HyTech: A Model Checker for Hybrid Systems* †

Thomas A. Henzinger
EECS Department
Univ. of California, Berkeley
tah@eecs.berkeley.edu

Pei-Hsin Ho
Strategic CAD Labs
Intel Corp., Hillsboro, Oregon
pho@ichips.intel.com

Howard Wong-Toi
Cadence Berkeley Labs
Berkeley, California
howard@cadence.com

Abstract

A *hybrid system* is a dynamical system whose behavior exhibits both discrete and continuous change. A *hybrid automaton* is a mathematical model for hybrid systems, which combines, in a single formalism, automaton transitions for capturing discrete change with differential equations for capturing continuous change. HyTech is a symbolic model checker for *linear hybrid automata*, a subclass of hybrid automata that can be analyzed automatically by computing with polyhedral state sets. A key feature of HyTech is its ability to perform parametric analysis, *i.e.* to determine the values of design parameters for which a linear hybrid automaton satisfies a temporal-logic requirement.

1 Introduction

A hybrid system typically consists of a collection of digital programs that interact with each other and with an analog environment. Examples of hybrid systems include manufacturing controllers, automotive and flight controllers, medical equipment, micro-electromechanical systems, and robots. When these systems occur in mission-critical applications, formal guarantees about the absence of logical and timing errors are desirable. The formal analysis of the mixed digital-analog nature of hybrid systems requires a mathematical model that incorporates the discrete behavior of computer programs with the continuous behavior of environment variables, such as time, position, and temperature. The first extensions of discrete state-transition models toward mixed discrete-continuous behavior concentrated on the single most important environment parameter—real time. One such model is the *timed automaton*—a finite automaton augmented with a finite number of clocks, which are real-valued variables whose values change continuously with the constant rate 1 [4]. Timed automata have been used successfully to analyze real-time protocols and asynchronous circuits. For modeling more general kinds of hybrid systems, we use the *hybrid automaton*—a finite automaton with a finite number of real-valued variables that change continuously, as specified by differential equations and differential inequalities, in more general ways than clocks [3, 36, 2].

For analyzing hybrid systems, we build on the model-checking technology, in which a formal model of the system is checked, fully automatically, for correctness with respect to a requirement

*A preliminary version of this paper appeared in the *Proceedings of the Ninth International Conference on Computer-Aided Verification* (CAV 97), *Lecture Notes in Computer Science* **1254**, Springer-Verlag, 1997, pp. 460–463, and an extended version appeared in *Software Tools for Technology Transfer* **1** (special issue on Timed and Hybrid Systems), 1997, pp. 110–122.

†This research was supported in part by the ONR YIP award N00014-95-1-0520, by the NSF CAREER award CCR-9501708, by the NSF grant CCR-9504469, by the AFOSR contract F49620-93-1-0056, by the ARO MURI grant DAAH-04-96-1-0341, by the ARPA grant NAG2-892, and by the SRC contract 95-DC-324.036.

expressed in temporal logic [11, 37]. For this purpose, the entire state space of the system is explored. This can be done enumeratively, by considering each state individually, or symbolically, by computing with constraints that represent state sets. Because of its ability to deal with very large state spaces, symbolic model checking has been proven an effective technique for the automatic analysis of complex finite state-transition systems [10]. In recent years, the model-checking approach has been extended to several classes of infinite state-transition systems, including timed automata [1]. Since clock values range over the infinite domain of the nonnegative reals, it is impossible to enumerate all states of a timed automaton, and symbolic representations of state sets must be employed. Specifically, the symbolic model checking of a timed automaton requires the manipulation of certain linear constraints on clock values, namely, disjunctions of inequalities of the form $\mathbf{x} \sim \mathbf{b}$ and $\mathbf{x} - \mathbf{y} \sim \mathbf{b}$, for clock vectors $\mathbf{x}$ and $\mathbf{y}$, an inequality operator $\sim \in \{\leq, \geq\}$, and a constant integer vector $\mathbf{b}$, whose components are bounded for any given automaton [29]. Since there are only finitely many of these constraints, all computations required for model checking are guaranteed to terminate.

By admitting more general linear constraints on continuous variables, namely, disjunctions of inequalities of the form $\mathbf{Ax} \sim \mathbf{c}$, where $\mathbf{A}$ is a constant matrix and $\mathbf{c}$ is a constant vector, the symbolic model-checking method for timed automata can be extended to a more general class of hybrid automata called *linear hybrid automata* [5]. In a linear hybrid automaton, the dynamics of the continuous variables are defined by linear differential inequalities of the form $\mathbf{A\dot{x}} \sim \mathbf{b}$, where $\mathbf{\dot{x}}$ is the vector of first derivatives of the variables $\mathbf{x}$.[1] Since the number of possible constraints is no longer finite, when moving from timed automata to linear hybrid automata, the price to pay for the increased generality is the loss of guaranteed termination for model checking. The method is still of practical interest, however, because termination happens naturally in many examples and can be enforced in others, say, by considering the behavior of a system over a bounded interval of time.

Model checking can be used to provide more than a mere "yes" or "no" answer to the question of whether a system satisfies a correctness requirement. HyTech provides also diagnostic information that aids in design and debugging. If a system description contains design parameters, whose values are not specified, then HyTech computes necessary and sufficient constraints on the parameter values that guarantee correctness. For example, for a railroad crossing, we will compute the exact cutoff point, in meters from the crossing, at which a train has to signal its approach in order for the gate to be closed by the time the train passes through the crossing. If a system fails to satisfy a correctness requirement, then HyTech generates an error trajectory, which illustrates a time-stamped sequence of events that leads to a violation of the requirement.

While linear hybrid automata are expressive compared to other formalisms for which model checking is possible, such as finite automata and timed automata, many embedded applications do not meet the linearity constraints. In such cases, we conservatively approximate the system using linear hybrid automata so that if the approximate automaton satisfies a correctness requirement, then the original system satisfies the requirement as well. If, on the other hand, the approximate system violates the requirement, and the generated error trajectory is not a possible trajectory of the original system, then the approximation must be refined.

This paper consists of three sections. Section 2 presents the model of hybrid automata, Section 3 illustrates the analysis techniques, and Section 4 briefly describes the tool HyTech. A simple thermostat is used as a running example to demonstrate modeling, approximation, safety

[1] It is important to realize that the definition of linearity for hybrid automata differs from the definition of linearity commonly used in systems theory. In particular, the differential inequalities of linear hybrid automata may not depend on the value of the variables, and thus dynamics of the form $\dot{x} = x$ are prohibited.

