Algorithmic Analysis of Nonlinear Hybrid Systems[*]

Thomas A. Henzinger and Pei-Hsin Ho

Computer Science Department, Cornell University, Ithaca, NY 14853
(tah|ho)@cs.cornell.edu

Abstract. Hybrid systems model discrete programs that are embedded in continuous environments. Model-checking tools are available for the analysis of *linear* hybrid systems, whose continuous variables are bounded by piecewise-linear trajectories. Most embedded programs, however, operate in *nonlinear* environments. We present, analyze, and apply two algorithms for translating nonlinear hybrid systems into linear hybrid systems.

The *clock translation* replaces nonlinear variables by clock variables; the *rate translation* approximates nonlinear variables by piecewise-linear envelopes. Both translations are sound for reachability; that is, if we establish a safety property of the translated linear system, we may conclude that the original nonlinear system satisfies the property. The clock translation is also complete for reachability; that is, the original system and the translated system satisfy the same safety properties. The two translations apply to incomparable classes of nonlinear hybrid systems. From the clock translation we obtain a new decidability result for hybrid systems.

With the help of HyTech, a symbolic model checker for linear hybrid systems, we automatically verify a nonlinear railroad gate control program using the clock translation, and a nonlinear temperature control program using the rate translation.

1 Introduction

Hybrid systems are digital real-time systems that are embedded in analog environments. Due to the rapid development of digital-processor technology, hybrid systems directly control much of what we depend on in our daily lives. Many hybrid systems, ranging from automobiles to aircraft, operate in safety-critical situations and therefore call for rigorous analysis techniques. Consequently, the formal specification and verification of hybrid systems has become an active area of research [7]. The symbolic model-checking approach, in particular, has been successfully extended from discrete systems [5] to real-time systems [9] and linear hybrid systems [1, 4]. While the symbolic model-checking method developed in these papers is limited to piecewise-linear dynamics, many typical real-time parameters (e.g., temperature) behave in a nonlinear fashion. The present paper extends the model-checking approach to the analysis of certain nonlinear systems.[2]

[*] This research was supported in part by the NSF grant CCR-9200794, by the AFOSR contract F49620-93-1-0056, and by the DARPA grant NAG2-892.

[2] Control theory, of course, has a long tradition of analyzing what we call nonlinear hybrid systems (in control theory, the term *linearity* usually refers to differential equations, not trajectories). There, however, the number of discrete modes of a controller is typically quite small. Model checking, on the other hand, allows the analysis of controllers that are defined by arbitrary finite-state programs. While the control theorists start with complex environments—differential equations—and steadily in-

We model hybrid systems as *hybrid automata* [2, 10] (Section 2). A hybrid automaton operates with a finite control over a data space of both discrete and continuous variables. The discrete variables are updated by automaton transitions; the continuous variables evolve according to differential equations. In [4], we introduced the symbolic model checker HYTECH (the Cornell *Hy*brid *Tech*nology Tool) for analyzing linear hybrid automata. Here, we turn to the analysis of nonlinear hybrid automata, which alone provide an accurate model for most real-time environments (for example, the temperature of a furnace decreases along an exponential curve with negative exponent).[3]

A hybrid automaton defines an infinite-state transition system, and the reachability analysis of a hybrid automaton requires the computation of weakest preconditions in the underlying transition system. As we know-how to compute weakest preconditions accurately only for linear hybrid automata, we propose a two-step methodology for verifying a nonlinear hybrid automaton A. In Step 1, we translate A into a linear hybrid automaton B. In Step 2, we apply the HYTECH tool to the translated automaton B. The translation is *$\mathcal{P}$-sound*, for a class $\mathcal{P}$ of properties, if all $\mathcal{P}$-properties of B are inherited by A; *$\mathcal{P}$-complete*, if all $\mathcal{P}$-properties of A are inherited by B. Incomplete translations may lead to false negatives: B may not satisfy the $\mathcal{P}$-property φ although A does. We therefore accompany incomplete translations with error analyses: given a metric d on hybrid automata, what is the automaton A' d-closest to A such that if A' satisfies φ, then so does B? If the d-difference between A and A' can be made arbitrarily small, then the translation is called *asymptotically $\mathcal{P}$-complete* under the metric d.

We present two translations, which transform two incomparable classes of nonlinear hybrid automata into linear hybrid automata. The *clock translation* (Section 3) replaces nonlinear variables by clocks (i.e., linear variables with slope 1). The clock translation is applicable to the nonlinear variable x if the value of x is always uniquely determined by the latest assignment to x and the time that has expired since that assignment. If the clock translation of the automaton A yields the automaton B, then the underlying transition systems are timed bisimilar. It follows that the clock translation is both sound and complete for all branching-time properties. If all variables of a nonlinear hybrid automaton can be replaced by clocks, then the resulting linear hybrid automaton is a timed automaton [3]. As a corollary, we obtain a new decidable class of hybrid automata. We verify a nonlinear railroad gate controller using the clock translation and the HYTECH model checker.

The *rate translation* (Section 4) approximates nonlinear variables by piecewise-linear variables. The rate translation is applicable to the nonlinear variable x if the value of x is bounded. If the rate translation of the automaton A yields the automaton B, then the transition system of B simulates the transition system of A (but not vice versa). It follows that the rate translation, while sound for all linear-time properties, is not complete for safety properties. We show that the rate translation is asymptotically complete for safety properties. We verify a nonlinear temperature controller using the rate translation and the HYTECH model checker. Technically, both the clock translation and the rate translation can be viewed as abstract interpretations of nonlinear hybrid automata [6].

crease the complexity of the controllers that can be analyzed, we computer scientists start with complex controllers—programs!—and steadily increase the complexity of the environments.

[3] We insist on representing nonlinear behavior accurately in our underlying *model*, because we feel that linearization or digitization ought to occur *after* the modeling phase, so that the errors that are introduced by these processes can be analyzed and bounded. Such an analysis, indeed, is performed in the present paper for both linearizations we propose.

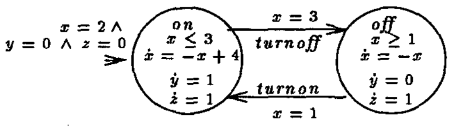


Fig. 1. Thermostat automaton

2 Hybrid Automata

Informally, a hybrid automaton consists of a finite vector $\mathbf{x}$ of real-valued variables and a labeled multigraph (V, E). The edges E represent discrete system actions and are labeled with nondeterministic guarded assignments to $\mathbf{x}$. The vertices V represent continuous environment activities and are labeled with constraints on the derivatives of $\mathbf{x}$. The state of the automaton changes either through instantaneous system actions or, while time elapses, through continuous activities.

For example, the hybrid automaton of Figure 1 models a thermostat that controls the temperature of a manufacturing plant. The nonlinear variable x represents the plant temperature. In control location *on*, a heater is turned on; in control location *off*, the heater is turned off. The variable x follow the differential equations $\dot{x} = -x + 4$ in location *on* and $\dot{x} = -x$ in location *off*. Initially, the temperature is 2 and the heater is turned on. If a thermometer detects that the plant temperature reaches 3, the heater is turned off. If the thermometer detects that the temperature falls to 1, the heater is turned on. The two variables y and z are auxiliary variables: y, a stop watch, records the accumulated time that the heater is turned on, and z, a clock, records the total elapsed time.

