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

Shankara Narayanan Krishna and Ashutosh Trivedi
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai, India 400076
Email: {krishnas,trivedi}@cse.iitb.ac.in

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 continuous parts of a cyber-physical system. 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.

I. 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 performing various functions 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 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 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 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 a 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 au-
to automatically refine—known as counterexample-guided abstraction refinement (CEGAR) [49], [48] 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 [78], [24] 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[1] of key verification problems like reachability and schedulability [10] and availability of mature verification tools—like UPPAAL [27], [96], Kronos [64], 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 [95], [41], [74], [55], [84], [93], [61] 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*) [77], [50], 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 systems 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 [20], [17], [52], [7], [45], [32] that allow the model to distinguish between controllable and uncontrollable non-determinism;
- probabilistic extensions [71], [25], [68], [6], [36], [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 [23], [33], [68], [28], [31] 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 [57], [53], [27] for efficient implementation of these results.

We begin (Section III) 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 III 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 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 [14]) three key subclasses of hybrid automata—timed automata, (initialized) rectangular hybrid automata, and (two dimensional) piecewise-constant derivative systems—with decidable model checking problem.
II. 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 $T = (S, S_0, \Sigma, \Delta)$ where:

- $S$ is a (potentially infinite) set of states;
- $S_0 \subseteq S$ is the set of initial states;
- $\Sigma$ 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 $T$ is finite (countable), if the sets $S$ and $\Sigma$ are finite (countable).

Given an action $a \in \Sigma$ and a state $s$ we write $\text{Post}(s, a)$ for the set of states that are reachable from $s$ on $a$ and $\text{Post}(s)$ 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 $T$ 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 $(s_0, a_1, s_1, a_2, s_2, \ldots, s_n)$ 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

\[
\langle \text{count, 0}, \text{tick}, \langle \text{count, 1}, \text{pause}, \text{pause, 1}, \text{tick}, \text{pause, 1}, \text{on}, \langle \text{count, 1}, \text{tick}, \text{count, 2} \rangle.
\]

Fig. 1. State transition graph for a mod-4 counter.

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 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 $\nu : X \rightarrow \mathbb{R}$ and we write $V(X)$ for the set of valuations on $X$. Abusing notation, we also treat a valuation $\nu$ 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 $\mathbb{R}^{\lvert X \rvert}$. For efficient computer-readable representation of predicates we often define them using non-linear algebraic equations involving $X$. We write $\text{pred}(X)$ for the set of predicates over $X$. For a predicate $\pi \in \text{pred}(X)$ we write $[\pi]$ for the set of valuations in $\mathbb{R}^{\lvert X \rvert}$ satisfying the equation $\pi$. 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 $\dot{\theta}$ and $\theta$ is

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

characterizing the motion of an idealized pendulum (Figure 2), where $\theta$ is the angle the pendulum forms with its rest position, $n$ is the mass of the pendulum, $m$ is the gravitational constant, and $\ell$ is the length of the pendulum.

We say that a predicate $P$ is polyhedral if it is defined as the conjunction of a finite set of linear constraints of the form $a_1x_1 + \cdots + a_nx_n \leq k$, 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 $\mathbb{R} \in \{<, \leq, =, >, \geq\}$. An example of a polyhedral predicate over the set $\{x, y, x\}$ 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 $\leq x \leq y \leq k$ or $x \leq y \leq k$, 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 $x \leq y \leq k$, where $k \in \mathbb{R}$, and $x \in X$. 


A. Discrete Dynamical Systems

Discrete dynamical systems can be conveniently modeled as extended finite state machines having finitely many modes (or 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 \( M_0 \subseteq M \),
- \( \Sigma \) 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,
- \( I : M \to \text{pred}(X) \) 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 \((m, \nu)\) where \( m \) is a control mode and \( \nu \) is a valuation of variables in \( X \). The execution of an extended finite state machine begins in a configuration \((m_0, \nu_0)\) such that the control mode \( m_0 \in M_0 \) is in the set of initial control modes and the valuation \( \nu_0 \in \text{val}(X) \) satisfies the invariant of mode \( m_0 \), i.e. \( \nu_0 \in [I(m_0)] \). At each discrete time-step the system executes a transition \((m, g, a, j, m')\) that is enabled in the current configuration \((m, \nu)\), i.e., \( \nu \in [g] \), and the configuration of the system jumps to a new configuration \((m', \nu')\) while respecting the jump constraints, i.e. \( (\nu, \nu') \in I[(m')] \) as well as the invariant condition of the resulting mode \( \nu' \in [I(m')] \). 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 \( T^\mathcal{M} = (S^\mathcal{M}, S^\mathcal{M}_0, \Sigma^\mathcal{M}, \Delta^\mathcal{M}) \) where:
- \( S^\mathcal{M} \subseteq (M \times \mathbb{R}^{\lvert X \rvert}) \) is the set of configurations of \( \mathcal{M} \) such that for all \((m, \nu) \in S^\mathcal{M} \) we have that \( \nu \in [I(m)] \);
- \( S^\mathcal{M}_0 \subseteq S^\mathcal{M} \) such that \((m, \nu) \in S^\mathcal{M}_0 \) if \( m \in M_0 \) and \( \nu \in V_0 \);
- \( \Sigma^\mathcal{M} \subseteq \Sigma \) is the set of labels;
- \( \Delta^\mathcal{M} \subseteq \Sigma^\mathcal{M} \times \Sigma^\mathcal{M} \times S^\mathcal{M} \) is the set of transitions such that \((\{(m, \nu), (m', \nu')\}) \in \Delta^\mathcal{M} \) if there exists a transition \( \delta = (m, g, a, j, m') \in \Delta \) such that the current valuation \( \nu \) satisfies the guard of \( \delta \), i.e. \( \nu \in [g] \); the pair of current and next valuations \((\nu, \nu')\) satisfies the jump constraint of \( \delta \), i.e. \((\nu, \nu') \in [I(m')]\); and the next valuation satisfies the invariant of the target mode of \( \delta \), i.e. \( \nu' \in [I(m')] \).