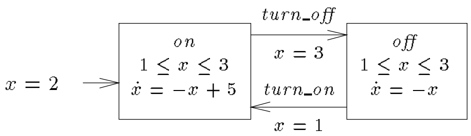

Figure 1: Thermostat automaton

verification, parametric analysis, and the use of the tool. For more involved model-checking procedures of general temporal-logic requirements, as well as for theoretical results on timed automata, the interested reader is encouraged to consult other literature [19].

2 Hybrid Automata

2.1 Example: a simple thermostat

A *euclidean dynamical system* prescribes how a set of *real*-valued variables evolve over time. A system state is a point in $\mathbb{R}^n$ (where n is the number of variables), and a system trajectory is a curve in $\mathbb{R}^n$ (called a *flow*). The deterministic evolution of real variables is naturally prescribed by an initial condition and differential equations; for example, the temperature $x \in \mathbb{R}$ of a heated plant may observe the initial condition $x = 2$ and the differential equation $\dot{x} = -x + 5$. The nondeterministic evolution of real-valued variables can be prescribed using differential inequalities, such as $\dot{x} \in [3, 4]$.

By contrast, a *boolean dynamical system* prescribes how a set of *boolean*-valued variables evolve over time. A system state is a point in $\mathbb{B}^m$ (where m is the number of variables), and a system trajectory is a sequence of states (each pair of consecutive states is called a *jump*). The nondeterministic evolution of boolean variables is naturally prescribed by an initial condition and a transition relation, which indicates for every state the set of possible successor states. For example, the status $y \in \{on, off\}$ of a heater may observe the initial condition $y = on$ and the transition relation $\{(on, off), (off, on)\}$, *i.e.* the heater may be turned on or off. A boolean dynamical system, thus, can be viewed as a finite automaton.

A *hybrid dynamical system* has both real-valued and boolean-valued variables, say n real variables and m boolean variables. A system state, then, is a point in $\mathbb{B}^m \times \mathbb{R}^n$, and a system trajectory is a sequence of flows and jumps: during flows, the boolean part of the state stays constant and the real part of the state evolves over time; at jumps, the entire state changes instantaneously. We describe hybrid dynamical systems using hybrid automata. A hybrid automaton annotates the control graph of a finite automaton with conditions on real-valued variables. Each node of the graph represents an operating mode of the system, and is annotated with differential inequalities that prescribe the possible evolutions (flows) of the real variables while the system remains in the given mode. Each edge of the graph represents a switch in operating mode, and is annotated with a condition that prescribes the possible changes (jumps) of the real variables when the system executes the given mode switch.

If we combine the temperature $x \in \mathbb{R}$ with the heater $y \in \{on, off\}$, we obtain a thermostat. The hybrid automaton of Figure 1 has two operating modes: either the heater is on (mode *on*), or the heater is off (mode *off*). Initially, the heater is on and the temperature x is 2 degrees. When the heater is on, the temperature rises at the rate of $-x+5$ degrees per minute; when the heater is off, the temperature falls at the rate of $-x$ degrees per minute. The heater can be turned off when the temperature reaches 3 degrees, and it can be turned on when the temperature falls to 1 degree. This is due to the edge conditions $x=3$ and $x=1$, which assert when a mode switch *may* occur. To force mode switches, such as forcing the heater to be turned off when the temperature reaches 3 degrees, we annotate the operating modes with so-called *invariant conditions* (in addition to the annotation with differential inequalities): the system can remain in a mode only as long as the corresponding invariant condition is satisfied. Thus, the invariant conditions $1 \leq x \leq 3$ of both operating modes prescribe that a mode switch *must* occur before the temperature leaves the operating interval of [1,3] degrees.

2.2 Formal definition

A *hybrid automaton* is a system $A = (X, V, \mathit{flow}, \mathit{inv}, \mathit{init}, E, \mathit{jump}, \Sigma, \mathit{syn})$ that consists of the following components [2]:

Variables A finite ordered set $X = \{x_1, x_2, \ldots, x_n\}$ of real-valued variables. For example, the thermostat automaton of Figure 1 uses the variable x to model the plant temperature (in this case, $n=1$).

Control modes A finite set V of control modes. For example, the thermostat automaton has two control modes, *on* and *off*.

Flow conditions A labeling function *flow* that assigns a flow condition to each control mode $v \in V$. The flow condition $\mathit{flow}(v)$ is a predicate over the variables in $X \cup \dot{X}$, where $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$. The dotted variable $\dot{x}_i$, for $1 \leq i \leq n$, refers to the first derivative of x_i with respect to time, *i.e.* $\dot{x}_i = dx_i/dt$. While the control of the hybrid automaton A is in mode v, the variables in X evolve along a differentiable curve such that at all points along the curve, the values of the variables and their first derivatives satisfy the flow condition $\mathit{flow}(v)$. For example, the control mode *on* of the thermostat automaton has the flow condition $\dot{x} = -x+5$; the control mode *off* has the flow condition $\dot{x} = -x$.

Invariant conditions A labeling function *inv* that assigns an invariant condition to each control mode $v \in V$. The invariant condition $\mathit{inv}(v)$ is a predicate over the variables in X. While the control of the hybrid automaton A is in mode v, the variables in X must satisfy the invariant condition $\mathit{inv}(v)$. For example, both control modes of the thermostat automaton have the invariant condition $1 \leq x \leq 3$.

Initial conditions A labeling function *init* that assigns an initial condition to each control mode $v \in V$. The initial condition $\mathit{init}(v)$ is a predicate over the variables in X. The control of the hybrid automaton A may start in the control mode v when the initial condition $\mathit{init}(v)$ is true. In the graphical representation of automata, initial conditions appear as labels on incoming arrows without source modes, and initial conditions of the form *false* are not depicted. For example, the control mode *on* of the thermostat automaton has the initial condition $x=2$; the control mode *off* has the initial condition *false*.

Control switches A finite multiset E of control switches. Each control switch (v, v') is a directed edge between a source mode $v \in V$ and a target mode $v' \in V$. For example, the thermostat automaton has two control switches, $(\mathit{on}, \mathit{off})$ and $(\mathit{off}, \mathit{on})$.

Jump conditions A labeling function *jump* that assigns a jump condition to each control switch $e \in E$. The jump condition *jump*(e) is a predicate over the variables in $X \cup X'$, where $X' = \{x'_1, \ldots, x'_n\}$. The unprimed symbol x_i, for $1 \le i \le n$, refers to the value of the variable x_i before the control switch, and the primed symbol x'_i refers to the value of x_i after the control switch. Thus, a jump condition relates the values of the variables before a control switch to the possible values after the control switch. In the graphical representation of automata, we use guarded assignments to represent jump conditions; for example, assuming $n = 2$, the guarded assignment $x_1 = x_2 \rightarrow x_1 := 2x_2$ stands for the jump condition $x_1 = x_2 \wedge x'_1 = 2x_2 \wedge x'_2 = x_2$ (notice that because the variable x_2 is not assigned a new value, its value after the jump is equal to its value before the jump). In the thermostat automaton, the control switch (on, off) has the jump condition $x = 3 \wedge x' = x$; the control switch (on, off) has the jump condition $x = 1 \wedge x' = x$.

