The Theory of Hybrid Automata[*][†]

Thomas A. Henzinger[‡]

Electrical Engineering and Computer Sciences
University of California at Berkeley

Abstract. A hybrid automaton is a formal model for a mixed discrete-continuous system. We classify hybrid automata acoording to what questions about their behavior can be answered algorithmically. The classification reveals structure on mixed discrete-continuous state spaces that was previously studied on purely discrete state spaces only. In particular, various classes of hybrid automata induce finitary trace equivalence (or similarity, or bisimilarity) relations on an uncountable state space, thus permitting the application of various model-checking techniques that were originally developed for finite-state systems.

1 Hybrid Automata

A hybrid system is a dynamical system with both discrete and continuous components. For example, an automobile engine whose fuel injection (continuous) is regulated by a microprocessor (discrete) is a hybrid system. As embedded computing becomes ubiquitous, hybrid systems are increasingly employed in safety-critical applications, thus making reliability a prime concern. Rigorous reliabilty analysis requires formal modeling. For this purpose, the hybrid automaton has been proposed as a formal model for hybrid systems.

1.1 Syntax

A paradigmatic example of a mixed discrete-continuous system is a digital controller of an analog plant. The discrete state of the controller is modeled by the

[*] This research was supported in part by the Office of Naval Research Young Investigator award N00014-95-1-0520, by the National Science Foundation CAREER award CCR-9501708, by the National Science Foundation grant CCR-9504469, by the Air Force Office of Scientific Research contract F49620-93-1-0056, by the Army Research Office MURI grant DAAH-04-96-1-0341, by the Advanced Research Projects Agency grant NAG2-892, and by the Semiconductor Research Corporation contract 96-DC-324.036.

[†] A preliminary version of this paper appeared in the Proceedings of the 11th Annual IEEE Symposium on *Logic in Computer Science* (LICS 96), pp. 278–292.

[‡] Email: tah@eecs.berkeley.edu.

vertices of a graph (control modes), and the discrete dynamics of the controller is modeled by the edges of the graph (control switches). The continuous state of the plant is modeled by points in $\mathbb{R}^n$, and the continuous dynamics of the plant is modeled by flow conditions such as differential equations. The behavior of the plant depends on the state of the controller: each control mode determines a flow condition, and each control switch may cause a discrete change in the state of the plant, as determined by a jump condition. Dually, the behavior of the controller depends on the state of the plant: each control mode continuously observes an invariant condition of the plant state, and by violating the invariant condition, a continuous change in the plant state will cause a control switch.

Definition 1.1 [Hybrid automata] [5, 43, 3] A *hybrid automaton* H consists of the following components.

Variables. A finite set $X = \{x_1, \ldots, x_n\}$ of real-numbered variables. The number n is called the *dimension* of H. We write $\dot{X}$ for the set $\{\dot{x}_1, \ldots, \dot{x}_n\}$ of dotted variables (which represent first derivatives during continuous change), and we write X' for the set $\{x'_1, \ldots, x'_n\}$ of primed variables (which represent values at the conclusion of discrete change).

Control graph. A finite directed multigraph (V, E). The vertices in V are called *control modes*. The edges in E are called *control switches*.

Initial, invariant, and flow conditions. Three vertex labeling functions *init*, *inv*, and *flow* that assign to each control mode $v \in V$ three predicates. Each initial condition $init(v)$ is a predicate whose free variables are from X. Each invariant condition $inv(v)$ is a predicate whose free variables are from X. Each flow condition $flow(v)$ is a predicate whose free variables are from $X \cup \dot{X}$.

Jump conditions. An edge labeling function *jump* that assigns to each control switch $e \in E$ a predicate. Each jump condition $jump(e)$ is a predicate whose free variables are from $X \cup X'$.

Events. A finite set Σ of events, and an edge labeling function $event\colon E \rightarrow \Sigma$ that assigns to each control switch an event. $\Box$

Example 1.1 [Temperature control] The hybrid automaton of Figure 1 models a thermostat. The variable x represents the temperature. In control mode *Off*, the heater is off, and the temperature falls according to the flow condition $\dot{x} = -0.1x$. In control mode *On*, the heater is on, and the temperature rises according to the flow condition $\dot{x} = 5 - 0.1x$. Initially, the heater is off and the temperature is 20 degrees. According to the jump condition $x < 19$, the heater may go on as soon as the temperature falls below 19 degrees. According to the invariant condition $x \geq 18$, at the latest the heater will go on when the temperature falls to 18 degrees. $\Box$

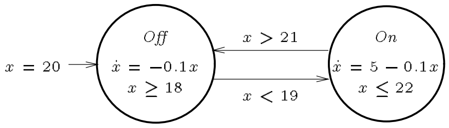

Figure 1: Thermostat automaton

1.2 Safe Semantics

The execution of a hybrid automaton results in continuous change (flows) and discrete change (jumps). The mixed discrete-continuous dynamics can be abstracted by a fully discrete transition system.

Definition 1.2 [Labeled transition systems] A *labeled transition system* S consists of the following components.

State space. A (possibly infinite) set Q of *states*, and a subset $Q^0 \subseteq Q$ of *initial states*.

Transition relations. A (possibly infinite) set A of *labels*, and for each label $a \in A$, a binary relation $\xrightarrow{a}$ on the state space Q. Each triple $q \xrightarrow{a} q'$ is called a *transition*.

A subset $R \subseteq Q$ of the state space is called a *region*. Given a region R and a label $a \in A$, we write $post_a(R) = \{q' \mid \exists q \in R.\, q \xrightarrow{a} q'\}$ for the region of a-successors of R, and we write $pre_a(R) = \{q \mid \exists q' \in R.\, q \xrightarrow{a} q'\}$ for the region of a-predecessors of R. □

For a given hybrid automaton, we define two labeled transition systems. Both transition systems represent discrete jumps by transitions. The timed transition system abstracts continuous flows by transitions, retaining only information about the source, the target, and the duration of each flow. The time-abstract transition system abstracts also the duration of flows.

Definition 1.3 [Transition semantics of hybrid automata] The *timed transition system* S^t_H of the hybrid automaton H is the labeled transition system with the components Q, Q^0, A, and $\xrightarrow{a}$ for each $a \in A$, defined as follows.

- Define $Q, Q^0 \subseteq V \times \mathbb{R}^n$ such that $(v, \mathbf{x}) \in Q$ iff the closed predicate $inv(v)[X := \mathbf{x}]$ is true, and $(v, \mathbf{x}) \in Q^0$ iff both $init(v)[X := \mathbf{x}]$ and $inv(v)[X := \mathbf{x}]$ are true. The set Q is called the *state space* of H, and the subsets of Q are called *H-regions*.
- $A = \Sigma \cup \mathbb{R}_{\geq 0}$.

- For each event $\sigma \in \Sigma$, define $(v, \mathbf{x}) \xrightarrow{\sigma} (v', \mathbf{x}')$ iff there is a control switch $e \in E$ such that (1) the source of e is v and the target of e is v', (2) the closed predicate $\mathit{jump}(e)[X, X' := \mathbf{x}, \mathbf{x}']$ is true, and (3) $\mathit{event}(e) = \sigma$.
- For each nonnegative real $\delta \in \mathbb{R}_{\geq 0}$, define $(v, \mathbf{x}) \xrightarrow{\delta} (v', \mathbf{x}')$ iff $v = v'$ and there is a differentiable function $f : [0, \delta] \rightarrow \mathbb{R}^n$, with the first derivative $\dot{f} : (0, \delta) \rightarrow \mathbb{R}^n$, such that (1) $f(0) = \mathbf{x}$ and $f(\delta) = \mathbf{x}'$, and (2) for all reals $\varepsilon \in (0, \delta)$, both $\mathit{inv}(v)[X := f(\varepsilon)]$ and $\mathit{flow}(v)[X, \dot{X} := f(\varepsilon), \dot{f}(\varepsilon)]$ are true. The function f is called a *witness* for the transition $(v, \mathbf{x}) \xrightarrow{\delta} (v', \mathbf{x}')$.

The *time-abstract transition system* S^a_H of H is the labeled transition system with the components Q, Q^0, B, and $\xrightarrow{b}$ for each $b \in B$, defined as follows.

- Q and Q^0 are defined as above.
- $B = \Sigma \cup \{\tau\}$, for some event $\tau \notin \Sigma$.
- For each event $\sigma \in \Sigma$, define $\xrightarrow{\sigma}$ as above.
- Define $(v, \mathbf{x}) \xrightarrow{\tau} (v', \mathbf{x}')$ iff there is a nonnegative real $\delta \in \mathbb{R}_{\geq 0}$ such that $(v, \mathbf{x}) \xrightarrow{\delta} (v', \mathbf{x}')$.

The time-abstract transition system S^a_H is called the *time abstraction* of the timed transition system S^t_H. □

Remark. [Definition 1.3] The state space Q and the timed label set A are infinite. The time-abstract label set B is finite. For all states q of a hybrid automaton, $q \xrightarrow{0} q$. Sequences of event transitions and time transitions with duration (label) 0 are permitted, which generalizes the interleaving semantics for discrete concurrent systems [7]. □

Remark. [Time vs. phase view] The time-abstract transition system S^a_H, which projects away the time dimension, can be viewed as the phase portrait of the timed transition system S^t_H [25]. □

Remark. [Time-silent transition semantics] None of the results presented in this paper change if the τ-transitions of time-abstract transition systems are considered silent [30]. □

1.3 Live Semantics

If we consider the infinite behavior of a hybrid automaton, then we are interested only in infinite sequences of transitions which do not converge in time. The divergence of time is a liveness assumption, and it is the only liveness assumption we need to consider [24, 33]. A hybrid automaton is nonzeno if it cannot prevent time from diverging. Clearly, only nonzeno designs of real-time systems can be realized.