Syntax Let $\mathbf{y}$ be a vector of real-valued variables. A *linear term* over $\mathbf{y}$ is a linear combination of variables from $\mathbf{y}$ with real coefficients.[4] A *linear inequality* over $\mathbf{y}$ is a nonstrict inequality between linear terms over $\mathbf{y}$. A *hybrid automaton* A consists of the following components [2]:

Data variables A finite vector $\mathbf{x} = (x_1, \ldots, x_n)$ of real-valued *data variables*. The thermostat automaton in Figure 1 has the vector (x, y, z) of data variables. A *data state* is a point in n-dimensional space $\mathbb{R}^n$. A *convex data region* is a convex polyhedron in $\mathbb{R}^n$; a *data region* is a finite union of convex data regions. A *convex data predicate* is a conjunction of linear inequalities (e.g., $x_3 \leq 2x_2 - 1 \wedge 6 \leq x_3$). A *data predicate* is a disjunction of convex data predicates. Each (convex) data predicate ϕ defines, then, a (convex) data region $[\![\phi]\!] \subseteq \mathbb{R}^n$ such that $\mathbf{s} \in [\![\phi]\!]$ iff $\phi[\mathbf{x} := \mathbf{s}]$ is true.
An *action predicate* is a conjunction of linear inequalities over $\mathbf{x} \cup \mathbf{x}'$, where $\mathbf{x}' = (x'_1, \ldots, x'_n)$ is the vector of primed data variables. Each action predicate ψ defines a function from data states $\mathbf{s} \in \mathbb{R}^n$ to convex data regions $[\![\psi]\!](\mathbf{s}) \subseteq \mathbb{R}^n$: $\mathbf{s}' \in [\![\psi]\!](\mathbf{s})$ iff $\psi[\mathbf{x}, \mathbf{x}' := \mathbf{s}, \mathbf{s}']$ is true.

Control locations A finite set V of vertices called *locations*. The thermostat automaton has the locations *on* and *off*. A *state* $(v, \mathbf{s})$ of the hybrid automaton A consists of a location $v \in V$ and a data state $\mathbf{s} \in \mathbb{R}^n$. A *region* $\bigcup_{v \in V}\{(v, S_v)\}$ is a collection of data regions $S_v \subseteq \mathbb{R}^n$, one for each location $v \in V$. A *state predicate* is a collection $\bigcup_{v \in V}\{(v, \phi_v)\}$ of data predicates ϕ_v, one for each location $v \in V$. When writing state predicates, we omit disjuncts of the form (v, false). Each state predicate $\varphi = \bigcup_{v \in V}\{(v, \phi_v)\}$ defines, then, the region $[\![\varphi]\!] = \bigcup_{v \in V}\{(v, [\![\phi_v]\!])\}$.

[4] We shall specify coefficients by symbolic expressions like *ln*3.

Initial conditions A labeling function *init* that assigns to each location $v \in V$ a convex data predicate $init(v)$, the *initial condition* of v. The control of A may start in the location v only when the initial condition $init(v)$ is true. In the graphical representation of hybrid automata, we usually suppress initial condition of the form *false*. In the thermostat automaton, the initial conditions of the locations *on* and *off* are $x = 2 \wedge y = 0 \wedge z = 0$ and *false*, respectively. We write I_A for the *initial region* $\bigcup_{v \in V}\{(v, [\![init(v)]\!])\}$ of A.

Location invariants A labeling function *inv* that assigns to each location $v \in V$ a convex data predicate $inv(v)$, the *invariant* of v. The control of A may reside in the location v only while the invariant $inv(v)$ is true. In the graphical representation, we usually suppress invariants of the form *true*. In the thermostat automaton, the invariants of the locations *on* and *off* are $x \leq 3$ and $x \geq 1$, respectively. We write Σ_A for the *admissible region* $\bigcup_{v \in V}\{(v, [\![inv(v)]\!])\}$ of A.

Continuous activities A labeling function *dif* that assigns to each location $v \in V$ and each data variable x_i an *activity* $dif(v, x_i)$, which is either (1) a linear term $dif(v, x_i) = f(\mathbf{x})$ over $\mathbf{x}$; or (2) a *rate interval* $dif(v, x_i) = [a, b]$, which is a bounded or unbounded, closed interval of the real line with $\pm\infty$ (that is, $a, b \in \mathbb{R} \cup \{-\infty, \infty\}$ and $a \leq b$). In the first case, if the control of A resides in the location v, the value of x_i evolves according to the first-order differential equation $\dot{x}_i = f(\mathbf{x})$. In the second case, the interval endpoints a and b give a lower bound and an upper bound on the derivative of x_i; that is, $a \leq \dot{x}_i \leq b$.

The data variable x is *linear* if for all locations $v \in V$, $dif(v, x)$ is a rate interval; otherwise, x is *nonlinear*. The hybrid automaton A is *linear* if all data variables in $\mathbf{x}$ are linear [4]; otherwise, A is *nonlinear*. In the thermostat automaton, the data variables y and z are linear. The variable x is a nonlinear variable with $dif(on, x) = -x + 4$ and $dif(off, x) = -x$.

Edges A finite multiset E of *edges*. Each edge (v, v') identifies a source location $v \in V$ and a target location $v' \in V$. For each location $v \in V$, there is a *stutter edge* $e_v = (v, v)$. In the graphical representation, we usually omit stutter edges. The thermostat automaton has four edges: (on, off), (off, on), (on, on), and (off, off).

Discrete actions A labeling function *act* that assigns to each edge $e \in E$ a convex action predicate $act(v, v')$, the *action* of e. If the control of A proceeds from the location v to the location v' via the edge $e = (v, v')$, then the values of all data variables change from $\mathbf{s}$ nondeterministically to any point within the data region $[\![act(e)]\!](\mathbf{s})$. For example, an edge with the action label $x_1 \leq 3 \wedge 3 \leq x_1' \leq 5 \wedge x_2' = x_2 \wedge x_3' = x_1 + 1$ can be traversed only when the value of x_1 is at most 3. By traversing the edge, the value of x_1 changes to any real number in the interval $[3, 5]$, the value of x_2 remains unchanged, and the new value of x_3 is 1 plus the old value of x_1. The stutter edges are labeled with the action *true*. In the graphical representation of actions, if the primed variable x' occurs only in the conjunct $x' = x$, then we usually omit that conjunct. In the thermostat automaton, $act(on, off)$ is $x = 3 \wedge x' = x \wedge y' = y \wedge z' = z$.

Edge labels A finite set L of *edge labels* and a labeling function *label* that assigns to each edge $e \in E$ a label from L. For all stutter edges e_v, $label(e_v) = v$. The edge labels are used to define the parallel composition of hybrid automata. In the thermostat automaton, $label(on, off)$ is *turnoff*.

Accepting conditions A labeling function *final* that assigns to each location $v \in V$ a data predicate $final(v)$, the *accepting condition* of v. We use the accepting conditions to check safety properties of hybrid automata. In the graphical representation of hybrid automata, we suppress accepting conditions of the form *true*. We write F_A for the *accepting region* region $\bigcup_{v \in V}\{(v, [\![final(v)]\!])\}$ of A.

The data or action predicate ϕ is *rational* if all constants that occur in ϕ are rational. The hybrid automaton A is *rational* if all data and action predicates that specify the initial and accepting conditions, invariants, and actions of A are rational. A state predicate is rational if it contains only rational data predicates.