Events A finite set Σ of events, and a labeling function *syn* that assigns an event in Σ to each control switch $e \in E$. For example, the control switch (on, off) of the thermostat automaton corresponds to the event *turn_off*; the control switch (off, on) corresponds to the event *turn_on*. Though not used in the thermostat example, events permit the synchronization of jumps between concurrent hybrid automata.

2.3 States and trajectories

A *state* of the hybrid automaton A is a pair $(v, \mathbf{a})$ consisting of a control mode $v \in V$ and a vector $\mathbf{a} = (a_1, \ldots, a_n)$ that represents a value $a_i \in \mathbb{R}$ for each variable $x_i \in X$. The state $(v, \mathbf{a})$ of A is *admissible* if the predicate $inv(v)$ is true when each variable x_i is replaced by the value a_i. The state $(v, \mathbf{a})$ is *initial* if the predicate $init(v)$ is true when each x_i is replaced by a_i. For example, the state $(on, 1.5)$ of the thermostat automaton is admissible; the state $(on, 0.5)$ is not. The thermostat automaton has exactly one initial state, $(on, 2)$.

Consider a pair (q, q') of two admissible states $q = (v, \mathbf{a})$ and $q' = (v', \mathbf{a}')$. The pair (q, q') is a *jump* of A if there is a control switch $e \in E$ with source mode v and target mode v' such that the predicate *jump*(e) is true when each variable x_i is replaced by the value a_i, and each primed variable x'_i is replaced by the value a'_i. The thermostat automaton has exactly two jumps, $((on, 3), (off, 3))$ and $((off, 1), (on, 1))$. The pair (q, q') is a *flow* of A if $v = v'$ and there is a nonnegative real $\delta \in \mathbb{R}_{\geq 0}$ (the duration of the flow) and a differentiable function $\rho : [0, \delta] \rightarrow \mathbb{R}^n$ (the curve of the flow) such that the following three requirements hold:

1. Endpoints: $\rho(0) = \mathbf{a}$ and $\rho(\delta) = \mathbf{a}'$.

2. Invariant condition: For all time instants $t \in (0, \delta)$, the state $(v, \rho(t))$ is admissible.

3. Flow condition: Let $\dot{\rho} : [0, \delta] \rightarrow \mathbb{R}^n$ be the first time derivative of ρ. For all time instants $t \in (0, \delta)$, the predicate *flow*(v) is true when each variable x_i is replaced by the i-th coordinate of the vector $\rho(t)$, and each dotted variable $\dot{x}_i$ is replaced by the i-th coordinate of $\dot{\rho}(t)$.

For example, $((off, 3), (off, 2))$ and $((off, 3), (off, 2.5))$ are flows of the thermostat automaton. If (q, q') is a jump, we say that q' is a jump successor of q; if (q, q') is a flow, then q' is called a flow successor of q (notice that every admissible state is a flow successor of itself, because there is always a flow of duration 0).

A *trajectory* of the hybrid automaton A is a finite sequence $q_0, q_1, \ldots, q_k$ of admissible states q_j such that (1) the first state q_0 of the sequence is an initial state of A, and (2) each pair $(q_j, q_j + 1)$

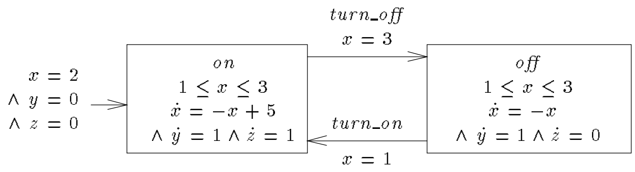

Figure 2: Thermostat automaton augmented for safety verification

of consecutive states in the sequence is either a jump of A or a flow of A. A state of A is *reachable* if it is the last state of some trajectory of A. While this is not usually the case, all admissible states of the thermostat automaton are reachable.

2.4 Safety requirements

A safety requirement asserts that nothing bad will happen during the evolution of a system. Safety requirements can often be specified by describing the "unsafe" values and value combinations of the system variables. Then, the system satisfies the safety requirement iff all reachable states are safe. Safety verification, therefore, amounts to computing the set of reachable states.

For hybrid automata, we specify safety requirements using state assertions. A *state assertion* φ for the hybrid automaton A is a function that assigns to each control mode $v \in V$ a predicate $\varphi(v)$ over the variables in X. We say that the state assertion φ is true (or false) for a state $(v, \mathbf{a})$ of A if the predicate $\varphi(v)$ is true (false) when each variable x_i is replaced by the value a_i. The states for which φ is true are called the φ-states. For example, the invariant conditions of A define a state assertion *inv*, and the *inv*-states are precisely the admissible states; similarly, the initial conditions define a state assertion *init* that is true precisely for the initial states (flow and jump conditions do not define state assertions). If *unsafe* is a state assertion for the hybrid automaton A, then *A satisfies the safety requirement* specified by *unsafe* if the state assertion *unsafe* is false for all reachable states of A.

Sometimes the given variables or control modes are not sufficient to specify a safety requirement, and the system description needs to be augmented with additional variables and control modes (or with so-called monitor automata, which are executed concurrently with the system and report when an unsafe state is entered; see below). For example, for the thermostat automaton, consider the requirement that the heater is active less than 2/3 of the first 60 minutes. To specify this requirement, we need a means of representing (1) the total elapsed time, say y, and (2) the total accumulated time that the heater has been active, say z. To this end, we add the two auxiliary variables y and z to the thermostat automaton, in a way that does not alter the behavior of the automaton: the variable y is a clock (*i.e.* $\dot{y} = 1$ for all control modes) that measures the elapsed time; the variable z is a stopwatch (*i.e.* $\dot{z} = 1$ or $\dot{z} = 0$) that measures the accumulated time spent in the control mode *on*. The augmented automaton is shown in Figure 2. Now we can specify the unsafe states using the state assertion *unsafe* that assigns the predicate $y = 60 \wedge z \geq 2y/3$ to both control modes. We will use HyTech to verify that the state assertion *unsafe* is false for all reachable states of the augmented thermostat (notice that it is no longer the case that all admissible states are reachable).