Definition 1.4 [Live transition systems] Consider a labeled transition system S and a state q_0 of S. A *q_0-rooted trajectory* of S is a finite or infinite sequence of pairs $\langle a_i, q_i\rangle_{i\geq 1}$ of labels $a_i \in A$ and states $q_i \in Q$ such that $q_{i-1}\xrightarrow{a_i}q_i$ for all $i \geq 1$. If q_0 is an initial state of S, then $\langle a_i, q_i\rangle_{i\geq 1}$ is an *initialized trajectory* of S. A *live transition system* (S, L) is a pair consisting of a labeled transition system S and a set L of infinite initialized trajectories of S. The set L of infinite initialized trajectories is *machine-closed* for S if every finite initialized trajectory of S is a prefix of some trajectory in L.[1] If (S, L) is a live transition system, and $\langle a_i, q_i\rangle_{i\geq 1}$ is either a finite initialized trajectory of S or a trajectory in L, then the corresponding sequence $\langle a_i\rangle_{i\geq 1}$ of labels is called a (finite or infinite) *trace* of (S, L). □

Definition 1.5 [Trace semantics of hybrid automata] We associate with each transition of the timed transition system S^t_H a *duration* in $\mathbb{R}_{\geq 0}$. For events $\sigma \in \Sigma$, the duration of $q\xrightarrow{\sigma}q'$ is 0. For reals $\delta \in \mathbb{R}_{\geq 0}$, the duration of $q\xrightarrow{\delta}q'$ is δ. An infinite trajectory $\langle a_i, q_i\rangle_{i\geq 1}$ of the timed transition system S^t_H *diverges* if the infinite sum $\sum_{i\geq 1} \delta_i$ diverges, where each δ_i is the duration of the corresponding transition $q_{i-1}\xrightarrow{a_i}q_i$. An infinite trajectory $\langle b_i, q_i\rangle_{i\geq 1}$ of the time-abstract transition system S^a_H *diverges* if there is a divergent trajectory $\langle a_i, q_i\rangle_{i\geq 1}$ of S^t_H such that for all $i \geq 1$, either $a_i = b_i$ or $a_i, b_i \notin \Sigma$. Let L^t_H be the set of divergent initialized trajectories of the timed transition system S^t_H, and let L^a_H be set of divergent initialized trajectories of the time-abstract transition system S^a_H. The hybrid automaton H is *nonzeno* if L^t_H is machine-closed for S^t_H (or equivalently, L^a_H is machine-closed for S^a_H). Each trace of the live transition system (S^t_H, L^t_H) is called a *timed trace* of H, and each trace of the live transition system (S^a_H, L^a_H) is called a *time-abstract trace* of H. □

Remark. [Traces vs. tubes] It may be argued that hybrid automata are unrealistically expressive in that they can enforce or detect an event at a specific real-numbered instance of time. Such instabilities can be avoided by interpreting the possible behaviors of a hybrid automaton not as a collection of timed traces but as a collection of timed *tubes*, which are bundles of almost identical timed traces. The resulting theory of "fuzzy" hybrid automata does not differ significantly from the results presented in this paper [22]. □

1.4 Composition

For two hybrid automata H_1 and H_2, we define the timed semantics and the time-abstract semantics of the parallel composition $H_1||H_2$. The two hybrid automata H_1 and H_2 interact via joint events: if event a is both an event of H_1 and an event of H_2, then H_1 and H_2 must synchronize on a-transitions; if a is an event of H_1 but not an event of H_2, then each a-transition of H_1 synchronizes with a 0-duration time transition of H_2, and vice versa. For each real $\delta > 0$, a

[1] Assuming that every initial state of S has a successor state.

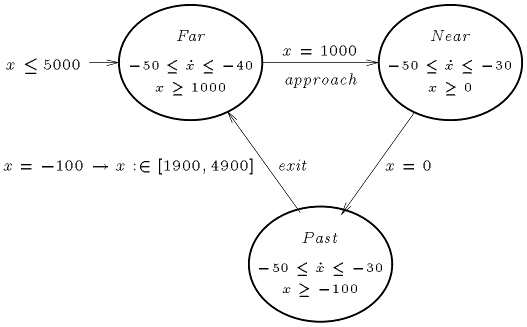
Far
$-50 \leq \dot{x} \leq -40$
$x \geq 1000$
$x \leq 5000$
$x = 1000$
approach
Near
$-50 \leq \dot{x} \leq -30$
$x \geq 0$
$x = -100 \rightarrow x :\in [1900, 4900]$
exit
$x = 0$
Past
$-50 \leq \dot{x} \leq -30$
$x \geq -100$

Figure 2: Train automaton

time transition of H_1 with duration δ must synchronize with a time transition of H_2 with the same duration.

Definition 1.6 [Product of transition systems] A *consistency check* for two labeled transition systems S_1 and S_2 is an associative partial function $\otimes$ on pairs consisting of a transition from S_1 and a transition from S_2. The *product* $S_1 \times S_2$ with respect to the consistency check $\otimes$ is the labeled transition system with the state space $Q_1 \times Q_2$, the set $Q_1^0 \times Q_2^0$ of initial states, the label set *range*$(\otimes)$, and the following transition relations: for each label $a \in range(\otimes)$, define $(q_1, q_2) \xrightarrow{a} (q_1', q_2')$ iff there is a label $a_1 \in A_1$ and a label $a_2 \in A_2$ such that a is the (defined) result of applying $\otimes$ to the two transitions $q_1 \xrightarrow{a_1} q_1'$ and $q_2 \xrightarrow{a_2} q_2'$. □

Definition 1.7 [Composition of hybrid automata] Consider two hybrid automata H_1 and H_2. A transition $q_1 \xrightarrow{a_1} q_1'$ of $S_{H_1}^t$ and a transition $q_2 \xrightarrow{a_2} q_2'$ of $S_{H_2}^t$ are *consistent* if one of the following three conditions is true.

1. $a_1 = a_2$. In this case, the consistency check $\otimes_=$ applied to the transitions $q_1 \xrightarrow{a_1} q_1'$ and $q_2 \xrightarrow{a_2} q_2'$ yields a_1.

2. $a_1 \in \Sigma_1 \backslash \Sigma_2$ and $a_2 = 0$. In this case, the consistency check $\otimes_=$ yields a_1.

3. $a_1 = 0$ and $a_2 \in \Sigma_2 \backslash \Sigma_1$. In this case, the consistency check $\otimes_=$ yields a_2.

The *timed transition system* $S_{H_1 \| H_2}^t$ is defined to be the product $S_{H_1}^t \times S_{H_2}^t$ with respect to the consistency check $\otimes_=$. The *time-abstract transition system* $S_{H_1 \| H_2}^a$ is defined to be the time abstraction of $S_{H_1 \| H_2}^t$. □

Example 1.2 [Railroad gate control] The hybrid automaton of Figure 2 models a train on a circular track with a gate. The variable x represents the distance of the train from the gate. Initially, the speed of the train is between 40 and

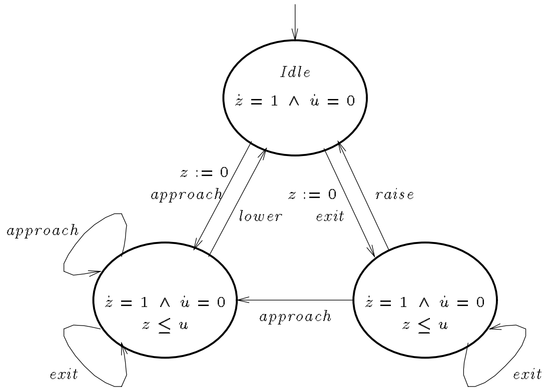
Idle
$\dot{z} = 1 \wedge \dot{u} = 0$
$z := 0$
approach
lower
$z := 0$
exit
raise
approach
$\dot{z} = 1 \wedge \dot{u} = 0$
$z \leq u$
approach
$\dot{z} = 1 \wedge \dot{u} = 0$
$z \leq u$
exit
exit

Figure 3: Controller automaton

50 meters per second. At the distance of 1000 meters from the gate, the train issues an *approach* event and may slow down to 30 meters per second. At the distance of 100 meters past the gate, the train issues an *exit* event. The circular track is between 2000 and 5000 meters long. We write jump conditions as guarded commands, which allows us to suppress conjuncts of the form $x' = x$. In particular, the jump condition of the control switch from *Near* to *Past* is $x = 0 \wedge x' = x$, and the jump condition from *Past* to *Far* is $x = -100 \wedge 1900 \leq x' \leq 4900$. The hybrid automaton of Figure 3 models the gate controller. The variable u is a symbolic constant that represents the reaction delay of the controller. The variable z is a clock for measuring elapsed time. When an *approach* event is received, the controller issues a *lower* event within u seconds, and when an *exit* event is received, the controller issues a *raise* event within u seconds. The hybrid automaton of Figure 4 models the gate. The variable y represents the position of the gate in degrees. Initially, the gate is open ($y = 90$). When a *lower* event is received, the gate starts closing at the rate of 9 degrees per second, and when a *raise* event is received, the gate starts opening at the same rate. Which values of the symbolic constant u ensure that the gate is fully closed ($y = 0$) whenever the train is within 10 meters of the gate ($-10 \leq x \leq 10$)? □

