Journal of the Indian Institute of Science
A Multidisciplinary Reviews Journal
ISSN: 0970-4140 Coden-JIISAD

REVIEWS

Hybrid Automata for Formal Modeling and Verification of Cyber-Physical Systems

Shankara Narayanan Krishna and Ashutosh Trivedi

Abstract | The presence of a tight integration between the discrete control (the "cyber") and the analog environment (the "physical")—via sensors and actuators over wired or wireless communication networks—is the defining feature of cyber-physical systems. Hence, the functional correctness of a cyber-physical system is crucially dependent not only on the dynamics of the analog physical environment, but also on the decisions taken by the discrete control that alter the dynamics of the environment. The framework of *Hybrid automata*—introduced by Alur, Courcoubetis, Henzinger, and Ho—provides a formal modeling and specification environment to analyze the interaction between the discrete and the continuous parts of cyber-physical systems. Hybrid automata can be considered as generalizations of finite state automata augmented with a finite set of real-valued variables whose dynamics in each state is governed by a system of ordinary differential equations. Moreover, the discrete transitions of hybrid automata are guarded by constraints over the values of these real-valued variables, and enable discontinuous jumps in the evolution of these variables. Considering the richness of the dynamics in a hybrid automaton, it is perhaps not surprising that the fundamental verification questions, like reachability and schedulability, for the general model are undecidable. In this article we present a review of hybrid automata as modeling and verification framework for cyber-physical systems, and survey some of the key results related to practical verification questions related to hybrid automata.

Keywords: *Cyber-Physical Systems, Dynamical Systems, Formal Modeling, Formal Verification, LTL Model-Checking, Timed Automata, Hybrid Automata.*

1 Introduction

The term "cyber-physical systems" refers to any network of digital and analog systems whose performance crucially depends on both the continuous dynamics of the analog parts and the real-time switching decisions made by the digital system. A typical cyber-physical system may consist of several processors connected with a set of physical systems via sensors and actuators over wired or wireless communication networks. Such systems are increasingly playing safety-critical role in modern life, where a fault in their design can be catastrophic.

Modern cars are an important paradigmatic example of such safety-critical cyber-physical systems. A modern premium car typically has 70 to 100 interconnected electronic control units (ECUs) with dozens of sensors[39] performing various functions[44] like air-bag control, cruise control, electronic stability control, antilock brakes, engine ignition, windshield-wiper control, engine control, and collision-avoidance system. Many of these ECUs are connected with analog environment via sensors and actuators, and are expected to perform their operations within hard time limits. For instance, the air-bag ECU needs to respond within 20–30 millisecond after the impact sensor connected to it detects a severe impact. As the number of ECUs in a typical car is increasing and performing more

Department of Computer Science and Engineering Indian Institute of Technology Bombay Mumbai, India 400076
krishnas@cse.iitb.ac.in
trivedi@cse.iitb.ac.in

autonomously, it is becoming increasingly difficult to ensure their correctness. The severity of the problem can perhaps be best realized by looking into the growing list of recalls[44] by leading car companies due to software-related problems. Some prominent examples include Toyota's recall of 160,000 of its 2004/05 Prius models because of a software problem causing the car to suddenly stall, Jaguar's 2011 recall of nearly 18,000 X-type cars due to a software bug resulting in driver's inability in turning off the cruise control, and Volkswagen's 2011 recall of about 4000 of its 2008 Passats models for engine-control module software problem. The list is long and underscores the challenges in designing and verifying safety-critical cyber-physical systems. Similar examples can also be cited for the cyber-physical system from other domains such as avionics, implantable medical devices, transportation networks, and energy sector.

Formal modeling and verification of systems is the set of techniques that employ rigorous mathematical reasoning to analyze properties of a system. In this article we concentrate on a celebrated[3,4] formal verification framework known as *model checking*.[50] Model Checking—pioneered by Clarke, Sifakis and Emerson[2]—is a widely used automated technique that, given a formal description of a system and a property, systematically checks whether this property holds for a given state of the system model. The three key steps of this framework are the following:

1. *formal modeling*: modeling a system under consideration using mathematically precise syntax that approximate a given system to a desired level of abstraction;
2. *formal specification*: specify the properties of the system using some mathematically precise specification language (typically in formal logic); and
3. *formal analysis*: analyze the formal model with respect to the formal specification and report counter-example in case the system model violates the specification.

The success of the model checking framework in formal verification of systems is largely due to it being highly automatic—a push-button technology[47]—in comparison to other competing approaches like theorem proving. The counterexamples generated in the model-checking process often are used to automatically refine—known as counterexample-guided abstraction refinement (CEGAR)[48,49] framework—the model and/or the property and the entire procedure can be repeated and thus removing the need of a very accurate initial model or specification.

Early research on formal modeling and verification of systems concentrated on simplified models of the systems as finite state-transition graphs. Since these models are finite in nature, it is—in theory—possible to exhaustively explore the state space of the system to verify the properties of interest. However, the biggest challenge in model-checking of finite state-transition graphs is so-called *state-space explosion problem*[50] characterizing the exponential blowup in the number of states in the explicit representation of the system where the system is naturally represented succinctly using state variables, or as a composition of a network of interacting finite state-transition graphs. In general, the state-space explosion problem renders the explicit exhaustive exploration of the system intractable. However, a number of techniques have been proposed to overcome the state-space explosion problem—including symmetry reduction,[46] partial-order reduction,[85] symbolic model checking[80] and bounded model checking[29,30]—that has culminated into efficient and mature tool support including SPIN[92] and NuSMV[82] for finite state model-checking. Examples of the use of finite-state model-checking in industry include the verification of hardware circuits,[67] communication[15] and security[24,78] protocols, and software device drivers.[23]

These finite state-transition graphs, however, often do not satisfactorily model cyber-physical systems as they disregard the continuous dynamics of the physical environment. Alur and Dill[10] were the first one to propose a formal model, known as timed automata, combining finite state-transition graphs with a finite set of real-valued variables that evolve as time progresses while the system occupies a state. In a timed automaton the real-valued variables—called clocks—simulate perfect clocks as they evolve with a uniform constant speed (rate) and hence can model asynchronous real-time systems interacting with a continuous physical environment. The clock variables can be used to constrain the evolution of the system by guarding the transitions of the graph, and can also be reset at the time of taking a transition to remember the time since that transition. These capabilities make timed automata quite expressive formalism to define real-time systems. Moreover, the decidability[a] of key verification problems like

[a] The concept of decidability is a central one in computer science and it characterizes the set of problems for which one can write computer programs that always terminate with a correct answer. The problems for which it is not possible to write such a program are known as undecidable problems. A most famous undecidable problem is the halting problem (similar to reachability problem) for the configurations of Turing machines (an abstract model of computation capturing the notion of algorithmic computation).

reachability and schedulability[10] and availability of mature verification tools—like UPPAAL,[27,96] Kronos,[66] and RED[89]—make timed automata an appealing tool for real-time system verification.

Alur, Courcoubetis, Henzinger, and Ho generalized the timed automata to hybrid automata[9] to include real-valued variables with arbitrary dynamics specified using ordinary differential equations. Considering the richness of dynamics of a hybrid automata, it is perhaps not surprising that the fundamental verification questions like reachability are undecidable for hybrid automata. A number of subclasses of hybrid automata has been proposed with decidable verification problems and some of the algorithms have been implemented as part of tools like HyTech[60] and PHAVer.[86]

Timed and hybrid automata provide an intuitive and semantically unambiguous way to model cyber-physical systems, and a number of case-studies[41,55,61,74,84,93,95] demonstrate their application for the analysis of cyber-physical systems. In this article we aim to provide a general introduction to verification using hybrid automata as we focus on model-checking classical LTL logic[77] over hybrid automata. To keep the discussion simple we do not cover other logics, for instance, computation tree logic (CTL, CTL*),[50,77] modal μ-calculus,[53] and real-time and hybrid extensions of these logics[14] including metric temporal logics (MTL)[65,83] and duration calculus (DC).[43]

The goal of this article is to introduce key concepts for cyber-physical system modeling and verification using hybrid automata with a focus on LTL model-checking. In order to better focus our attention, we will not cover several useful extensions of hybrid automata that capture certain natural aspects of modeling hybrid systems, including

- game-theoretic extensions[7,17,20,32,45,52] that allow the model to distinguish between controllable and uncontrollable non-determinism;
- probabilistic extensions[6,25,36,68,71,75] that permit modeling of stochastic behavior arising due to, e.g., faulty or unreliable sensors or actuators, uncertainty in timing delays, and performance characteristics of (third-party) components; and
- priced extensions[28,31,33,73,88] that permit modeling of resource consumption and payoffs associated with decisions.

We also restrict our attention to theoretical results regarding decidability of LTL model-checking problems, and do not cover data structures and algorithms[27,54,57] for efficient implementation of these results.

We begin (Section 2) this survey by introducing two formalisms to model discrete and continuous dynamical systems, and then we present hybrid automata model that combines features from these two models. Section 3 introduces syntax and semantics of linear temporal logic (LTL) followed by a formal definition of corresponding model-checking problem over a hybrid automata, and using two-counter Minsky machines[81] we prove the in general LTL model-checking over hybrid automata is undecidable. In this section, we also introduce the idea of state-space reduction using a well-established technique called *quotienting* which we later exploit to show decidability of model checking problem for some variants of hybrid automata. We conclude the survey by discussing (Section 4) three key subclasses of hybrid automata—timed automata, (initialized) rectangular hybrid automata, and (two dimensional) piecewise-constant derivative systems—with decidable model checking problem.

2 Hybrid Automata

A *dynamical system* is simply a system whose "state" evolves with "time" governed by a fixed set of rules or "dynamics". The state of a dynamical system is specified as valuations of the variables of interest in the system. Depending upon the nature of variables (discrete or continuous) and the notion of time (discrete or continuous) the dynamics of variables can be specified by differential equations or discrete assignments. For the purpose of this paper, we classify the dynamical systems into the following three broad classes: i) *discrete systems* where both the notion of time and the variables are discrete, ii) *continuous systems* where the notion of time is continuous, while the variables are continuous, and iii) *hybrid systems* where some variables are continuous and some are discrete, and although the notion of time is continuous, special dynamic-changing events can happen at discrete instants. Notice that both discrete and continuous systems can be considered as subclasses of hybrid systems.

On an abstract level any dynamical system can simply be represented as a graph whose nodes represent the states and edges represent transition between the states. Formally, a state transition graph can be defined in the following manner.

Definition 1 (State Transition Graphs): A state transition graph is a tuple $\mathcal{T} = (S, S_0, \Sigma, \Delta)$ where:

– S is a (potentially infinite) set of *states*;
– $S_0 \subseteq S$ is the set of *initial states*;
– Σ is a (potentially infinite) set of *actions*; and
– $\Delta \subseteq S \times \Sigma \times S$ is the *transition relation*.

We say that a state transition graph $\mathcal{T}$ is finite (countable), if the sets S and Σ are finite (countable).

Given an action $a \in \Sigma$ and a state s we write Post (s, a) for the set of states that are reachable from s on a and Post (s) for the states reachable in one step from s, i.e.

$$\text{Post}(s, a) = \{s' : (s, a, s') \in \Delta\}$$
$$\text{Post}(s) = \bigcup_{a \in \Sigma} \text{Post}(s, a).$$

A run—an execution or a trajectory—of a dynamical system modeled as a state transition graph $\mathcal{T}$ is a (finite or infinite) alternating sequence of states and actions that begins with an initial state and all consecutive states are connected with their predecessor via the transition relation. Formally, a finite run is a sequence $\langle s_0, a_1, s_1, a_2, s_2, \ldots, s_n \rangle$ such that $s_0 \in S_0$ and for all $0 \leq i < n$ we have that $s_{i+1} \in \text{Post}(s_i, a_{i+1})$. An infinite run is defined analogously.

Example 1: A graphical description of a state transition graph depicting a mod-4 counter with pause is shown in Figure 1. We represent a state using a rounded rectangle and a transition using a labeled edge between participating states. An initial state is marked using an incoming arrow to that state labeled "start". An example of a run is the finite sequence:

$\big\langle$(count, 0), tick, (count, 1), pause, (pause, 1), tick, (pause, 1), on, (count, 1), tick, (count, 2)$\big\rangle$.

A state transition graph is a feasible way to represent and computationally analyze dynamical systems with finitely many states. However, to enable computational analysis of a general infinite state dynamical system we need a finitary way to represent a potentially infinite space of states. We

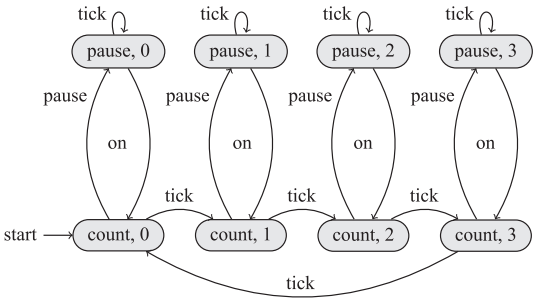


Figure 1: State transition graph for a mod-4 counter.

begin this section by introducing concepts and notation used throughout this article, followed by discussing such syntactical models to represent purely discrete and purely continuous dynamical system. After introducing these models we present hybrid automata capable of modeling hybrid dynamical systems.

Variables and predicates

Let $\mathbb{R}$ be the set of real numbers, $\mathbb{R}_{\geq 0}$ be the set of non-negative real numbers, and $\mathbb{Z}$ be the set of integers.

Let X be a set of real-valued variables. A *valuation* on X is a function $v : X \to \mathbb{R}$ and we write $V(X)$ for the set of valuations on X. Abusing notation, we also treat a valuation v as a point in $\mathbb{R}^n$ that is equipped with the standard *Euclidean norm* $\|\cdot\|$ where n is the cardinality of X.

We define a predicate over a set X as a subset of $\mathbb{R}^{|X|}$. For efficient computer-readable representation of predicates we often define them using non-linear algebraic equations involving X. We write pred(X) for the set of predicates over X. For a predicate $\pi \in$ pred(X) we write $[\![\pi]\!]$ for the set of valuations in $\mathbb{R}^{|X|}$ satisfying the equation π. We write $\top$ for the predicate that is true for all valuations, while $\bot$ for the predicate which is false for all the valuations.