3 Analysis of Hybrid Automata

3.1 Computing the reachable states

To check if the hybrid automaton A satisfies the safety requirement specified by the state assertion *unsafe*, we attempt to compute another state assertion, *reach*, which is true exactly for the reachable states of A. Then we check if there is any state for which both *reach* and *unsafe* are true: if so, the safety requirement is violated, and we produce an error trajectory from an initial state to an unsafe state (this is useful for debugging the system); if not, the safety requirement is satisfied. We attempt to compute the state assertion *reach* as follows. For a state assertion φ, let $Post(\varphi)$ be a state assertion that is true precisely for the jump and flow successors of the φ-states, *i.e.* $Post(\varphi)$ is true for a state q' iff there exists a φ-state q such that (q, q') is either a jump or a flow of A. If we succeed in computing the state assertion $\varphi_1 = Post(init)$, then we have characterized all states that are reachable by trajectories of length 1 (*i.e.* by a single jump or flow); if we succeed in computing the state assertion $\varphi_2 = Post(\varphi_1)$, then we have characterized all states that are reachable by trajectories of length 2; etc. Finally, if for some natural number k, we find that φ_k and $\varphi_{k+1} = Post(\varphi_k)$ are equivalent (*i.e.* they are true for the same states), then we can conclude that φ_k characterizes all states that are reachable by trajectories of *any* length, and therefore $reach = \varphi_k$ (notice that, because every admissible state is a flow successor of itself, if a state is reachable by a trajectory of length k, then it is also reachable by a trajectory of length $k + 1$).

The success of this computation of *reach* hinges on two issues. First, for a state assertion φ, we need to be able to compute the state assertion $Post(\varphi)$. This can be done reasonably efficiently for a restricted class of hybrid automata called linear hybrid automata. Second, the iterative computation of *reach* must converge within a finite number of *Post* applications. This can be guaranteed for certain restricted classes of linear hybrid automata, such as the class of *timed automata*, all of whose variables are clocks. While we address the first issue below, the second issue is more of theoretical than practical interest: if a verification attempt does not succeed, by exhausting all available space or time resources, it is of little value to know that with unlimited resources the computation would have converged. The convergence issue, therefore, is not discussed further, and we refer the interested reader to the literature, where decidability results for several subclasses of linear hybrid automata can be found [18, 17, 28, 26, 34, 27].

3.2 Linear hybrid automata

The hybrid automaton model is very expressive. While convenient for providing formal descriptions of hybrid systems, the very generality of the model prohibits automatic analysis. We therefore consider a restricted class of hybrid automata, the linear hybrid automata, for which the function *Post* on state assertions can be computed efficiently.

An *atomic linear predicate* is an inequality between a rational constant and a linear combination of variables with rational coefficients, such as $3x_1 - x_2 + 7x_5 \leq 3/4$. A *convex linear predicate* is a finite conjunction of linear inequalities. A *linear predicate* is a finite disjunction of convex linear predicates. The hybrid automaton A is a *linear hybrid automaton* if it satisfies the following two requirements [5]:

1. Linearity: For every control mode $v \in V$, the flow condition *flow*(v), the invariant condition *inv*(v), and the initial condition *init*(v) are convex linear predicates. For every control switch $e \in E$, the jump condition *jump*(e) is a convex linear predicate.

2. Flow independence: For every control mode $v \in V$, the flow condition *flow*(v) is a predicate over the variables in $\dot{X}$ only (and does not contain any variables from X).

The second requirement ensures that the possible flows are independent from the values of the variables, and depend only on the control mode. While this requirement is quite limiting, and prohibits flow conditions such as $\dot{x} = x$, it does permit many kinds of variables that typically arise in real-time computing, such as clocks ($\dot{x} = 1$), stopwatches ($\dot{x} = 1$ or $\dot{x} = 0$), and clocks with bounded drift ($\dot{x} \in [1-\epsilon, 1+\epsilon]$ for some constant ϵ).

A state assertion φ for A is *linear* if for every control mode $v \in V$, the predicate $\varphi(v)$ is linear. For linear hybrid automata, we have the following theorem [5]: if A is a linear hybrid automaton, and φ is a linear state assertion for A, then *Post*(φ) can be computed and the result is again a linear state assertion for A. This is because for linear hybrid automata, every flow curve can be replaced by a straight line between the two endpoints. The theorem enables the automatic analysis—safety verification as well as more general temporal-logic model checking [5]—of linear hybrid automata.

3.3 From nonlinear to linear hybrid automata

The thermostat automaton of Figure 1 is not a linear hybrid automaton, because the requirement of flow independence is violated in both control modes. Since we have no direct means of automatically verifying nonlinear hybrid automata, we have developed two techniques for replacing a nonlinear hybrid automaton by a linear hybrid automaton [25]. The first technique, called clock translation, replaces variables that cause nonlinearity by clocks. The second technique, called linear phase-portrait approximation, replaces nonlinear predicates by more relaxed linear predicates.

Clock translation

The idea behind clock translation is that sometimes the value of a variable can be determined from a past value and the time that has elapsed since the variable had that value. For instance, the variable x_i of the hybrid automaton A is *clock-translatable* if the following two requirements hold:

1. Solvability: In each flow condition *flow*(v), all occurrences of x_i and $\dot{x}_i$ are within a conjunct of the form $\dot{x}_i = g_i^v(x_i)$, where $g_i^v : \mathbb{R} \rightarrow \mathbb{R}$ is an integrable function with constant sign over the invariant condition (*i.e.* for any two admissible states $(v, \mathbf{a})$ and $(v, \mathbf{b})$, either both $g_i^v(a_i)$ and $g_i^v(b_i)$ are positive, or both are negative). In each invariant, initial, and jump condition, all occurrences of x_i and x_i' are within conjuncts of the form $x_i' = x_i$ or $x_i \sim c$ or $x_i' \sim c$, for inequality operators $\sim$ and rational constants c.

2. Initialization: For every control mode v, the initial condition *init*(v) implies $x_i = c$ for some constant c. For every control switch (v, v'), either $g_i^v = g_i^{v'}$ and *jump*(v, v') implies $x_i' = x_i$, or *jump*(v, v') implies $x_i' = c$ for some constant c.

Under the conditions above, the value of x_i is determined by (a) the time since it was last reassigned to a constant, and (b) the value of that constant. Therefore, all invariant, initial, and jump conditions on the clock-translatable variable x_i can be translated to conditions on a clock t_i^x that is restarted whenever x_i is reassigned to a constant. If necessary, control modes may need to be duplicated to account for reassignments of x_i to different constants.