Remark. [Shared variables] The consistency check $\otimes_{=}$ depends only on the transition labels, and not on the source and target states of transitions. Alternative consistency checks can be used to model read-shared and even write-shared variables of hybrid automata [9]. □

Remark. [Time-abstract hybrid automata] The time-abstract transition system $S^a_{H_1 \| H_2}$ is generally different from the product $S^a_{H_1} \times S^a_{H_2}$ of the time-abstract component systems. This is not the case for *time-abstract hybrid au-*

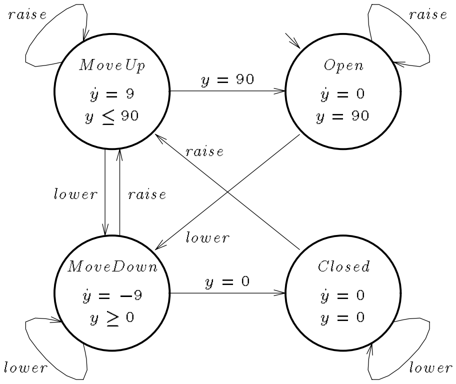

Figure 4: Gate automaton

tomata [25]. Time-abstract design is desirable, because many useful properties of time-abstract component systems are inherited by the compound system. □

Remark. [Receptiveness] The composition of two nonzeno hybrid automata is not necessarily nonzeno. It is an interesting modeling problem for real-time systems to guarantee the liveness of compound designs [1, 21, 39, 8]. □

2 On the Trace Languages of Hybrid Automata

We identify which requirements on the traces of a hybrid automaton can be checked algorithmically, and which cannot.

2.1 Verification Tasks

We study four paradigmatic questions about the traces of a hybrid automaton. The reachability problem is a fundamental subtask for the verification of safety requirements, and the emptiness problem is a fundamental subtask for the verification of liveness requirements. The timed trace inclusion problem compares the traces of a hybrid automaton against a timed specification, and the time-abstract trace inclusion problem compares the traces of a hybrid automaton against a time-abstract specification.

Definition 2.1 [Reachability, emptiness, and trace inclusion] The *reachability problem* for a class $\mathcal{H}$ of hybrid automata asks, given a hybrid automaton H from $\mathcal{H}$ and a control mode v of H, if there is an initialized trajectory of S^t_H (or equivalently, S^a_H) that visits a state of the form $(v, \mathbf{x})$. The *emptiness problem* for $\mathcal{H}$ asks, given a hybrid automaton H from $\mathcal{H}$, if there is a divergent initialized trajectory of S^t_H (or equivalently, S^a_H). The (*finitary*) *timed trace inclusion* *problem* for $\mathcal{H}$ asks, given two hybrid automata H_1 and H_2 from $\mathcal{H}$, if every (finite) timed trace of H_1 is also a timed trace of H_2. The (*finitary*) *time-abstract trace inclusion problem* for $\mathcal{H}$ asks, given two hybrid automata H_1 and H_2 from $\mathcal{H}$, if every (finite) time-abstract trace of H_1 is also a time-abstract trace of H_2. □

Remark. [Definition 2.1] Some of these problems are harder than others. In particular, reachability can be reduced to finitary time-abstract trace inclusion, and emptiness can be reduced to time-abstract trace inclusion. Also, finitary trace inclusion can be reduced to trace inclusion. □

2.2 Rectangular Automata

A hybrid automaton is rectangular if the flow conditions are independent of the control modes, and the variables are pairwise independent. Specifically, in each control mode of a rectangular automaton, the first derivative of each variable is given a range of possible values, and that range does not change with control switches. With each control switch of a rectangular automaton, the value of each variable is either left unchanged, or changed nondeterministically to a new value within a given range of possibilities. The behaviors of the variables are decoupled, because the ranges of possible values and derivative values for one variable cannot depend on the value or derivative value of another variable.

Definition 2.2 [Rectangular automata] [45, 32] A *rectangle* $\mathbf{I} = \prod_{1 \le i \le n} \mathbf{I}_i$ *of dimension* n is the product of n intervals $\mathbf{I}_i \subseteq \mathbb{R}$ of the real line, each with rational or infinite endpoints. The rectangle $\mathbf{I}$ is *bounded* (a *singleton*) if each interval $\mathbf{I}_i$, for $1 \le i \le n$, is bounded (a singleton). A hybrid automaton H is a *rectangular automaton* if the following three restrictions are met. Let $X = \{x_1, \ldots, x_n\}$ be the set of variables of H.

1. For each control mode v of H, the initial condition $init(v)$ has the form $X \in \mathbf{I}^{init(v)}$ for a bounded n-dimensional rectangle $\mathbf{I}^{init(v)}$, and the invariant condition $inv(v)$ has the form $X \in \mathbf{I}^{inv(v)}$ for an n-dimensional rectangle $\mathbf{I}^{inv(v)}$.

2. There is a bounded n-dimensional rectangle $\mathbf{I}^{flow}$ such that for each control mode v of H, the flow condition $flow(v)$ has the form $\dot{X} \in \mathbf{I}^{flow}$.

3. For each control switch e of H, the jump condition $jump(v)$ has the form $X \in \mathbf{I}^{pre(e)} \wedge Y' = Y \wedge X' \in \mathbf{I}^{post(e)}$ for two n-dimensional rectangles $\mathbf{I}^{pre(e)}$ and $\mathbf{I}^{post(e)}$, and a set $Y \subseteq X$ of variables. The control switch e is said to *reinitialize* the variables in $X \backslash Y$. For all $1 \le i \le n$, if the variable x_i is reinitialized by e, then the interval $\mathbf{I}_i^{post(e)}$ must be bounded.

The rectangular automaton H is a *singular automaton* if the flow rectangle $\mathbf{I}^{flow}$ is a singleton. The singular automaton H is a *timed automaton* if $\mathbf{I}^{flow} = [1, 1]^n$. □

Remark. [Clocks and drifting clocks] A clock can be modeled by a variable x_i with the flow interval $\mathbf{I}_i^{\mathit{flow}} = [1,1]$. All variables of a timed automaton are clocks [6]. A clock with drift ε, for $\varepsilon \in \mathbb{Q}_{\geq 0}$, can be modeled by a variable with the flow interval $[1-\varepsilon, 1+\varepsilon]$ [13, 37]. □

Remark. [Composition] Timed, singular, and rectangular automata are closed under composition: for two timed (singular; rectangular) automata H_1 and H_2, we can construct a timed (singular; rectangular) automaton H such that $S_H^t = S_{H_1 \| H_2}^t$ (and therefore, $S_H^a = S_{H_1 \| H_2}^a$). If the dimension of H_1 is n_1 and the dimension of H_2 is n_2, then the dimension of H is $n_1 + n_2$. □

We define two generalizations of rectangular automata. Multirectangular automata allow flow conditions that vary with control switches, and triangular automata allow the comparison of variables.

Definition 2.3 [Multirectangular and triangular automata] A hybrid automaton H is a *multirectangular automaton* if the restrictions of Definition 2.2 are met, except that different control modes v and v' of H may have different flow rectangles $\mathbf{I}^{\mathit{flow}(v)}$ and $\mathbf{I}^{\mathit{flow}(v')}$. The multirectangular automaton H is a *multisingular automaton* if all flow rectangles of H are singletons. The intersection of an n-dimensional rectangle with any number of half-spaces of $\mathbb{R}^n$ that are defined by inequalities of the form $x_i \leq x_j$, for $1 \leq i, j \leq n$, is called a *triangle of dimension n*. A hybrid automaton is a *triangular automaton* if the restrictions of Definition 2.2 are met, except that every rectangle may be a triangle. □

Remark. [Stopwatches and symbolic constants] A stopwatch can be modeled by a multisingular variable x_i with the two flow intervals $\mathbf{I}_i^{\mathit{flow}(v)} = [1,1]$ (the stopwatch is turned on) and $\mathbf{I}_i^{\mathit{flow}(v')} = [0,0]$ (the stopwatch is turned off). Stopwatches are useful for measuring accumulated durations, such as the cumulative amount of time that is spent in control mode v [38, 4]. An unknown system constant can be modeled by a singular variable x_j with the flow interval $\mathbf{I}_j^{\mathit{flow}} = [0,0]$ such that (1) x_j is not reinitialized by any control switch, and (2) the behaviors of other variables may depend on the (unknown but constant) value of x_j [10]. □

Remark. [Initialized multirectangular automata] Some multirectangular automata can be translated to rectangular automata by increasing the dimension. In particular, this is the case for *initialized multirectangular automata*, where for each variable x_i and each control switch e, if the flow interval $\mathbf{I}_i^{\mathit{flow}(v)}$ of the source v of e is different from the flow interval $\mathbf{I}_i^{\mathit{flow}(v')}$ of the target v' of e, then x_i is reinitialized by e [45]. □

Example 2.1 [Railroad gate control] The train automaton of Figure 2 is an initialized 1D multirectangular automaton and can be translated to a 2D rectangular automaton with the same timed traces. The controller automaton of

Figure 3 is a 2D triangular automaton with a clock z and a symbolic constant u. If the reaction delay u of the controller is known (say, 5 seconds), then the controller can be modeled by a 1D timed automaton. The gate automaton of Figure 4 is a 1D multisingular automaton (not initialized). If the direction of the gate cannot be reversed midway, then the gate can be modeled by a singular automaton. □

Remark. [Abstract interpretation] Nonsingular flow intervals permit the conservative approximation of complex continuous behavior with arbitrary accuracy [26]: we may split the state space into regions and within each region, use lower and upper bounds on the first derivatives of all variables. □

2.3 Verification Results

The following theorem ensures the verifiability of rectangular automata against time-abstract finite-state specifications.