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 \( \{\text{tick}, \text{on}, \text{pause}\} \) 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 \((\text{count}, x = 3, t, x' = 0, \text{count})\) is shown in the Figure 2 as a self-loop labeled with \((x = 1, t, x' = 0)\) on the mode labeled count. It is straightforward to see that the extended finite state machine in Figure 2 models a modulo-4 counter 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 jump constraints, i.e. \((\nu, \nu') \in [I(m')] \). In the following, we refer to \((m', \nu')\) by \((m', \nu') \in S^\mathcal{M} \) while respecting the guard \((\nu, \nu') \in [I(m')] \) and the jump constraint \((\nu, \nu') \in [I(m')] \).

\[ x < 3, \text{tick}, x' = x + 1 \]
\[ \top, \text{on}, x' = x \]
\[ \top, \text{tick}, x' = x \]
\[ x = 3, \text{tick}, x' = 0 \]

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

B. 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 \( F : \mathbb{R}^{\lvert X \rvert} \to \mathbb{R}^{\lvert X \rvert} \) characterizing the system of ordinary differential equations:

\[ \dot{X} = F(X) \]
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 \( \ddot{\theta} + \frac{g}{\ell} \sin(\theta) = 0 \) can be written as a system of first-order differential equations \( \dot{\theta} = y, \dot{y} = -\frac{g}{\ell} \sin(\theta) \).

Formally, a continuous dynamical system is defined in the following manner.

**Definition 4 (Continuous Dynamical System):** A continuous dynamical system is a tuple \( \mathcal{M} = (X, F, \nu_0) \) such that
- \( X \) is a finite set of real-valued variables,
- \( F : R^{|X|} \to R^{|X|} \) is the flow function characterizing the set of ordinary differential equations \( \dot{X} = F(X) \), and
- \( \nu_0 \in R^{|X|} \) is the initial valuation.

A run of a continuous dynamical system \( \mathcal{M} = (X, F, \nu_0) \) is given as a solution to the system of differential equations (1) with initial valuation \( \nu_0 \). Let a differentiable function \( f : R_{\geq 0} \to R^{|X|} \) be a solution to (1), that provides the evaluations of the variables as a function of time, such that:

\[
\begin{align*}
 f(0) &= \nu_0 \\
 \dot{f}(t) &= F(f(t)) \text{ for every } t \in R_{\geq 0},
\end{align*}
\]

where \( \dot{f} : R_{\geq 0} \to R^{|X|} \) 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 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-Lindelo\v{f} Theorem [90]):** If a function \( F : R^{|X|} \to R^{|X|} \) is Lipschitz-continuous then the differential equation \( \dot{X} = F(X) \) with initial valuation \( \nu_0 \in R^{|X|} \) has a unique solution \( f : R_{\geq 0} \to R^{|X|} \) for all \( \nu_0 \in R^{|X|} \).

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

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

**Theorem 2 (Stability wrt initial valuation [74]):** Let \( F \) be a Lipschitz-continuous function with constant \( K > 0 \) and let \( f : R_{\geq 0} \to R^{|X|} \) and \( f' : R_{\geq 0} \to R^{|X|} \) be solutions to the differential equation \( \dot{X} = F(X) \) with initial valuation \( \nu_0 \in R^{|X|} \) and \( \nu'_0 \in R^{|X|} \), respectively. Then, for all \( t \in R_{\geq 0} \) we have that \( \| f(t) - f'(t) \| \leq \| \nu_0 - \nu'_0 \| e^{Kt} \).

This theorem implies that, under Lipschitz-continuous assumption on the flow function \( F \), any two runs whose initial valuation is 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:

\[
\begin{align*}
\dot{\theta} &= y, \\
\dot{y} &= -\frac{g}{\ell} \sin(\theta),
\end{align*}
\]

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

\[
\dot{\theta} = y \quad \text{and} \quad \dot{y} = -\frac{g}{\ell} \theta.
\]

Hence our continuous dynamical system is \( \mathcal{M} = (X, F, \nu_0) \) where \( X = \{ \theta, y \} \), \( F \) is such that \( F(\theta) = y \) and \( F(y) = -\frac{g}{\ell} \theta \) and \( \nu_0 = (\theta_0, 0) \). The solution for these differential equations is

\[
\begin{align*}
\theta(t) &= A \cos(Kt) + B \sin(Kt) \\
y(t) &= -AK \sin(Kt) + BK \cos(Kt),
\end{align*}
\]

where \( K = \sqrt{\frac{g}{\ell}} \). 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 \( f : R_{\geq 0} \to \{ \theta, y \} \) as \( t \mapsto (\theta_0 \cos(Kt), -\theta_0 K \sin(Kt)) \).