A special case of a linear hybrid automaton is a timed automaton [3]. The data variable x of the hybrid automaton A is a *clock* if $dif(v,x) = [1,1]$ for all locations v of A; that is, each clock always increases with the rate at which time advances. An atomic data predicate is *simple* if it has the form $x \leq c$ or $x \geq c$, for some $c \in \mathbb{R}$; an atomic action predicate is *simple* if it is a simple atomic data predicate or has the form $x' = c$ or $x' = x$. The data variable x of A is *simple* if in all initial and accepting conditions, invariants, and actions of A, x and x' occur only in atomic data and action predicates that are simple. The hybrid automaton A is a *timed automaton* if all data variables of A are simple clocks.

Semantics At any time instant, the state of a hybrid automaton A specifies a location and values for all data variables. The state can change in two ways: (1) by an instantaneous move through an edge that changes both the location and the values of data variables according to the corresponding action; or (2) by a time delay that changes only the values of data variables in a continuous manner according to the activities of the current control location. A *data trajectory* (δ, ρ) in location v of the hybrid automaton A consists of a nonnegative *duration* $\delta \in \mathbb{R}_{\geq 0}$ and a differentiable function $\rho\colon [0,\delta] \rightarrow \mathbb{R}^n$ such that

1. for all reals $t \in [0,\delta]$, $\rho(t) \in [\![inv(v)]\!]$;
2. for each data variable x_i with $dif(v,x_i) = f(\mathbf{x})$, for all reals $t \in (0,\delta)$, $d\rho(x_i)(t)/dt = f(\mathbf{x})$ (where $\rho(x_i)(t)$ denotes the ith component of the data state $\rho(t)$); and
3. for each data variable x_i with $dif(v,x_i) = [a,b]$, for all reals $t \in (0,\delta)$, $d\rho(x_i)(t)/dt \in [a,b]$.

We define the the following two transition relations on the admissible states of the hybrid automaton A:

Edge step For all states $\sigma_1 = (v,\mathbf{s}_1)$ and $\sigma_2 = (v',\mathbf{s}_2)$ of A, and all edge labels ℓ, $\sigma_1 \xrightarrow{\ell} \sigma_2$ if $\sigma_1, \sigma_2 \in \Sigma_A$, and there exists an edge e from v to v' such that $label(e) = \ell$ and $\mathbf{s}_2 \in [\![act(e)]\!](\mathbf{s}_1)$.

Time step For all states $\sigma_1 = (v,\mathbf{s}_1)$ and $\sigma_2 = (v,\mathbf{s}_2)$ of A, and all durations $\delta \in \mathbb{R}_{\geq 0}$, $\sigma_1 \xrightarrow{\delta} \sigma_2$ if there exists a data trajectory (δ,ρ) in location v such that $\rho(0) = \mathbf{s}_1$ and $\rho(\delta) = \mathbf{s}_2$.

The hybrid automaton A defines the labeled transition system $[\![A]\!] = \langle \Sigma_A, I_A, \mathcal{L}, \rightarrow_A, F_A \rangle$ that consists of (1) the infinite state space Σ_A, (2) the set I_A of initial states, (3) the alphabet $\mathcal{L} = L \cup \mathbb{R}_{\geq 0}$, (4) the transition relation $\rightarrow_A = \bigcup\{\xrightarrow{\delta} \mid \delta \geq 0\} \cup \bigcup\{\xrightarrow{\ell} \mid \ell \in L\}$, and (5) the set F_A of accepting states. A *trajectory* τ of A is a finite path $\sigma_0 \xrightarrow{m_0} \sigma_1 \xrightarrow{m_1} \cdots \xrightarrow{m_{k-1}} \sigma_k$ in $[\![A]\!]$ such that $\sigma_0 \in I_A$ and for all $i \in \{0,\ldots,k-1\}$, $(\sigma_i \xrightarrow{m_i} \sigma_{i+1}) \in \rightarrow_A$. The trajectory τ is *accepting* if the final state of τ is accepting; that is, $\sigma_k \in F_A$. The *reachable region* $R(A) \subseteq \Sigma_A$ of the hybrid automaton A is the set of all final states on trajectories of A.

Parallel composition A hybrid system typically consists of many components that operate concurrently and communicate with each other. We describe each component as a hybrid automaton. The edge labels can be used to synchronize various system components. The hybrid automaton that models the entire system is then constructed from the component automata using a product operation that is defined in the standard way [4].

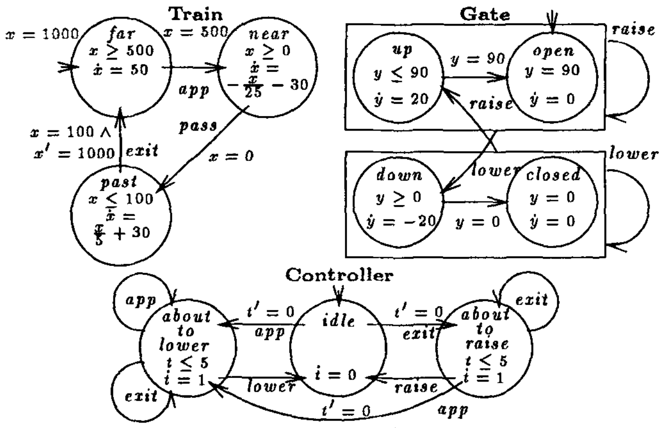


Fig. 2. A railroad gate controller

Example: Nonlinear railroad gate controller We present a nonlinear variation of a familiar example [4]. The three hybrid automata of Figure 2 model three processes—a train, a gate, and a controller. The nonlinear variable x represents the distance of the train from the gate. Initially, the train is far from the gate and moves at the speed 50 meters per second. When the train approaches the gate, a sensor placed at a distance of 500 meters from the crossing detects the train and sends the signal *app* to the controller. The train starts to slow down and the controller waits 5 seconds before sending the command *lower* to the gate; the delay of the controller is modeled by the clock t. Consequently, the gate is lowered from 90 radius degrees to 0 degrees at the constant rate of 20 degrees per second; the position of the gate in degrees is represented by variable y. After passing the gate, the train begins to accelerate. A second sensor placed at 100 meters past the crossing detects the leaving train and signals *exit* to the controller, which, after another delay of 5 seconds, sends the command *raise* to the gate. The distance between consecutive trains is (at least) 1,000 meters. Notice that the synchronization signals like *app*, *exit* and *raise* are modeled by edge labels.

The emptiness problem The *emptiness problem* for hybrid automata asks, given a hybrid automaton A, if A has an accepting trajectory. If the accepting region F_A represents the set of "unsafe" states specified by a safety property, then the safety property can be verified by checking the emptiness of A, namely, the safety property is satisfied by A iff A has no accepting trajectory. The emptiness problem is decidable for rational timed automata [3] and certain rational simple linear hybrid automata [8], and semidecidable for rational linear hybrid automata [2].

For the thermostat automaton of Figure 1, we will verify the safety property whose "unsafe" region is characterized by the state predicate $\bigcup_{v \in \{on, off\}} \{(v, 6 \leq z \leq 2y-1)\}$; that is, after 6 time units the heater has always been on at most half of the time plus 1 time unit. For the railroad gate controller of Figure 2, we will verify the safety property whose unsafe region is characterized by the state predicate $\bigcup_{v \not\models closed} \{(v, x \leq 100)\}$; that is, whenever the train is within 100 meters from the gate, the gate is closed (we write $v \models closed$ if the gate component of location v is *closed*).

3 Clock Translation

The clock translation of a hybrid automaton replaces each nonlinear data variable x by a clock t_x that is restarted whenever the value of x is changed by a discrete action. The clock translation is applicable if at every point of a trajectory, the value of x is uniquely determined by the value of t_x.

