). Compositional falsification of cyber-physical systems with machine learning components. In *NASA Formal Methods. Lecture Notes in Computer Science* (Vol. 10227). Berlin: Springer.
61. Dreossi, T., Ghosh, S., Sangiovanni-Vincentelli, A., & Seshia, S.A. (2017). Systematic testing of convolutional neural networks for autonomous driving. In *Reliable Machine Learning in*

the Wild (RMLW) Workshop, Cf. https://people.eecs.berkeley.edu/~tommasodreossi/papers/rmlw2017.pdf
62. Duggirala, P. S., Fan, C., Mitra, S., & Viswanathan, M. (2015). Meeting a powertrain verification challenge. In *Proceedings of the 27th International Conference on Computer Aided Verification. Part I* (pp. 536–543). Cham: Springer.
63. Duggirala, P. S., Potok, M., Mitra, S., & Viswanathan, M. (2015). C2E2: A tool for verifying annotated hybrid systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control (HSCC'15)* (pp. 307–308). New York: ACM.
64. Dutta, S., Jha, S., Sankaranarayanan, S., & Tiwari, A. (2018). Learning and verification of feedback control systems using feedforward neural networks. *IFAC-PapersOnLine, 51*(16), 151–156.
65. Dutta, S., Jha, S., Sankaranarayanan, S., & Tiwari, A. (2018). Output range analysis for deep feedforward neural networks. In *Proceedings of NASA Formal Methods Symposium (NFM). Lecture Notes in Computer Science* (Vol. 10811, pp. 121–138). Berlin: Springer.
66. Dutta, S., Kushner, T., & Sankaranarayanan, S. (2018). Robust data-driven control of artificial pancreas systems using neural networks. In M. Češka, & D. Šafránek (Eds.), *Computational methods in systems biology* (pp. 183–202). Cham: Springer.
67. Ehlers, R. (2017). Formal verification of piece-wise linear feed-forward neural networks. In *International Symposium on Automated Technology for Verification and Analysis. Lecture Notes in Computer Science* (Vol. 10482, pp. 269–286). Berlin: Springer.
68. Fainekos, G. E., & Pappas, G. J. (2009). Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science, 410*(42), 4262–4291.
69. Fan, C., Kapinski, J., Jin, X., & Mitra, S. (2018). Simulation-driven reachability using matrix measures. *ACM Transactions on Embedded Computing Systems, 17*(1), 21:1–21:28.
70. Finkbeiner, B., Rabe, M. N., & Sánchez, C. (2015). Algorithms for model checking HyperLTL and HyperCTL*. In *International Conference on Computer Aided Verification* (pp. 30–48). Berlin: Springer.
71. Forejt, V., Kwiatkowska, M., Norman, G., & Parker, D. (2011). Automated verification techniques for probabilistic systems. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems* (pp. 53–113). Berlin: Springer.
72. Fränzle, M., Herde, C., Ratschan, S., Schubert, T., & Teige, T. (2007). Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. *Journal on Satisfiability, Boolean Modeling and Computation, 1*, 209–236.
73. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., et al. (2011). SpaceEx: Scalable verification of hybrid systems. In *International Conference on Computer Aided Verification (CAV'11). Lecture Notes in Computer Science* (Vol. 6806, pp. 379–395). Berlin: Springer.
74. Fulton, N., Mitsch, S., Quesel, J.D., Völp, M., & Platzer, A. (2015). KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In *Proceedings of International Conference on Automated Deduction* (Vol. 9195, pp. 527–538). Cham: Springer. https://doi.org/10.1007/978-3-319-21401-6_36
75. Gadkari, A., Yeolekar, A., Suresh, J., Ramesh, S., Mohalik, S., & Shashidhar, K. (2008). Automotgen: Automatic model oriented test generator for embedded control systems. In A. Gupta & S. Malik (Eds.), *Computer aided verification. Lecture Notes in Computer Science* (Vol. 5123, pp. 204–208). Berlin: Springer.