Theorem 2.1 [Time-abstract traces] [32] *For every rectangular automaton H, the set of finite time-abstract traces of H is regular, and the set of infinite time-abstract traces of H is ω-regular.*

Proof. Given an rectangular automaton H of dimension n, we can construct a singular automaton H' of dimension $2n$ such that H and H' have the same timed traces. The construction replaces each variable x_i of H by a variable x_i^ℓ of H' that tracks the smallest possible value of x_i, and a variable x_i^u of H' that tracks the largest possible value of H'. In particular, if x_i has the flow interval $[\ell, u]$, then x_i^ℓ has the flow interval $[\ell, \ell]$ and x_i^u has the flow interval $[u, u]$. Alur and Dill have shown that for every timed automaton H' one can construct a Büchi automaton H'' whose traces are exactly the time-abstract traces of H' (see Theorem 3.2 below). Their construction can be generalized to singular automata. □

Corollary 2.1 [Time-abstract trace inclusion] *The time-abstract trace inclusion problem for rectangular automata can be decided in EXPSPACE.*

Remark. [Emptiness] The emptiness problem for rectangular automata is in PSPACE, and the additional exponential for time-abstract trace inclusion is caused by an intermediate complementation step. PSPACE emptiness checking is optimal, because already the reachability problem for timed automata (and other real-time formalisms) is PSPACE-hard [6]. □

Rectangular automata characterize an exact boundary between the decidability and undecidability of verification problems. If the flow conditions are allowed to vary with control switches (multirectangular automata), or if the values of different variables may be related (triangular automata), then already the reachability problem cannot be decided.

Theorem 2.2 [Reachability] [5] *The reachability problems for multisingular automata and for triangular automata are undecidable.*

Proof. Reduction from the halting problem for 2-counter machines. □

Remark. [Theorem 2.2] Theorem 2.2 can be sharpened to more specific statements [10, 32]. For example, the combination of clocks with a single stopwatch causes undecidability, and so does the combination of clocks with symbolic constants. □

We have focused on time-abstract trace inclusion, because there is no hope for deciding timed trace inclusion.

Theorem 2.3 [Timed trace inclusion] [6] *The finitary timed trace inclusion problem for timed automata is undecidable.*

Remark. [Complementation] Theorem 2.3 does not contradict the decidability of the emptiness problem for timed automata (which follows from Theorem 2.1), because the (finitary) timed trace sets of timed automata are not closed under complement [6]. □

3 On the State Spaces of Hybrid Automata

Since the state space of a nontrivial hybrid automaton is infinite, it cannot be explored by enumerating states. We analyze the state space of a hybrid automaton by computing with finite symbolic representations of infinite regions. For example, if x is a real-numbered variable, then the predicate $1 \leq x \leq 5$ is a finite symbolic representation of an infinite set of real numbers.

3.1 Symbolic Analysis of Transition Systems

A labeled transition system can be analyzed using symbolic representations of regions if there are algorithms for performing certain operations on the symbolic representations.

Definition 3.1 [Theories for transition systems] Consider a labeled transition system S with the state space Q. A *theory* T for S is a set of predicates that are assigned truth values by the states in Q. Given a predicate p of T, we write $[\![p]\!]$ for the set of states in Q that satisfy p, and we say that p *defines* the region $[\![p]\!] \subseteq Q$. A set Φ of predicates from T *induces* an equivalence relation $\approx^{\Phi}$ on Q: for all states q and r of S, define $q \approx^{\Phi} r$ iff q and r satisfy the same predicates in Φ. The theory T is *decidable* if for each predicate p of T, it can be decided whether $[\![p]\!]$ is empty. The theory T is *effectively closed under boolean operations* if for all predicates p_1 and p_2 of T, one can construct a predicate $Or(p_1, p_2)$ of T that defines the region $[\![p_1]\!] \cup [\![p_2]\!]$, and a predicate $Not(p_1)$ that defines the region $Q \backslash [\![p_1]\!]$. The theory T is *effectively closed under transitions* if for each predicate p of T, and each label a of S, one can construct a predicate $Post(p, a)$ of T that defines the region $post_a([\![p]\!])$,[2] and a predicate $Pre(p, a)$ that defines the region $pre_a([\![p]\!])$. The theory T *permits the symbolic analysis* of S if (1) T is decidable, (2) T is effectively closed under boolean operations and transitions, and (3) there is a predicate of T that defines the set of initial states of S. □

Definition 3.2 [Similarity, bisimilarity, and trace equivalence] Consider a labeled transition system S with the state space Q, and an equivalence relation $\approx$ on Q. A *$\approx$-simulation* $\preceq$ of S is a binary relation on Q such that $q \preceq r$ implies (1) $q \approx r$ and (2) for each label a of S, if $q \xrightarrow{a} q'$, then there exists a state r' such that $r \xrightarrow{a} r'$ and $q' \preceq r'$. A symmetric $\approx$-simulation is called a *$\approx$-bisimulation*. Two states q and r of S are *$\approx$-similar* if there is a $\approx$-simulation $\preceq$ such that $q \preceq r$ and $r \preceq q$. The two states q and r are *$\approx$-bisimilar* if there is a $\approx$-bisimulation $\simeq$ such that $q \simeq r$. The two states q and r are *trace equivalent* if the labeled transition systems $S[Q^0 := \{q\}]$ and $S[Q^0 := \{r\}]$ have the same finite traces. If T is a theory for S, and Φ is a set of predicates from T, then the $\approx^{\Phi}$-(bi)similarity relation of S is called *Φ-(bi)similarity*. □

Remark. [Definition 3.2] We remind the reader of some well-known facts from concurrency theory. If two states q and r are $\approx$-bisimilar, then q and r are also $\approx$-similar. If there is an equivalence relation $\approx$ such that the two states q and r are $\approx$-similar, then q and r are trace equivalent. The converse of either statement is not necessarily true. □

Bisimilarity and similarity relations of a labeled transition system S can be defined as greatest fixpoints of a monotonic operator, and approximated by iterating the operator. The iteration can be performed in a theory that permits the symbolic analysis of S. The iteration terminates iff the approximated equivalence relation has finitely many equivalence classes.

Definition 3.3 [Finitary equivalences] An equivalence relation $\simeq$ is called *finitary* if there are finitely many $\simeq$-equivalence classes. □

Proposition 3.1 [Symbolic bisimilarity approximation] *Let S be a labeled transition system with a finite label set, let T be a theory that permits the symbolic analysis of S, and let Φ be a finite set of predicates from T. Each step of the procedure BisimApprox (Figure 5) is effective, and upon termination the procedure returns the Φ-bisimilarity relation of S. Furthermore, the procedure BisimApprox terminates iff the Φ-bisimilarity relation of S is finitary.* □

Proposition 3.2 [Symbolic similarity approximation] [23] *Let S, T, and Φ be as in Proposition 3.1. Each step of the procedure SimApprox (Figure 6) is*

[2] Effective closure under *post* is not required for the results presented in this paper.

```
procedure $BisimApprox$:
  Input: a labeled transition system $S$ with label set $A$, and a set
    $\Phi$ of predicates.
  Output: the set $\Pi$ of equivalence classes of the $\Phi$-bisimilarity re-
    lation of $S$.

  let $\Pi$ be the set of $\approx^{\Phi}$-equivalence classes;
  while there are two regions $R, R' \in \Pi$ and a label $a \in A$ such that
        both $R \cap pre_a(R')$ and $R \backslash pre_a(R')$ are nonempty do
    $(R_1, R_2) := (R \cap pre_a(R'), R \backslash pre_a(R'))$;
    $\Pi = (\Pi \backslash \{R\}) \cup \{R_1, R_2\}$
    od;
  return $\Pi$.
```

Figure 5: Symbolic bisimilarity computation

effective, and upon termination the procedure returns the Φ-similarity relation of S. Furthermore, the procedure SimApprox terminates iff the Φ-similarity relation of S is finitary. $\square$

Remark. [Proposition 3.2] For two states q and r of S, there is a $\approx$-simulation $\preceq$ with $q \preceq r$ iff upon termination of the procedure *SimApprox*, $q \in R$ and $r \in sim(R)$ for some region $R \in \Pi$. $\square$

If a finitary bisimilarity or similarity relation of an infinite-state transition system S can be computed, then many verification problems for S can be reduced to finite-state problems. Alternatively, if a verification task can be stated in the μ-calculus, then we may compute directly on the infinite state space without computing a finitary reduction. The μ-calculus defines monotonic operators on regions, and the iteration of these operators can be performed in a theory that permits the symbolic analysis of S. The iteration is guaranteed to terminate if a suitable finitary reduction of the state space exists.