Solvable automata Let A be a hybrid automaton. The simple nonlinear data variable x of A is *(rationally) determined* in the location v of A if $dif(v,x) = f(x)$ and for all (rational) initial values $x_0 \in \mathbb{R}$, the initial-value problem "$\dot{x}(t) = f(x)$; $x(0) = x_0$" has an algebraic solution $x_{v,x_0}(t)$ such that for each constant c that appears in an atomic data predicate $x \sim c$, for $\sim \in \{\leq, \geq\}$, or an atomic action predicate $x' = c$ of A, the function $x_{v,x_0}(t) - c$ has a finite number of (rational) roots. For example, suppose that the function $x_{v,x_0}(t) - c_0$ has the two roots r_0 and r_1, and all trajectories that enter the location v have the initial value x_0 for x. Then the value of the variable x is $x_{v,x_0}(t_x)$ for all reachable states in v. Thus an exit edge of location v guarded with $x = c_0$ can be replaced by two exit edges guarded with $t_x = r_0$ and $t_x = r_1$, respectively.

The location v of A is *definite* for the data variable x if the initial condition $init(v)$ implies $x = c$, for some *initial value* $c \in \mathbb{R}$. The edge e of A is *definite* for x if the action $act(e)$ implies $x' = c$, for some *arrival value* $c \in \mathbb{R}$. The simple nonlinear data variable x of A is *solvable* if the following three conditions hold:

1. For all locations v of A, x is determined in v.
2. All locations of A are definite for x.
3. For all edges $e = (v, v')$ of A, if $dif(v,x) \not\equiv dif(v',x)$, then e is definite for x;

For example, the nonlinear variable x of the thermostat automaton of Figure 1 is solvable, and so is the the nonlinear variable x of the train automaton of Figure 2. The hybrid automaton A is *solvable* if all nonlinear data variables of A are simple and solvable. The automaton A is *rationally solvable* if A is (1) rational, (2) solvable, and (3) all data variables of A are rationally determined in all locations of A. All (rational) timed automata are (rationally) solvable and the class of (rationally) solvable hybrid automata is closed under parallel composition. For each solvable data variable x, we collect the initial values of x for all locations and the arrival values of x for all definite edges in the finite set $CritVal(x) \subseteq \mathbb{R}$ of *critical values* for x.

The clock translation algorithm Given a solvable nonlinear hybrid automaton A, we construct a linear hybrid automaton A^c—the *clock translation* of A—by replacing each nonlinear data variable x with a new clock t_x. For each nonlinear data variable x, the construction proceeds in two steps:

1. Let $CritVal(x) = \{c_1, \ldots, c_n\}$ with $c_1 < \cdots < c_n$. Each location v of A is split into a collection $v_{c_1}, \ldots, v_{c_n}$ of locations, one for each critical value c_i of x. We then add the clock t_x such that the value of x in location v_{c_i} is $x(t_x)$, where $x(t)$ is the solution of the initial-value problem "$\dot{x}(t) = dif(v,x)$; $x(0) = c_i$".
2. All initial and accepting conditions, invariants, and actions are translated from conditions on x to conditions on t_x.

We now provide more details.

Step 1. Splitting locations and edges After the application of Step 1, each new location v_{c_i} has the same activities as v and, in addition, the new activity $dif(v_{c_i}, t_x) = 1$ for the clock t_x. The new location v_{c_i} has the the initial condition $init(v) \wedge t_x = 0$, the accepting condition $final(v)$, and the invariant $inv(v)$. For each indefinite edge $e = (v, v')$, we introduce all edges of the form (v_{c_i}, v'_{c_i}) with the action $act(e)$ and the

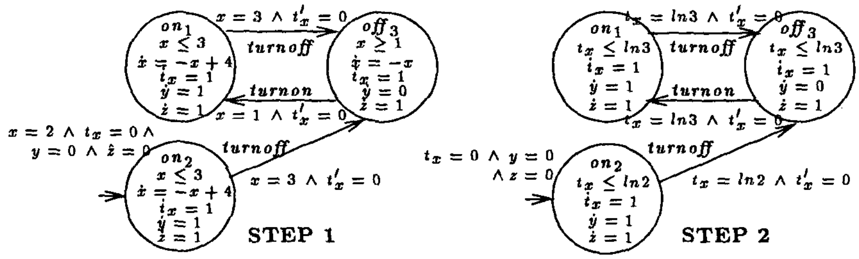


Fig. 3. Clock translation of the thermostat automaton

label *label*(e); for each definite edge $e = (v, v')$ with the arrival value c_j, we introduce all edges of the form (v_{c_i}, v'_{c_j}) with the action $act(e) \wedge t'_x = 0$ and the label *label*(e). For example, the thermostat automaton of Figure 1 has only definite edges. The critical values of x are $1, 2$, and 3, so we split both locations *on* and *off* into three locations each. Since the locations on_3, off_1, and off_2 are not reachable by a sequence of automaton edges from the initial location on_2, we remove these three locations from the clock-translated automaton. The result of Step 1 is shown on the left in Figure 3.

Step 2. Updating accepting conditions, invariants, and actions Let the function $x(t)$ be the solution of the initial-value problem "$\dot{x}(t) = dif(v, x)$; $x(0) = c_i$". We now eliminate the nonlinear variable x from the initial and accepting conditions, invariant, and exit-edge actions of each new location v_{c_i}.

First, we simply remove all the atomic data predicates that involve the variable x from the initial condition $init(v_{c_i})$. Second, we translate the accepting condition of location v_{c_i}. Suppose that $x \leq c$ is an atomic data predicate of the accepting condition $final(v_{c_i})$ (other atomic data predicates are handled similarly). We find all finite roots $r_0, \ldots, r_k$ of $x(t) - c$ (count roots with zero derivatives twice). If no such root exists and $c_i \leq c$, then $x \leq c$ is always satisfied and we replace $x \leq c$ by *true*; if no root exists and $c_i > c$, then $x \leq c$ is not satisfiable and we replace $x \leq c$ by *false*. Otherwise, if $c_i \leq c$, then $x \leq c$ is satisfied when the value of t_x is in any of the intervals $[0, r_0], [r_1, r_2], \ldots$; if $c_i > c$, then $x \leq c$ is satisfied when the value of t_x is in any of the intervals $[r_0, r_1], [r_2, r_3], \ldots$ We therefore replace $x \leq c$ by the disjunction $\bigvee_{[r_i, r_{i+1}] \in I} r_i \leq t_x \leq r_{i+1}$, where I is the set of *root intervals* during which $x \leq c$ is satisfied. The result can be transformed into disjunctive normal form.

Third, we translate the invariant of location v_{c_i}. Suppose that $x \leq c$ is a conjunct of the invariant $inv(v_{c_i})$ (other conjuncts are handled similarly). If $c_i > c$, then the arrival value of x does not satisfy the invariant, and thus we remove the location v_{c_i}. Otherwise, we find the smallest nonnegative finite root r of $x(t) - c$. If such a root r exists, then the automaton control can reside in the location v_{c_i} up to r time units, and thus we replace the conjunct $x \leq c$ of the invariant by the conjunct $t_x \leq r$. If no such root r exists, then the automaton control can reside in the location v_{c_i} forever, and we replace the conjunct $x \leq c$ by *true*.