The variable x of the thermostat automaton is clock-translatable. Clock translation results in the hybrid automaton of Figure 3. The control mode *on* is split into two control modes, one for each of the two values that x may have when entering the mode *on*. The initial value of

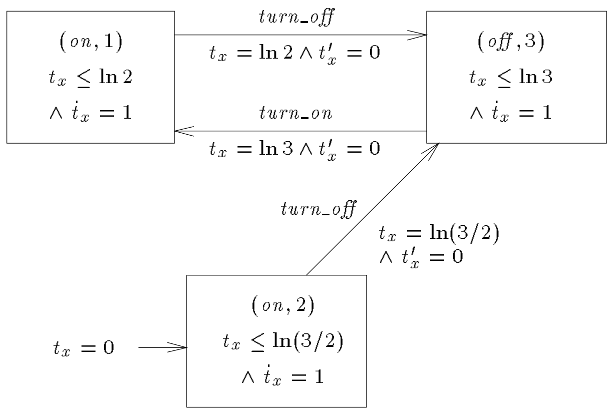

Figure 3: Clock translation of the thermostat automaton

x is 2, from which x follows the curve $x(t) = -3e^{-t} + 5$. It takes $\ln(3/2)$ minutes to reach the threshold temperature of 3 degrees. Thus the invariant condition $x \leq 3$ for mode *on* is translated to $t_x \leq \ln(3/2)$ at mode $(on, 2)$. The jump condition $x = 3 \wedge x' = x$ of the switch (on, off) is translated to the jump condition $t_x = \ln(3/2) \wedge t'_x = 0$ for the switch $((on, 2), (off, 3))$. The translated jump condition resets the clock t_x, since the differential equations prescribing the evolution of x differ at the source and target modes of the switch. Whenever the automaton control reenters the mode *on*, the variable x has the value 1 and follows the curve $x(t) = -4e^{-t} + 5$ for $\ln 2$ minutes before reaching the value 3.

Linear phase-portrait approximation

The idea behind linear phase-portrait approximation is to relax nonlinear flow, invariant, initial, and jump conditions using weaker linear conditions: each nonlinear predicate p is replaced by a linear predicate p' such that p implies p'. For example, the linear hybrid automaton of Figure 4 is a linear phase-portrait approximation of the thermostat automaton. Since the invariant, initial, and jump conditions of the thermostat are all linear, only the flow conditions need to be relaxed. For the control mode *on*, the invariant condition $1 \leq x \leq 3$ and the flow condition $\dot{x} = -x + 5$ imply that the first derivative of x is bounded from above by 4 and bounded from below by 2. Hence the flow condition $\dot{x} = -x + 5$ can be relaxed to the linear condition $\dot{x} \in [2, 4]$. Similarly, for the control mode *off*, the nonlinear flow condition can be relaxed to the linear condition $\dot{x} \in [-3, -1]$.

While clock translation preserves the trajectories of a system, linear phase-portrait approximation adds trajectories to a system. Hence, if we prove that the relaxed system satisfies a safety property, we can be sure that the original system also satisfies the property. However, if the relaxed system violates a safety property, then we must check if the discovered error trajectory is a valid trajectory of the original system. If not, then the analysis is inconclusive and the approximation needs to be refined, perhaps by splitting control modes in order to gain more accurate overapproximations of the possible flows. For example, the control mode *on* of the thermostat automaton can be split into two control modes, on_1 and on_2, each with the flow condition $\dot{x} = -x + 5$, mode on_1 with the invariant condition $1 \leq x \leq 2$, mode on_2 with the invariant condition $2 \leq x \leq 3$, and a

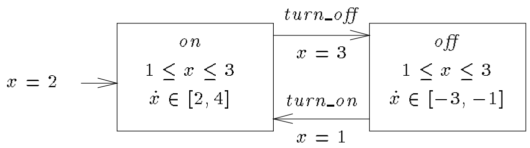

Figure 4: Linear phase-portrait approximation of the thermostat automaton

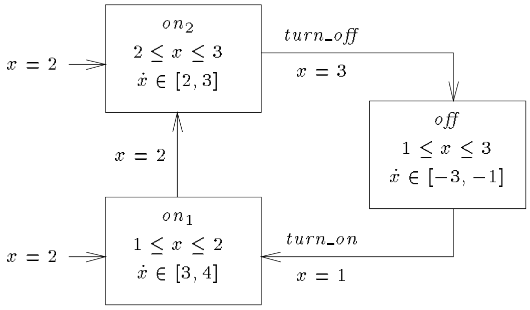

Figure 5: Tighter linear phase-portrait approximation of the thermostat automaton

control switch from on_1 to on_2 labeled $x = 2$. Relaxation, then, yields the linear flow conditions $\dot{x} \in [3,4]$ for on_1 and $\dot{x} \in [2,3]$ for on_2. The resulting linear hybrid automaton appears in Figure 5. It has strictly more trajectories than the original nonlinear automaton, but strictly fewer than the earlier approximation from Figure 4. Since tighter linear approximations yield more control modes, analysis becomes more expensive.

3.4 Safety verification of the thermostat

Recall that we wish to verify that the heater is active for less than 2/3 of the first hour of operation. By adding the auxiliary variables y and z, described above, to the linear phase-portrait approximation of the thermostat automaton, we obtain the linear hybrid automaton of Figure 6. To ensure convergence of the computation of the reachable states, the conjunct $y \leq 60$ has been added to the invariant conditions, so that trajectories are tracked only for the first 60 minutes. As above, the unsafe states are specified by the linear state assertion *unsafe* that assigns the predicate $y = 60 \wedge z \geq 2y/3$ to both control modes. Since the linear hybrid automaton of Figure 6 has strictly more trajectories than the original nonlinear thermostat from Figure 2, it suffices to prove the safety requirement for the linear version.

We write $\{(on, p_1), (\mathit{off}, p_2)\}$ for the state assertion that assigns the predicate p_1 to the control mode *on*, and assigns p_2 to *off*. The computation of the reachable states starts from the state assertion

$$\varphi_0 \;=\; \mathit{init} \;=\; \{(on, x = 2 \wedge y = 0 \wedge z = 0),\ (\mathit{off}, \mathit{false})\},$$

i.e. all initial states have the control mode *on*, and x is initially 2 degrees. We compute the state

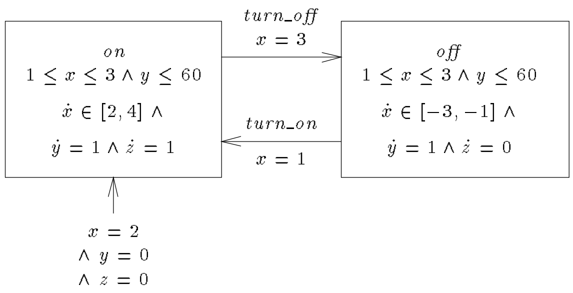

Figure 6: Linear thermostat automaton for safety verification

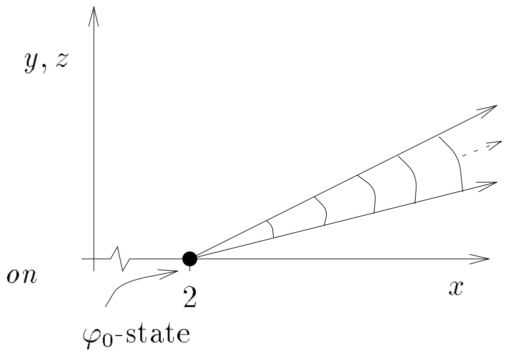

Figure 7: Flow successors of the φ_0-state, if $inv(on)$ were $true$

assertion $\varphi_1 = Post(\varphi_0)$ in two steps. First, we find all jump successors of φ_0-states: there are none, because the control switch from *on* to *off* requires that x is 3 degrees. Second, we find all flow successors of φ_0-states. For this purpose, observe that for a state assertion φ, the predicate