Definition 3.4 [The μ-calculus] Consider a labeled transition system S with the state space Q, and a theory T for S. The formulas of the (S, T)-*based μ-calculus* are generated by the grammar

$$\phi \ ::= \ p \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \exists @ \phi \mid \mu R.\, \phi \mid R$$

for predicates p of T, labels a of S, and region variables R. A formula of the (S, T)-based μ-calculus is *legal* if each occurrence of a region variable is bound by a μ-quantifier and separated from the quantifier by an even number of negations. Given a map F that assigns to each region variable a region of S, every subformula of a legal formula of the (S, T)-based μ-calculus defines a region $[\![\phi]\!]^F \subseteq Q$:

```
procedure $SimApprox$:
  Input: a labeled transition system $S$ with label set $A$, and a set
    $\Phi$ of predicates.
  Output: the set $\Pi$ of equivalence classes of the $\Phi$-similarity relation of $S$.

  let $\Pi$ be the set of $\approx^{\Phi}$-equivalence classes;
  for each region $R \in \Pi$ do $sim(R) := R$ od;
  while there are two regions $R, R' \in \Pi$ and a label $a \in A$ such that
          both $R \cap pre_a(sim(R'))$ and $sim(R) \backslash pre_a(sim(R'))$ are
          nonempty do
    $(R_1, R_2) := (R \cap pre_a(sim(R')), R \backslash pre_a(sim(R')))$;
    $\Pi = (\Pi \backslash \{R\}) \cup \{R_1\}$;
    $sim(R_1) := sim(R) \cap pre_a(sim(R'))$;
    if $R_2$ is nonempty then $\Pi := \Pi \cup \{R_2\}$; $sim(R_2) := sim(R)$ fi
    od;
  return $\Pi$.
```

Figure 6: Symbolic similarity computation

$$\begin{aligned}
&\llbracket p \rrbracket^F = \llbracket p \rrbracket \\
&\llbracket \phi_1 \vee \phi_2 \rrbracket^F = \llbracket \phi_1 \rrbracket^F \cup \llbracket \phi_2 \rrbracket^F \\
&\llbracket \neg\phi \rrbracket^F = Q \backslash \llbracket \phi \rrbracket^F \\
&\llbracket \exists @ \phi \rrbracket^F = pre_a(\llbracket \phi \rrbracket^F) \\
&\llbracket \mu R.\, \phi \rrbracket^F = \bigcap \{ Q' \subseteq Q \mid Q' = \llbracket \phi \rrbracket^{F[R := Q']} \} \\
&\llbracket R \rrbracket^F = F(R)
\end{aligned}$$

The legal formula ϕ of the (S,T)-based μ-calculus *defines* the region $\llbracket \phi \rrbracket = \llbracket \phi \rrbracket^F$, for some map F. The formula ϕ is *existential* if every occurrence of an $\exists\bigcirc$ connective lies within an even number of negations, and ϕ is *universal* if every occurrence of an $\exists\bigcirc$ connective lies within an odd number of negations. □

Remark. [Negation-free μ-formulas] Let $\forall @ \phi$ stand for $\neg\exists @ \neg\phi$, and let $\nu R.\, \phi$ stand for $(\neg\mu R.\, \neg\phi[R := \neg R])$. Using the defined connectives $\wedge$, $\forall\bigcirc$, and ν, every formula of the μ-calculus can be translated into an equivalent formula in negation-free form, where all $\neg$ connectives occur in front of predicates. If ϕ is a formula in negation-free form, then ϕ is existential iff ϕ contains no occurrence of the $\forall\bigcirc$ connective, and ϕ is universal iff ϕ contains no occurrence of the $\exists\bigcirc$ connective. □

Remark. [Reachability and controllability] We mention two of the many system requirements that can be checked using the μ-calculus. Let S be a labeled transition system with a finite label set A, let p^0 be a predicate that defines the set of initial states of S, and let p^* be a predicate that defines a

```
procedure MuApprox(S, $\phi$):
  Input: a labeled transition system $S$, and a legal formula $\phi$ of the
   $\mu$-calculus.
  Output: a predicate $p$ such that $p$ and $\phi$ define the same region
   of $S$.

  case $\phi$ is a predicate $p$:
    return $p$
  case $\phi$ has the form $\phi_1 \vee \phi_2$:
    return $Or(MuApprox(S, \phi_1), MuApprox(S, \phi_2))$
  case $\phi$ has the form $\neg\phi'$:
    return $Not(MuApprox(S, \phi'))$
  case $\phi$ has the form $\exists\textcircled{a}\phi'$:
    return $Pre(MuApprox(S, \phi'), a)$
  case $\phi$ has the form $\mu R.\, \phi'$:
    $p_1 := false$;
    repeat
      $p_2 := p_1$; $p_1 := MuApprox(S, \phi'[R := p_2])$
      until $[\![p_1]\!] = [\![p_2]\!]$;
    return $p_1$
  end case.
```

Figure 7: Symbolic μ-calculus model checking

set of error states of S. There is no trajectory from an initial state to an error state iff the existential μ-formula $p^0 \wedge (\mu R.\, p^* \vee \bigvee_{a \in A} \exists\textcircled{a}R)$ defines the empty region. Suppose that the behavior of S can be influenced by applying a control map that maps each state of S to a label in A [41]: during the execution of S, in each state q, the control map chooses a label a, and the system proceeds to an a-successor of q. There is a control map that keeps the system from entering an error state iff the existential μ-formula $p^0 \wedge (\mu R.\, p^* \vee \bigwedge_{a \in A} \exists\textcircled{a}R)$ defines the empty region. □

Proposition 3.3 [Symbolic μ-calculus approximation] *Let T be a theory that permits the symbolic analysis of the labeled transition system S, let ϕ be a legal formula of the (S,T)-based μ-calculus, and let Φ be the set of predicates that occur in ϕ. Each step of the procedure MuApprox (Figure 7) is effective, and upon termination the procedure returns a predicate p of T such that $[\![p]\!] = [\![\phi]\!]$. Furthermore, the procedure MuApprox is guaranteed to terminate if the Φ-bisimilarity relation of S is finitary. If ϕ is existential or universal, then the procedure MuApprox is guaranteed to terminate if the Φ-similarity relation of S is finitary.* □

Proof. For termination, observe that each predicate that is generated by the procedure *MuApprox* defines a union of equivalence classes of the Φ-bisimilarity relation of S. Furthermore, if ϕ is existential, then the operator *Pre* is applied only to regions R that are closed under simulators; that is, if $q \in R$ and $q \preceq r$ for some Φ-simulation $\preceq$, then $r \in R$. □

Remark. [Termination] If the procedure *BisimApprox* terminates, then the procedure *SimApprox* terminates also. The converse is not necessarily true. For any given input formula, the procedure *MuApprox* may terminate even if the procedures *SimApprox* and *BisimApprox* do not terminate. This encourages practical experimentation, especially since in practice, with concrete time and space constraints, strong termination guarantees are always elusive. □

3.2 Linear Hybrid Automata

The hybrid automata that can be analyzed symbolically in the theory of the reals with addition are called linear.

Definition 3.5 [Linear hybrid automata] A *linear term* is an expression of the form $k_0 + k_1x_1 + \cdots + k_mx_m$, for real-numbered variables $x_1, \ldots, x_m$ and integer constants $k_0, \ldots, k_m$. If t_1 and t_2 are linear terms, then $t_1 \leq t_2$ is a *linear inequality*. A hybrid automaton H is *linear* if the following two restrictions are met.

1. The initial, invariant, flow, and jump conditions of H are boolean combinations of linear inequalities.

2. If X is the set of variables of H, then the flow conditions of H contain free variables from $\dot{X}$ only. □

Remark. [Definition 3.5] The linear hybrid automata are closed under composition. All (multi)rectangular automata and all triangular automata are linear hybrid automata. The use of general linear hybrid automata for approximating complex continuous behavior can be more efficient than the use of rectangular automata [35, 44]. □

Definition 3.6 [Theories for hybrid automata] Consider a hybrid automaton H with the set X of variables and the set V of control modes. An *H-predicate* is a predicate whose free variables are boolean-valued variables from V and real-valued variables from X. A state $(v, \mathbf{x})$ of H *satisfies* the H-predicate p if the closed predicate $p[v, V\backslash\{v\}, X := \mathit{true}, \mathbf{false}, \mathbf{x}]$ is true. The H-predicate p is *linear* if p is a boolean combination of (1) boolean-valued variables from V and (2) linear inequalities whose (real-valued) variables are from X. The linear H-predicate p is a *rectangular H-predicate* if each linear inequality in p can be written in the form $x \leq k$ or $x \geq k$, for a variable x and a rational constant k. An *H-formula of the rectangular μ-calculus* is a formula of the (S_H^a, T)-based μ-calculus, for the set T of rectangular H-predicates. □

Theorem 3.1 [Symbolic analysis of linear hybrid automata] [5, 9] *For every linear hybrid automaton H, the set of linear H-predicates permits the symbolic analysis of the time-abstract transition system S_H^a.*

Proof. The linear H-predicates are quantifier-free formulas of the first-order theory $(\mathbb{R}, +, \leq, 0, 1)$ of the reals with addition, comparison, and integer constants. Each time transition of a linear hybrid automaton has a witness that can be decomposed into a finite sequence of straight lines. Then, using quantification over the reals, the *Post* and *Pre* operations can be expressed in the theory $(\mathbb{R}, + \leq, 0, 1)$. The proof concludes with the observation that the first-order theory $(\mathbb{R}, +, \leq, 0, 1)$ admits quantifier elimination. □

Remark. [Timed automata] The symbolic analysis of singular automata does not require the full theory of linear predicates, which leads to more efficient implementations [20, 11, 12, 19]. For a hybrid automaton A, a linear H-predicate p is a *triangular H-predicate* if each linear inequality in p can be written in the form $x \leq k$, $x \geq k$, $x \leq y + k$, or $x \geq y + k$, for variables x and y and a rational constant k. For every timed automaton H, the set of triangular H-predicates permits the symbolic analysis of the time-abstract transition system S_H^a [34]. □

Remark. [Polynomial hybrid automata] Since the theory of the reals with addition *and* multiplication is decidable, it is possible to define a class of hybrid automata that are more general than linear hybrid automata and can be analyzed symbolically in the more powerful theory. The practicality of such a generalization has not been studied. □

3.3 Bisimilarity and Similarity Relations

From Theorem 2.1 it follows that for every rectangular automaton H, the trace equivalence relation of the time-abstract transition system S_H^a is finitary. If we can identify subclasses of rectangular automata whose time-abstract transition systems have finitary similarity or bisimilary relations, then we obtain termination guarantees for the procedure *MuApprox* applied to hybrid automata.