Figure 4 shows the change in valuations of the variables \( \theta \) and \( y \) as a function of time.
C. 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 $H = (M, M_0, \Sigma, X, \Delta, I, F, V_0)$ where:

- $M$ is a finite set of control modes including a distinguished initial set of control modes $M_0 \subseteq M$,
- $\Sigma$ 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 $(m, \nu)$ where $m \in M$ is a mode and $\nu \in \mathbb{R}^{|X|}$ is a variable valuation. For a Lipschitz-continuous flow function $F : M \to (\mathbb{R}^X \to \mathbb{R}^X)$, a valuation $\nu \in \mathbb{R}^{|X|}$, a mode $m \in M$, and a time delay $t \in \mathbb{R}_{\geq 0}$ we define $(\nu \oplus F(m)t)$ for the unique valuation $f(t)$ where $f$ is the unique run of the continuous dynamical system $(X, F(m), \nu)$. For a jump predicate $j \in \text{pred}(X \cup X')$ and valuation $\nu$ we define $\nu[j]$ for the set of valuations $\nu' \in \mathbb{R}^{|X|}_{\geq 0}$ such that $(\nu, \nu') \in j$.

The execution of a hybrid automaton begins in an initial configuration $(m_0, \nu_0)$ where $m_0 \in M_0$ is an initial mode and $\nu_0 \in V_0$ is an initial valuation satisfying $\nu_0 \in [I(m_0)]$. 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 $(\nu_0 \oplus F(m_0)t_1) \in [g]$ and all the intermediate valuations that system passes through from $\nu_0$ to $(\nu_0 \oplus F(m_0)t_1)$ satisfy the invariant of the mode $m_0$, i.e. for all $t \in [0, t_1]$ we have that $(\nu_0 \oplus F(m_0)t) \in [I(m_0)]$. After the execution of the transition $(m_0, g, a, j, m_1)$ the state of the system jumps to a new configuration $(m_1, \nu_1)$ 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 $(m_1, \nu_1)$. We can formalize this semantics using a (uncountably infinite) state transition graph.

Definition 6 (Hybrid Automata: Semantics): The semantics of a hybrid automaton $H=(M, M_0, \Sigma, X, \Delta, I, F, V_0)$ is given as a state transition graph $T^H=(S^H, S^H_0, \Sigma^H, \Delta^H)$ where:

- $S^H \subseteq (M \times \mathbb{R}^{|X|})$ is the set of configurations of $H$ such that for all $(m, \nu) \in S^H$ we have that $\nu \in [I(m)]$;
- $S^H_0 \subseteq S^H$ s.t. $(m, \nu) \in S^H_0$ if $m \in M_0$ and $\nu \in V_0$;
- $\Sigma^H = \mathbb{R}_{\geq 0} \times \Sigma$ is the set of labels;
- $\Delta^H \subseteq \Sigma^H \times \Sigma^H \times \mathbb{S}^H$ is the set of transitions such that $((m, \nu), (a, j), (m', \nu')) \in \Delta^H$ if there exists a transition $\delta = (m, g, a, j, m') \in \Delta$ such that

$$
\nu' \in (\nu \oplus F(m) t)[j];
\nu' \in [I(m')].
$$

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 $x_1 = x_2$ and $x_2 = -g$. 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 the discrete
transition labeled impact. For the starting valuation we assume
\( x_1 = \ell \) meters and \( x_2 = 0 \). Formally the hybrid automaton \( H = (M, M_0, \Sigma, X, \Delta, I, F, V_0) \) models the bouncing ball where:
\[
- M = M_0 = \{m_0\}, \\
- \Sigma = \{\text{impact}\}, \\
- X = \{x_1, x_2\}, \\
- \Delta \text{ contains the following transition }
\]
\[
(\ell, 0) \wedge x_2 \leq 0, \text{ impact, } x'_1 = x_1 \wedge x'_2 = -cx_2, m),
\]
\[
- I(m) = x_1 \geq 0, \\
- F(m) = \dot{x}_1 = x_2 \wedge \dot{x}_2 = -g, \text{ and } \\
- V_0 = \{(\ell, 0)\}.
\]
The transition diagram corresponding to this automaton is
shown in Figure 6(a). The transition diagram of a hybrid au-
tomaton 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 corre-
sponding to the flow function is
\[
x_1(t) = -\frac{1}{2}gt^2 + Ct + D \text{ and } x_2(t) = -gt + C. \tag{2}
\]
For the initial configuration is \( (m, (\ell, 0)) \) solving (2) we
get \( C = 0 \) and \( D = \ell \). Hence from \( (m, (\ell, 0)) \) 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 \( (0, -gt_1) = (\ell, 0) \oplus F(m) \times t_1 \).