76. Gao, S., Kong, S., & Clarke, E. M. (2013). dReal: An SMT solver for nonlinear theories over the reals. In *International Conference on Automated Deduction (CADE'13). Lecture Notes in Computer Science* (Vol. 7898, pp. 208–214). Berlin: Springer.
77. Geiger, A., Lenz, P., & Urtasun, R. (2012) Are we ready for autonomous driving? The Kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 3354–3361). Piscataway: IEEE.
78. Girard, A. (2005). Reachability of uncertain linear systems using zonotopes. In *International Workshop on Hybrid Systems: Computation and Control. Lecture Notes in Computer Science* (Vol. 3414, pp. 291–305). Berlin: Springer.

79. Girard, A., & Pappas, G. J. (2005). Approximate bisimulations for nonlinear dynamical systems. In *Proceedings of the 44th IEEE Conference on Decision and Control* (pp. 684–689). Piscataway: IEEE.
80. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Cambridge, MA: MIT Press. http://www.deeplearningbook.org
81. Goubault, E., Jourdan, J. H., Putot, S., & Sankaranarayanan, S. (2014). Finding non-polynomial positive invariants and lyapunov functions for polynomial systems through darboux polynomials. In *Proceedings of the American Control Conference (ACC)* (pp. 3571–3578). New York: IEEE Press.
82. Hainry, E. (2008). Reachability in linear dynamical systems. In *Logic and theory of algorithms* (pp. 241–250). Berlin: Springer.
83. Henzinger, T. A. (1996). The theory of hybrid automata. In *Proceedings of the Logic in Computer Science* (pp. 278–292). Piscataway: IEEE.
84. Henzinger, T. A., Kopke, P. W., Puri, A., & Varaiya, P. (1998). What's decidable about hybrid automata? *Journal of Computer and System Sciences, 57*(1), 94–124.
85. Herde, C., Eggers, A., Franzle M., & Teige, T. (2008). Analysis of hybrid systems using HySAT. In *Third International Conference on Systems, 2008* (pp. 13–18). Piscataway: IEEE.
86. Hovorka, R. (2005). Continuous glucose monitoring and closed-loop systems. *Diabetic Medicine, 23*(1), 1–12.
87. Huang, X., Kwiatkowska, M., Wang, S., & Wu, M. (2017). Safety verification of deep neural networks. In *Proceedings of the Computer Aided Verification* (pp. 3–29). Cham: Springer.
88. Jiang, Z., Pajic, M., Moarref, S., Alur, R., & Mangharam, R. (2012). Modeling and verification of a dual chamber implantable pacemaker. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Lecture Notes in Computer Science* (Vol. 7214, pp. 188–203). Berlin: Springer.
89. Junghanns, A., Mauss, J., & Tatar, M. (2008). Tatar: Testweaver—a tool for simulation-based test of mechatronic designs. In *6th International Modelica Conference*, Bielefeld, March 3. Citeseer
90. Kapinski, J., Deshmukh, J.V., Sankaranarayanan, S., & Aréchiga, N. (2014). Simulation-guided lyapunov analysis for hybrid dynamical systems. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control* (pp. 133–142). New York: ACM.
91. Kapinski, J., Krogh, B. H., Maler, O., & Stursberg, O. (2003). On systematic simulation of open continuous systems. In *International Workshop on Hybrid Systems: Computation and Control* (pp. 283–297). Berlin: Springer.
92. Kato, K., Ishikawa, F., & Honiden, S. (2018). Falsification of cyber-physical systems with reinforcement learning. In *2018 IEEE Workshop on Monitoring and Testing of Cyber-Physical Systems (MT-CPS)* (pp. 5–6). Piscataway: IEEE.
93. Katz, G., Barrett, C., Dill, D., Julian, K., & Kochenderfer, M. (2017). Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification* (pp. 97–117). Berlin: Springer.
94. Koymans, R. (1990). Specifying real-time properties with metric temporal logic. *Real-Time System, 2*(4), 255–299.
95. Kurzhanski, A. B., & Varaiya, P. (2000). Ellipsoidal techniques for reachability analysis. In *International Workshop on Hybrid Systems: Computation and Control. Lecture Notes in Computer Science* (Vol. 1790, pp. 202–214). Berlin: Springer.