Fourth, we translate the actions of all edges that leave the location v_{c_i}. Suppose that $x \leq c$ is a conjunct of the action $act(e)$, where $e = (v, v')$. We find all finite roots $r_0, \ldots, r_k$ of $x(t) - c$ (count roots with zero derivatives twice). If no such root exists and $c_i \leq c$, then the edge e is always enabled and we replace the conjunct $x \leq c$ by *true*; if no root exists and $c_i > c$, we remove the edge e. Otherwise, if $c_i \leq c$, then the edge e is enabled when the value of t_x is in any of the intervals $[0, r_0], [r_1, r_2], \ldots$; if $c_i > c$, then e is enabled when the value of t_x is in any of the intervals $[r_0, r_1], [r_2, r_3], \ldots$ For each root interval $[r_i, r_{i+1}]$ during which e is enabled, we introduce an edge with the action $act(e)$ and the label $label(e)$ except that (1) the conjunct $x \leq c$ is replaced by the conjunct $r_i \leq t_x \leq r_{i+1}$ and (2) any atomic subformula involving x' is removed. In the thermostat example, we have $x_{on_1}(t) = -3e^{-t} + 4$, $x_{on_2}(t) = -2e^{-t} + 4$, and $x_{off_3}(t) = 3e^{-t}$. Consider the action $x = 3$ of the edge from on_2 to off_3. Since $ln2$ is the unique root of $-2e^{-t} + 4 - 3$, it follows that $x = 3$ iff $t_x = ln2$. Hence we replace the action $x = 3$ with the action $t_x = ln2$. The final result of Step 2 is shown on the right in Figure 3.

Soundness, completeness, and decidability We show that the clock translation is both sound and complete for checking the emptiness of solvable automata. Let A be a solvable hybrid automaton, and let A^c be the automaton clock translated from A by translating a nonlinear variable x into a clock t_x. We show that A and A^c are bisimilar.

We first recall the definition of (timed) bisimulation. Let $T_1 = \langle \Sigma_1, I_1, \mathcal{L}, \rightarrow_1 \rangle$ and $T_2 = \langle \Sigma_2, I_2, \mathcal{L}, \rightarrow_2 \rangle$ be two labeled transition systems. The binary relation $\approx \subseteq \Sigma_1 \times \Sigma_2$ is a *bisimulation* between T_1 and T_2 if for all states $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$, $\sigma_1 \approx \sigma_2$ implies for every letter $m \in \mathcal{L}$ that (1) if $\sigma_1 \xrightarrow{m} \sigma_1'$, then there exists a state σ_2' such that $\sigma_2 \xrightarrow{m} \sigma_2'$ and $\sigma_1' \approx \sigma_2'$; and (2) if $\sigma_2 \xrightarrow{m} \sigma_2'$, then there exists a state σ_1' such that $\sigma_1 \xrightarrow{m} \sigma_1'$ and $\sigma_1' \approx \sigma_2'$. The two states $\sigma \in \Sigma_1$ and $\sigma' \in \Sigma_2$ are *bisimilar* if there exists a bisimulation $\approx$ between T_1 and T_2 such that $\sigma \approx \sigma'$. The labeled transition systems T_1 and T_2 are *bisimilar*, denoted $T_1 \approx T_2$, if for each initial state $\sigma \in I_1$, there is a initial state $\sigma' \in I_2$ such that $\sigma \approx \sigma'$, and vice versa. The two hybrid automata A and B are *bisimilar* if $[\![A]\!] \approx [\![B]\!]$.

We define the function $\alpha_x : \Sigma_{A^c} \rightarrow \Sigma_A$ such that $\alpha_x(v_c, \mathbf{s}) = (v, \mathbf{s}')$, where the location v_c is split from the location v for the critical value c, the states $\mathbf{s}$ and $\mathbf{s}'$ agree on all variables except x and t_x, and $\mathbf{s}'(x) = x(\mathbf{s}(t_x))$ if the function $x(t)$ is the solution of the initial-value problem "$\dot{x}(t) = dif(v, x)$; $x(0) = c$".

Lemma 1. *Let A be a solvable hybrid automaton, and let A^c be the clock translation of A that results from replacing the nonlinear variable x by the clock t_x. Then for all states $\sigma, \sigma' \in \Sigma_{A^c}$, $\sigma \xrightarrow{m} \sigma'$ in $[\![A^c]\!]$ iff $\alpha_x(\sigma) \xrightarrow{m} \alpha_x(\sigma')$ in $[\![A]\!]$.*

It follows that the relation $\{(\sigma, \alpha_x(\sigma)) \mid \sigma \in \Sigma_{A^c}\}$ is a bisimulation between $[\![A]\!]$ and $[\![A^c]\!]$, and that $[\![A]\!] \approx [\![A^c]\!]$. Since bisimilarity is transitive, if A^c results from A by replacing several nonlinear variables with clocks, A and A^c are still bisimilar.

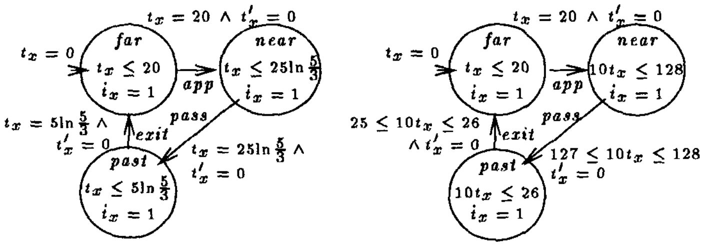
Theorem 2. *If A is a solvable hybrid automaton and A^c is a clock translation of A, then A and A^c are bisimilar.*

It follows that the clock translation is sound and complete for all branching-time properties. In particular, for safety properties, we have the following corollary.

Corollary 3. *Let A be a solvable hybrid automaton, and let A^c be a clock translation of A. If A has an accepting trajectory iff A^c has an accepting trajectory.*

We conclude that for solving the emptiness problem for the nonlinear automaton A, it suffices to solve the emptiness problem for the linear automaton A^c. The emptiness problem for A^c, however, can be solved exactly only if the clock translation A^c is rational. This gives us the following decidability result, which covers a class of nonlinear hybrid automata, while all previously published decidability results refer to linear hybrid automata [8].

Corollary 4. *The emptiness problem is decidable for rationally solvable hybrid automata.*



$t_x = 20 \wedge t'_x = 0$
$t_x = 0$
far
$t_x \leq 20$
$\dot{t}_x = 1$
app
near
$t_x \leq 25\ln\frac{5}{3}$
$\dot{t}_x = 1$
$t_x = 5\ln\frac{5}{3} \wedge t'_x = 0$
exit
pass
$t_x = 25\ln\frac{5}{3} \wedge t'_x = 0$
past
$t_x \leq 5\ln\frac{5}{3}$
$\dot{t}_x = 1$
$t_x = 20 \wedge t'_x = 0$
$t_x = 0$
far
$t_x \leq 20$
$\dot{t}_x = 1$
app
near
$10t_x \leq 128$
$\dot{t}_x = 1$
$25 \leq 10t_x \leq 26 \wedge t'_x = 0$
exit
pass
$127 \leq 10t_x \leq 128$
$t'_x = 0$
past
$10t_x \leq 26$
$\dot{t}_x = 1$

Fig. 4. Clock translation and 0.1-approximate clock translation of the train automaton

δ-approximate clock translation If the clock translation A^c is not rational, we approximate A^c by a rational automaton, and show soundness for emptiness checking. To preserve soundness when approximating irrational roots numerically, we *over*-approximate all root intervals. For example, the action $t_x = ln2$ can be overapproximated by the rational data predicate $693 \leq 1000t_x \leq 694$ with an error bounded by $\delta = 1/1000$. Formally, a *δ-approximation* of the data predicate $t_x \leq c$ ($t_x \geq c$), for $\delta \in \mathbb{R}$, is of the form $t_x \leq c'$ ($t_x \geq c'$) such that $c \leq c' \leq c + \delta$ ($c - \delta \leq c' \leq c$) and c' is rational. A *δ-approximate clock translation* $[A^c]_\delta$ of A is a rational linear hybrid automaton that is obtained from the clock translation A^c by replacing all atomic data predicates in initial and accepting conditions, invariants, and actions by δ-approximations. Note that an action predicate that involves the primed variable t'_x for a new clock t_x are of the form $t'_x = 0$ or $t'_x = t_x$.