After taking the transition impact this valuation changes
according to the jump function \( x'_1 = x_1 \wedge x'_2 = -cx_2 \) result-
ing in the new valuation \( (0, cgt_1) \). Again, in our no-
tation we write \( (0, cgt_1) \in (0, -gt_1) \times x'_1 = x_1 \wedge x'_2 = -cx_2 \).
The run of the system, so far, can be written as
\( (m, (\ell, 0)), (t_1, \text{impact}), (m, (0, cgt_1)) \). Now from the con-
figuration \( (m, (0, cgt_1)) \) 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 \( x_1(t) = -\frac{1}{2}gt^2 + cgt_1t \) 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:
\[
(\ell, 0), (t_1, \text{impact}), (m, (0, cgt_1)), \tag{3}
(2ct_1, \text{impact}), (m, (0, c^2gt_1)), \tag{3}
(2^2t_1, \text{impact}), (m, (0, c^3gt_1)), \ldots \tag{3}
\]
where \( t_1 = \sqrt{2\ell/g} \). The first two transitions of the run for
\( \ell = 10 \) and \( c = 1 \) are shown in Figure 6(b).

For a given run \( r = (m_0, \nu_0), (t_1, a_1), (m_1, \nu_1), \ldots \) of a
hybrid automaton we define its time \( T(r) \) is defined as
\[
T(r) = \sum_{i=1}^{\infty} t_i.
\]
We say that a run \( r \) time-diverging if \( T(r) = \infty \). For an ex-
ample of a time-diverging run consider \( \ell \) for \( c = 1 \) as shown
in the system Figure 6(b) 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 \( \ell \) is \( T(r) = t_1(1+c)/(1-c) \) 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.

D. Composition of a Network of Hybrid Automata

While modeling a complex hybrid system using a hybrid automata, 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 \( C \), whose states are products of states of individual component automata. We define this construction as the following.

**Definition 7 (Composition):** Let \( C = \{ \mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_n \} \) be a network of hybrid automata where for each \( 1 \leq i \leq n \) let \( \mathcal{H}_i \) be \( (M_i, M_0^i, \Sigma^i, X^i, \Delta_i, I^i, F^i, V^i_0) \). For an action \( a \in \bigcup_{i=1}^{n} \Sigma_i \) we define \( E(a) = \{ i : a \in \Sigma_i^i \} \). The product automata \( \mathcal{H}_1 \otimes \mathcal{H}_2 \otimes \cdots \otimes \mathcal{H}_n \) of \( C \) is defined as a hybrid automaton \( \mathcal{H} = (M, M_0, \Sigma, X, \Delta, I, F, V_0) \) where

\[
\begin{align*}
M &= M^1 \times M^2 \times \cdots \times M^n, \\
M_0 &= M_0^1 \times M_0^2 \times \cdots \times M_0^n, \\
\Sigma &= \Sigma^1 \cup \Sigma^2 \cup \cdots \cup \Sigma^n, \\
X &= X^1 \cup X^2 \cup \cdots \cup X^n,
\end{align*}
\]

\( \Delta \subseteq (M \times \text{pred}(X) \times \Sigma \times \text{pred}(X \cup X') \times M) \) is defined s.t. \( ((m_1, \ldots, m_n), g, a, j, (m_1', \ldots, m_n')) \in \Delta \) 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 \( (m_i, g, a, j, m_i') \) such that \( g = \land_{i \in E(a)} g_i \) and \( j = \land_{i \in E(a)} j_i \).

- \( I \) is such that \( I(m_1, \ldots, m_n) = \wedge_{i=1}^{n} I^i(m_i) \);
- \( F \) is such that \( F(m_1, \ldots, m_n) = F^i(m_i)(x) \) 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 \( J = \{ j_1, \ldots, j_n \} \) of jobs to be processed on a finite set \( M = \{ m_1, \ldots, m_k \} \) of machines. There is a strict precedence requirement between the jobs given as a strict partial order \( \prec \) over the set of jobs in \( J \). A mapping \( \zeta : J \rightarrow 2^M \) specifies the set of machines where a job can be executed, while the function \( \delta : J \rightarrow \mathbb{R}_{\geq 0} \) 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, \( j_i = \{ j' : j' \prec j \} \), have terminated; 2) each machine \( m \in M \) can process atmost one job at a time; and 3) a job, once started, cannot be preempted.

**Modeling Jobs.** We model each job \( j_i \in J \) 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 \( \text{done}_i \) 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 is guarded by the condition that all of the preceding jobs according to \( \prec \) has been finished, i.e. \( \bigwedge_{k : k \prec i} (\text{done}_k = 1) \). The transition from a mode \( S_i \) to \( F_i \) with action finish 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 \( \text{done}_i' = 1 \).

**Modeling Machines.** We model each machine \( m_i \in M \) 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. \( m_j \in \zeta(j_j) \) there is a mode \( P_{i,j} \) (corresponding to processing job \( j_j \) \( j_j \in J \) on machine \( m_i \in M \)). For each mode \( P_{i,j} \) there is a transition from \( I_i \) to \( P_{i,j} \) with action begin, and a transition from \( P_{i,j} \) to \( I_i \) with action finish, 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 \( H_{j_1}, H_{j_2} \), and \( H_{m_1} \) corresponding to the jobs \( j_1 \) and \( j_2 \), and the machine \( m_1 \) respectively. This figure also shows the composition of these automata \( H_{j_1} \otimes H_{j_2} \otimes H_{m_1} \) representing the hybrid automata corresponding to the complete job-shop problem.