Definition 3.7 [Timed and time-abstract (bi)similarity] Consider a hybrid automaton H and a set Φ of H-predicates. The Φ-(bi)similarity relation of the timed transition system S_H^t is called the *timed Φ-(bi)similarity relation* of H, and the Φ-(bi)similarity relation of the time-abstract transition system S_H^a is called the *time-abstract Φ-(bi)similarity relation* of H. □

The fundamental theorem of timed automata shows the existence of finitary time-abstract bisimilarity relations for timed automata. This result can be extended to singular automata.

Theorem 3.2 [Time-abstract bisimilarity of singular automata] [6, 5] *If H is a singular automaton, and Φ is a finite set of rectangular H-predicates, then the time-abstract Φ-bisimilarity relation of H is finitary.*

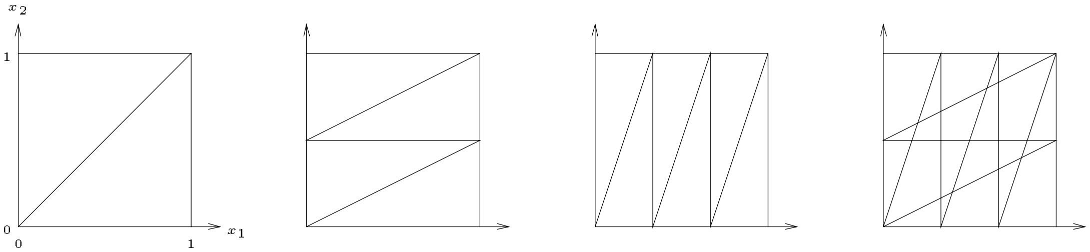

Figure 8: Four finitary equivalence relations on the unit square

Proof. A rectangular automaton H and a finite set Φ of rectangular H-predicates are *normalized* if all non-flow interval endpoints in H and all constants in Φ are integers. Normalization can be achieved by multiplying all non-flow interval endpoints in H and all constants in Φ by a suitably chosen integer constant. Assuming that H and Φ are normalized, let K be the largest integer constant that occurs in H and Φ. If H is a timed automaton, then the (finite) set Φ of triangular H-predicates with integer constants no larger than K induces a finitary Φ-bisimulation on the time-abstract transition system S_H^a. The first panel of Figure 8 shows the induced bisimulation on the unit square $[0,1]^2$ for a 2D timed automaton H. For instance, if $v = v'$ and $0 < \mathbf{x}_1 < \mathbf{x}_2 < 1$ and $0 < \mathbf{x}'_1 < \mathbf{x}'_2 < 1$, then the two states $(v, \mathbf{x})$ and $(v', \mathbf{x}')$ of H are time-abstract Φ-bisimilar. If H is a singular automaton, then a slight extension of triangular H-predicates needs to be considered. For example, the second and third panels of Figure 8 show the induced bisimulations on the unit square for 2D singular automata with the flow rectangles $[2,2] \times [1,1]$ and $[1,1] \times [3,3]$, respectively. □

Corollary 3.1 [Symbolic μ-calculus model checking for singular automata] *The procedure MuApprox terminates if given the time-abstract transition system of a singular automaton H and an H-formula ϕ of the rectangular μ-calculus.*

Remark. [Nonsingular automata and nonrectangular predicates] Singular automata with rectangular predicates identify a boundary between the existence and nonexistence of finitary bisimilarity relations. In fact, for the following three generalizations, bisimilarity degenerates to equality on infinite state spaces. Let $\Phi_1 = \{x_1 = 1, x_2 = 1, x_1 \leq x_2\}$ and $\Phi_2 = \{x_1 = 1, x_2 = 1\}$.

1. There is an (infinite-state) singular automaton H such that the time-abstract Φ_1-bisimilarity relation of H is equality.

2. There is an (infinite-state) multisingular automaton H such that the time-abstract Φ_2-bisimilarity relation of H is equality.

3. There is an (infinite-state) 2D rectangular automaton H such that the time-abstract Φ_2-bisimilarity relation of H is equality. □

The boundary between the existence and nonexistence of finitary similarity relations lies at 2D rectangular automata.

Theorem 3.3 [Time-abstract similarity of 2D rectangular automata] [23] *If H is a 2D rectangular automaton, and Φ is a finite set of rectangular H-predicates, then the time-abstract Φ-similarity relation of H is finitary.*

Proof. The structure of the finitary time-abstract similarity relation is best illustrated with an example. Let H be a 2D rectangular automaton with the flow rectangle $[1, 2] \times [1, 3]$. Assuming that H and Φ are normalized, the fourth panel of Figure 8 shows a finitary kernel of a time-abstract Φ-simulation on the unit square. The simulation is obtained by intersecting the bisimulations for the two cases of extremal flow: maximal $\dot{x}_1$ and minimal $\dot{x}_2$, and vice versa. □

Corollary 3.2 [Symbolic μ-calculus model checking for 2D rectangular automata] *The procedure MuApprox terminates if given the time-abstract transition system of a 2D rectangular automaton H and an existential or universal H-formula ϕ of the rectangular μ-calculus.*

Theorem 3.3 does not generalize to rectangular automata of arbitrary dimension.

Theorem 3.4 [Time-abstract similarity of 3D rectangular automata] [30] *Let $\Phi = \{x_1 = 1, x_2 = 1, x_3 = 1\}$. There is an (infinite-state) 3D rectangular automaton H such that the time-abstract Φ-similarity relation of H is equality.*

In summary, rectangular automata are a maximal class of hybrid automata with finitary time-abstract trace equivalence relations, 2D rectangular automata are a maximal class of hybrid automata with finitary time-abstract similarity relations, and singular automata are a maximal class of hybrid automata with finitary time-abstract bisimilarity relations.

Remark. [Context-free equivalences] We have restricted ourselves to decidability results that can be obtained by relating hybrid automata to finite automata. Additional decidability results can be obtained by relating hybrid automata to pushdown automata [14, 16]. Little is known, however, about which classes of hybrid automata are time-abstract trace equivalent (similar; bisimilar) to pushdown automata. □

Remark. [Timed (bi)similarity] We have focused on time-abstract state space equivalences, because the timed counterparts are infinitary already for nontrivial timed automata. From Theorem 2.3 it follows that for timed automata, the timed trace equivalence of two states cannot be decided. It should be noted, however, that timed similarity and timed bisimilarity can be decided for timed automata. Specifically, if H is a timed automaton, Φ is a finite set of rectangular H-predicates, and q and r are two states of H, then it can be decided if q and r are timed Φ-(bi)similar [17, 48, 47]. □

Remark. [Dense vs. discrete time] In our model, jumps may happen at any real-numbered times. By contrast, in sampling control, all control switches occur at multiples of a rational sampling interval. Sampling control can therefore be modeled by *discrete-time hybrid automata*, where all jumps happen (without loss of generality) at integer times [31]. For discrete-time hybrid automata, verification questions can be answered even in the *multi*rectangular case. This is because for every discrete-time multirectangular hybrid automaton H, the set of rectangular H-predicates permits the symbolic analysis of the time-abstract transition system S_H^a. It follows that if H is a discrete-time multirectangular hybrid automaton with either bounded invariant rectangles or nonnegative flow rectangles, and Φ is a finite set of rectangular H-predicates, then the time-abstract Φ-bisimilarity relation of H is finitary. □

3.4 Computation Tree Logics

We have studied the structure of the timed and time-abstract transition systems of hybrid automata. These transition systems, however, may not be directly useful for (dis)proving assertions about the behavior of a hybrid automaton H, because each trajectory of S_H^t and S_H^a only samples a piecewise-continuous trajectory of H at certain discrete points. In the following, we restrict ourselves to the time-abstract view. Since each time transition of S_H^a abstracts all information about intermediate states that are visited, by looking only at a trajectory of S_H^a, it is impossible to check if the corresponding piecewise-continuous trajectory of H visits any given state or region. We solve this problem by defining (time-abstract) observational transition systems, where each time transition is labeled with a region: the time transition t is labeled with the region R iff all intermediate states and the target state of t lie within R. Thus, an observational transition system results from the continuous observation of a hybrid automaton, with limited observational power: for a given set $\mathcal{R}$ of regions, it can be observed whether a continuous trajectory fragment stays within any of the regions from $\mathcal{R}$.

Definition 3.8 [Piecewise-continuous semantics of hybrid automata] Consider a hybrid automaton H and a set $\mathcal{R}$ of H-regions. The *$\mathcal{R}$-observational transition system* $S_H^{\mathcal{R}}$ of H is the labeled transition system with the components Q, Q^0, A, and $\xrightarrow{c}$ for each $c \in C$, defined as follows.[3]

- Q and Q^0 are defined as in Definition 1.3.
- $C = \Sigma \cup \mathcal{R}$.
- For each event $\sigma \in \Sigma$, define $\xrightarrow{\sigma}$ as in Definition 1.3.

[3] If $\mathcal{R} = \{R\}$ for a single H-region R, then we write S_H^R for the $\mathcal{R}$-observational transition system $S_H^{\mathcal{R}}$.

- For each region $R \in \mathcal{R}$, define $(v,\mathbf{x})\xrightarrow{R}(v',\mathbf{x}')$ iff there is a nonnegative real $\delta \in \mathbb{R}_{\geq 0}$ and a witness f for the transition $(v,\mathbf{x})\xrightarrow{\delta}(v',\mathbf{x}')$ such that for all reals $\varepsilon \in (0,\delta]$, $(v, f(\varepsilon)) \in R$. The real δ is a *possible duration* of the transition $(v,\mathbf{x})\xrightarrow{R}(v',\mathbf{x}')$.