96. Kushner, T., Bortz, D., Maahs, D., & Sankaranarayanan, S. (2018). A data-driven approach to artificial pancreas verification and synthesis. In *International Conference on Cyber-Physical Systems (ICCPS'18)*. New York: IEEE Press.
97. Labinaz, G., Bayoumi, M. M., & Rudie, K. (1997). A survey of modeling and control of hybrid systems. *Annual Reviews in Control, 21*, 79–92.
98. Lafferriere, G., Pappas, G. J., & Sastry, S. (2000). O-minimal hybrid systems. *Mathematics of Control, Signals and Systems, 13*(1), 1–21.

99. Leitner, F., & Leue, S. (2008). Simulink design verifier vs. SPIN a comparative case study. In *Proceedings of the 13th International Workshop on Formal Methods for Industrial Critical Systems*.
100. Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., et al. (2011). Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE Intelligent Vehicles Symposium (IV)* (pp. 163–168). Piscataway: IEEE.
101. Lomuscio, A., & Maganti, L. (2017). An approach to reachability analysis for feed-forward ReLU neural networks. http://arxiv.org/abs/1706.07351
102. Loos, S. M., Platzer, A., & Nistor, L. (2011). Adaptive cruise control: Hybrid, distributed, and now formally verified. In *International Symposium on Formal Methods* (pp. 42–56). Berlin: Springer.
103. Maahs, D. M., Calhoun, P., Buckingham, B. A., Chase, H. P., Hramiak, I., Lum, J., et al. (2014). A randomized trial of a home system to reduce nocturnal hypoglycemia in type 1 diabetes. *Diabetes Care, 37*(7), 1885–1891.
104. Magdici, S., & Althoff, M. (2017). Adaptive cruise control with safety guarantees for autonomous vehicles. *IFAC-PapersOnLine, 50*(1), 5774–5781.
105. Makino, K., & Berz, M. (2003). Taylor models and other validated functional inclusion methods. *Journal of Pure and Applied Mathematics, 4*(4), 379–456.
106. Maler, O., & Nickovic, D. (2004). Monitoring temporal properties of continuous signals. In *Proceedings of Formal Modeling and Analysis of Timed Systems* (pp. 152–166). Berlin: Springer.
107. Meiss, J. D. (2007). *Differential dynamical systems*. Philadelphia: SIAM.
108. Mitchell, I., & Tomlin, C. (2000). Level set methods for computation in hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control. Lecture Notes in Computer Science* (Vol. 1790, pp. 310–323). Berlin: Springer.
109. Mover, S., Cimatti, A., Tiwari, A., & Tonetta, S. (2013). Time-aware relational abstractions for hybrid systems. In *Proceedings of the Eleventh ACM International Conference on Embedded Software (EMSOFT '13)* (pp. 14:1–14:10). Piscataway: IEEE Press.
110. National Transportation Safety Board (NTSB) (2016). Collision between a car operating with automated vehicle control systems and a tractor-semitrailer truck. https://www.ntsb.gov/news/events/Documents/2017-HWY16FH018-BMG-abstract.pdf
111. Nghiem, T., Sankaranarayanan, S., Fainekos, G.E., Ivancic, F., Gupta, A., & Pappas, G.J. (2010). Monte-Carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proceedings of Hybrid Systems: Computation and Control* (pp. 211–220). New York: ACM.
112. Nguyen, L. V., Kapinski, J., Jin, X., Deshmukh, J. V., & Johnson, T. T. (2017). Hyperproperties of real-valued signals. In *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design* (pp. 104–113). New York: ACM.
113. Nicolescu, G., & Mosterman, P. J. (2009). *Model-based design for embedded systems* (1st ed.). Boca Raton: CRC Press.
114. Nilsson, P., Hussien, O., Chen, Y., Balkan, A., Rungger, M., Ames, A., et al. (2014). Preliminary results on correct-by-construction control software synthesis for adaptive cruise control. In *2014 IEEE 53rd Annual Conference on Decision and Control (CDC)* (pp. 816–823). Piscataway: IEEE.