Example 2: An example of a predicate over the variables $\ddot{\theta}$ and θ is

$$m\ell\ddot{\theta} = -mg\sin(\theta),$$

characterizing the motion of an idealized pendulum (Figure 3) where θ is the angle the pendulum forms with its rest position, $\ddot{\theta}$ is second derivative of θ, m is the mass of the pendulum, g is the gravitational constant, and ℓ is the length of the pendulum.

We say that a predicate P is *polyhedral* if it is defined as a conjunction of a finite set of linear constraints of the form $a_1 x_1 + \cdots + a_n x_n \bowtie k$, where $k \in \mathbb{Z}$, for all $1 \leq i \leq n$ we have that $a_i \in \mathbb{R}$, $x_i \in X$, and $\bowtie \in \{<, \leq, =, >, \geq\}$. An example of a polyhedral predicate over the set $\{x, y, z\}$ is $2x + 3y - 9z \leq 5$. We define an *octagonal* predicate as the conjunction of a finite set of linear constraints over X of the form $\pm x \pm y \bowtie k$ or $x \bowtie k$, where $k \in \mathbb{R}$, $x, y \in X$. Similarly a *rectangular* predicate is defined as the conjunction of a finite set of linear constraints over X of the form $x \bowtie k$, where $k \in \mathbb{R}$, and $x \in X$.

2.1 *Discrete dynamical systems*

Discrete dynamical systems can be conveniently modeled as *extended finite state machines* having finitely many modes and transitions between these

modes. The values of variables remain unchanged while the system is in some mode, and changes only when a transition takes place where they can "jump" to new values assigned by the transition. These jumps are specified using *predicates* over the set $X \cup X'$ that relates the current values of variables of system, given as the set X, to the values in the next time-step, given as the set X' of primed-versions of variables in X. Transitions are often guarded by *predicates* over variables specifying the enabledness condition of the transition. Starting from some initial valuation to the variables, a system modeled using an extended finite state machine evolves in discrete time-steps. At each discrete step the system can take any enabled transition, i.e. satisfied by the current variable valuation, and after executing the transition the valuation of the variables is changed according to the jump condition. The system continues evolving in this fashion forever. An extended finite state machine is formally defined as the following.

Definition 2 (Extended Finite State Machines: Syntax): An *extended finite state machine* is a tuple $\mathcal{M} = (M, M_0, \Sigma, X, \Delta, I, V_0)$ such that:

- M is a finite set of control *modes* including a distinguished initial set of control modes $M_0 \subseteq M$,
- Σ is a finite set of *actions*,
- X is a finite set of real-valued *variable*,
- $\Delta \subseteq M \times \text{pred}(X) \times \Sigma \times \text{pred}(X \cup X') \times M$ is the *transition relation*,
- $I : M \rightarrow \text{pred}(X)$ is the mode-invariant function, and
- $V_0 \in \text{pred}(X)$ is the set of initial valuations.

For a transition $\delta = (m, g, a, j, m') \in \Delta$ we refer to $m \in M$ as its *source mode*, $g \in \text{pred}(X)$ as its *guard*, $a \in A$ as its *action*, $j \in \text{pred}(X \cup X')$ as its *jump constraint*, and $m' \in M$ as the *target mode*.

A configuration of an extended finite state machine is a tuple (m, v) where m is a control mode and v is a valuation of variables in X. The execution of an extended finite state machine begins in a configuration (m_0, v_0) such that the control mode $m_0 \in M_0$ is in the set of initial control modes and the valuation $v_0 \in V_0$ satisfies the invariant of mode m_0, i.e. $v_0 \in [\![I(m_0)]\!]$. At each discrete time-step the system executes a transition (m, g, a, j, m') that is enabled in the current configuration (m, v), i.e., $v \in [\![g]\!]$, and the configuration of the system jumps to a new configuration (m', v') while respecting the jump constraints, i.e. $(v, v') \in [\![j]\!]$ as well as the invariant condition of the resulting mode $v' \in [\![I(m')]\!]$. The system continues its execution from the resulting configuration in the similar fashion. Hence, we can define the semantics of an extended finite state machine as a state transition graph in the following manner.

Definition 3 (Extended Finite State Machine: Semantics): The semantics of an extended finite state machine $\mathcal{M} = (M, M_0, \Sigma, X, \Delta, I, V_0)$ is given as a state transition graph $T^{\mathcal{M}} = (S^{\mathcal{M}}, S_0^{\mathcal{M}}, \Sigma^{\mathcal{M}}, \Delta^{\mathcal{M}})$ where:

- $S^{\mathcal{M}} \subseteq (M \times \mathbb{R}^{|X|})$ is the set of configurations of $\mathcal{M}$ such that for all $(m, v) \in S^{\mathcal{M}}$ we have that $v \in [\![I(m)]\!]$;
- $S_0^{\mathcal{M}} \subseteq S^{\mathcal{M}}$ such that $(m, v) \in S^{\mathcal{M}}$ if $m \in M_0$ and $v \in V_0$;
- $\Sigma^{\mathcal{M}} = \Sigma$ is the set of labels;
- $\Delta^{\mathcal{M}} \subseteq S^{\mathcal{M}} \times \Sigma^{\mathcal{M}} \times S^{\mathcal{M}}$ is the set of transitions such that $((m, v), a, (m', v')) \in \Delta^{\mathcal{M}}$ if there exists a transition $\delta = (m, g, a, j, m') \in \Delta$ such that the current valuation v satisfies the guard of δ, i.e. $v \in [\![g]\!]$; the pair of current and next valuations (v, v') satisfies the jump constraint of δ, i.e. $(v, v') \in [\![j]\!]$; and the next valuation satisfies the invariant of the target mode of δ, i.e. $v' \in [\![I(m')]\!]$.

Let us consider an example of the syntax and semantics of an extended finite state machine.

Example 3 (Modulo-4 counter): Let us consider a modulo-4 counter with reset and pause functionality shown in Figure 2. This extended finite state machine $\mathcal{M} = (M, M_0, \Sigma, X, \Delta, I, V_0)$ has two control modes $M = \{\text{count}, \text{pause}\}$ with count being the initial mode. The variable x is the only variable, while the set of action is $\Sigma = \{tick, on, pause\}$ where tick, on, and pause stand for clock-tick, start-counting, and pause-counting actions, respectively. While drawing an extended finite state machine, we depict modes by rounded rectangles and transitions by arrows connecting the modes labeled by a triplet (g, a, j) showing the guard, the action, and the jump predicate of the transition. For example the transition (count, $x = 3$, *tick*, $x' = 0$, count) is shown in the Figure 2 as a self-loop labeled with $(x = 3, tick, x' = 0)$ on the mode labeled count. It is straightforward to see that the extended finite state machine in Figure 2 models a modulo-4 counter

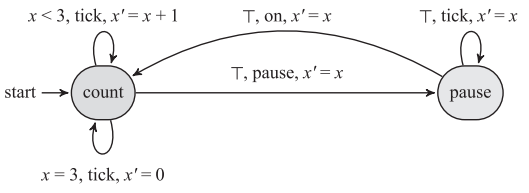


Figure 2: An EFSM description of a mod-4 counter with reset and pause.

with reset and pause. The corresponding state transition graph is shown in the Figure 1.

In the rest of the article, to minimize clutter, we will omit the guard if it is the predicate $\top$, and we omit the jump predicates specifying that the value of a variable remains unchanged, i.e. predicates of the form $x' = x$.

2.2 *Continuous dynamical systems*

For the purpose of this article, a continuous dynamical system is a finite set of continuous variables along with a set of ordinary differential equations characterizing the dynamics or the flow of these variables as a function of time. We represent the flow of a continuous dynamical system using a flow function $F : \mathbb{R}^{|X|} \to \mathbb{R}^{|X|}$ characterizing the system of ordinary differential equations:

$$\dot{X} = F(X) \tag{1}$$

where, following Newton's dot notation for differentiation, $\dot{X}$ represents the set of first-order derivatives of the variables in the set X. Information about the higher-order derivatives can be represented using only first-order derivatives introducing auxiliary variables. For example the second-order differential equation $\ddot{\theta} + (g/\ell)\sin(\theta) = 0$ can be written as a system of first-order differential equations $\dot{\theta} = y, \dot{y} = -(g/\ell)\sin(\theta)$. Formally, a continuous dynamical system is defined in the following manner.

Definition 4 (Continuous Dynamical System): A *continuous dynamical system* is a tuple $\mathcal{M} = (X, F, v_0)$ such that

- X is a finite set of real-valued *variable*,
- $F : \mathbb{R}^{|X|} \to \mathbb{R}^{|X|}$ is the flow function characterizing the the set of ordinary differential equation $\dot{X} = F(X)$, and
- $v_0 \in \mathbb{R}^{|X|}$ is the initial valuation.

A *run* of a continuous dynamical system $\mathcal{M} = (X, F, v_0)$ is given as a solution to the system of differential equations (1) with initial valuation v_0. Let a differentiable function $f : \mathbb{R}_{\geq 0} \to \mathbb{R}^{|X|}$ be a solution to (1), that provides the valuations of the variables as a function of time, such that:

$$f(0) = v_0$$
$$\dot{f}(t) = F(f(t)) \text{ for every } t \in \mathbb{R}_{\geq 0},$$

where $\dot{f} : \mathbb{R}_{\geq 0} \to \mathbb{R}^{|X|}$ is the time derivative of the function f. We call such a function f a run of the continuous dynamical system $\mathcal{M}$. Since, in general, a solution of (1) may not exist or may not be unique, a run of a continuous dynamical system may not exist or may not be unique.[74] To ensure the existence and the uniqueness of the run we enforce Lipschitz-continuity[b] assumption on F. The following result states the existence and uniqueness of the set of ordinary differential equations (1) under Lipschitz-continuity assumption.

Theorem 1 (*Picard-Lindelöf Theorem*):[90] If a function F: $\mathbb{R}^{|X|} \to \mathbb{R}^{|X|}$ is Lipschitz-continuous then the differential equation $\dot{X} = F(X)$ with initial valuation $v_0 \in \mathbb{R}^{|X|}$ has a unique solution $f : \mathbb{R}_{\geq 0} \to \mathbb{R}^{|X|}$ for all $v_0 \in \mathbb{R}^{|X|}$.

In addition, Lipschitz-continuity offers the following advantage while numerically simulating an approximate solution to the differential equations (1).

Theorem 2 (Stability wrt initial valuation):[74] Let F be a Lipschitz-continuous function with constant $K > 0$ and let $f : \mathbb{R}_{\geq 0} \to \mathbb{R}^{|X|}$ and f': $\mathbb{R}_{\geq 0} \to \mathbb{R}^{|X|}$ be solutions to the differential equation $\dot{X} = F(X)$ with initial valuation $v_0 \in \mathbb{R}^{|X|}$ and $v'_0 \in \mathbb{R}^{|X|}$, respectively. Then, for all $t \in \mathbb{R}_{\geq 0}$ we have that $||f(t) - f'(t)|| \leq ||v - v_0|| e^{Kt}$.

This theorem implies that, under Lipschitz-continuous assumption on the flow function F, any two runs whose initial valuations are close to one-another remain close as the time progresses. Since it is not always possible to analytically solve differential equations, this property permits us to numerically simulate the behaviour of continuous dynamical system using approximation methods, e.g. Euler's method or Runge-Kutta method, that are readily available in tools such as Matlab[79] and Mathematica.[98]

Example 4 (Simple Pendulum): Consider a simple pendulum shown in Figure 3 and its the motion equations:

$$\dot{\theta} = y,$$
$$\dot{y} = -(g/\ell)\sin(\theta),$$

with initial valuations $(\theta, y) = (\theta_0, 0)$. To analytically solve these equations let us assume small enough angular displacement θ and $\sin(\theta) \approx \theta$. Now the equations simplify to

$$\dot{\theta} = y \text{ and } \dot{y} = -(g/\ell)\theta.$$

Hence our continuous dynamical system is $\mathcal{M} = (X, F, v_0)$ where $X = \{\theta, y\}$, F is such that $F(\dot{\theta}) = y$ and $F(\dot{y}) = -(g/\ell)\theta$ and $v_0 = (\theta_0, 0)$. The solution for these differential equations is

$$\theta(t) = A\cos(Kt) + B\sin(Kt)$$
$$y(t) = -AK\sin(Kt) + BK\cos(Kt),$$

[b] We say that a function $F : \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz-continuous if there exists a constant K > 0, called the Lipschitz constant, such that for all x, y $\in \mathbb{R}^n$ we have that $||F(x) - F(y)|| < K ||x - y||$.

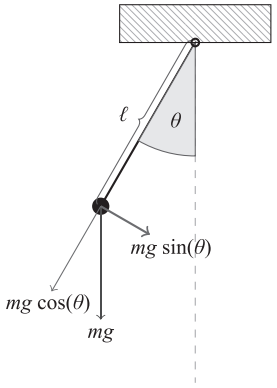
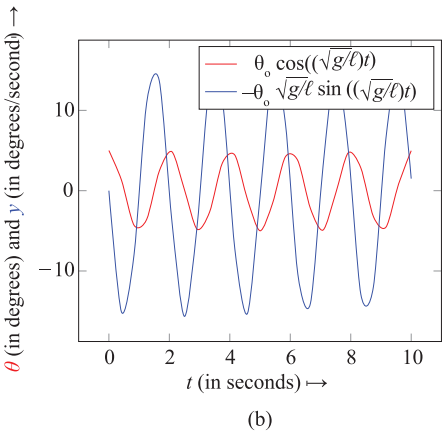


Figure 3: An idealized pendulum with length *l* and mass *m*.



(b)

Figure 4: The variables θ (angle displacement) and y (angular velocity) are plotted with respect to the time for a pendulum with l = 1 meter with θ_0 = 5 degrees.

where $K = \sqrt{g/\ell}$. Substituting $\theta(0) = \theta_0$ and $y(0) = 0$ from the initial valuation, we get that $A = \theta_0$ and $B = 0$. Hence the unique run of the pendulum system can be given as the function $f: \mathbb{R}_{\geq 0} \to \{\theta, y\}$ as $t \mapsto (\theta_0 \cos(Kt), -\theta_0 K \sin(Kt))$. Figure 4 shows the change in valuations of the variables θ and y as a function of time.