### III. 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 change of behaviour of a system over time. Manna and Pnueli [77], [1] 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* [77], [50], and modal \( \mu \)-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 [37], [11] 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 broad 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 III-A) 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 III-B 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 III-C we show that LTL model-checking can be algorithmically solved for finite Kripke structures. Finally, in Section III-D we introduce the notion of bisimulation, and show that the existence of a finite bisimulation implies the decidability of LTL model-checking problem.

#### A. 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.

**Definition 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
- \( L : S \rightarrow 2^P \) 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
- \( L : M \rightarrow 2^P \) 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 \(r = (\langle m_0, v_0 \rangle, \langle t_1, a_1 \rangle, (m_1, v_1), \ldots, (m_n, v_n) \ldots)\) of \(\mathcal{H}\), we define a trace corresponding to \(r\), denoted as \(\text{Trace}(r)\), as the sequence \((L(m_0), L(m_1), L(m_2), \ldots)\). Let \(\text{Trace}(\mathcal{H}, P, L)\) be the set of traces of the Hybrid Kripke Structure \(\mathcal{H}\). For a trace \(\sigma = \langle P_0, P_1, \ldots, P_n, \ldots \rangle \in \text{Trace}(\mathcal{H}, P, L)\) 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 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 \(\phi\) and \(\psi\) are valid LTL formulas then so are \(\neg \phi\), \(\phi \land \psi\) and \(\phi \lor \psi\);
- if \(\phi\) and \(\psi\) are valid LTL formulas then so are \(\Box \phi\), \(\Diamond \phi\), \(\Box \Diamond \psi\), and \(\Diamond \Box \psi\).

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

- \(\sigma \models \top\) and \(\sigma \models \bot\); 
- \(\sigma \models p\) if \(p \in P\); 
- \(\sigma \models \neg \phi\) if \(\sigma \models \phi\); 
- \(\sigma \models \phi \land \psi\) if \(\sigma \models \phi\) and \(\sigma \models \psi\); 
- \(\sigma \models \phi \lor \psi\) if \(\sigma \models \phi\) or \(\sigma \models \psi\); 
- \(\sigma \models \Box \phi\) if \(\sigma[1] \models \phi\); 
- \(\sigma \models \Diamond \phi\) if there exists \(i \geq 0\) such that \(\sigma[i] \models \phi\); 
- \(\sigma \models \Box \Diamond \psi\) if for all \(i \geq 0\) we have that \(\sigma[i] \models \psi\); and
- \(\sigma \models \phi U \psi\) if there exists \(i \geq 0\) such that \(\sigma[i] \models \psi\), and for all \(0 \leq j < i\) or \(\sigma[j] \models \phi\).

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

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 “the door is open”, “the lift is at floor \(i\) and is not moving” and “there is a request for the lift to move to the \(i\)th floor” respectively. The following are some specifications in English and their LTL counterparts:

1. Reachability property: The lift will visit the ground floor sometime.
   \[ \phi_1 \overset{def}{=} \Diamond fl_0. \]
2. Safety property: The door of the lift is never open at a floor if the lift is not present there.
   \[ \phi_2 \overset{def}{=} \Box (\bigwedge_{i=0}^{k} (\neg fl_i \Rightarrow \neg op_i)). \]
3. Recurrence property: The lift keeps coming back to the ground floor.
   \[ \phi_3 \overset{def}{=} \Box (\neg fl_0 \Rightarrow \Diamond fl_0) \wedge \Box \Diamond fl_0. \]
4. Persistence property: Eventually always a requested floor will be eventually served.
   \[ \phi_4 \overset{def}{=} \Diamond (\bigwedge_{i=0}^{k} (req_i \Rightarrow \Diamond fl_i)). \]

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

**B. 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 \(\phi\), the LTL model-checking problem is to decide whether all traces of \(\mathcal{H}\) satisfy \(\phi\), 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 following Kripke structure \(T\) with set of atomic propositions \(\{p, q\}\). We are depicting the labeling function by writing the set of propositions inside the
A module simulating \( l_i \): increment \( c \), goto \( l_j \)

\[
x_2 = 1?, x'_2 = 0 \\
x_1 = 1? \\
y = 0? \\
z' = 0 \\
x_1 > 1? x_2, y < 1? \\
x'_1, z'_1 = 0 \\
z = z_1?, y = 1? x_2 < 1? \\
y' = 0 \\
x_2 = 1?, x'_2 = 0
\]

Fig. 9. Module simulating \( l_i \): increment \( c \), goto \( l_j \)

\[
\text{start} \xrightarrow{a} m_0 \xrightarrow{a} m_1 \xrightarrow{b} m_2 \xrightarrow{b}
\]

Fig. 10. A Kripke structure \( \mathcal{T} \).

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

Proof. Given a two-counter machine \( A \), we construct a hybrid Kripke structure \( \mathcal{H} \) and an LTL formula \( \phi \) such that \( \mathcal{H} \models \phi \) iff \( A \) 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 \( A \). The increment, decrement and test instructions are encoded by suitable modules 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:

\[
x'_1 = 1 \land x'_2 = 1 \land y' = 1 \land z = 1 \land z'_1 = 2.
\]

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 \( x_1 = \frac{1}{2^c} \) and \( x_2 = \frac{1}{2^d} \), before incrementing counter \( c \), then at the end of simulating the increment instruction, we will have \( x_1 = \frac{1}{2^{c+1}} \) and \( x_2 = \frac{1}{2^d} \).