An infinite trajectory $\langle c_i, q_i\rangle_{i\geq 1}$ of the $\mathcal{R}$-observational transition system $S_H^{\mathcal{R}}$ *diverges* if there is an infinite sequence $\langle \delta_i\rangle_{i\geq 0}$ of reals such that (1) the infinite sum $\sum_{i\geq 1}\delta_i$ diverges, and (2) for all $i \geq 0$, either $a_i \in \Sigma$ and $\delta_i = 0$, or $a_i \in \mathcal{R}$ and δ_i is a possible duration of the corresponding transition $q_{i-1}\xrightarrow{a_i}q_i$. A set Φ of H-predicates *permits the observational symbolic analysis* of the hybrid automaton H if Φ permits the symbolic analysis of the observational transition system $S_H^{\mathcal{R}}$, where $\mathcal{R}$ is the set of H-regions that are definable by predicates in Φ. An equivalence relation $\simeq$ on Q is an *observational Φ-(bi)similarity relation* of H if $\simeq$ is the Φ-(bi)similarity relation of the observational transition system $S_T^{\mathcal{R}}$, where $\mathcal{R}$ is the set of $\simeq$-equivalence classes. □

Since observational transition systems are defined in a time-abstract way, the results of Theorems 3.1, 3.2, and 3.3 carry over from time-abstract to observational transition systems.

Proposition 3.4 [Observational symbolic analysis of linear hybrid automata] *For every linear hybrid automaton H, the set of linear H-predicates permits the observational symbolic analysis of H. Consider a finite set Φ of rectangular H-predicates. If H is a singular automaton, then there is a finitary observational Φ-bisimilarity relation of H. If H is a 2D rectangular automaton, then there is a finitary observational Φ-similarity relation of H.*

For (dis)proving assertions about the infinite behavior of hybrid automata, we need to take into account the liveness assumption that time diverges. Computation tree logics for hybrid automata are branching-time temporal logics for stating requirements about divergent piecewise-continuous trajectories.

Definition 3.9 [Computation tree logics] Consider a linear hybrid automaton H with the state space Q. The *H-formulas of linear* Ctl are generated by the grammar

$$\phi \ ::= \ p \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \phi_1 \exists\mathcal{U} \phi_2 \mid \exists\Box\phi$$

for linear H-predicates p. Every H-formula of linear Ctl defines an H-region $[\![\phi]\!] \subseteq Q$:

$[\![\phi_1 \vee \phi_2]\!] = [\![\phi_1]\!] \cup [\![\phi_2]\!]$
$[\![\neg\phi]\!] = Q \backslash [\![\phi]\!]$
$q \in [\![\phi_1 \exists\mathcal{U} \phi_2]\!]$ iff the observational transition system $S_H^{[\![\phi_1 \vee \phi_2]\!]}$ has a (finite) q-rooted trajectory that visits a state in $[\![\phi_2]\!]$
$q \in [\![\exists\Box\phi]\!]$ iff the observational transition system $S_H^{[\![\phi]\!]}$ has a divergent q-rooted trajectory

The H-formula ϕ of linear CTL is an *H-formula of rectangular* CTL if all predicates that occur in ϕ are rectangular H-predicates. The H-formula ϕ of rectangular CTL is an H-formula of *rectangular* $\exists$CTL if each occurence of the $\exists\mathcal{U}$ connective in ϕ and each occurrence of the $\exists\Box$ connective in ϕ lies within the scope of an even number of negations, and ϕ is an H-formula of *rectangular* $\forall$CTL if all occurrences of $\exists\mathcal{U}$ and $\exists\Box$ lie within the scope of odd numbers of negations. $\Box$

Remark. [Definition 3.9] The semantics of linear CTL is defined to be piecewise-continuous (it refers to trajectories of observational rather than time-abstract transition systems), live (it refers to divergent trajectories), and strict (the temporal connectives do not impose requirements on the current state; for example, a nonstrict version of $\phi_1\exists\mathcal{U}\phi_2$ can be defined as $\phi_2 \vee (\phi_1\exists\mathcal{U}\phi_2)$). The disjunction in the semantic clause for the $\exists\mathcal{U}$ connective is necessary, because a switch from ϕ_1 being true to ϕ_2 being true may occur from a right-closed to a left-open interval. In the following, we write $\exists\Diamond\phi$ for $true\exists\mathcal{U}\phi$, and $\forall\Box\phi$ for $\neg\exists\Diamond\neg\phi$. $\Box$

If a system requirement is given as a formula of a computation tree logic, then the corresponding verification task is a model-checking problem.

Definition 3.10 [Model checking] The *model-checking problem* for a class $\mathcal{H}$ of hybrid automata and a computation tree logic $\mathcal{L}$ asks, given a hybrid automaton H from $\mathcal{H}$ and an H-formula ϕ from $\mathcal{L}$, if $[\![\phi]\!]$ contains all initial states of H. $\Box$

For a linear hybrid automaton H, an H-formula ϕ of linear CTL can be translated into a formula ϕ' of the μ-calculus. The piecewise-continuity of linear CTL is taken care of by interpreting ϕ' over an observational transition system $S_H^{\mathcal{R}}$. The liveness of linear CTL is taken of by interpreting ϕ' over an extension of $S_H^{\mathcal{R}}$ with a clock variable that can observe the divergence of time. The translation leads to a model-checking procedure for linear hybrid automata and linear CTL.

Definition 3.11 [Clock extension] A *clock automaton* H_z is a timed automaton with a single variable, z, a single control mode with the initial condition $z = 0$ and the invariant condition $true$, and a single control switch with the jump condition $z' = 0$. If H is a hybrid automaton and z is not a variable of H, then the composition $H\|H_z$ is called a *clock extension* of H. A procedure is an *effective procedure for the observational symbolic analysis* of the hybrid automaton H if each step of the procedure is either effective or a subroutine call of the form $MuApprox(S_{H'}^{[\![p]\!]}, \phi)$, for a clock extension H' of H, a linear H'-predicate p, and a formula ϕ of the $(S_{H'}^{[\![p]\!]}, T)$-based μ-calculus, where T is the set of linear H'-predicates. $\Box$

Theorem 3.5 [From CTL to the μ-calculus] [34, 9] *Let H be a linear hybrid automaton and let ϕ be an H-formula of linear* CTL. *There is an effective procedure for the observational symbolic analysis of H which, upon termination,* *returns a linear H-predicate p with $[\![p]\!] = [\![\phi]\!]$. Furthermore, the procedure is guaranteed to terminate if H is a singular automaton and ϕ is a formula of rectangular* CTL*, and if H is a 2D rectangular automaton and ϕ is a formula of rectangular* $\exists$CTL *or of rectangular* $\forall$CTL*.*

Proof. The CTL formula $\phi_1 \exists\mathcal{U} \phi_2$ can be translated to the formula $(\mu R.\, \exists Ⓒ(\phi_2 \vee R))$, for $c = [\![\phi_1 \vee \phi_2]\!]$ of the μ-calculus. The H-formula $\exists\Box\phi$ can be translated to the H'-formula $(\nu R.\, \phi\exists\mathcal{U}(\phi \wedge z = 1 \wedge \phi\exists\mathcal{U}(\phi \wedge z = 0 \wedge R)))$, where $H' = H \| H_z$ is a clock extension of H. The latter formula asserts that ϕ is true throughout some infinite trajectory along which $z = 1$ is true infinitely often and $z = 0$ is true infinitely often. This can be the case if and only if the trajectory diverges. □

Corollary 3.3 [CTL model checking] *The model-checking problem for rectangular* CTL *is PSPACE-decidable for singular automata. The model-checking problems for rectangular* $\exists$CTL *and rectangular* $\forall$CTL *are PSPACE-decidable for 2D rectangular automata.*

Remark. [HyTech] The procedure of Theorem 3.5 for checking linear CTL requirements of linear hybrid automata has been implemented in the tool HyTech [27, 28, 29]. The procedure has been found to terminate on several examples of practical interest that do not fall into any of the classes for which a priori termination guarantees can be given [37, 42, 18, 36, 46]. □

Example 3.1 [Railroad gate control] Recall the safety requirement for the railroad gate controller from Example 1.2, namely, that the gate is fully closed whenever the train is within 10 meters of the gate. This requirement is expressed by the formula $(\mathit{Far} \wedge \mathit{Idle} \wedge \mathit{Open}) \rightarrow \forall\Box(-10 \leq x \leq 10 \rightarrow \mathit{Closed})$ of rectangular $\forall$CTL. HyTech simplifies this CTL formula, fully automatically, to a linear predicate whose projection onto the u-dimension is $5u < 49$. It follows that the safety requirement is met if and only if the reaction delay u of the controller is less than 9.8 seconds. □

Remark. [Nonzenoness] The semantics of the $\exists\mathcal{U}$ connective of linear CTL is defined over finite trajectories. The alternative interpretation of $\exists\mathcal{U}$ over finite prefixes of divergent trajectories requires that the underlying hybrid automaton is nonzeno. For a linear hybrid automaton H and a clock extension $H \| H_z$, the rectangular existential formula $\phi_{nz} = (\nu R.\, \exists\Diamond(z = 1 \wedge \exists\Diamond(z = 0 \wedge R)))$ defines the set of states q with divergent q-rooted trajectories. Hence, if ϕ_{nz} can be simplified to a linear H-predicate p_{nz}, then the addition of p_{nz} as a conjunct to all invariant conditions of H results in a nonzeno linear hybrid automaton H_{nz} such that H and H_{nz} have the same divergent timed traces. From Theorem 3.5 it follows that this is always possible for singular and 2D rectangular automata. □

Remark. [Ctl with clocks, stopwatches, and symbolic constants] The H-formulas of linear Ctl can be generalized to permit real-numbered variables that are not variables of the hybrid automaton H. In this way, linear Ctl has been extended to include clocks (Tctl) [7, 2], stopwatches [15, 9], and symbolic constants [49]. The symbolic-analysis result of Theorem 3.5 continues to hold for these logics, and the decidability results of Corollary 3.3 continue to hold for Tctl. Isolated decidability results are known also for computation tree logics with a limited use of stopwatches or symbolic constants [49, 4]. □

Acknowledgments. My view of mixed discrete-continuous systems has been shaped in collaborations with Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Pei-Hsin Ho, Peter Kopke, Zohar Manna, Xavier Nicollin, Amir Pnueli, Anuj Puri, Joseph Sifakis, Howard Wong-Toi, Pravin Varaiya, Moshe Vardi, and Sergio Yovine. My special thanks go to Oded Maler, Zohar, and Amir for introducing me to the problem domain [40], to Pei-Hsin and Howard for implementing HyTech, and to Peter for many valuable comments on several drafts of this manuscript.