115. Norris, J. (1998). *Markov chains*. Cambridge: Cambridge University Press.
116. Øksendal, B. K. (2000). *Stochastic differential equations: An introduction*. Berlin: Springer.
117. Pajic, M., Mangharam, R., Sokolsky, O., Arney, D., Goldman, J., & Lee, I. (2014). Model-driven safety analysis of closed-loop medical systems. *IEEE Transactions on Industrial Informatics, 10*(1), 3–16.
118. Papachristodoulou, A., & Prajna, S. (2005). Analysis of non-polynomial systems using the sum of squares decomposition. In *Positive Polynomials in Control* (pp. 23–43). Berlin: Springer.

119. Pei, Y., Entcheva, E., Grosu, R., & Smolka, S. (2005) Efficient modeling of excitable cells using hybrid automata. In *Proceedings of the Computational Methods in Systems Biology* (pp. 216–227).
120. Platzer, A. (2008). Differential dynamic logic for hybrid systems. *Journal of Automated Reasoning, 41*(2), 143–189.
121. Platzer, A. (2010). Logical analysis of hybrid systems: Proving theorems for complex dynamics. Heidelberg: Springer. https://doi.org/10.1007/978-3-642-14509-4
122. Platzer, A., & Clarke, E. M. (2008). Computing differential invariants of hybrid systems as fixedpoints. In A. Gupta & S. Malik (Eds.), *Proceedings of computer aided verification. Lecture Notes in Computer Science* (Vol. 5123, pp. 176–189). Berlin: Springer.
123. Pnueli, A. (1977). The temporal logic of programs. In Proceedings of Symposium on Foundations of Computer Science (pp. 46–57). Piscataway: IEEE.
124. Podelski, A., & Wagner, S. (2007). *Region stability proofs for hybrid systems* (pp. 320–335). Berlin: Springer.
125. Prabhakar, P., Duggirala, P. S., Mitra, S., & Viswanathan, M. (2013). Hybrid automata-based CEGAR for rectangular hybrid systems. In R. Giacobazzi, J. Berdine, I. Mastroeni (Eds.), *Verification, model checking, and abstract interpretation* (pp. 48–67). Berlin: Springer.
126. Prajna, S. (2005). *Optimization-based methods for nonlinear and hybrid systems verification*. Ph.D. thesis, California Institute of Technology, Caltech, Pasadena, CA, USA.
127. Prajna, S., & Jadbabaie, A. (2004). Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control* (pp. 477–492). Berlin: Springer.
128. Pulina, L., & Tacchella, A. (2012). Challenging smt solvers to verify neural networks. *AI Communications, 25*(2), 117–135.
129. Ratschan, S., & She, Z. (2005). Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *International Workshop on Hybrid Systems: Computation and Control. Lecture Notes in Computer Science* (Vol. 3414, pp. 573–589). Berlin: Springer.
130. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Transactions on Embedded Computing Systems, 6*(1), 8. http://doi.acm.org/10.1145/1210268.1210276
131. Reactive Systems Inc. (2003). *Model-based testing and validation of control software with reactis*. http://www.reactive-systems.com/papers/bcsf.pdf
132. Roohi, N., Prabhakar, P., & Viswanathan, M. (2016). Hybridization based CEGAR for hybrid automata with affine dynamics. In M. Chechik, & J. F. Raskin (Eds.), *Tools and algorithms for the construction and analysis of systems* (pp. 752–769). Berlin: Springer.
133. Ruan, W., Wu, M., Sun, Y., Huang, X., Kroening, D., & Kwiatkowska, M. (2018). *Global robustness evaluation of deep neural networks with provable guarantees for L0 norm*. http://arxiv.org/abs/1804.05805
134. Sankaranarayanan, S., & Fainekos, G. E. (2012). Falsification of temporal properties of hybrid systems using the cross-entropy method. In *ACM International Conference on Hybrid Systems: Computation and Control* (pp. 125–134). New York: ACM.