We illustrate here 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 \( l_1 \) is entered with \( y = 0 \), \( x_1 = \frac{1}{2^c} \) and \( x_2 = \frac{1}{2^d} \). On entering mode \( A_1 \), we have \( x_1 = 1, y = 1 - \frac{1}{2^c}, x_2 = \frac{1}{2^d} + (1 - \frac{1}{2^d}) \) or \( x_2 = 1 - \frac{1}{2^d} - \eta \) if \( \frac{1}{2^d} + \eta = 1, \eta \leq 1 - \frac{1}{2^d} \) and \( z = 0 \). Mode \( B_1 \) can be entered if \( x_2, y < 1 \) and \( x_1 > 1 \). Assume \( k = 0 \) units of time was spent at mode \( A_1 \). This gives \( y = 1 - \frac{1}{2^c} + k, x_2 = \frac{1}{2^d} + (1 - \frac{1}{2^d}) + k \) (or \( 1 - \frac{1}{2^d} - \eta - k, or 1 - \eta' if \frac{1}{2^d} - \eta + \eta' = 1, \eta' \leq k \)), \( z = k, x_1 = 0, z_1 = 0 \) on entering mode \( B_1 \). 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 \( B_1 \). Then \( z = k + l, z_1 = 2l, x_2 = \frac{1}{2^d} + (1 - \frac{1}{2^d} + k + l, x_1 = l, y = 1 - \frac{1}{2^c} + k + l \).

To satisfy the constraints \( z_1, y = 1 \), we have \( k = l \) and \( k + l = 2k = \frac{1}{2^d} \) giving \( x_1 = \frac{1}{2^{c+1}}, x_2 = \frac{1}{2^d}, y = 0 \) at \( l_j \).

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

C. 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 \( \omega \)-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 the 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 correctness of this algorithm follows from the observation that do not satisfy \( \phi \).

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

Based on this result, the LTL model checking for a finite Kripke structure \( K \) can be performed in the following manner:

1. Construct a Büchi automaton \( A_\neg \phi \) corresponding to the negation of the LTL property.
2. Construct the composition \( K \otimes A_\neg \phi \) of the Kripke structure \( K \) with the Büchi automaton \( A_\neg \phi \).
3. If the Büchi automaton \( H \otimes A_\neg \phi \) 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 \( H \otimes A_\neg \phi \) as a counter-example.

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

Theorem 5 (LTL model-Checking for Finite Structures [97]):

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 [92, 82].

D. 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 \( H \) there exists a bisimulation relation with some finite state Kripke structure \( K \), then the problem of LTL model-checking for \( H \) can be reduced to the decidable problem of LTL model-checking for finite Kripke structure \( K \).

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

Definition 12 (Bisimulation Relation): Let \( K = (T = (S, S_0, \Sigma, \Delta), P, L) \) and \( K' = (T' = (S', S'_0, \Sigma', \Delta'), P, L') \) be two Kripke structures. A bisimulation relation between \( K \) and \( K' \) is a binary relation \( R \subseteq S \times S' \) such that:

- every initial state of \( T \) is related to some initial state of \( T' \), and vice-versa, i.e. for every \( s \in S_0 \) there exists \( s' \in S'_0 \) such that \( (s, s') \in R \) and for every \( s' \in S'_0 \) there exists a \( s \in S_0 \) such that \( (s, s') \in R \);
- for every \( (s, s') \in R \) 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 \( (t, t') \in R \), 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 \( (t, t') \in R \).

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

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

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

Proof. Let \( T \sim T' \). 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 \( T \) there is a run \( a' = \langle s_0' a_1', s_1, a_2, \ldots \rangle \) of \( T' \) such that \( L(a_i) = L'(a_i') \) for every \( i \geq 0 \). This implies that \( \text{Trace}(r) = \text{Trace}(r') \) and hence \( \text{Trace}(T) \subseteq \text{Trace}(T') \).

Similarly, we can show that \( \text{Trace}(T') \subseteq \text{Trace}(T) \). Hence it follows that \( T \sim T' \) implies \( \text{Trace}(T) = \text{Trace}(T') \).

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 \( \phi \) we have that \( T \sim T' \) implies that \( T \models \phi \) if and only if \( T' \models \phi \).

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

- \( L(s) = L'(s') \);
- if \( t \in \text{POST}(s) \), then there exists an \( t' \in \text{POST}(s') \) such that \( (t, t') \in R \);
— 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 \( s \sim_\mathcal{R} s' \) 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 \( \mathcal{K} = (\mathcal{T} = (S, S_0, \Sigma, \Delta), P, \mathcal{L}) \) 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 \( \mathcal{K}_\mathcal{R} = (\mathcal{T}_\mathcal{R} = (S_\mathcal{R}, S^0_\mathcal{R}, \Sigma_\mathcal{R}, \Delta_\mathcal{R}, P, \mathcal{L}_\mathcal{R}) \) where:

- the state space of \( \mathcal{T}_\mathcal{R} \) is the quotient space of \( \mathcal{T} \), i.e. \( S_\mathcal{R} = \{[s]_\mathcal{R} : s \in S\} \);
- the set of initial states is the set of \( \mathcal{R} \)-equivalence classes of the initial states, i.e. \( S^0_\mathcal{R} = \{[s]_\mathcal{R} : s \in S_0\} \);
- \( \Sigma_\mathcal{R} = \{\tau\} \);
- Each transition \((s, a, s') \in \Delta \) induces a transition from \([s]_\mathcal{R} \) to \([s']_\mathcal{R} \) in \( \Delta_\mathcal{R} \), i.e. \( \Delta_\mathcal{R} = \{([s]_\mathcal{R}, \tau, [s']_\mathcal{R}) : (s, a, s') \in \Delta \} \);
- \( \mathcal{L}_\mathcal{R} \) is defined such that \( \mathcal{L}_\mathcal{R}([s]) = \mathcal{L}(s) \).