2.3 *Hybrid dynamical systems*

In the previous two subsections we discussed modeling of purely discrete and purely continuous dynamical systems. We saw that in a discrete dynamical system the state of the system changes during a discrete transition where it "jumps" (see Figure 5) to the new value governed by the transition relation, while in a continuous system the state of the system continuously "flows" (see Figure 5) in a fashion governed by ordinary differential equations. Hybrid systems share their properties with both discrete as well as continuous systems, as their state progresses with time in both discrete jumps as well as continuous flows. In this section we present hybrid automata, a combination of extended finite state machines and continuous dynamical systems, where in every control mode the dynamics of the variables of the system can be specified using ordinary differential equations.

Definition 5 (Hybrid Automata: Syntax): A hybrid automaton is a tuple $\mathcal{H} = (M, M_0, \Sigma, X, \Delta, I, F, V_0)$ where:

- M is a finite set of control *modes* including a distinguished initial set of control modes $M_0 \subseteq M$,
- Σ is a finite set of *actions*,
- X is a finite set of real-valued *variables*,
- $\Delta \subseteq M \times \text{pred}(X) \times \Sigma \times \text{pred}(X \cup X') \times M$ is the *transition relation*,
- $I: M \to \text{pred}(X)$ is the mode-invariant function,
- $F: M \to (\mathbb{R}^{|X|} \to \mathbb{R}^{|X|})$ is the mode-dependent flow function characterizing the flow for each mode $m \in M$ as the set of ODEs $\dot{X} = F(m)(X)$, and
- $V_0 \in \text{pred}(X)$ is the set of initial valuations.

To ensure existence of unique solutions of the ODEs in flow functions, we assume that for each mode $m \in M$ the flow function $F(m)$ is Lipschitz-continuous.

Just like in an extended finite state machine, a configuration of a hybrid automaton is a tuple (m, v) where $m \in M$ is a mode and $v \in \mathbb{R}^{|X|}$ is a variable valuation. For a Lipschitz-continuous flow function $F: M \to (\mathbb{R}^{|X|} \to \mathbb{R}^{|X|})$, a valuation $v \in \mathbb{R}^{|X|}$, a mode $m \in M$, and a time delay $t \in \mathbb{R}_{\geq 0}$ we define $(v \oplus_{F(m)} t)$ for the unique valuation $f(t)$ where f is the unique run of the continuous dynamical system $(X, F(m), v)$. For a jump predicate $j \in \text{pred}(X \cup X')$ and valuation v we define $v[j]$ for the set of valuations $v' \in \mathbb{R}^{|X|}_{\geq 0}$ such that $(v, v') \in j$.

The execution of a hybrid automaton begins in an initial configuration (m_0, v_0) where $m_0 \in M_0$ is an initial mode and $v_0 \in V_0$ is an initial valuation satisfying $v_0 \in [\![I(m_0)]\!]$. The system stays in a mode for some time, say $t_1 \in \mathbb{R}_{\geq 0}$, and while the system stays in a control mode m the valuation of the variables changes according to ODE specified by the flow $F(m)$ of the corresponding mode. After spending $t_1 \in \mathbb{R}_{\geq 0}$ time in mode m_0 an enabled transition (m_0, g, a, j, m_1) is non-deterministically chosen and executed. Notice that we say that a transition (m_0, g, a, j, m_1) is enabled if $(v_0 \oplus_{F(m_0)} t_1) \in [\![g]\!]$ and all the intermediate valuations that system passes through from v_0 to $(v_0 \oplus_{F(m_0)} t_1)$ satisfy the invariant of the mode m_0, i.e. for all $t \in [0, t_1]$ we have that $(v_0 \oplus_{F(m_0)} t) \in [\![I(m_0)]\!]$. After executing

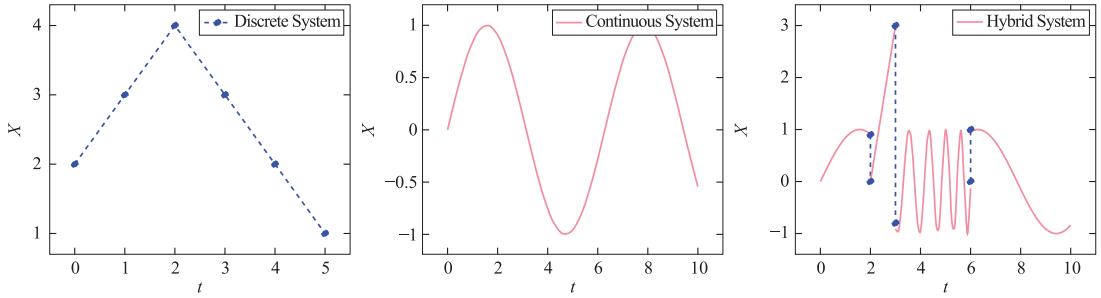


Figure 5: Runs of discrete, continuous, and hybrid systems.

the transition (m_0, g, a, j, m_1) the state of the system jumps to a new configuration (m_1, ν_1) such that $\nu_1 \in [\![I(m_1)]\!]$ and $\nu_1 \in (\nu_0 \oplus_{F(m_0)} t_1)[j]$. The system continues its operation in a similar manner from the resulting configuration (m_1, ν_1). We can formalize this semantics using a (uncountably infinite) state transition graph.

Definition 6 (Hybrid Automata: Semantics): The semantics of a hybrid automaton $\mathcal{H} = (M, M_0, \Sigma, X, \Delta, I, F, V_0)$ is given as a state transition graph $T^{\mathcal{H}} = (S^{\mathcal{H}}, S_0^{\mathcal{H}}, \Sigma^{\mathcal{H}}, \Delta^{\mathcal{H}})$ where:

- $S^{\mathcal{H}} \subseteq (M \times \mathbb{R}^{|X|})$ is the set of configurations of $\mathcal{H}$ such that for all $(m, \nu) \in S^{\mathcal{H}}$ we have that $\nu \in [\![I(m)]\!]$;
- $S_0^{\mathcal{H}} \subseteq S^{\mathcal{H}}$ s.t. $(m, \nu) \in S_0^{\mathcal{H}}$ if $m \in M_0$ and $\nu \in V_0$;
- $\Sigma^{\mathcal{H}} = \mathbb{R}_{\geq 0} \times \Sigma$ is the set of labels;
- $\Delta^{\mathcal{H}} \subseteq S^{\mathcal{H}} \times \Sigma^{\mathcal{H}} \times S^{\mathcal{H}}$ is the set of transitions such that $((m, \nu), (t, a), (m', \nu')) \in \Delta^{\mathcal{H}}$ if there exists a transition $\delta = (m, g, a, j, m') \in \Delta$ such that
  - $(\nu \oplus_{F(m)} t) \in [\![g]\!]$;
  - $(\nu \oplus_{F(m)} \tau) \in [\![I(m)]\!]$ for all $\tau \in [0, t]$;
  - $\nu' \in (\nu \oplus_{F(m)} t)[j]$; and
  - $(\nu' \in [\![I(m')]\!])$.

Example 5 (A bouncing ball): In Figure 6 we model a bouncing ball using a hybrid automaton with one control mode m and two variables: the variable x_1, representing the vertical position of the ball, and the variable x_2, representing the vertical velocity of the ball.

The differential equations governing the free fall of the ball can be given using Newton's law of motion as $\dot{x}_1 = x_2$ and $\dot{x}_2 = -g$. The valuations of the variables flow according to these equations until the ball comes in the contact with ground, and at that time it reverses the direction of its velocity, while losing some energy proportional to its restitution coefficient c, i.e. after the impact we have $x_1' = x_1$ and $x_2' = -cx_2$. Observe that the bouncing ball system is a hybrid system since its dynamics involve both flows and jumps. The continuous dynamics of the system is captured using flow function of the unique mode m, while the jump is modeled with

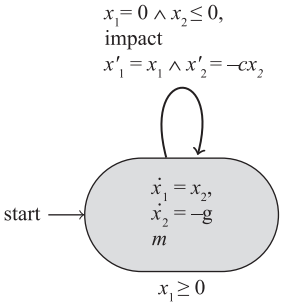


Figure 6: A hybrid automaton modeling the dynamics of a bouncing ball.

the discrete transition labeled *impact*. For the starting valuation we assume $x_1 = \ell$ meters and $x_2 = 0$. Formally the hybrid automata $H = (M, M_0, \Sigma, X, \Delta, I, F, V_0)$ models the bouncing ball where:

- $M = M_0 = \{m_0\}$,
- $\Sigma = \{\text{impact}\}$,
- $X = \{x_1, x_2\}$,
- Δ contains the following transition $(m, x_1 = 0 \wedge x_2 \leq 0, \text{impact}, x_1' = x_1 \wedge x_2' = -cx_2, m)$,
- $I(m) = x_1 \geq 0$,
- $F(m) = \dot{x}_1 = x_2 \wedge \dot{x}_2 = -g$, and
- $V_0 = \{(\ell, 0)\}$.

The transition diagram corresponding to this automaton is shown in Figure 6. The transition diagram of a hybrid automaton follows the similar conventions as that of an extended finite state machine, with the exception of flow conditions. We write flow conditions of a mode inside the rounded rectangle representing the mode.

Now let us explain the unique run of the system starting from the configuration $(m, (\ell, 0))$. The solution to ODE corresponding to the flow function is

$$x_1(t) = -\frac{1}{2}gt^2 + Ct + D \text{ and } x_2(t) = -gt + C \qquad (2)$$

For the initial configuration is $(m, (\ell, 0))$ solving (2) we get $C = 0$ and $D = \ell$. Hence from

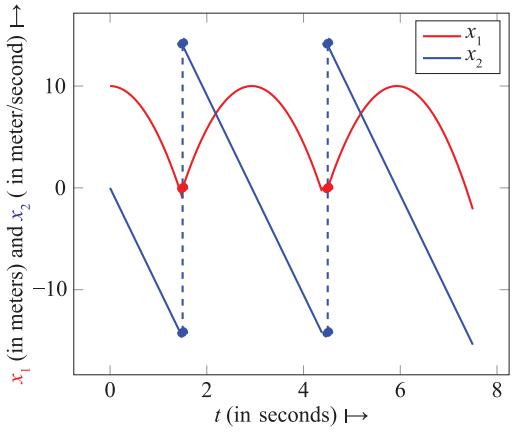


Figure 7: A run of the system where the initial vertical position is ℓ = 10 meters and the coefficient of restitution c = 1.

$(m, (\ell, 0))$ system flows according to the equations $x_1(t) = -\frac{1}{2}gt^2 + \ell$ and $x_2(t) = -gt$. According to these equations the value of variable x_1 continue to fall for the next $t_1 = \sqrt{2\ell/g}$ time units when x_1 becomes 0, and the transition *impact* becomes available and must be taken (since the invariant of the mode requires x_1 to be non-negative). Immediately before taking the transition the configuration is $(0, -gt_1)$. Using our notations we can write it as $(0, -gt_1) = (\ell, 0) \oplus_{F(m)} t_1$.

After taking the transition *impact* this valuation changes according to the jump function $x_1' = x_1 \wedge x_2' = -cx_2$ resulting in the new valuation $(0, cgt_1)$. Again, in our notation we write $(0, cgt_1) \in (0, -gt_1)[x_1' = x_1 \wedge x_2' = -cx_2]$. The run of the system, so far, can be written as $\langle (m, (\ell, 0)), (t_1, \text{impact}), (m, (0, cgt_1)) \rangle$. Now from the configuration (m, $(0, cgt_1)$) the system can flow continuously according to $F(m)$. Solving (2) for this initial valuation we get $C = cgt_1$ and $D = 0$. Hence from $(m, (0, cgt_1))$ the system flows according to the equations $x_1(t) = -\frac{1}{2}gt^2 + cgt_1 t$ and $x_2(t) = -gt + cgt_1$ for the next $t_2 = 2ct_1$ time units till it reaches the valuation $x_1 = 0$ (the ball hits the ground again). At this point the resulting configuration will be $(0, -cgt_1)$ and after the transition the configuration will be $(0, c^2gt_1)$. The system continues in this fashion forever and realizes the following infinite run of the system:

$$\begin{aligned} \langle (m,(\ell,0)), \quad & (t_1, \text{impact}), \quad (m,(0, cgt_1)), \\ & (2ct_1, \text{impact}), \quad (m,(0,\ c^2gt_1)), \\ & (2c^2t_1, \ \text{impact}), \quad (m,(0,\ c^3gt_1)),\ldots\rangle, \end{aligned} \tag{3}$$

where $t_1 = \sqrt{2\ell/g}$. The first two transitions of the run for $\ell = 10$ and $c = 1$ are shown in Figure 7.

For a given run $r = \langle (m_0, v_0), (t_1, a_1), (m_1, v_1), \ldots \rangle$ of a hybrid automaton we define its time $T(r)$ as

$$T(r) = \sum_{i=1}^{\infty} t_i.$$

We say that a run r *time-diverging* if $T(r) = \infty$. For an example of a time-diverging run consider (3) for $c = 1$ as shown in Figure 7 where time between every consecutive transition is $2\sqrt{2\ell/g}$. The infinite run in this example seems natural since we assume the restitution coefficient $c = 1$, and under this unrealistic situation we expect the ball to bounce indefinitely. However, given the generality of the model of hybrid automata the time divergence of a run is not always guaranteed. As an example consider again the bouncing ball system now with restitution coefficient $0 < c < 1$. In this case the time of the run (3), $T(r) = t_1(1 + c)/(1 - c)$ is finite for any $0 < c < 1$. Runs that are not time-diverging, on an intuitive level, are not physically realizable since they execute infinitely many discrete transitions in a finite amount of time. Assuming the possibility of realizing infinitely many discrete actions in a finite time often lead to paradoxical situations, commonly known as Zeno's paradoxes, and the runs that do not diverge also go by the name of Zeno runs. We call a hybrid automaton *non-Zeno* if it does not permit any Zeno run. We will later see that the ability of hybrid automata to model Zeno runs often cause difficulty in their analysis.