References

[1] M. Abadi and L. Lamport. An old-fashioned recipe for real time. *ACM Transactions on Programming Languages and Systems*, 16(5):1543–1571, 1994.

[2] R. Alur, C. Courcoubetis, and D.L. Dill. Model checking in dense real time. *Information and Computation*, 104(1):2–34, 1993.

[3] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[4] R. Alur, C. Courcoubetis, and T.A. Henzinger. Computing accumulated delays in real-time systems. *Formal Methods in System Design*, 11(2):137–156, 1997.

[5] 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 R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems I*, Lecture Notes in Computer Science 736, pages 209–229. Springer-Verlag, 1993.

[6] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[7] R. Alur and T.A. Henzinger. Logics and models of real time: a survey. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors,

Real Time: Theory in Practice, Lecture Notes in Computer Science 600, pages 74–106. Springer-Verlag, 1992.

[8] R. Alur and T.A. Henzinger. Modularity for timed and hybrid systems. In A. Mazurkiewicz and J. Winkowski, editors, *CONCUR 97: Concurrency Theory*, Lecture Notes in Computer Science 1243, pages 74–88. Springer-Verlag, 1997.

[9] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[10] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, pages 592–601. ACM Press, 1993.

[11] R. Alur and R.P. Kurshan. Timing analysis in COSPAN. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 220–231. Springer-Verlag, 1996.

[12] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL: a tool-suite for automatic verification of real-time systems. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 232–243. Springer-Verlag, 1996.

[13] D. Bosscher, I. Polak, and F. Vaandrager. Verification of an audio-control protocol. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *FTRTFT 94: Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science 863, pages 170–192. Springer-Verlag, 1994.

[14] A. Bouajjani, R. Echahed, and R. Robbana. Verification of context-free timed systems using linear hybrid observers. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science, pages 118–131. Springer-Verlag, 1994.

[15] A. Bouajjani, R. Echahed, and J. Sifakis. On model checking for real-time properties with durations. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science*, pages 147–159. IEEE Computer Society Press, 1993.

[16] A. Bouajjani and R. Robbana. Verifying ω-regular properties for subclasses of linear hybrid systems. In P. Wolper, editor, *CAV 95: Computer-aided Verification*, Lecture Notes in Computer Science 939, pages 437–450. Springer-Verlag, 1995.

[17] K. Čerāns. Decidability of bisimulation equivalence for parallel timer processes. In G. von Bochmann and D.K. Probst, editors, *CAV 92: Computer-aided Verification*, Lecture Notes in Computer Science 663, pages 302–315. Springer-Verlag, 1992.

[18] J.C. Corbett. Timing analysis of ADA tasking programs. *IEEE Transactions on Software Engineering*, 22(7):461–483, 1996.

[19] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 208–219. Springer-Verlag, 1996.

[20] D.L. Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *CAV 89: Automatic Verification Methods for Finite-state Systems*, Lecture Notes in Computer Science 407, pages 197–212. Springer-Verlag, 1989.

[21] R. Gawlick, R. Segala, J.F. Sogaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. In S. Abiteboul and E. Shamir, editors, *ICALP 94: Automata, Languages, and Programming*, Lecture Notes in Computer Science 820, pages 166–177. Springer-Verlag, 1994.

[22] V. Gupta, T.A. Henzinger, and R. Jagadeesan. Robust timed automata. In O. Maler, editor, *HART 97: Hybrid and Real-time Systems*, Lecture Notes in Computer Science 1201, pages 331–345. Springer-Verlag, 1997.

[23] M.R. Henzinger, T.A. Henzinger, and P.W. Kopke. Computing simulations on finite and infinite graphs. In *Proceedings of the 36rd Annual Symposium on Foundations of Computer Science*, pages 453–462. IEEE Computer Society Press, 1995.

[24] T.A. Henzinger. Sooner is safer than later. *Information Processing Letters*, 43(3):135–141, 1992.

[25] T.A. Henzinger. Hybrid automata with finite bisimulations. In Z. Fülöp and F. Gécseg, editors, *ICALP 95: Automata, Languages, and Programming*, Lecture Notes in Computer Science 944, pages 324–335. Springer-Verlag, 1995.

[26] T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In P. Wolper, editor, *CAV 95: Computer-aided Verification*, Lecture Notes in Computer Science 939, pages 225–238. Springer-Verlag, 1995.

[27] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: the next generation. In *Proceedings of the 16th Annual Real-time Systems Symposium*, pages 56–65. IEEE Computer Society Press, 1995.

[28] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, editors, *TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1019, pages 41–71. Springer-Verlag, 1995.

[29] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. In O. Grumberg, editor, *CAV 97: Computer-aided Verification*, Lecture Notes in Computer Science 1254, pages 460–463. Springer-Verlag, 1997.

[30] T.A. Henzinger and P.W. Kopke. State equivalences for rectangular hybrid automata. In U. Montanari and V. Sassone, editors, *CONCUR 96: Concurrency Theory*, Lecture Notes in Computer Science 1119, pages 530–545. Springer-Verlag, 1996.

[31] T.A. Henzinger and P.W. Kopke. Discrete-time control for rectangular hybrid automata. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *ICALP 97: Automata, Languages, and Programming*, Lecture Notes in Computer Science 1256, pages 582–593. Springer-Verlag, 1997.

[32] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the 27th Annual Symposium on Theory of Computing*, pages 373–382. ACM Press, 1995.

[33] T.A. Henzinger, P.W. Kopke, and H. Wong-Toi. The expressive power of clocks. In Z. Fülöp and F. Gécseg, editors, *ICALP 95: Automata, Languages, and Programming*, Lecture Notes in Computer Science 944, pages 417–428. Springer-Verlag, 1995.

[34] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[35] T.A. Henzinger and H. Wong-Toi. Linear phase-portrait approximations for nonlinear hybrid systems. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 377–388. Springer-Verlag, 1996.

[36] T.A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, Lecture Notes in Computer Science 1165, pages 265–282. Springer-Verlag, 1996.

[37] P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In P. Wolper, editor, *CAV 95: Computer-aided Verification*, Lecture Notes in Computer Science 939, pages 381–394. Springer-Verlag, 1995.

[38] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: a class of decidable hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 179–208. Springer-Verlag, 1993.

[39] N.A. Lynch, R. Segala, F. Vaandrager, and H.B. Weinberg. Hybrid I/O Automata. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 496–510. Springer-Verlag, 1996.

[40] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 447–484. Springer-Verlag, 1992.

[41] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In E.W. Mayr and C. Puech, editors, *STACS 95: Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 900, pages 229–242. Springer-Verlag, 1995.

[42] S. Nadjm-Tehrani and J.-E. Strömberg. Proving dynamic properties in an aerospace application. In *Proceedings of the 16th Annual Real-time Systems Symposium*, pages 2–10. IEEE Computer Society Press, 1995.

[43] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems I*, Lecture Notes in Computer Science 736, pages 149–178. Springer-Verlag, 1993.

[44] A. Puri, V. Borkar, and P. Varaiya. ε-approximation of differential inclusions. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 362–376. Springer-Verlag, 1996.

[45] A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pages 95–104. Springer-Verlag, 1994.

[46] T. Stauner, O. Müller, and M. Fuchs. Using HyTech to verify an automotive control system. In O. Maler, editor, *HART 97: Hybrid and Real-time Systems*, Lecture Notes in Computer Science 1201, pages 139–153. Springer-Verlag, 1997.

[47] S. Taşiran, R. Alur, R.P. Kurshan, and R.K. Brayton. Verifying abstractions of timed systems. In U. Montanari, editor, *CONCUR 96: Concurrency Theory*, Lecture Notes in Computer Science, pages 546–562. Springer-Verlag, 1996.

[48] K. Čerāns, J.C. Godskesen, and K.G. Larsen. Timed modal specification: Theory and tools. In C. Courcoubetis, editor, *CAV 93: Computer-aided Verification*, Lecture Notes in Computer Science 697, pages 253–267. Springer-Verlag, 1993.

[49] F. Wang. Timing behavior analysis for real-time systems. In *Proceedings of the Tenth Annual Symposium on Logic in Computer Science*, pages 112–122. IEEE Computer Society Press, 1995.