$$\exists x_1, \ldots, x_n.\ \exists \delta \geq 0.\ \exists \dot{x}_1, \ldots, \dot{x}_n.\ \varphi(v) \wedge \mathit{flow}(v) \wedge\ x'_1 = x_1 + \delta \dot{x}_1 \wedge \ldots \wedge x'_n = x_n + \delta \dot{x}_n$$

is true for the values $a_1, \ldots, a_n$ of the variables $x'_1, \ldots, x'_n$ iff the state $(v, \mathbf{a})$ is a flow successor of a state for which φ is true, assuming the unconstraining invariant condition $inv(v) = true$. From linear predicates, the existential quantifiers can be eliminated effectively. In particular, for $\varphi = \varphi_0$ and $v = on$, we obtain the predicate

$$\begin{aligned} & \exists x, y, z.\ \exists \delta \geq 0.\ \exists \dot{x}, \dot{y}, \dot{z}.\ x = 2 \wedge y = 0 \wedge z = 0 \\ & \qquad \wedge\ \dot{x} \in [2, 4] \wedge \dot{y} = 1 \wedge \dot{z} = 1 \wedge x' = x + \delta \dot{x} \wedge\ y' = y + \delta \dot{y} \wedge z' = z + \delta \dot{z} \\ = \ & (\exists \delta \geq 0.\ 2 + 2\delta \leq x' \leq 2 + 4\delta \wedge y' = \delta \wedge z' = \delta) \\ = \ & (2z' + 2 \leq x' \leq 4z' + 2 \wedge y' = z'), \end{aligned}$$

which corresponds to the unbounded cone in Figure 7. After renaming primed symbols to unprimed symbols and intersecting with the invariant condition of the control mode *on*, we obtain

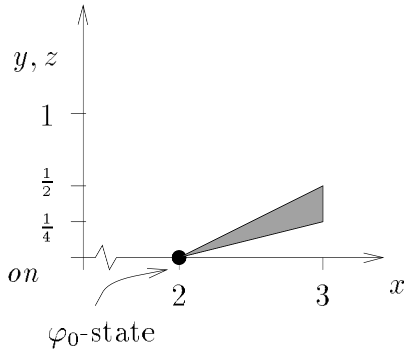

Figure 8: Flow successors of the φ_0-state

the predicate

$$x \leq 3 \wedge 2z + 2 \leq x \leq 4z + 2 \wedge y = z,$$

which corresponds to the shaded region in Figure 8. This predicate characterizes the states with the control mode *on* that can be reached from an initial state by a single flow. Since there are no states with the control mode *off* that can be reached from an initial state by a single flow, we conclude

$$\begin{aligned}\varphi_1 &= \mathit{Post}(\varphi_0)\\ &= \{(on, x \leq 3 \wedge 2z + 2 \leq x \leq 4z + 2 \wedge y = z), (\mathit{off}, \mathit{false})\}.\end{aligned}$$

Next, we compute $\varphi_2 = \mathit{Post}(\varphi_1)$. The jump successors of φ_1-states are those states for which the state assertion

$$\{(on, \mathit{false}),\ (\mathit{off}, x = 3 \wedge \frac{1}{4} \leq z \leq \frac{1}{2} \wedge y = z)\}$$

is true. This is because the control switch from *on* to *off* may happen only when $x = 3$, and the values of x, y, and z are not changed by the jump, and $2z + 2 \leq 3 \leq 4z + 2$ simplifies to $1/4 \leq z \leq 1/2$. Since φ_1 is already closed under flow successors (*i.e.* all flow successors of φ_1-states are also φ_1-states), we conclude

$$\begin{aligned}\varphi_2 &= \mathit{Post}(\varphi_1)\\ &= \{(on, x \leq 3 \wedge 2z + 2 \leq x \leq 4z + 2 \wedge y = z), (\mathit{off}, x = 3 \wedge \tfrac{1}{4} \leq z \leq \tfrac{1}{2} \wedge y = z)\}.\end{aligned}$$

For computing $\varphi_3 = \mathit{Post}(\varphi_2)$, there are no new jump successors and no new flow successors with the control mode *on*. The addition of all flow successors with the control mode *off* yields

$$\begin{aligned}\varphi_3 &= \mathit{Post}(\varphi_2)\\ &= \left\{\begin{array}{l}(on, x \leq 3 \wedge 2z + 2 \leq x \leq 4z + 2 \wedge y = z),\\ (\mathit{off}, 1 \leq x \leq 3 \wedge z + \frac{2}{3} \leq y \leq z + 2 \wedge\ 2z \leq x \leq 4z)\end{array}\right\}.\end{aligned}$$

Now the control switch from *off* to *on* adds new jump successors:

$$\begin{aligned}\varphi_4 &= \mathit{Post}(\varphi_3)\\ &= \left\{\begin{array}{l}(on, (x \leq 3 \wedge 2z + 2 \leq x \leq 4z + 2 \wedge y = z) \vee (x = 1 \wedge \frac{1}{4} \leq z \leq \frac{1}{2} \wedge z + \frac{2}{3} \leq y \leq z + 2)),\\ (\mathit{off}, 1 \leq x \leq 3 \wedge z + \frac{2}{3} \leq y \leq z + 2 \wedge 2z \leq x \leq 4z)\end{array}\right\}.\end{aligned}$$

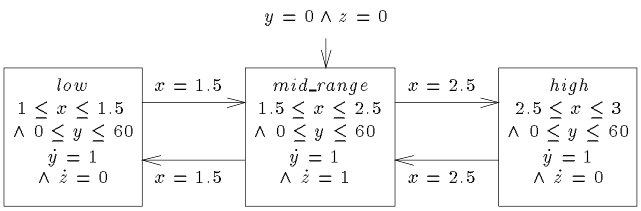

Figure 9: Monitor automaton

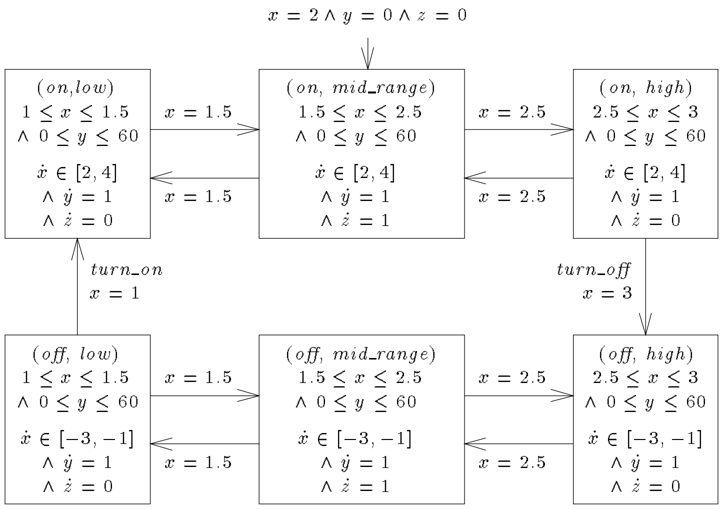

Figure 10: Parallel composition of thermostat automaton and the monitor automaton