2.4 *Composition of a network of hybrid automata*

While modeling a complex hybrid system using a hybrid automaton, it is often convenient to represent various components of the system as a network of hybrid automata $c = \{\mathcal{H}^1, \mathcal{H}^2, \ldots, \mathcal{H}^n\}$ that communicate with each other using shared variables and action. Specifying a system as a composition of various subsystems offer two main advantages, namely abstraction and modularity. The first advantage (abstraction) is that it allows the system designer to concentrate on the details of one subsystem at a time without getting overwhelmed by the complexity of the interaction of this subsystem with other. The second advantage (modularity) is that in a system designed in this fashion, it is easy to add, remove, and modify subsystems. The semantics of such a network can also be given as a single hybrid automaton $\mathcal{H}$, called the product automaton of $\mathcal{C}$, whose states are products of states of individual component automata. We define this construction as the following.

Definition 7 (Composition): Let $\mathcal{C} = \{\mathcal{H}^1, \mathcal{H}^2, \ldots, \mathcal{H}^n\}$ be a network of hybrid automata where for each $1 \leq i \leq n$ let $\mathcal{H}^i$ be $(M^i, M_0^i, \Sigma^i, X^i, \Delta^i, I^i, F^i, V_0^i)$. For an action $a \in \cup_{i=1}^n \Sigma_i$ we define $E(a) \stackrel{\text{def}}{=} \{i : a \in \Sigma^i\}$. The product automata $\mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \ldots \mathcal{H}_n$ of $\mathcal{C}$ is defined as a hybrid automaton $H = (M, M_0, \Sigma, X, \Delta, I, F, V_0)$ where

- $M = M^1 \times M^2 \times \ldots M^n$,
- $M_0 = M_0^1 \times M_0^2 \times \cdots M_0^n$,
- $\Sigma = \Sigma^1 \cup \Sigma^2 \cup \ldots \Sigma^n$,
- $X = X^1 \cup X^2 \cup \ldots X^n$,
- $\Delta \subseteq (M \times \text{pred}(X) \times \Sigma \times \text{pred}(X \cup X') \times M)$ is defined s.t. $((m_1,...,m_n), g, a, j, (m_1',...,m_n')) \in \Delta$ if and only if for all $i \notin E(a)$ we have that $m_i = m_i'$ and for all $i \in E(a)$ there exists a transition (m_i, g_i, a, j_i, m_i') such that $g = \wedge_{i \in E(a)} g_i$ and $j = \wedge_{i \in E(a)} j_i$.
- I is such that $I(m_1,...,m_n) = \wedge_{i=1}^n I^i(m_i)$;
- F is such that $F(m_1, \ldots, m_n)(x) = F^i(m_i)(x)$ if $x \in X^i$; and
- V_0 is such that $V_0 = \wedge_{i=1}^n V_0^i$.

As an example of modeling a system using a composition of a network of hybrid automata, we consider the job-shop scheduling problem modeled as a collection of hybrid automata. In the next section, we show that solving the job-shop problem reduces to solving a verification problem (reachability) over the resulting hybrid automata.

Example 6 (Job-shop Scheduling Problem): The job-shop scheduling problem is an important optimization problem studied frequently in both computer science as well as in operations research. It consists of a finite set $\mathbb{J} = \{j_1, \ldots, j_n\}$ of jobs to be processed on a finite set $\mathbb{M} = \{m_1, \ldots, m_k\}$ of machines. There is a strict precedence requirement between the jobs given as a strict partial order $\prec$ over the set of jobs in $\mathbb{J}$. A mapping $\zeta : \mathbb{J} \to 2^{\mathbb{M}}$ specifies the set of machines where a job can be executed, while the function $\delta : \mathbb{J} \to \mathbb{R}_{\geq 0}$ specify the time duration of a job. We can model the job-shop scheduling problem using a network of hybrid automata where each job and each machine is specified using a separate hybrid automaton. We have the following constraints on the job execution: i) a job j can be executed iff all jobs in its precedence, $j{\downarrow} = \{j' : j' \prec j\}$, have terminated; 2) each machine $m \in \mathbb{M}$ can process atmost one job at a time; and 3) a job, once started, cannot be preempted.

Modeling Jobs. We model each job $j_i \in \mathbb{J}$ as a hybrid automaton H_i with three modes U_i (unscheduled), S_i (scheduled), and F_i (finished) where U_i being the initial mode. With each automaton H_i we associate two variables: variable x_i, measuring the time while the job j_i is being executed on a machine; and variable done_i with values 0 and 1 denoting whether the job is unfinished (0) or finished (1). For each job j_i the initial valuation of variable x_i is 0, while the valuation for $\text{done}_i = 0$. For each mode $m \in \{U_i, S_i, F_i\}$ we have that $F(m)(\text{done}_i) = 0$ and $F(S_i)(x_i) = 1$ (to measure time spent during processing of the job) and $F(U_i)(x_i) = 0$ and $F(F_i)(x_i) = 0$. The transition from a mode U_i to S_i with action begin_i is guarded by the condition that all of the preceding jobs according to $\prec$ has been finished, i.e. $\wedge_{k:k \prec i}(\text{done}_k = 1)$. The transition from a mode S_i to F_i with action finish_i is guarded by predicate $\text{done}_i' = \delta(j_i)$ specifying that job j_i takes exactly $\delta(j_i)$ time units, and the jump of this transition includes $\text{done}_j' = 1$.

Modeling Machines. We model each machine $m_i \in \mathbb{M}$ using a hybrid automaton with no variable and $k + 1$ modes where k is the number of jobs that can be scheduled to this machine: there is a unique mode I_i (idle), and for each job j_j that can be scheduled to this machine, i.e. $m_i \in \zeta(j_i)$ there is a mode $P_{i,j}$ (corresponding to processing job $j_j \in \mathbb{J}$ on machine $m_i \in \mathbb{M}$). For each mode $P_{i,j}$ there is a transition from I_i to $P_{i,j}$ with action begin_j and a transition from $P_{i,j}$ to I_i with action finish_j denoting the scheduling and the finishing, respectively, of job j_j on machine m_i. Since there are no variables associated with these automata the guard and the jump predicate of these transitions is simply $\top$.

As an example of such modeling, consider the job-shop problem with $J = \{j_1, j_2\}$, $M = \{m_1\}$, $\zeta(j_1) = \zeta(j_2) = m_1$, $j_1 \prec j_2$, and $\delta(j_1) = 3$ and $\delta(j_2) = 4$. Figure 8 shows hybrid automata $\mathcal{H}_{j1}$, $\mathcal{H}_{j2}$, and $\mathcal{H}_{m1}$ corresponding to the jobs j_1 and j_2, and the machine m_1 respectively. This figure also shows the composition of these automata $\mathcal{H}_{j1} \otimes \mathcal{H}_{j2} \otimes \mathcal{H}_{m1}$ representing the hybrid automata corresponding to the complete job-shop problem.

3 Formal Verification of Hybrid Systems

Formal modeling and verification of systems is the set of techniques that employ rigorous mathematical reasoning to analyze properties of a system. In this article we concentrate on model checking—a formal verification framework introduced by Clarke, Sifakis and Emerson[47]—that, given a formal description of a system and its specification, systematically verifies whether the specification holds for the system model. Since, by definition the states of a dynamical system changes with time, classical propositional logic is not sufficient to reason with temporal properties of such dynamical systems. Temporal logics extend propositional or predicate logics by modalities that are useful to capture the

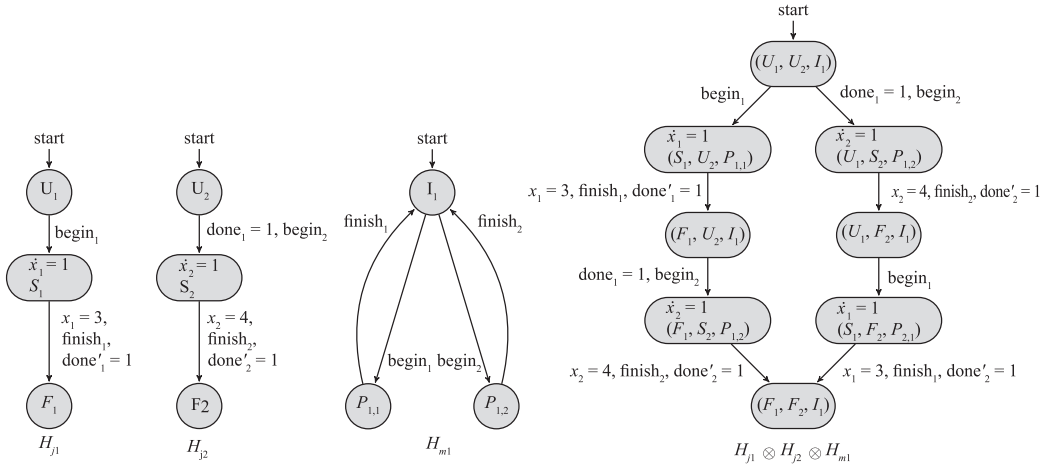


Figure 8: Network of hybrid automata $\mathcal{H}_{j1}$, $\mathcal{H}_{j1}$, and $\mathcal{H}_{m1}$ corresponding to jobs j_1 and j_2, and a machine m_1, and their product automata $\mathcal{H}_{j1} \otimes \mathcal{H}_{j2} \otimes \mathcal{H}_{m1}$.

change of behaviour of a system over time. Manna and Pnueli[1,77] were the first one to propose and promote the use of temporal logic to specify properties of dynamical systems in the context of system verification. *Linear temporal logic* (LTL),[77] *computation tree logic* (CTL) and its generalization CTL*,[50,77] and modal μ-calculus[53] are some of the popular temporal logics used for the system specification. Timed and weighted extensions of these logics e.g. metric temporal logics (MTL and MITL),[83] duration calculus (DC),[43] and weighted logics[31,37] have also been proposed to specify more involved quantitative properties of hybrid dynamical systems.

In this article we limit the discussion to simple qualitative properties of hybrid systems that broadly can be classified into the following two categories:[76]

- The *reachability or guarantee properties*, that ask whether the system can reach a configuration satisfying certain property p? (symbolically, we write $\Diamond p$ and we say *eventually* p); and
- The *safety properties* that ask whether the system can stay forever in configurations satisfying certain property p? (symbolically, we write $\Box p$ and we say *always* or *globally* p).

The linear temporal logic, LTL, provides a formal language to specify more involved nesting of such properties with ease. We begin this section (Section 3.1) by introducing Kripke structures that provide a way to mark states of the hybrid automata with properties of interest, and present the syntax and semantics of LTL that are interpreted over Kripke structures. In Section 3.2 we formally introduce LTL model-checking problem for hybrid automata, and show that in general this problem is undecidable. On a positive note, in Section 3.3, we show that LTL model-checking can be algorithmically solved for finite Kripke structures. Finally, in Section 3.4 we introduce the notion of bisimulation, and show that the existence of a finite bisimulation implies the decidability of LTL model-checking problem.

3.1 *Hybrid Kripke structures and linear temporal logic*

The formal specification of the underlying system begins by identifying key properties of interests (called atomic propositions) regarding the states of the system under verification. Kripke structures provide a way to label the states of state-transition graphs with such atomic propositions, and the linear temporal logic specifies properties of the sequence of the truth values of these propositions, called *traces*, for the runs of corresponding transition system. Hence, before we introduce linear temporal logic LTL we need to introduce Kripke structures and their corresponding hybrid extension, and the concept of traces.

Defintion 8 (Hybrid Kripke Structure): A *Kripke Structure* is a tuple $(\mathcal{T}, P, L)$ where:

- $\mathcal{T} = (S, S_0, \Sigma, \Delta)$ is a state transition graph,
- P is a finite set of *atomic propositions*, and
- $L : S \to 2^P$ is a labeling function that labels every state with a subset of P.

Similarly, we define a *Hybrid Kripke Structure* as a tuple $(\mathcal{H}, P, L)$ where:

- $\mathcal{H} = (M, M_0, \Sigma, X, \Delta, I, F, V_0)$ is a hybrid automaton,
- P is a finite set of *atomic propositions*, and

– $L : M \to 2^P$ is a labeling function that labels every mode with a subset of P.

Observe that the semantics of a hybrid Kripke structure is a Kripke structure.

Let us fix a hybrid Kripke structure $(\mathcal{H}, P, L)$ and its semantics Kripke structure $([\![\mathcal{H}]\!], P, L)$ for the rest of this section. When the set of propositions and labeling function is clear from the context, we use the terms state transition graph and Kripke structure, and the terms hybrid Kripke structure and hybrid automaton interchangeably.

Given a hybrid Kripke structure $(\mathcal{H}, P, L)$ and an infinite run $r = \langle (m_0, v_0), (t_1, a_1), (m_1, v_1), \ldots, (m_n, v_n), \ldots \rangle$ of $\mathcal{H}$, we define a trace corresponding to r, denoted as $Trace(r)$, as the sequence $\langle L(m_0), L(m_1), L(m_2), \ldots L(m_n), \ldots \rangle$. Let $Trace(\mathcal{H}, P, L)$ be the set of traces of the Hybrid Kripke Structure $\mathcal{H}$. For a trace $\sigma = \langle P_0, P_1, \ldots, P_n, \ldots \rangle \in Trace(\mathcal{H}, P, L)$ we write $\sigma[i] = \langle P_i, P_{i+1}, \ldots, \rangle$ for the suffix of the trace starting at the index $i \geq 0$.

Now we are in a position to define the syntax and semantics of linear temporal logic.

Definition 9 (Linear Temporal Logic (Syntax)): The set of valid LTL formulas over a set P of atomic propositions can be inductively defined as the following:

– $\top$ and $\bot$ are valid LTL formulas;
– if $p \in P$ then p is a valid LTL formula;
– if ϕ and ψ are valid LTL formulas then so are $\neg\phi$, $\phi \wedge \psi$ and $\phi \vee \psi$;
– if ϕ and ψ are valid LTL formulas then so are $\bigcirc\phi$, $\Diamond\phi$, $\Box\phi$ and $\phi \mathcal{U} \psi$.