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 5 and Theorem 5.

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

**IV. 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 decidability of LTL 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 IV-B) that under certain restriction (initialized) recover decidability of LTL model-checking via reductions to similar problem on timed automata. Finally, in Section IV-C 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.

---

3\(^{\text{Observe that the definition of bisimulation ensures that the state labeling \( L_\mathcal{R} \) is well defined.}}\)
the partial orders of the clocks, determined by their fractional parts in \( \nu \) and \( \nu' \), 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 \( \nu \) and \( \nu' \) are region equivalent, and we write \( \nu \sim_R \nu' \) if and only if:
- either for \( x \in X \) we have \( \nu(x) > K \) and \( \nu'(x) > K \), or
- for any \( x, y \in X \) with \( \nu(x), \nu'(x) \leq K \) and \( \nu(y), \nu'(y) \leq K \) the following conditions hold:
  - \( |\nu(x)| = |\nu'(x)| \) and \( |\nu(x)| = 0 \) if \( |\nu'(x)| = 0 \),
  - \( |\nu(y)| = |\nu'(y)| \) and \( |\nu(y)| = 0 \) if \( |\nu'(y)| = 0 \),
  - \( |\nu(x)| \leq |\nu'(x)| \) if and only if \( |\nu'(x)| \leq |\nu(y)| \).

where \( c \triangleq |c| \) represents the fractional part of \( c \in \mathbb{R}_{\geq 0} \).

It is easy to see that \( \sim_R \) is an equivalence relation. For a clock valuation \( \nu \) we write \( [\nu] \) for the region equivalence class of \( \nu \). 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 \( (m, \nu) \) and \( (m', \nu') \) are region equivalent, and we write \( [(m, \nu)] = [(m', \nu')] \), if and only if \( m = m' \) and \( [\nu] = [\nu'] \).

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 \( (M, M_0, \Sigma, X, \Delta, I, F, \nu_0) \) are bounded from above by \( |M| \cdot |X|! \cdot 2^{|X|} \cdot |\Sigma|^{|X|} \cdot (2K + 2) \).

**Theorem 8 ([17]):** 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 [95, 51]. 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 [73, 24, 33, 88, 63], uncontrollable nondeterminism [19, 20, 17, 71, 31, 38], stochastic behavior [8], [69, 23, 68, 70, 62, 75, 86], and recursion [24, 5]. For a detailed overview of these extensions we refer to [97].

### B. Multi-Rate and Rectangular Hybrid Automata

Multi-rate hybrid automata, introduced by Henzinger and Kopke [58, 87, 59], 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 reset 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, \( c, c' \in \mathbb{R} \) and \( c, c' \in \mathbb{N} \). We write \( \text{rect}(X) \) for this class of rectangular predicates over the set \( X \). Formally, we define a multi-rate hybrid automaton 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 \( \mathcal{H} = (M, M_0, \Sigma, X, \Delta, I, F, \nu_0) \) 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 \( [x] \), i.e. \( g \in \text{rect}(X) \) and
  - the jump predicate \( j \) only permits variable resets to real numbers, i.e. \( j \) is of the form

\[
\land_{x \in Y} (x' = c_x)
\]

where \( Y \subseteq X \) and \( c_x \in \mathbb{R} \) 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:

\[
\land_{x \in X} (\dot{x} = c_{x,m})
\]

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

The semantics of multi-rate automata and the concept of multi-rate Kripke structures is defined in 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

![Fig. 11. A time-sensitive login protocol implemented as a timed automaton.](image-url)
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.

To prove the decidability for an initialized rectangular automaton \(H_r\), we reduce the problem to corresponding problem for an initialized multi-rate automaton \(H_m\). Each variable \(x\) of \(H_r\) with rate in the rectangle \(a \leq \dot{x} \leq b\) is simulated using two variables \(x_1, x_u\) such that \(\dot{x}_1 = a\) and \(\dot{x}_u = b\). The variables \(x_1, x_u\) 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) \(x_1 \leq 10\) 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.

\[\square\]

C. 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.

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 on a generalization of constant-rate multi-mode systems to bounded-rate multi-mode system and showed the decidability of the schedulability problem.

V. Summary

In this article we presented hybrid automata for modeling and formal verification of cyber-physical systems. We begin by showing how 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. We discussed a general approach of finding finite bisimulation quotients 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, UPAL, KRONOS, RED, HYTECH, PHAVER, able to perform automatic verification of systems modeled using them. A growing number of case-studies using these tools have shown dempomice in extending the state-of-the-art to industrial-sized examples.
E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla. Symmetry reductions in model checking. In Computer Aided Verification, pages 147–158. Springer, 1998.