The first disjunct of the *on* part of φ_4 characterizes the states that can be reached without jumps; the *off* part characterizes the states that can be reached by flows and at most one jump, from *on* to *off*; and the second disjunct of the *on* part characterizes the states that can be reached by flows and at most two jumps, from *on* to *off* and from *off* to *on*. HyTech performs these computations for us, fully automatically, until neither new jump successors nor new flow successors can be found. After 73 iterations, it returns the linear state assertion *reach* that is true precisely for the reachable states. Last, HyTech verifies that the variable-free predicate $\exists X. \bigvee_{v \in V}(reach(v) \wedge unsafe(v))$ is false. Hence there is no state for which both *reach* and *unsafe* are true, which confirms that the thermostat satisfies the safety requirement.

3.5 Parallel composition and monitors

Safety requirements cannot always be specified using state assertions. In the thermostat example, it was necessary to embellish the original automaton from Figure 1 with the variables y and z in order to specify the desired heater utilization requirement. Sometimes, it is convenient to build a separate automaton, called a monitor, whose role is to enter an unsafe state precisely when the original system violates a requirement. The monitor must observe the original system without changing its behavior. Reachability analysis is then performed on the parallel composition of the system with the monitor.

For the thermostat example, consider the task of verifying that the temperature lies in the midrange $[1.5, 2.5]$ at least 25% of the first 60 minutes of operation. The monitor automaton of Figure 9 uses the variable y to measure the total elapsed time, as before, and uses the variable z to measure the accumulated time that the temperature has been in the range $[1.5, 2.5]$. The unsafe states are specified by the state assertion that assigns the predicate $y = 60 \wedge z < y/4$ to all control modes of the monitor automaton.

The parallel composition of the monitor automaton and the thermostat automaton of Figure 1 is depicted in Figure 10. The variable set of the compound automaton is the union of the variable sets of both component automata, and the control graph of the compound automaton is the cartesian product of the component graphs. Each control mode of the compound automaton corresponds to both a control mode from the first component automaton and a control mode from the second component automaton (hence there are $2 \times 3 = 6$ control modes in our example). Each control switch of the compound automaton corresponds either to simultaneous control switches (with matching event labels) of both component automata (this cannot happen in the example), or to a control switch (with event label not appearing in the event set of the other component) of one of the component automata. This notion of parallel composition corresponds to the interleaving model of discrete concurrent computation. HyTech constructs the compound automaton automatically and confirms the desired safety requirement, namely, that all reachable states of the compound automaton correspond to safe states of the monitor automaton.

3.6 Parametric analysis

High-level system descriptions often use *design parameters*—symbolic constants with unknown, fixed values. The parameters are not assigned values until the implementation phase of design. The goal of parametric analysis is to determine necessary and sufficient constraints on the parameters under which safety violations cannot occur. Thus, rather than merely verifying (or falsifying) systems for certain values of the design parameters, quantitative information is extracted, further aiding the design process. Common uses for parametric analysis include determining minimum and maximum bounds on variables, and finding cutoff values for timers and cutoff points for the placement of sensors.

In a linear hybrid automaton A, a design parameter α can be represented as a variable whose value never changes, *i.e.* all flow conditions must imply $\dot{\alpha} = 0$ and all jump conditions must imply $\alpha' = \alpha$. Then, in all states of a trajectory of A, the parameter α has the same value (but the value of α may differ from trajectory to trajectory). The value $a \in \mathbb{R}$ is called *safe for* α if whenever we add the conjunct $\alpha = a$ to all initial conditions of A, then no unsafe state is reachable. This is the case precisely when there is no trajectory of A such that (1) the last state of the trajectory is unsafe, and (2) the parameter α has the value a in all states of the trajectory. Since the value of α is constant along each trajectory, requirements (1) and (2) are equivalent to the single requirement that the last state of the trajectory is unsafe and assigns the value a to the parameter α. Thus, the predicate $\exists X \setminus \{\alpha\}. \bigvee_{v \in V}(\mathit{reach}(v) \wedge \mathit{unsafe}(v))$ is a predicate over the variable α which is true precisely for the unsafe values for α. If *reach* and *unsafe* are linear state assertions, then the existential quantifier can be eliminated effectively, and we obtain, by negation, a linear predicate that characterizes exactly the safe values for the parameter α. Multiple parameters can be handled analogously.

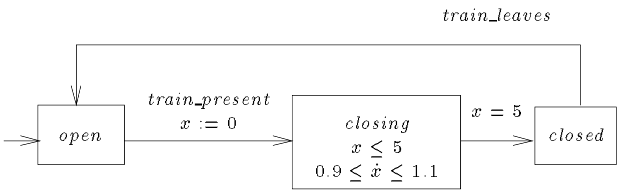

Figure 11: Railroad-gate controller automaton

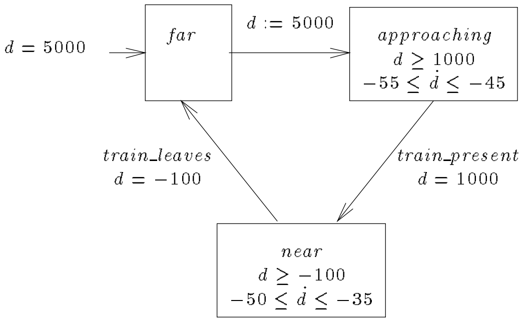

Figure 12: Train automaton

Example: railroad-gate controller

We consider the railroad-gate controller from Figures 17 and 19 of [6]. The controller, modeled by the automaton of Figure 11, lowers and raises a gate at a railroad crossing. Whenever it detects the presence of an oncoming train, it closes the gate after 5 time units, as measured with the local clock x. The clock is subject to 10% drift, and thus the gate may be closed at any time between 4.5 time units and 5.5 time units after the approaching train is detected. The controller raises the gate when the train exits the vicinity of the crossing. The train is modeled by the automaton of Figure 12. It approaches the crossing at a speed between 45 and 55 meters per time unit. When it is 1000 meters from the crossing, a sensor signals its approach to the controller, and the speed of the train is reduced to the range of 35 to 50 meters per time unit. A second sensor, at 100 meters past the crossing, signals the exit of the train. The train may return to the crossing, but only on a route that is at least 5100 meters long.