We often use $\phi \Rightarrow \psi$ as a shorthand for $\neg\phi \vee \psi$. Before we define the semantics of LTL formula formally, let us give an informal description of the temporal operators $\bigcirc$, $\Diamond$, $\Box$, and $\mathcal{U}$. LTL formulas are interpreted over traces of (Hybrid) Kripke structures. The formula $\bigcirc\phi$, read as next ϕ, holds for a trace $\sigma = \langle P_0, P_1, P_2, \ldots \rangle$ if ψ holds for the trace $\sigma[1]$. The formula $\Diamond\phi$, read as eventually ϕ, holds for a trace $\sigma = \langle P_0, P_1, P_2, \ldots \rangle$ if there exists $i \geq 0$ such that the formula ψ holds for the trace $\sigma[i]$. The formula $\Box\phi$, read as globally or always ϕ, holds for a trace $\sigma = \langle P_0, P_1, P_2, \ldots \rangle$ if for all $i \geq 0$ the formula ψ holds for traces $\sigma[i]$. Finally, the formula $\phi \mathcal{U} \psi$, read as ϕ until ψ, holds for a trace $\sigma = \langle P_0, P_1, P_2, \ldots \rangle$ if there is an index i such that ψ holds for the trace $\sigma[i]$, and for every index j before i the formula ϕ holds for the trace $\sigma[j]$, i.e the formula ϕ holds until formula ψ holds.

Definition 10 (Linear Temporal Logic (Semantics)): For a trace $\sigma = \langle P_0, P_1, P_2, \ldots \rangle$ of a (Hybrid) Kripke structure we write $\sigma \models \phi$ to say that the trace σ satisfies the formula ϕ. The satisfaction of LTL formulas is defined as follows:

– $\sigma \vDash \top$ and $\sigma \nvDash \bot$;
– $\sigma \vDash p$ if $p \in P_0$;
– $\sigma \vDash \neg\phi$ if $\sigma \nvDash \phi$;
– $\sigma \vDash \phi \wedge \psi$ if $\sigma \vDash \phi$ and $\sigma \vDash \psi$;
– $\sigma \vDash \phi \vee \psi$ if $\sigma \vDash \phi$ or $\sigma \vDash \psi$;
– $\sigma \vDash \bigcirc\phi$ if $\sigma[1] \vDash \phi$;
– $\sigma \vDash \Diamond\phi$ if there exists $i \geq 0$ such that $\sigma[i] \vDash \phi$;
– $\sigma \vDash \Box\phi$ if for all $i \geq 0$ we have that $\sigma[i] \vDash \phi$; and
– $\sigma \vDash \phi \mathcal{U} \psi$ if there exists $i \geq 0$ such that $\sigma[i] \vDash \psi$, and for all $0 \leq j < i$ or $\sigma[j] \vDash \phi$.

For a (hybrid) Kripke structure $(\mathcal{H}, P, L)$, and an LTL formula ϕ we say that $(\mathcal{H}, P, L) \vDash \phi$ if for all $\sigma \in Trace(\mathcal{H}, P, L)$ we have that $\sigma \vDash \phi$.

Lamport[72] observed that most of the system specifications can be classified in safety properties (*something will not happen*) and liveness properties (*something must happen*). Manna and Pnueli[76] further refined the class of specifications starting from reachability and safety properties to introduce a hierarchy of temporal properties using nesting of LTL operators, for instance

– The *recurrence properties* that ask whether the system can infinitely often visit configurations satisfying certain property p? (symbolically, we write $\Box\Diamond p$ and we say *infinitely often p*); and
– The *persistence properties* that ask whether the system visits configurations not satisfying a certain property p only finitely often? (symbolically, we write $\Diamond\Box p$ and we say *eventually always p*).

Some examples for expressing reachability, safety, and liveness properties using LTL are shown in the following example.

Example 7: As an example let us write LTL specifications for an elevator serving k different floors. Let op_i, fl_i and req_i be atomic propositions representing the situations that "the door at floor i is open", "the lift is at floor i and is not moving" and "there is a request for the lift to move to the ith floor" respectively. The following are some specifications in English and their LTL counterparts:

1. Reachability property: *The lift will visit the ground floor sometime.*

$$\phi_1 \stackrel{def}{=} \Diamond fl_0.$$

2. Safety property: *The door of the lift is never open at a floor if the lift is not present there.*

$$\phi_2 \stackrel{def}{=} \Box\left(\bigwedge_{i=0}^{k} (\neg fl_i \Rightarrow \neg op_i)\right)$$

3. Recurrence property: *The lift keeps coming back to the ground floor.*

$$\phi_3 \stackrel{def}{=} \Box(\neg fl_0 \Rightarrow \Diamond fl_0) \wedge \Box\Diamond fl_0.$$

4. Persistence property: *Eventually always a requested floor will be eventually served.*

$$\phi_4 \stackrel{def}{=} \Diamond\Box\left(\bigwedge_{i=0}^{k} (req_i \Rightarrow \Diamond fl_i)\right).$$

We refer the reader to[22,50,76,77] for a detailed overview of LTL for system specification.

3.2 *LTL model checking for hybrid automata*

LTL model-checking problem for hybrid automata can be formally stated in the following manner.

Definition 11 (LTL Model-Checking): Given a system modeled as a (Hybrid) Kripke structure $(\mathcal{H}, P, L)$, and a specification written as an LTL formula ϕ, the *LTL model-checking* problem is to decide whether all traces of $\mathcal{H}$ satisfy ϕ, i.e. $(\mathcal{H}, P, L) \models \phi$. Moreover, if the system does not satisfy the property give a counterexample (run of the system) violating the property.

Example 8: Consider the Kripke structure $\mathcal{T}$ shown in Figure 9 with set of atomic propositions $\{p, q\}$. We are depicting the labeling function by writing the set of propositions inside the state, and we omit other non-relevant details. Let us consider the LTL formulas $\phi_1 = \Diamond(p \wedge \neg q)$ and $\phi_2 = \Box q \vee \Diamond\Box p$. Observe that $\mathcal{T} \not\models \phi_1$ as is clear from the counterexample $r = \langle m_0, a, m_1, a, m_0, \ldots\rangle$ as it never visits the configuration satisfying $(p \wedge \neg q)$ as is clear from its trace $Trace(r) = \{q\}\{p, q\}\ \{q\}\ \{p, q\}$. On the other hand, it is easy to verify that $\mathcal{T}$ satisfies ϕ_2 as any run of $\mathcal{T}$ either never visits m_2 (and in that case satisfies $\Box q$, or it eventually visits m_2 and never leaves it (and thus satisfies $\Diamond\Box p$).

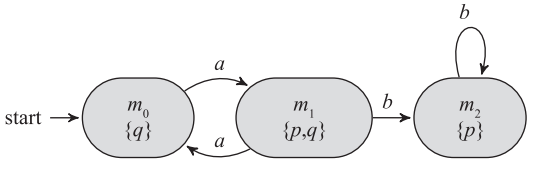


Figure 9: A Kripke structure $\mathcal{T}$.

Example 9 (Job-Shop Scheduling Revisited): Consider the job-shop scheduling problem modeled as a network of hybrid automata in Figure 8. Consider the atomic propositions j_1.finish and j_2.finish that are true only in modes F_1 and F_2. The counterexample produced in model-checking LTL property $\neg(\Diamond(j_1.\text{finish} \wedge j_2.\text{finish}))$ gives a valid schedule for the job-shop scheduling problem.

Next, we show that LTL model-checking problem for hybrid Kripke structures is undecidable. To prove this result, we show a reduction from a well-known undecidable problem of reachability (halting) for two-counter *Minsky machines*.[81]

A Minsky machine $\mathcal{A}$ is a tuple (L, C) where: $L = \{\ell_0, \ell_1, \ldots, \ell_n\}$ is the set of instructions. There is a distinguished terminal instruction ℓ_n called HALT. $C = \{c_1, c_2\}$ is the set of two *counters*; the instructions L are one of the following types:

1. (increment c) ℓ_i : $c = c + 1$; goto ℓ_k,
2. (test-and-decrement c) ℓ_i : if $(c > 0)$ then $(c = c - 1)$; goto ℓ_k else goto ℓ_m,
3. (Halt) ℓ_n : HALT.

where $c \in C$, $\ell_i, \ell_k, \ell_m \in L$.

A configuration of a Minsky machine is a tuple (ℓ, c, d) where $\ell \in L$ is an instruction, and c, d are natural numbers that specify the value of counters c_1 and c_2, respectively. The initial configuration is $(\ell_0, 0, 0)$. A run of a Minsky machine is a (finite or infinite) sequence of configurations $\langle k_0, k_1, \ldots\rangle$ where k_0 is the initial configuration, and the relation between subsequent configurations is governed by transitions between respective instructions. The run is a finite sequence if and only if the last configuration is the terminal instruction ℓ_n. Note that a Minsky machine has exactly one run starting from the initial configuration. The *halting problem* for a Minsky machine asks whether its unique run ends at the terminal instruction ℓ_n. It is well known[81] that the halting problem for two-counter Minsky machines is undecidable.

Theorem 3: The LTL model-checking problem for hybrid Kripke structures is undecidable.

Proof. Given a two counter machine $\mathcal{A}$, we construct a hybrid Kripke structure $\mathcal{H}$ and an LTL formula ϕ such that $\mathcal{H} \models \phi$ iff $\mathcal{A}$ halts. The modes of $\mathcal{H}$ are labeled with the labels l_i of instructions. There is a unique mode of $\mathcal{H}$ labeled with atomic proposition "HALT" which corresponds to the terminal instruction of $\mathcal{A}$. The increment, decrement and test instructions are encoded by suitable modules

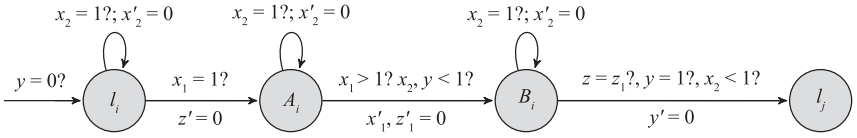


Figure 10: Module simulating l_i: increment c, goto l_j.

in $\mathcal{H}$. The variables of $\mathcal{H}$ are $X = \{x_1, x_2, y, z, z_1\}$ with $F(m)$ for all modes is defined as the following:

$$\dot{x}_1 = 1 \wedge \dot{x}_2 = 1 \wedge \dot{y} = 1 \wedge \dot{z} = 1 \wedge \dot{z}_1 = 2.$$

The initial mode is labeled by l_0, the label of the first instruction. The values of the counters *c, d* are encoded as $x_1 = \frac{1}{2^c}$ and $x_2 = \frac{1}{2^d}$. After the execution of each instruction, x_1, x_2 will contain the current values of counters *c, d* encoded in the above form. For instance, if we have $x_1 = \frac{1}{2^c}, x_2 = \frac{1}{2^d}$ before incrementing counter *c*, then at the end of simulating the increment instruction, we will have $x_1 = \frac{1}{2^{c+1}}$ and $x_2 = \frac{1}{2^d}$.

In Figure 10, we illustrate the case of the increment instruction l_i : increment *c* and goto l_j. The case for the decrement instruction is similar, and hence omitted. Mode l_i is entered with $y = 0$, $x_1 = \frac{1}{2^c}$ and $x_2 = \frac{1}{2^d}$. On entering mode A_i, we have $x_1 = 1, y = 1 - \frac{1}{2^c}, x_2 = \frac{1}{2^d} + (1 - \frac{1}{2^c})$ or $x_2 = 1 - \frac{1}{2^c} - \eta$ if $\frac{1}{2^d} + \eta = 1, \eta \leq 1 - \frac{1}{2^c}$ and $z = 0$. Mode B_i can be entered if x_2, $y < 1$ and $x_1 > 1$. Assume $k > 0$ units of time was spent at mode A_i. This gives $y = 1 - \frac{1}{2^c} + k, x_2 = \frac{1}{2^d} + (1 - \frac{1}{2^c}) + k$ (or $1 - \frac{1}{2^c} - \eta + k$, or $1 - \eta'$ if $1 - \frac{1}{2^c} - \eta + \eta' = 1, \eta' \leq k$), $z = k$, $x_1 = 0$, $z_1 = 0$ on entering mode B_i. We can reach mode l_j only if the values of z and z_1 are the same. Assume l units of time was spent at B_i. Then $z = k + l$, $z_1 = 2l$, $x_2 = \frac{1}{2^d} + (1 - \frac{1}{2^c}) + k + l, x_1 = l, y = 1 - \frac{1}{2^c} + k + l$. To satisfy the constraints $z = z_1, y = 1$, we have $k = 1$ and $k + l = 2k = \frac{1}{2^c}$ giving $x_1 = \frac{1}{2^{c+1}}, x_2 = \frac{1}{2^d}, y = 0$ at l_j.

The LTL formula $\phi = l_0 \wedge \Diamond$ HALT will be satisfied by $\mathcal{H}$ iff $\mathcal{A}$ halts. This shows that LTL model checking of hybrid Kripke structures is undecidable.

3.3 *LTL model-checking for finite Kripke structures*

As we discussed in previous section the LTL model-checking problem is undecidable for general hybrid automata. However, for finite Kripke structures Wolper, Vardi, and Sistla[99] developed an elegant automata-theoretic algorithm for solving the LTL model-checking problem. The algorithm exploits the connection between LTL formulas and a type of ω-automata—automata that extend the theory of finite automata to infinite inputs—called Büchi automata.[40,56] The syntax for the Büchi automata specifies a finite state transition graph $\mathcal{T}$ along with a set F of accepting states, and the semantics of Büchi automata restricts the set of valid runs to the runs of $\mathcal{T}$ that visit F infinitely often. In general Büchi automata are closed under all Boolean operations including union, intersection, and complementation, however deterministic variant of Büchi automata is not closed under complementation. Emptiness checking for Büchi automata can be decided efficiently (linear in time) by analyzing strongly connected components of $\mathcal{T}$.

The LTL model-checking problem exploits the following connection between linear temporal logic and Büchi automata.