E. M. Clarke, E. A. Emerson, and J. Sifakis. Model checking: algorithmic verification and debugging. Communications of the ACM, 52(11):74–84, 2009.

E. M. Clarke, L. Chen, F. Fehnker, Z. Han, 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):583–604, 2003.

E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In CAV, pages 154–169, 2000.

E. M. Clarke, O. Grumberg, and D. Peled. Model Checking. MIT Press, 1999.

C. Courcoubetis and M. Yannakakis. Minimum and maximum delay problems in real-time systems. In Formal Methods in System Design, volume 1, pages 385–415, Dordrecht, 1992. Kluwer.

L. de Alfaro, M. Faella, T. A. Henzinger, and R. Majumdar. Combined partial order reductions with on-the-fly model checking. In Computer aided verification, pages 377–390. Springer, 1994.

Phaver. Algorithmic verification of hybrid systems past hytech. pages 258–273. Springer, 2005.

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, pages 4604–4609, 2011.

E. Grädel, W. Thomas, and T. Wilke, editors. Automata, Logics, and Infinite Games. A Guide to Current Research, volume 2500 of LNCS. Springer, 2002.

T. A. Henzinger, P. Ho, and H. Wong-ito. Hytech: A model checker for hybrid systems. Software Tools for Technology Transfer, 1:460–463, 1997.

T. A. Henzinger and P. W. Kopke. Discrete-time control for rectangular hybrid automata. Theor. Comput. Sci., 221(1-2):369–392, 1999.

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):94–124, 1998.

Hytech. http://embedded.eecs.berkeley.edu/research/hytech/

Zhihao Jiang, Miroslav Pajic, and Rahul Mangharam. Cyber-physical modeling of implantable cardiac medical devices. Proceedings of the IEEE, 100(1):122–137, 2012.

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.

M. Jurdziński and A. Trivedi. Concavely-priced timed automata. In In Hybrid and Real-Time Systems, volume 5215 of Lecture Notes in Computer Science, pages 1–13. Springer Berlin Heidelberg, 2008.

J. Lygeros, C. Tomlin, and S. Sastry. Hybrid Systems: Modeling, Analysis and Control. In Preparation., 2008. Unpublished manuscript, http://www-verimag.imag.fr/~lygeros/papers/lygeros_book.pdf.

M. Z. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. In ARTS, pages 75–95, 1999.

L. Lamport. Proving the correctness of multiprocess programs. Software Engineering, IEEE Transactions on, SE-3(2):125–143, 1977.

K. G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, T. Hune, P. Pettersson, and J. Růžička. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In G. Berry, H. Conon, and A. Finkel, editors, Proc. CAV’01, volume 2102 of LNCS, pages 493–505, Heidelberg, 2001. Springer.

O. Maler, K. G. Larsen, and B. Krogh. On zone-based analysis of duration probabilistic automata. In INFINITY, pages 33–46, 2010.

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, pages 205–205, New York, NY, USA, 1987. ACM.

Z. Manna and A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems. Springer-Verlag, 1992.

W. Marrisio, E. M. Clarke, and S. Jha. Model checking for security protocols. Technical report, DTIC Document, 1997.

MATLAB. version 7.10.0 (R2010a). The MathWorks Inc., Natick, Massachusetts, 2010.

Kenneth L. McMillan. Symbolic model checking. Springer, 1993.

Marvin L. Minsky. Computation: finite and infinite machines. Prentice-Hall, Inc., 1967.

Usnsv. http://usnsv.fbk.eu/.

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, pages 1–13, Springer Berlin Heidelberg, 2008.

William Pasillas-Lépine. Hybrid model checking and limit cycle analysis for a class of five-phase anti-lock brake algorithms. Vehicle System Dynamics, 44(2):173–188, 2006.

Doron Peled. Combining partial order reductions with on-the-fly model-checking. In Computer aided verification, pages 377–390. Springer, 1994.

Phaver. http://www-verimag.imag.fr/~frehse/phaver_web/

A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusion. In Computer Aided Verification (CAV), pages 95–104, 1994.

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):97–114, 2006.

RED. http://cc.ece.illinois.edu/~finn/red/

Jaajiet Röychowdhury. Numerical simulation and modelling of electronic and biochemical systems. Foundations and Trends in Electronic Design Automation, 3(2—3):97–303, February 2009.

A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. Journal of the ACM, 32(3):733–749, July 1985.

Spin. http://spinroot.com/.

T. Stauner, O. Müller, 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, pages 139–153, Springer Berlin Heidelberg, 1997.

A. Trivedi and D. Wojtczak. Recursive timed automata. In Formal Modeling and Analysis of Timed Systems (FORMATS), volume 5215 of LNCS, pages 48–62. Springer, 2008.

Y. Kesten, A. Pnueli, J. Sifakis, and Yovine. S. Integration graphs: A class of five-phase anti-lock brake algorithms. In Parallel and Distributed Processing, 3(3):12–31, 2002. IEEE.

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):97–114, 2006.

Inc. Wolfram Research. Mathematica Edition: Version 8.0. Wolfram Research, Inc., 2010. Champaign, Illinois.

P. Wolper, M.Y. Vardi, and A. P. Sistla. Reasoning about infinite computation paths. In Foundations of Computer Science, pages 185–194, 1983.
This figure "pend-phase.png" is available in "png" format from:

http://arxiv.org/ps/1503.04928v1