We show that $[A^c]_\delta$ simulates A. To see this, recall the definition of (timed) simulation. Let $T_1 = \langle \Sigma_1, I_1, \mathcal{L}, \rightarrow_1 \rangle$ and $T_2 = \langle \Sigma_2, I_2, \mathcal{L}, \rightarrow_2 \rangle$ be two labeled transition systems. The binary relation $\succeq \subseteq \Sigma_1 \times \Sigma_2$ is a *simulation* of T_2 by T_1 if for all states $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$, $\sigma_1 \succeq \sigma_2$ implies for every letter $m \in \mathcal{L}$ that if $\sigma_2 \xrightarrow{m} \sigma'_2$, then there exists a state σ'_1 such that $\sigma_1 \xrightarrow{m} \sigma'_1$ and $\sigma'_1 \succeq \sigma'_2$. The state $\sigma \in \Sigma_1$ *simulates* the state $\sigma' \in \Sigma_2$ if there exists a simulation $\succeq$ of T_2 by T_1 such that $\sigma \succeq \sigma'$. The labeled transition system T_1 *simulates* the labeled transition system T_2, denoted $T_1 \succeq T_2$, if each initial state of T_2 is simulated by an initial state of T_1. The hybrid automaton A *simulates* the hybrid automaton B, written $A \succeq B$, if $[\![A]\!] \succeq [\![B]\!]$. It is clear that $[A^c]_\delta$ simulates A^c. Moreover, since A and A^c are bisimilar, we know that $[A^c]_\delta$ simulates A.

Proposition 5. *Let A be a solvable hybrid automaton. For all $\delta \in \mathbb{R}_{\geq 0}$, if $[A^c]_\delta$ is a δ-approximate clock translation of A, then $[A^c]_\delta$ simulates A.*

It follows that approximate clock translation is sound for all linear-time properties. Again, for safety properties, we have the following corollary.

Corollary 6. *Let A be a solvable hybrid automaton, and let $[A^c]_\delta$ be a δ-approximate clock translation of A. If A has an accepting trajectory, then so does $[A^c]_\delta$.*

Example: Railroad gate controller The performance data in this paper was measured on a Sun 670MP workstation. The clock translation B_1^c of the train automaton B_1 from Figure 2 is shown on the left in Figure 4, next to a 0.1-approximate clock translation $[B_1]_{0.1}$ on the right. By taking the product of $[B_1]_{0.1}$ with the gate and controller automata from Figure 2, the HyTech verifier automatically checks (in 25 seconds of CPU time) that whenever the train is within 100 meters from the gate, then the gate is closed. On the other hand, a 1.0-approximate clock translation of the train automaton is not sufficient for proving this safety property. (After clock translation, the safety property of the thermostat automaton from Figure 1 is checked by HyTech in 7 seconds of CPU time.)

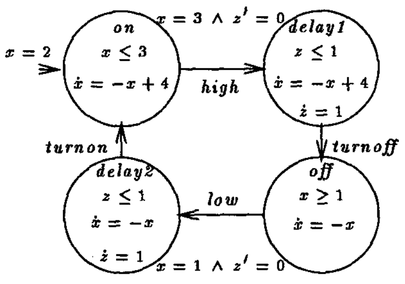


Fig. 5. A temperature controller with delays

Error analysis The approximate clock translation $[A^c]_\delta$ may have an accepting trajectory even if A does not. We now show that there is a hybrid automaton that is very close to A and has an accepting trajectory. The following error analysis relaxes all atomic data and action predicates of A to provide an upper bound on the error of $[A^c]_\delta$. If the hybrid automaton A models a hybrid system with sensors and actuators, then the *ε-relaxed automaton* A^ε, for $\varepsilon \in \mathbb{R}_{\geq 0}$, models the same system with sensors and actuators that suffer from errors bounded by ε: (1) all atomic data predicates of the form $x \leq c$ and $x \geq c$ (in initial and accepting conditions, invariants, and actions) are replaced by $x \leq c+\varepsilon$ and $x \geq c-\varepsilon$, respectively; and (2) all atomic action predicates of the form $x' = c$ are replaced by $c - \varepsilon \leq x' \leq c + \varepsilon$. Notice that if $\varepsilon \to 0$, then $A^\varepsilon \to A$. We define the metric $d_\succeq$ on hybrid automata such that $d_\succeq(A, B)$ is the infimum of all nonnegative reals ε such that $A^\varepsilon \succeq B \succeq A$ or $B^\varepsilon \succeq A \succeq B$, if such an ε exists; otherwise, $d_\succeq(A, B) = \infty$. The *error* of the δ-approximate clock translation $[A^c]_\delta$ is $d_\succeq(A, [A^c]_\delta)$. The following lemma bounds this error.

Lemma 7. *Let A be a solvable hybrid automaton, and let $[A^c]_\delta$ be a δ-approximate clock translation of A. If $\lambda \in \mathbb{R}_{\geq 0}$ bounds the absolute values of the derivatives of all data variables of A in all locations of A, then $A^{\delta \cdot \lambda}$ simulates $[A^c]_\delta$; that is, the error of the δ-approximate clock translation $[A^c]_\delta$ is bounded by $\delta \cdot \lambda$.*

It follows that the approximate clock translation is asymptotically complete, under the metric $d_\succeq$, for checking the emptiness of hybrid automata.

Theorem 8. *Let A be a solvable hybrid automaton. For all reals $\varepsilon > 0$, there is a real $\delta > 0$ such that for all δ-approximate clock translations $[A^c]_\delta$ of A, $A^\varepsilon \succeq [A^c]_\delta \succeq A$.*

Corollary 9. *Let A be a solvable hybrid automaton. For all reals $\varepsilon > 0$, there is a real $\delta > 0$ such that if a δ-approximate clock translation of A has an accepting trajectory, then so does A^ε.*

4 Rate Translation

The rate translation of a hybrid automaton replaces each nonlinear data variable x by a piecewise-linear variable that approximates x. The rate translation may be applicable also to unsolvable automata. Consider, for example, the nonlinear hybrid automaton of Figure 5, which models a temperature controller with delays: after the thermometer detects that the temperature is low or high, there may be a delay of up to 1 time unit before the heater is turned on or off. We wish to verify that the plant temperature is always between $\frac{1}{5}$ and $\frac{19}{5}$. The automaton is not solvable, because the edge from *delay*$_1$ to *off* is indefinite for x. Hence we cannot apply the clock translation to eliminate the nonlinear variable x. Instead, we approximate x by a piecewise-linear variable. In location v with the bounded invariant region $[\![inv(v)]\!]$, we bound the derivative of x by its minimum a and its maximum b, and then replace the activity $dif(v,x)$ by the rate interval $[a,b]$. For a better approximation, we split the location v into several locations and limit the size of the rate intervals. Clearly, smaller rate intervals yield a more accurate overapproximation of the automaton trajectories.