Theorem 4 (LTL-to-Büchi Automata):[99] For every LTL formula ϕ we can effectively construct a finite (Büchi) automaton $\mathcal{A}_\phi$ (of size exponential in ϕ) such that words recognized by $\mathcal{A}_\phi$ are precisely the set of traces that satisfy ϕ.

Based on this result, the LTL model checking for a finite Kripke structure $\mathcal{K}$ can be performed in the following manner:

1. Construct a Büchi automaton $\mathcal{A}_{\neg\phi}$ corresponding to the negation of the LTL property.
2. Construct the composition $\mathcal{K} \otimes \mathcal{A}_{\neg\phi}$ of the Kripke structure $\mathcal{K}$ with the Büchi automaton $\mathcal{A}_{\neg\phi}$.
3. If the Büchi automaton $\mathcal{H} \otimes \mathcal{A}_{\neg\phi}$ is empty, then return "TRUE"
4. Else, return a lasso-shaped (a finite prefix followed by a cycle that contains an accepting state) infinite run accepted by $\mathcal{H} \otimes \mathcal{A}_{\neg\phi}$ as a counter-example.

The correctness of this algorithm follows from the observation that the set of traces for this composition $\mathcal{K} \otimes \mathcal{A}_{\neg\phi}$ characterize the set of traces that are generated by $\mathcal{K}$ that do not satisfy ϕ. Hence, the Kripke structure $\mathcal{K}$ satisfies the LTL property ϕ if and only if $\mathcal{H} \otimes \mathcal{A}_{\neg\phi}$ is empty.

Theorem 5 (LTL model-Checking for Finite Structures):[91] LTL model checking problem for finite Kripke structures is decidable in PSPACE.

LTL model-checking for finite Kripke structures is implemented by a number of mature tools, notably SPIN[92] and NuSMV,[82] and has been applied to a number of practical case-studies.[82,92]

3.4 *Finite bisimulation and decidability*

In this section we introduce the concept of bisimulation relation between two Kripke structures, and show that for two bisimilar systems (systems having a bisimulation relation between their states) we have that both systems have the same set of traces, and hence precisely the same set of LTL formulas are satisfied by both of them. Using this idea, we show that if for a given hybrid Kripke structure $\mathcal{H}$ there exists a bisimulation relation with some finite state Kripke structure $\mathcal{K}$, then the problem of LTL model-checking for $\mathcal{H}$ can be reduced to the decidable problem of LTL model-checking for finite Kripke structure $\mathcal{K}$.

We say that a Kripke structure $\mathcal{K}' = (\mathcal{T}', P, L')$ can *simulate* a Kripke structure $\mathcal{K} = (\mathcal{T}, P, L)$ if every step of $\mathcal{K}$ can be matched (with respect to atomic propositions) by one or more steps of $\mathcal{K}'$. A Bisimulation equivalence denotes the presence of a mutual simulation between two structures $\mathcal{K}$ and $\mathcal{K}'$. Formally, bisimulation relation is defined in the following manner.

Definition 12 (Bisimulation Relation): Let $\mathcal{K} = (\mathcal{T} = S, S_0, \Sigma, \Delta), P, L)$ and $\mathcal{K}' = (\mathcal{T} = (S', S_0', \Sigma', \Delta'), P, L')$ be two Kripke structures. A bisimulation relation between $\mathcal{K}$ and $\mathcal{K}'$ is a binary relation $\mathcal{R} \subseteq S \times S'$ such that:

- every initial state of $\mathcal{T}$ is related to some initial state of $\mathcal{T}'$, and vice-versa, i.e. for every $s \in S_0$ there exists $s' \in S_0'$ such that $(s, s') \in \mathcal{R}$ and for every $s' \in S_0'$ there exists a $s \in S_0$ such that $(s, s') \in \mathcal{R}$;
- for every $(s, s') \in \mathcal{R}$ the following holds:
  - $L(s) = L'(s')$,
  - every outgoing transition of s is matched with some outgoing transition of s', i.e. if $t \in \text{Post}(s)$ then there exists $t' \in \text{Post}(s')$ with $(t, t') \in \mathcal{R}$, and
  - every outgoing transition of s' is matched with some outgoing transition of s, i.e. if $t' \in \text{Post}(s')$ then there exists $t \in \text{Post}(s)$ with $(t, t') \in \mathcal{R}$.

We say that $\mathcal{T}$ and $\mathcal{T}'$ (analogously, $\mathcal{K}$ and $\mathcal{K}'$) are bisimilar or bisimulation equivalent, and we write $\mathcal{T} \sim \mathcal{T}'$, if there exists a bisimulation relation $\mathcal{R} \subseteq S \times S'$.

The following Proposition follows from the definition of bisimulation and the semantics of LTL.

Proposition 6: If $\mathcal{T} \sim \mathcal{T}'$ then $Trace(\mathcal{T}) = Trace(\mathcal{T}')$. Moreover, if $\mathcal{T} \sim \mathcal{T}'$ then for every LTL formula ϕ we have that $\mathcal{T} \models \phi$ if and only if $\mathcal{T}' \models \phi$.

Proof. Let $\mathcal{T} \sim \mathcal{T}'$. Using a simple inductive argument, one can show that for every run $a = \langle s_0, a_1, s_1, a_2, \ldots \rangle$ of $\mathcal{T}$ there is a run $r' = \langle s_0', a_1', s_1', a_2', \ldots \rangle$ of $\mathcal{T}'$ such that $L(s_i) = L'(s_i')$ for every $i \geq 0$. This implies that $Trace(r) = Trace(r')$ and hence $Trace(\mathcal{T}) \subseteq Trace(\mathcal{T}')$. Similarly, we can show that $Trace(\mathcal{T}') \subseteq Trace(\mathcal{T})$. Hence it follows that $\mathcal{T} \sim \mathcal{T}'$ implies $Trace(\mathcal{T}) = Trace(\mathcal{T}')$. To prove the other part of the proposition, observe LTL formulae are interpreted over traces of structures, and since two bisimilar Kripke structures have the same set of traces, it follows that for every LTL formula ϕ we have that $\mathcal{T} \sim \mathcal{T}'$ implies that $\mathcal{T} \models \phi$ if and only if $\mathcal{T}' \models \phi$.

This proposition shows that LTL model checking problem can be reduced to solving LTL model checking problem over a bisimilar Kripke structure. We next show how to extend this idea to define bisimulation over the states of a Kripke structure, and use it to produce a bisimilar Kripke structure with fewer states.

Definition 13 (Bisimulation Relation on $\mathcal{K}$): Let $\mathcal{K} = (\mathcal{T} = (S, S_0, \Sigma, \Delta), P, L)$ be a Kripke structure. A bisimulation on $\mathcal{K}$ is a binary relation $\mathcal{R} \subseteq S \times S$ such that for all $(s, s') \in \mathcal{R}$ we have that:

- $L(s) = L(s')$;
- if $t \in \text{Post}(s)$, then there exists an $t' \in \text{Post}(s')$ such that $(t, t') \in \mathcal{R}$;
- if $t' \in \text{Post}(s')$, then there exists an $t \in \text{Post}(s)$ such that $(t, t') \in \mathcal{R}$.

It is easy to see that a bisimulation relation $\mathcal{R}$ over the state space of $\mathcal{K}$ is an equivalence relation. For a state $s \in S$ we write $[s]_{\mathcal{R}}$ for the equivalence class of $\mathcal{R}$ containing s. We say that states $s, s' \in S$ are bisimulation equivalent, and we write $s \sim_{\mathcal{T}} s'$, if there exists a bisimulation relation $\mathcal{R}$ for $\mathcal{T}$ with $(s, s') \in \mathcal{R}$.

Given a Kripke structure $\mathcal{T}$, we use a bisimulation relation $\mathcal{R}$ for reducing the state space of $\mathcal{T}$ using the following quotient construction.

Definition 14 (Bisimulation Quotient): Given a Kripke structure $\mathcal{K} = (\mathcal{T} = (S, S_0, \Sigma, \Delta), P, L)$ and a bisimulation relation $\mathcal{R} \subseteq S \times S$ over $\mathcal{K}$, the bisimulation quotient $\mathcal{K}_{\mathcal{R}}$ is defined as a Kripke structure $\mathcal{K}_{\mathcal{R}} = (\mathcal{T}_{\mathcal{R}} = (S_{\mathcal{R}}, S_{\mathcal{R}}^0, \Sigma_{\mathcal{R}}, \Delta_{\mathcal{R}}), P, L_{\mathcal{R}})$ where:

- The state space of $\mathcal{T}_{\mathcal{R}}$ is the quotient space of $\mathcal{T}$, i.e. $\mathcal{S}_{\mathcal{R}} = \{[s]_{\mathcal{R}} : s \in S\}$;
- The set of initial states is the set of $\mathcal{R}$-equivalence classes of the initial states, i.e. $S_{\mathcal{R}}^0 = \{[s]_{\mathcal{R}} : s \in S_0\}$;
- $\Sigma_{\mathcal{R}} = \{\tau\}$;

[c] Observe that the definition of bisimulation ensures that the state labeling $L_{\mathcal{R}}$ is well defined.

– Each transition $(s, a, s') \in \Delta$ induces a transition from $[s]_{\mathcal{R}}$ to $[s']_{\mathcal{R}}$ in $\Delta_{\mathcal{R}}$, i.e. $\Delta_{\mathcal{R}} = \{([s]_{\mathcal{R}}, \tau, [s']_{\mathcal{R}}) : (s, \alpha, s') \in \Delta\}$, and
– $L_{\mathcal{R}}$ is defined such that $L_{\mathcal{R}}([s]) = L(s)$[c].

We say that a bisimulation quotient is *finite* if there are finitely many equivalence classes of $\mathcal{R}$, i.e. $|S_{\mathcal{R}}| < \infty$.

The proof of the following theorem is immediate from Proposition 6 and Theorem 5.

Theorem 7: The existence of a finite bisimulation quotient for a hybrid Kripke structure imply the decidability of LTL model-checking problem.

4 Decidable Subclasses of Hybrid Automata

Given the expressiveness of hybrid automata it is not surprising that simple reachability questions are undecidable for general hybrid kripke structures. In this section we discuss some prominent subclasses of hybrid automata for which LTL model checking problem is decidable. In the previous section we discussed that showing the existence of a finite bisimulation quotient guarantees decidable model-checking. Timed automata were among the first hybrid automata shown to have decidable model-checking using this approach. We begin this section by presenting timed automata and discuss this bisimulation known as region-equivalence relation. We will also review multi-rate and rectangular hybrid automata (Section 4.2) that under certain restriction (initialized) recover decidability of LTL model-checking via reductions to similar problem on timed automata. Finally, in Section 4.3 we discuss a relatively simple class of hybrid systems, called piecewise-constant derivative systems, that capture the essence of undecidability and provide references to its variants that permit algorithmic analysis.

4.1 *Timed automata*

Timed automata, introduced by Alur and Dill,[10,11] is a popular formalism to model real-time systems. A timed automaton is a hybrid automaton where all variables grow with a constant and uniform rate (for all variables $x \in X$ we have that $\dot{x} = 1$) and the only jump permitted during the discrete transitions is reset to zero. Moreover, the set of predicates permitted to appear as guard on transitions is restricted to the following kind of octagonal predicates:

$$g := x \bowtie c \,|\, x - y \bowtie c \,|\, g \wedge g \tag{4}$$

where x, y are clock variables, $\bowtie \in \{<, \leq, =, >, \geq\}$ and $c \in \mathbb{N}$. We write $\mathcal{Z}(X)$ for this class of octagonal predicates over the set X. Formally, we define a timed automata as a restriction of hybrid automata in the following manner.

Definition 15 (Timed Automata: Syntax): A timed automaton is a hybrid automaton $\mathcal{T} = (M, M_0, \Sigma, X, \Delta, I, F, V_0)$ with the following restrictions:

– The transition relation $\Delta \subseteq M \times \text{pred}(X) \times \Sigma \times \text{pred}(X \cup X') \times M$ is such that if $(m, g, a, j, m') \in \Delta$ then
  – the guard g is of the form (4), i.e. $g \in \mathcal{Z}(X)$ and
  – the jump predicate j only permits variable resets to zero, i.e. j is of the form

    $$\wedge_{x \in Y}(x' = 0),$$

    for some $Y \subseteq X$. We denote such set Y as reset(j).
– The mode-invariant function $I : M \to \text{pred}(X)$ is such that for all $m \in M$ we have that $I(m) \in \mathcal{Z}(X)$;
– The flow function $F : \text{M} \to (\mathbb{R}^{|X|} \to \mathbb{R}^{|X|})$ is such that for all $m \in M$ we have that $F(m)$ characterizes:

  $$\wedge_{x \in X}(\dot{x} = 1), \text{ and}$$

– $V_0 \in \text{pred}(X)$ is the set of initial valuations is such that $V_0 = \wedge_{x \in X}(x = 0)$.

The semantics of timed automata and the concept of timed Kripke structures is defined in a similar way as for hybrid automata.

Example 10: The hybrid automaton corresponding to the job-shop scheduling problem, shown in Figure 8, can also be modeled as a timed automaton by requiring that the rates of variables x_1 and x_2 is 1 in all the modes (unlike the current example where these clocks are paused in certain modes).

Example 11: As an example of a timed automaton consider Figure 11 that models a login protocol using a timed automaton. The system starts in the "standby" mode. If the user gives a correct password within 60 time-units after giving the user name, a connection will be established; if, however, the password given is wrong, the system restarts after a delay of at least 10 time units. Moreover, if no password is given within 60 time units after supplying user name, then the system restarts in the standby mode. This system is modeled using a timed automaton with five modes and one clock in Figure 11.

Alur and Dill[11] proposed the notion of region equivalence to define a bisimulation relation over

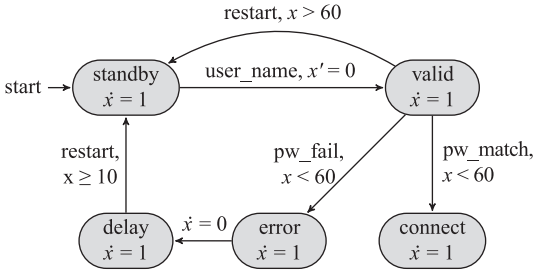


Figure 11: A time-sensitive login protocol implemented as a timed automaton.

the timed Kripke structures $(\llbracket \mathcal{T} \rrbracket, P, L)$. We say that two clock valuations v and v' are *region equivalent*, and we write $v \sim_{\mathcal{R}} v'$, if and only if all clocks have the same integer parts in v and v', and if the partial orders of the clocks, determined by their fractional parts in v and v', are the same.