135. Sankaranarayanan, S., Kumar, S. A., Cameron, F., Bequette, B. W., Fainekos, G., & Maahs, D. M. (2017). Model-based falsification of an artificial pancreas control system. *ACM SIGBED Review, 14*(2), 24–33.
136. Sankaranarayanan, S., & Tiwari, A. (2011). Relational abstractions for continuous and hybrid systems. In *International Conference on Computer Aided Verification. Lecture Notes in Computer Science* (Vol. 6806, pp. 686–702). Berlin: Springer.
137. Siper, M. J. (2005). *An Introduction to mathematical theory of computation* (2nd ed.). Toronto: Thompson Publishing (Course Technology)
138. Skyler, J. S. (Ed.). (2012). *Atlas of diabetes* (4th ed.). Berlin: Springer.
139. Sontag, E. D. (1981). Nonlinear regulation: The piecewise linear approach. *IEEE Transactions on Automatic Control, 26*(2), 346–358.
140. Steil, G., Panteleon, A., & Rebrin, K. (2004). Closed-sloop insulin delivery—the path to physiological glucose control. *Advanced Drug Delivery Reviews, 56*(2), 125–144.

141. Steil, G. M. (2013). Algorithms for a closed-loop artificial pancreas: The case for proportional-integral-derivative control. *Journal of Diabetes Science and Technology, 7*, 1621–1631.
142. Sutton, R. S., & Barto, A. G. (1998). *Reinforcement learning: An introduction* (Vol. 1). Cambridge: MIT Press.
143. Teixeira, R. E., & Malin, S. (2008). The next generation of artificial pancreas control algorithms. *Journal of Diabetes Science and Technology, 2*, 105–112.
144. Tjeng, V., & Tedrake, R. (2017). Verifying neural networks with mixed integer programming. http://arxiv.org/abs/1711.07356
145. Topcu, U., & Packard, A. (2009). Stability region analysis for uncertain nonlinear systems. *IEEE Transactions on Automatic Control, 54*, 1042–1047.
146. Topcu, U., Seiler, P., & Packard, A. (2008). Local stability analysis using simulations and sum-of-squares programming. *Automatica, 44*, 2669–2675.
147. Tuncali, C. E., Fainekos, G., Ito, H., & Kapinski, J. (2018). Simulation-based adversarial test generation for autonomous vehicles with machine learning components. In *Proceedings of IEEE Intelligent Vehicles Symposium (IV)*
148. Tuncali, C. E., Kapinski, J., Ito, H., & Deshmukh, J. V. (2018). Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018* (pp. 30:1–30:6). New York: ACM.
149. Ulus, D. (2017). Montre: A tool for monitoring timed regular expressions. In *Proceedings of the International Conference on Computer Aided Verification* (pp. 329–335). Berlin: Springer.
150. Ulus, D., Ferrère, T., Asarin, E., & Maler, O. (2014). Timed pattern matching. In *Proceedings of the International Conference on Formal Modeling and Analysis of Timed Systems* (pp. 222–236). Berlin: Springer.
151. Zutshi, A., Sankaranarayanan, S., Deshmukh, J., & Jin, X. (2016). Symbolic-numeric reachability analysis of closed-loop control software. In *Hybrid Systems: Computation and Control (HSCC)* (pp. 135–144). New York: ACM Press.
152. Zutshi, A., Sankaranarayanan, S., Deshmukh, J., & Kapinski, J. (2013). A trajectory splicing approach to concretizing counterexamples for hybrid systems. In *IEEE Conference on Decision and Control (CDC)* (pp. 3918–3925). New York: IEEE Press.
153. Zutshi, A., Sankaranarayanan, S., Deshmukh, J., & Kapinski, J. (2014). Multiple-shooting CEGAR-based falsification for hybrid systems. In *International Conference on Embedded Software (EMSOFT)* (pp. 5:1–5:10). New York: ACM Press.
154. Zutshi A., Sankaranarayanan S., & Tiwari A. (2012). Timed relational abstractions for sampled data control systems. In P. Madhusudan & S. A. Seshia (Eds.), *Computer Aided Verification. Lecture Notes in Computer Science* (Vol. 7358). Berlin: Springer.