Bounded automata Let A be a hybrid automaton. The data variable x of A is *nondecreasing* (*nonincreasing*) in location v of A if $inv(v)$ implies $dif(x,v) \geq 0$ ($dif(x,v) \leq 0$). The data variable x of A is *bounded* with the *window* $[c,d] \subseteq \mathbb{R}$ if any one of the following three conditions holds:

1. For all states in $R(A)$, the value of x is always within the bounded interval $[c,d]$. In particular, this is the case if for all locations v of A, $inv(v)$ implies $c \leq x \leq d$.
2. A does not contain constants smaller than c or larger than d, and either x is nondecreasing in all locations of A, or x is nonincreasing in all locations.
3. $\neg F_A$ implies $c \leq x \leq d$.

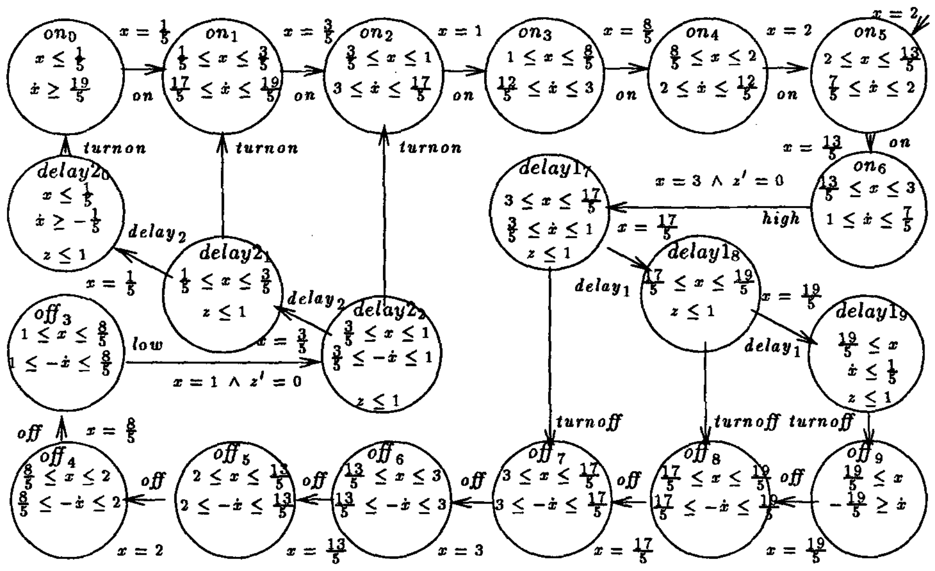
If the data variable x is bounded with the window $[c,d]$, and its value lies outside $[c,d]$, then the exact value of x is irrelevant for checking the emptiness of A. The hybrid automaton A is *bounded* if all nonlinear variables of A are simple and bounded.

The rate translation algorithm Let A be a bounded hybrid automaton, and let $\delta \in \mathbb{R}_{\geq 0}$ be a nonnegative real. A *δ-approximate rate translation* $[A^r]_\delta$ of A is a linear hybrid automaton that is obtained by the following construction. Consider a data variable x and a location v of A. First assume that the activity $dif(v,x) = f(x)$ is a function of x only. Let $[c,d]$ be the window of x. We partition the window $[c,d]$ into k subintervals $I_1 = [c_0 = c, c_1], \ldots, I_k = [c_{k-1}, c_k = d]$, each of size at most δ. The location v is split into $k+2$ locations $v_0, \ldots, v_{k+1}$. Each new location v_i has the invariant $inv(v) \wedge c_{i-1} \leq x \leq c_i$, where $c_{-1} = -\infty$ and $c_{k+1} = \infty$. For each v_i, we compute the minimum a and the maximum b of the function $dif(v_i,x)$ for $c_{i-1} \leq x \leq c_i$. We then approximate the derivative of x in the location v_i by the rate interval $dif(v_i,x) = [a,b]$. Finally, we introduce all edges of the form (v_i, v_{i+1}) and (v_{i+1}, v_i) with the action $x = c_i$ and the label v (which is the label of the stutter edge e_v); and for each edge $e = (v,v')$, all edges of the form (v_i, v'_j) with the action $act(e)$ and the label $label(e)$.

Now consider the general case that $dif(v,x_i) = f(x_1,\ldots,x_n)$. We approximate all nonlinear variables $x_1,\ldots,x_n$ simultaneously. Suppose that the window for x_i is I_i. We partition I_i into k_i subintervals $[c_{i,0}, c_{i,1}], \ldots, [c_{i,k_i-1}, c_{i,k_i}]$, each of size at most δ. The location v is split into the set $U_v = \{v(a_1,\ldots,a_n) \mid 0 \leq a_i \leq k_i+1\}$ of $(k_1+2)\cdots(k_n+2)$ locations, all with the initial condition $init(v)$ and the accepting condition $final(v)$. The invariant of $v(a_1,\ldots,a_n)$ is $inv(v) \wedge \bigwedge_{i=1,\ldots,n} c_{i,a_i-1} \leq x_i \leq c_{i,a_i}$, where $c_{i,-1} = -\infty$ and $c_{i,k_i+1} = \infty$. For each new location, we compute the rate intervals for all x_i. For each pair $v(\mathbf{a})$ and $v(\mathbf{b})$ of new locations, we introduce all edges of the form $(v(\mathbf{a}), v(\mathbf{b}))$ with the axtion $\mathbf{x}' = \mathbf{x}$ and the label v (many of these edges are inconsistent and can be omitted); and for each edge $e = (v,v')$, we introduce all edges of the form $(v(\mathbf{a}), v'(\mathbf{b}))$ with the action $act(e)$ and the label $label(e)$.

Soundness We show that the rate translation is sound for checking the emptiness of bounded automata. We define the onto function $\beta : \Sigma_{[A^r]_\delta} \rightarrow \Sigma_A$ such that $\beta(v',\mathbf{s}) = (v,\mathbf{s})$ if $v' \in U_v$. It is clear that for all states $\sigma_1, \sigma_2 \in \Sigma_{[A^r]_\delta}$, if $\beta(\sigma_1) \xrightarrow{m} \beta(\sigma_2)$ in $[\![A]\!]$, then $\sigma_1 \xrightarrow{m} \sigma_2$ in $[\![[A^r]_\delta]\!]$. Hence the relation $\{(\sigma, \beta(\sigma)) \mid \sigma \in \Sigma_{[A^r]_\delta}\}$ is a simulation of $[\![A]\!]$ by $[\![[A^r]_\delta]\!]$.