Definition 16 (Region Equivalence): Let $\mathcal{T}$ be a timed automaton and let K be the maximum constant used in the guards of $\mathcal{T}$. We say that two clock valuations v and v' are *region equivalent*, and we write $v \sim_{\mathcal{R}} v'$ if and only if:

- either for $x \in X$ we have $v(x) > K$ and $v'(x) > K$, or
- for any $x, y, \in X$ with $v(x), v'(x) \leq K$ and $v(y), v'(y) \leq K$ the following conditions hold:
  - $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$, and $\lceil v(x) \rceil = 0$ iff $\lceil v'(x) \rceil = 0$,
  - $\lfloor v(y) \rfloor = \lfloor v'(y) \rfloor$, and $\lceil v(y) \rceil = 0$ iff $\lceil v'(y) \rceil = 0$,
  - $\lceil v(x) \rceil \leq \lceil v(y) \rceil$ if and only if $\lceil v'(x) \rceil \leq \lceil v'(y) \rceil$,

where $\lceil c \rceil \stackrel{def}{=} (c - \lfloor c \rfloor)$ represents the fractional part of $c \in \mathbb{R}_{\geq 0}$.

It is easy to see that $\sim_{\mathcal{R}}$ is an equivalence relation. For a clock valuation v we write $[v]$ for the region equivalence class of v. Region equivalence relation can be extended from valuations to configurations of a timed automaton $\mathcal{T}$ in a straightforward manner: we say that two configurations (m,v) and (m',v') are region equivalent, and we write $[(m,v)] = [(m',v')]$, if and only if $m = m'$ and $[v] = [v']$.

Alur and Dill[11] showed that region equivalence relations characterize finite bisimulation quotients for timed Kripke structures by showing that the number of equivalence classes for a timed automaton $(M, M_0, \Sigma, X, \Delta, I, F, V_0)$ are bounded from above by $|M| \cdot |X|! \cdot 2^{|X|} \cdot \Pi_{i=1}^{|X|} \cdot (2.K + 2)$.

Theorem 8:[11] Region equivalence relation characterizes a finite bisimulation quotient for timed Kripke structures.

This theorem combined with Theorem 7 proves the decidability of LTL model checking for timed Kripke structures. The complexity of LTL model checking was considered by Courcoubetis and Yannakakis[51] who showed that simple reachability problem for timed Kripke structures with three or more clocks is PSPACE-complete. Despite the high computational complexity of verification, algorithms based on region equivalence relation coupled with clever data-structures[27] to symbolically represent sets of regions have been shown to perform well in practice on medium-sized applications.[41,95] UPPAAL,[96] KRONOS,[66] and RED[89] are some of the leading tools that can perform timed automata based verification. The theory of timed automata has also been extended in several directions to allow them to model more realistic real-time systems, e.g. real-time systems with cost and rewards,[26,33,63,73,88] uncontrollable nondeterminism,[7,17,19,20,31,38] stochastic behavior,[8,25,36,62,68,69,70,75] and recursion.[5,94] We refer the reader to Waez, Dingel, and Rudie[97] for a detailed survey fo these extensions.

4.2 *Multi-rate and rectangular hybrid automata*

Multi-rate hybrid automata, introduced by Henzinger and Kopke,[58,59,87] are a subclass of hybrid automata where the dynamics of variables is restricted to constant rates. However, unlike timed automata, different variables can have different rates, and it can vary among different modes. Moreover, during discrete transitions these variables can be reseted to real numbers. Also in a multi-rate hybrid automaton the set of predicates permitted to appear as guard on transitions is restricted to the following kind of rectangular predicates:

$$g := c' \bowtie x \bowtie c, \tag{5}$$

where x is a variable, $\bowtie \in \{<, \leq, =, >, \geq\}$ and c, $c' \in \mathbb{N}$. We write rect(X) for this class of rectangular predicates over the set X. Formally, we define a multi-rate hybrid automata as a restriction of hybrid automata in the following manner.

Definition 17 (Multi-rate Hybrid Automata: Syntax): A multi-rate hybrid automaton is a hybrid automaton $\mathcal{H} = (M, M_0, \Sigma, X, \Delta, I, F, V_0)$ with the following restrictions:

- the transition relation $\Delta \subseteq M \times \text{pred}(X) \times \Sigma \times \text{pred}(X \cup X') \times M$ is such that if $(m, g, a, j, m') \in \Delta$ then
  - the guard g is of the form (5), i.e. $g \in$ rect(X) and
  - the jump predicate j only permits variable resets to real numbers, i.e. j is of the form

$$\wedge_{x \in Y} (x' = c_x)$$

where $Y \subseteq X$ and $c_x \in \mathbb{Z}$ for each $x \in Y$. We denote such set Y as reset(j).

- the mode-invariant function $I : M \to \text{pred}(X)$ is such that for all $m \in M$ we have that $I(m) \in \text{rect}(X)$;
- the flow function $F : M \to (\mathbb{R}^{|X|} \to \mathbb{R}^{|X|})$ is such that for all $m \in M$ we have that $F(m)$ characterize:

$$\wedge_{x \in X} (\dot{x} = c_{x,m}),$$

where $c_{x,m} \in \mathbb{Z}$ for each $x \in X$; and
- $V_0 \in \text{pred}(X)$ is the set of initial valuations is such that $V_0 = \wedge_{x \in X} x = 0$.

The semantics of multi-rate automata and the concept of multi-rate Kripke structures is defined is a similar way as for hybrid automata. Rectangular hybrid automata[58,59] are a generalization of multi-rate hybrid automata where within each mode the rate of a variable can change non-deterministically within a given mode-dependent interval.

Using a reduction from two counter Minsky machine, one can easily show that the LTL model checking problem for multi-rate hybrid automata is undecidable.

Theorem 9:[59] LTL model-checking problem for multi-rate hybrid automata is undecidable.

We say that a multi-rate (or rectangular) hybrid automaton is *initialized* if it satisfies the property that every transition between two modes with different rates (rate intervals, resp.) for a variable, resets that variable, i.e. for every transition $(m, g, a, j, m') \in \Delta$ with $F(m)(x) \neq F(m')(x)$ we have $x \in \text{reset}(j)$. Figure 12 shows an initialized rectangular automaton.

Henzinger et al.[59] showed the decidability of initialized rectangular and multi-rate hybrid automata.

Theorem 10: The LTL model-checking problem for initialized rectangular (multi-rate) hybrid automata is decidable.

Proof. The decidability of LTL model-checking problem for initialized multi-rate automata by reducing the problem to similar problem for timed automata by rescaling the rate of all variables to one via appropriate adjustment of the constraints on the mode invariants and guards in all the transitions.

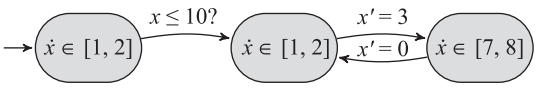


Figure 12: An initialized rectangular automaton.

To prove the decidability for an initialized rectangular automaton $\mathcal{H}_r$, we reduce the problem to corresponding problem for an initialized multi-rate automaton $\mathcal{H}_m$. Each variable x of $\mathcal{H}_r$ with rate in the rectangle $a \leq \dot{x} \leq b$ is simulated using two variables x_l, x_u such that $\dot{x}_l = a$ and $\dot{x}_u = b$. The variables x_l, x_u keep track of the lower and upper bounds of x respectively. With this replacement, the invariant conditions of modes, as well as guards and resets on transitions have to be adjusted appropriately. For example, if we had a transition with guard $x \leq 10$, then it is replaced with (i) $x_l \leq 10$ and (ii) $x_u > 10, x'_u = 10$. This conversion from initialized rectangular to initialized multirate automata is language preserving. Hence, from the decidability of LTL model checking problem for initialized multi-rate hybrid automata, the decidability for initialized rectangular hybrid follows.

4.3 *Piecewise-constant derivative systems and their variants*

Asarin, Maler, and Pnueli[18] initiated the study of hybrid dynamical systems with piecewise-constant derivatives (PCD) defined as a partition of the Euclidean space into a finite set of regions (polyhedral predicates), where the dynamics in a region is defined by a constant rate vector. They defined PCD systems as completely deterministic systems where a discrete transition occurs at region boundaries, where runs change their directions according to the rate vector available in the new region. Given the simplicity of such systems, it is perhaps surprising that the reachability problem for PCD systems with three or more variables is undecidable.[18] In fact, Asarin and Maler[16] observed that, due to the capability of such systems to perform Zeno runs, every set of arithmetical hierarchy (a hierarchy of undecidable problems) can be recognized by a PCD system of some finite dimension. On the positive side, Asarin, Maler, and Pnueli[18] gave an algorithm to solve the reachability problem for two-dimensional PCD systems. Cerans and Viksna[42] later generalized this decidability result to more general piecewise-Hamiltonian systems. We also mention the work of Asarin, Schneider, Yovine[21] who extended the decidability result for two-dimensional PCD systems to a non-deterministic setting of simple planar differential inclusion systems (SPDIs) where a number of rate vectors are available in each region.

Kesten, Pnueli, Sifakis, and Yovine[64] also studied another variant of constant-rate hybrid systems, called *integration graphs*, that can be considered as a subset of multi-rate automaton

where no test of non-clock (integrator) variables is allowed to appear on a loop. Kesten et al.[64] showed the decidability for the two subclasses of integration graphs: The class with a single clock variable, and the class where integrators are tested only once.

Recently, Bouyer et al.[35] introduced timed automata with energy constraints, that can be considered as multi-rate automata with a single non-clock variable (energy variable) that does not appear on guards, and showed decidability of schedulability problem where the energy variable is required to be greater than a given lower-bound. Bouyer, Fahrenberg, Larsen, and Markey[34] later generalized this result to give an EXPTIME algorithm for a subclass where energy variables can grow exponentially.

Alur, Trivedi, and Wojtczak recently studied constant-rate multi-mode systems,[13] that can be considered as multi-rate automata with the exception that there is no structure in the automata, i.e. any mode can be used after any other mode, and there is only a global invariant over variables. They showed that reachability and schedulability problems for these systems can be solved in polynomial time for starting states strictly inside the global invariant space. Alur, Trivedi, and Wojtczak also showed that introducing either local invariants or guards make the reachability problem undecidable. Alur et al.[12] later studied this problem for a generalization of constant-rate multi-mode systems to bounded-rate multi-mode system and showed the decidability of the schedulability problem.

5 Summary

In this article we presented hybrid automata for modeling and formal verification of cyber-physical systems. Hybrid automata naturally combine features from continuous dynamical systems and discrete finite state machines, and provide an elegant and expressive model. This expressiveness, however, comes with a price—the simple reachability problem for simple subclasses of hybrid automata, like piecewise-constant derivative systems, turned out to be highly undecidable. In this article we discussed a general approach of finding finite bisimulation quotient to show decidability of subclasses of hybrid automata, and sketched the proof for the decidability for two key subclasses: timed automata and initialized rectangular hybrid automata. Hybrid automata provide an intuitive and semantically unambiguous way to model cyber-physical systems. These formalisms provide a rich theory and a mature set of tools, UPPAAL,[96] Kronos,[66] RED,[89] HyTECH,[60] and PHAVer,[86] able to perform automatic verification of systems modeled using them. A growing number of case-studies using these tools have shown promise in extending the state-of-the-art to industrial-sized examples.

Received 22 August 2013.

References

1. ACM Turing award citation for A. Pnueli. http://awards.acm org/citation.cfm?id=4725172&srt=alpha&alpha=P&aw=140&ao=AMTURING&yr=1996, 1996. For seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification.
2. ACM Turing award citation for E.M. Clarke, E.A. Emerson, and J. Sifakis. http://awards.acm.org/citation.cfm?id=1167964&srt=alpha&alpha=C&aw=140&ao=AMTURING&yr=2007, 2007. For their role in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries.
3. ACM Kanellakis theory and practice award citation for G.J. Holzmann, R.P. Kurshan, M.Y. Vardi, and P. Wolpe. http://awards.acm.org/citation.cfm?id=1625680&srt=all&aw=147&ao=KANELLAK&yr=2005, 2005. For the development of automata-theoretic techniques for reactive-systems verification, and the practical realization of powerful formal-verification tools based on these techniques.
4. ACM Kanellakis theory and practice award citation for R.E. Bryant, E.M. Clarke, E.A. Emerson, K.L. Mcmillan. http://awards.acm.org/citation.cfm?id=1167964&srt=all&aw=147&ao=KANELLAK&yr=1998, 1998. For their invention of "symbolic model checking".
5. P.A. Abdulla, M.F. Atig, and J. Stenman. Dense-timed pushdown automata. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, LICS '12, pp. 35–44, Washington, DC, USA, 2012. IEEE Computer Society.
6. R. Alur and M. Bernadsky. Bounded model checking for GSMP models of stochastic real-time systems. In *Proceedings of HSCC*, volume 3927, pp. 19–33. LNCS, 2006.
7. R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sannella, editors, *Proc. ICALP'04*, volume 3142 of *LNCS*, pp. 122–133. Springer, 2004.
8. R. Alur, C. Courcoubetis, and D.L. Dill. Model-checking for probabilistic real-time systems. In *Automata, Languages and Programming: Proceedings of the 18th ICALP, Lecture Notes in Computer Science 510*, pp. 115–126. Springer, 1991.
9. R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems I*, volume 736 of *Lecture Notes in Computer Science*, pp. 209–229. Springer-Verlag, 1993.

10. R. Alur and D. Dill. Automata for modeling real-time systems. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 443 of *LNCS*, pp. 322–335. Springer, 1990.
11. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126: pp. 183–235, 1994.
12. R. Alur, V. Forejt, S. Moarref, and A. Trivedi. Safe schedulability of bounded-rate multi-mode systems. In *Hybrid Systems: Computation and Control (HSCC)*, pp. 243–252, 2013.
13. R. Alur, A. Trivedi, and D. Wojtczak. Optimal scheduling for constant rate multi-mode systems. In *Hybrid Systems: Computation and Control (HSCC)*, pp. 75–84, 2012.
14. Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *Real-Time: Theory in Practice*, pp. 74–106. Springer, 1992.
15. P. Argón, G. Delzanno, S. Mukhopadhyay, and A. Podelski. Model checking communication protocols. In *Proceedings of the 28th Conference on Current Trends in Theory and Practice of Informatics Piestany: Theory and Practice of Informatics*, SOFSEM '01, pp. 160–170, London, UK, 2001. Springer-Verlag.
16. E. Asarin and O. Maler. Achilles and the tortoise climbing up the arithmetical hierarchy. *Journal of Computer and System Sciences*, 57(3): pp. 389–398, 1998.
17. E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In F.W. Vaandrager and J.H. van Schuppen, editors, *Proc. HSCC'99*, volume 1569 of *LNCS*, pp. 19–30. Springer, 1999.
18. E. Asarin, O. Maler, and A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138(1): pp. 35–65, 1995.
19. E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, volume 999 of *LNCS*, pp. 1–20. Springer, 1995.
20. E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In P. Antsaklis, W. Kohn, A. Nerode, and Sastry S., editors, *Proceedings of IFAC Symposium on System Structure and Control*, pp. 469–474. Elsevier, 1998.
21. E. Asarin, G. Schneider, and S. Yovine. On the decidability of the reachability problem for planar differential inclusions. In *Hybrid Systems: Computation and Control*, pp. 89–104. Springer, 2001.
22. C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
23. T. Ball, V. Levin, and S.K. Rajamani. A decade of software model checking with SLAM. *Communications of the ACM*, 54(7): pp. 68–76, July 2011.
24. D. Basin, C. Cremers, and C. Meadows. Model checking security protocols. *Handbook of Model Checking*, 2011. http://people.inf.ethz.ch/cremersc/publications/index.html
25. D. Beauquier. Probabilistic timed automata. *Theoretical Computer Science*, 292(1): pp. 65–84, 2003.
26. G. Behrmann, A. Fehnker, T. Hune, K.G. Larsen, P. Pettersson, J. Romijn, and F.W. Vaandrager. Minimum-cost reachability for priced timed automata. In M.D. Di Benedetto and A.L. Sangiovanni-Vincentelli, editors, *Proc. HSCC'01*, volume 2034 of *LNCS*, pp. 147–161, Heidelberg, 2001. Springer.
27. J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, pp. 87–124, 2003.
28. J. Berendsen, D. Jansen, and J.-P. Katoen. Probably on time and within budget—on reachability in priced probabilistic timed automata. In *Proc. QEST'06*, pp. 311–322, Washington, DC, USA, 2006. IEEE.
29. A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in computers*, 58: pp. 117–148, 2003.
30. A. Biere, A. Cimatti, E.M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Handbook of Satisfiability*, 185: pp. 457–481, 2009.
31. P. Bouyer. Weighted timed automata: Model-checking and games. *Electronic Notes Theoretical Computer Science*, 158: pp. 3–17, 2006.
32. P. Bouyer, T. Brihaye, M. Jurdzinski, R. Lazic, and M. Rutkowski. Average-price and reachability-price games on hybrid automata with strong resets. In *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 5215 of *LNCS*, pp. 63–77. 2008.
33. P. Bouyer, E. Brinksma, and K.G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1): pp. 3–23, 2008.
34. P. Bouyer, U. Fahrenberg, K.G. Larsen, and N. Markey. Timed automata with observers under energy constraints. In *Hybrid Systems: Computation and Control (HSCC)*, 2010.
35. P. Bouyer, U. Fahrenberg, K.G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In *Formal Modeling and Analysis of Timed Systems*, pp. 33–47. Springer, 2008.
36. P. Bouyer, and V. Forejt. Reachability in stochastic timed games. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5556 of *LNCS*, pp. 103–114. Springer, 2009.
37. T. Brihaye, V. Bruyere, and J.-F. Raskin. Model-checking for weighted timed automata. In Yassine Lakhnech and Sergio Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, volume 3253 of *Lecture Notes in Computer Science*, pp. 277–292. Springer Berlin Heidelberg, 2004.
38. T. Brihaye, T.A. Henzinger, V.S. Prabhu, and J. Raskin. Minimum-time reachability in timed games. In *Proc. ICALP'07*, volume 4596 of *LNCS*, pp. 825–837. Springer, 2007.
39. M. Broy, I.H. Kruger, A. Pretschner, and C. Salzmann. Engineering automotive software. *Proceedings of the IEEE*, 95(2): pp. 356–373, 2007.
40. J.R. Büchi. On a decision method in restricted second-order arithmetic. In *Int. Congr. for Logic Methodology and Philosophy of Science*, pp. 1–11. Standford University Press, Stanford, 1962.

41. F. Cassez, J. Jessen, K. Larsen, J. Raskin, and P. Reynier. Automatic synthesis of robust and optimal controllers: an industrial case study. In F.W. Vaandrager and J.H. van Schuppen, editors, *HSCC'09*, volume 5469 of *LNCS*, 2009.
42. K. Cerans and J. Viksna. Deciding reachability for planar multi-polynomial systems. In *Hybrid Systems*, pp. 389–400, 1995.
43. Zhou Chaochen, Anders P. Ravn, and Michael R. Hansen. An extended duration calculus for hybrid real-time systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pp. 36–59. Springer Berlin Heidelberg, 1993.
44. Robert N. Charette. This car runs on code. http://news.discovery.com/autos/toyota-recall-software-code.htm, February 2013.
45. K. Chatterjee, T.A. Henzinger, and V. Prabhu. Timed parity games: Complexity and robustness. In *Proceedings of the Sixth International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'08)*, volume 5215 of *LNCS*, pp. 124–140, 2008.
46. E.M. Clarke, E.A. Emerson, S. Jha, and A.P. Sistla. Symmetry reductions in model checking. In *Computer Aided Verification*, pp. 147–158. Springer, 1998.
47. E.M. Clarke, E.A. Emerson, and J. Sifakis. Model checking: algorithmic verification and debugging. *Communications of the ACM*, 52(11): pp. 74–84, 2009.
48. E.M. Clarke, A. Fehnker, Z. Han, J. Krogh, B.H. and Ouaknine, O. Stursberg, and M. Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *International Journal of Foundations of Computer Science*, 14(4): pp. 583–604, 2003.
49. E.M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, pp. 154–169, 2000.
50. E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
51. C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In *Formal Methods in System Design*, volume 1, pp. 385–415, Dordrecht, 1992. Kluwer.
52. L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In R. Amadio and D. Lugiez, editors, *Proc. CONCUR'03*, volume 2761 of *LNCS*, pp. 144–158. Springer, 2003.
53. E.A. Emerson. Model checking and mu-calculus. In N. Immerman and Ph. G. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 185–214. American Mathematical Society, 1996.
54. G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. pp. 258–273. Springer, 2005.
55. Hong Fu, Guangyu Tian, Quanshi Chen, and Yiding Jin. Hybrid automata of an integrated motor-transmission powertrain for automatic gear shift. In *American Control Conference (ACC), 2011*, pp. 4604–4609, 2011.
56. E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games. A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.
57. T.A. Henzinger, P. Ho, and H. Wong-toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1: pp. 460–463, 1997.
58. T.A. Henzinger and P.W. Kopke. Discrete-time control for rectangular hybrid automata. *Theor. Comput. Sci.*, 221(1–2): pp. 369–392, 1999.
59. T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1): pp. 94–124, 1998.
60. Hytech.http://embedded.eecs.berkeley.edu/research/hytech/
61. Zhihao Jiang, Miroslav Pajic, and Rahul Mangharam. Cyber–physical modeling of implantable cardiac medical devices. *Proceedings of the IEEE*, 100(1): pp. 122–137, 2012.
62. M. Jurdziński, J. Sproston, and F. Laroussinie. Model checking probabilistic timed automata with one or two clocks. *Logical Methods in Computer Science*, 4(3):12, 2008.
63. M. Jurdziński and A. Trivedi. Concavely-priced timed automata. In F. Cassez and C. Jard, editors, *Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 5215 of *LNCS*, pp. 48–62. Springer, 2008.
64. Y. Kesten, A. Pnueli, J. Sifakis, and Yovine. S. Integration graphs: A class of decidable hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *LNCS*, pp. 179–208. Springer, 1992.
65. Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4): pp. 255–299, 1990.
66. Kronos.http://www-verimag.imag.fr/TEMPORISE/kronos/
67. R.P. Kurshan. Verification technology transfer. In Orna Grumberg and Helmut Veith, editors, *25 Years of Model Checking*, pp. 46–64. Springer-Verlag, Berlin, Heidelberg, 2008.
68. M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *FMSD*, 29: pp. 33–78, 2006.
69. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Verifying quantitative properties of continuous probabilistic timed automata. In *Proc. of 11th International Conference on Concurrency Theorey, (CONCUR'00)*, volume 1877 of *LNCS*, pp. 123–137, 2000.
70. M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 205(7):1027–1077, 2007.
71. M.Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. In *ARTS*, pp. 75–95, 1999.
72. L. Lamport. Proving the correctness of multiprocess programs.*Software Engineering, IEEE Transactions on*, SE-3(2): pp. 125–143, 1977.

73. K.G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In G. Berry, H. Comon, and A. Finkel, editors, *Proc. CAV'01*, volume 2102 of *LNCS*, pp. 493–505, Heidelberg, 2001. Springer.
74. J. Lygeros, C. Tomlin, and S. Sastry. *Hybrid Systems: Modeling, Analysis and Control*. In preparation, 2008. Unpublished anuscript, http://www-inst.cs.berkeley.edu/_ee291e/sp09/handouts/book.pdf
75. O. Maler, K.G. Larsen, and B. Krogh. On zone-based analysis of duration probabilistic automata. In *INFINITY*, pp. 33–46, 2010.
76. Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, PODC '87, pp. 205–205, New York, NY, USA, 1987. ACM.
77. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.
78. W. Marrero, E.M. Clarke, and S. Jha. Model checking for security protocols. Technical report, DTIC Document, 1997.
79. MATLAB. *version 7.10.0 (R2010a)*. The MathWorks Inc., Natick, Massachusetts, 2010.
80. Kenneth L. McMillan. *Symbolic model checking*. Springer, 1993.
81. Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.
82. Nusmv. http://nusmv.fbk.eu/
83. J. Ouaknine and J. Worrell. Some recent results in metric temporal logic. In Franck Cassez and Claude Jard, editors, *Formal Modeling and Analysis of Timed Systems*, volume 5215 of *Lecture Notes in Computer Science*, pp. 1–13. Springer Berlin Heidelberg, 2008.
84. William Pasillas-Lépine. Hybrid modeling and limit cycle analysis for a class of five-phase anti-lock brake algorithms. *Vehicle System Dynamics*, 44(2): pp. 173–188, 2006.
85. Doron Peled. Combining partial order reductions with on-the-fly model-checking. In *Computer aided verification*, pp. 377–390. Springer, 1994.
86. Phaver. http://www-verimag.imag.fr/~frehse/phaver web/.
87. A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusion. In *Computer Aided Verification (CAV)*, pp. 95–104, 1994.
88. J.I. Rasmussen, K.G. Larsen, and K. Subramani. On using priced timed automata to achieve optimal scheduling. *Formal Methods in System Design*, 29(1): pp. 97–114, 2006.
89. RED. http://cc.ee.ntu.edu.tw/-farn/red/
90. Jaijeet Roychowdhury. Numerical simulation and modelling of electronic and biochemical systems. *Foundations and Trends in Electronic Design Automation*, 3(2–3): pp. 97–303, February 2009.
91. A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3): pp. 733–749, July 1985.
92. Spin. http://spinroot.com/
93. T. Stauner, O. Muller, and M. Fuchs. Using hytech to verify an automotive control system. In Oded Maler, editor, *Hybrid and Real-Time Systems*, volume 1201 of *Lecture Notes in Computer Science*, pp. 139–153. Springer Berlin Heidelberg, 1997.
94. A. Trivedi and D. Wojtczak. Recursive timed automata. In*Proceedings of the 8th International Symposium on Automated Technology for Veri-fication and Analysis*, volume 6252 of *LNCS*, pp. 306–324. Springer-Verlag, September 2010.
95. Uppaal case-studies. http://www.it.uu.se/research/group/darts/uppaal/ examples.shtml
96. Uppaal. http://www.uppaal.com/
97. Md Tawhid Bin Waez, Juergen Dingel, and Karen Rudie. A survey of timed automata for the development of real-time systems. *Computer Science Review*, 2013.
98. Inc. Wolfram Research. *Mathematica Edition: Version 8.0*. Wolfram Research, Inc., 2010. Champaign, Illinois.
99. P. Wolper, M.Y. Vardi, and A.P. Sistla. Reasoning about infinite computation paths. In *Foundations of Computer Science*, pp. 185–194, 1983.

Krishna S. received her Ph.D. from the Indian Institute of Technology Madras in 2003, and she is faculty member in the department of computer science and engineering at the Indian Institute of Technology (IIT) Bombay. Her research focuses on studying decidable models for real-time systems, model-checking timed systems with weighted timed logic, synthesis games for timed systems, and expressiveness and decidability issues in timed logic. She is also a member of the centre for formal design and verification of software (CFDVS) at IIT Bombay.

Ashutosh Trivedi received his Ph.D. from the University of Warwick for studying competitive optimisation on Timed Automata in 2009. After this he worked on probabilistic timed systems at the University of Oxford in the group of Marta Kwiatkowska. In 2010 he joined the University of Pennsylvania as a postdoctoral researcher with Rajeev Alur to work on verification of hybrid systems. He is now an assistant professor in the department of computer science and engineering at the Indian Institute of Technology (IIT) Bombay. His research focuses on developing theory, techniques, and tools for formal analysis, verification, and synthesis of hybrid systems. He is also a member of the centre for formal design and verification of software (CFDVS) at IIT Bombay.