Proposition 10. *Let A be a bounded hybrid automaton. For all $\delta \in \mathbb{R}_{\geq 0}$, if $[A^r]_\delta$ is a δ-approximate rate translation of A, then $[A^r]_\delta$ simulates A.*



on_0
$x \le \frac{1}{5}$
$\dot{x} \ge \frac{19}{5}$
$x = \frac{1}{5}$
on
on_1
$\frac{1}{5} \le x \le \frac{3}{5}$
$\frac{17}{5} \le \dot{x} \le \frac{19}{5}$
$x = \frac{3}{5}$
on
on_2
$\frac{3}{5} \le x \le 1$
$3 \le \dot{x} \le \frac{17}{5}$
$x = 1$
on
on_3
$1 \le x \le \frac{8}{5}$
$\frac{12}{5} \le \dot{x} \le 3$
$x = \frac{8}{5}$
on
on_4
$\frac{8}{5} \le x \le 2$
$2 \le \dot{x} \le \frac{12}{5}$
$x = 2$
on
on_5
$2 \le x \le \frac{13}{5}$
$\frac{7}{5} \le \dot{x} \le 2$
$x = 2$
$x = \frac{13}{5}$
on
on_6
$\frac{13}{5} \le x \le 3$
$1 \le \dot{x} \le \frac{7}{5}$
$x = 3 \wedge z' = 0$
high
turnon
turnon
turnon
$delay2_0$
$x \le \frac{1}{5}$
$\dot{x} \ge -\frac{1}{5}$
$z \le 1$
$delay_2$
$x = \frac{1}{5}$
$delay2_1$
$\frac{1}{5} \le x \le \frac{3}{5}$
$z \le 1$
$delay_2$
$delay2_2$
$\frac{3}{5} \le x \le 1$
$\frac{3}{5} \le -\dot{x} \le 1$
$z \le 1$
$x = \frac{3}{5}$
off_3
$1 \le x \le \frac{8}{5}$
$1 \le -\dot{x} \le \frac{8}{5}$
low
$x = 1 \wedge z' = 0$
$delay1_7$
$3 \le x \le \frac{17}{5}$
$\frac{3}{5} \le \dot{x} \le 1$
$z \le 1$
$x = \frac{17}{5}$
$delay_1$
$delay1_8$
$\frac{17}{5} \le x \le \frac{19}{5}$
$z \le 1$
$x = \frac{19}{5}$
$delay_1$
$delay1_9$
$\frac{19}{5} \le x$
$\dot{x} \le \frac{1}{5}$
$z \le 1$
turnoff
turnoff
turnoff
off
$x = \frac{8}{5}$
off_4
$\frac{8}{5} \le x \le 2$
$\frac{8}{5} \le -\dot{x} \le 2$
$x = 2$
off
off_5
$2 \le x \le \frac{13}{5}$
$2 \le -\dot{x} \le \frac{13}{5}$
$x = \frac{13}{5}$
off
off_6
$\frac{13}{5} \le x \le 3$
$\frac{13}{5} \le -\dot{x} \le 3$
$x = 3$
off
off_7
$3 \le x \le \frac{17}{5}$
$3 \le -\dot{x} \le \frac{17}{5}$
$x = \frac{17}{5}$
off
off_8
$\frac{17}{5} \le x \le \frac{19}{5}$
$\frac{17}{5} \le -\dot{x} \le \frac{19}{5}$
$x = \frac{19}{5}$
off
off_9
$\frac{19}{5} \le x$
$-\frac{19}{5} \ge \dot{x}$

Fig. 6. Rate translation of the temperature controller with delays

It follows that the rate translation is sound for all linear-time properties. In particular, for safety properties, we have the following corollary.

Corollary 11. *Let A be a bounded hybrid automaton, and let $[A^r]_\delta$ be a rate translation of A. If A has an accepting trajectory, then so does $[A^r]_\delta$.*

Example: Temperature controller with delays Recall the emptiness problem for the automaton B_2 of Figure 5 with the accepting condition $\bigcup_v \{(v, x \le \frac{1}{5} \vee x \ge \frac{19}{5})\}$. For the rate translation of the automaton B_2, we partition the window $[\frac{1}{5}, \frac{19}{5}]$ into the eight intervals $[\frac{1}{5}, \frac{3}{5}]$, $[\frac{3}{5}, 1]$, $[1, \frac{8}{5}]$, $[\frac{8}{5}, 2]$, $[2, \frac{13}{5}]$, $[\frac{13}{5}, 3]$, $[3, \frac{17}{5}]$, and $[\frac{17}{5}, \frac{19}{5}]$ of uneven size at most 0.6 (it is a good idea to separate intervals at the point c if $x = c$ is a conjunct of an invariant or action). After removing inconsistent edges and unreachable locations, we obtain the linear hybrid automaton $[B_2^r]_{0.6}$ of Figure 6, which is a 0.6-approximate rate translation of B_2. The HyTech verifier reports that $[B_2^r]_{0.6}$ satisfies the safety property that the value of x stays within the interval $(\frac{1}{5}, \frac{19}{5})$ (using 45 seconds of CPU time).

Error analysis To analyze the error of the δ-approximate rate translation $[A^r]_\delta$, we define the metric d_R such that $d_R(A, B)$ is the infimum of all nonnegative reals ε such that $R(A) \subseteq R(B) \subseteq R(A^\varepsilon)$ or $R(B) \subseteq R(A) \subseteq R(B^\varepsilon)$, if such an ε exists; otherwise, $d_R(A, B) = \infty$. The *error* of the δ-approximate rate translation $[A^r]_\delta$ is $d_R(A, [A^r]_\delta)$, where $R([A^r]_\delta)$ is interpreted as $\beta(R([A^r]_\delta))$ when compared with $R(A)$. In the full paper, we give an error analysis for the rate translation of *monotonic* bounded hybrid automata, where in each location each nonlinear variable is nondecreasing or nonincreasing. The rate translation is asymptotically complete, under the metric d_R, for checking the emptiness of monotonic bounded hybrid automata.

Theorem 12. *Let A be a monotonic bounded hybrid automaton. For all reals $\varepsilon > 0$, there is a real $\delta > 0$ such that for all δ-approximate rate translations $[A^r]_\delta$ of A, $R(A) \subseteq R([A^r]_\delta) \subseteq R(A^\varepsilon)$.*

Corollary 13. *Let A be a monotonic bounded hybrid automaton. For all reals $\varepsilon > 0$, there is a real $\delta > 0$ such that if a δ-approximate rate translation of A has an accepting trajectory, then so does A^ε.*

5 Discussion

The two translations presented here provide algorithmic methods for verifying safety properties of two classes of nonlinear hybrid systems: solvable systems and bounded systems. Whenever both translations are applicable, the clock translation is generally preferable, because the size of the translated automaton does not depend on the precision of the translation. This work can be extended in two ways, which are discussed in the full version of the paper. First, symbolic model-checking techniques can be used to verify, in addition to safety properties, arbitrary branching-time properties of solvable and bounded systems. Second, the classes of solvable and bounded systems can be generalized by admitting nonlinear variables that are not simple, and the class of solvable systems can be generalized by admitting linear variables whose dynamics is given by rate intervals [4, 11, 12].

Acknowledgement We thank Howard Wong-Toi for a careful reading.

References

1. R. Alur, C. Coucoubetis, 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.
2. R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. *Hybrid Systems*, Lecture Notes in Computer Science 736, pp. 209–229. Springer, 1993.
3. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
4. R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Real-time Systems Symposium*, pp. 2–11, 1993.
5. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
6. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for the static analysis of programs by construction or approximation of fixpoints. *ACM Symposium on Principles of Programming Languages*, 1977.
7. R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems.* Lecture Notes in Computer Science 736. Springer, 1993.
8. T.A. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *ACM Symposium on Theory of Computing*, 1995.
9. 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.
10. X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. *Hybrid Systems*, Lecture Notes in Computer Science 736, pp. 149–178. Springer, 1993.
11. A. Olivero, J. Sifakis, and S. Yovine. Using abstractions for the verification of linear hybrid systems. *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pp. 81–94. Springer, 1994.
12. A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pp. 95–104. Springer, 1994.