The complete system is represented by the parallel composition of the controller automaton and the train automaton. In this example, event labels are used to synchronize the concurrent execution of the controller and the train: the control switches labeled *train_present* must be executed simultaneously, thus ensuring that the controller receives the approach signal from the train; similarly, the control switches labeled *train_leaves* must be executed simultaneously, ensuring that the controller receives the exit signal from the train (as before, control switches without event labels are executed individually). The resulting compound automaton is shown in Figure 13. Many of

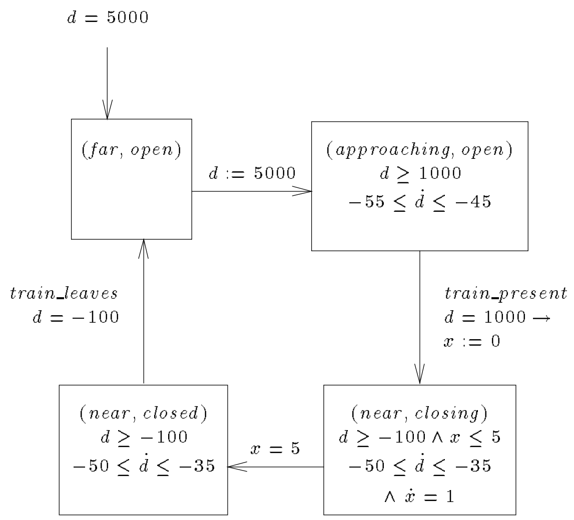

Figure 13: Parallel composition of train automaton and controller automaton

the control modes of the compound automaton are not reachable in the control graph via a path of directed edges from the initial mode (*far*, *open*). No state in such a mode can satisfy the state assertion *reach*, so these modes are excluded from the figure. The safety requirement of interest asserts that whenever the train is within 10 meters of the crossing, then the gate is closed. Accordingly, the unsafe states are specified by the state assertion that assigns the predicate *false* to the control modes of the compound automaton whose controller component is *closed*, and assigns $-10 \leq d \leq 10$ to all other control modes. HyTech automatically verifies this property.

In this example, parametric analysis can be used to determine how much advance notice the controller requires in order to meet the safety requirement. For this purpose, we replace the constant 1000, which indicates the location of the sensor that detects oncoming trains, by a parameter α. Then, the controller is alerted when an approaching train is α meters from the crossing. The parametric train automaton is shown in Figure 14. HyTech computes that for the same controller and the same safety requirement as before, the necessary and sufficient constraint for correctness is $\alpha > 287\frac{7}{9}$. Thus the controller will lower the gate in time if and only if it is warned of the approaching train before the train is $287\frac{7}{9}$ meters from the crossing.

Example: heater utilization

For the thermostat example, we can use parametric analysis to determine an upper bound on the time the heater is active during the first hour of operation. For this purpose, we introduce a parameter α and specify the unsafe states by the state assertion that assigns the predicate $y = 60 \wedge z \geq \alpha$ to all control modes. We then let HyTech compute the values of α for which an unsafe state is reachable. The largest such α value is an upper bound on the value of z after 60 minutes of elapsed time. For the linear phase-portrait approximation from Figure 6, HyTech returns the constraint $\alpha \leq 36$, implying that the thermostat is active for no more than 36 minutes during the first hour. For the more accurate approximation from Figure 5, HyTech computes the

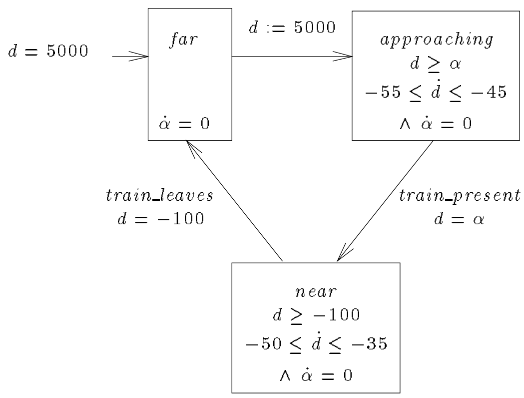

Figure 14: Parametric train automaton

tighter upper bound of $33\frac{1}{3}$ minutes.

In this paper, we use linear phase-portrait approximations of the thermostat automaton to demonstrate safety verification and parametric analysis. However, the clock translation of the thermostat automaton, depicted in Figure 3, could also be used. This hybrid automaton is not itself a linear hybrid automaton, since its description involves irrational constants such as $\ln 2$. Linear phase-portrait approximation must be applied before the automaton can be analyzed with HyTech. For example, the value of $\ln 2$ is approximately 0.69315, so the invariant $t_x \leq \ln 2$ for the mode $(on, 1)$ may be replaced by $t_x \leq 70/100$, and the constraint $t_x = \ln 2$ in the jump condition for the switch $((on, 1), (off, 3))$ may be replaced by $69/100 \leq t_x \leq 70/100$. When $\ln 3$ and $\ln(3/2)$ are likewise approximated to the nearest 1/100, HyTech computes an upper bound of 23.51 minutes of heater utilization.

4 HyTech

We give only a very brief introduction to HyTech—a detailed tutorial appears in the user guide [24]. A HyTech input file consists of two parts. The first part contains the textual description of a collection of linear hybrid automata, which are automatically composed for the analysis. The second part of the input contains a sequence of analysis commands. The analysis language is a simple while-programming language that provides as primitive the data type "state assertion" with a variety of operations, including *Post*, boolean operators, and existential quantification. This gives the user a flexible framework for writing state-space exploration programs. For added convenience, there are built-in macros for reachability analysis, parametric analysis, the conservative approximation of state assertions [22], and the generation of error trajectories (see below).

For example, the following command lines are taken from the analysis script for determining, for the thermostat from Figure 6, the duration α that the heater is active during the first hour of operation:

```
Time:  0.00
Location:  on
     x = 2 & y = 0 & z = 0
---------------
  VIA 0.50 time units
---------------
Time:  0.50
Location:  on
     x = 3 & 2y = 1 & 2z = 1
-------------------------------
  VIA: turn_off
-------------------------------
Time:  0.50
Location:  off
     x = 3 & 2y = 1 & 2z = 1
---------------
  VIA 0.67 time units
---------------
Time:  1.17
Location:  off
     x = 1 & 6y = 7 & 2z = 1
-------------------------------
  VIA: turn_on
-------------------------------
Time:  1.17
Location:  on
     x = 1 & 6y = 7 & 2z = 1
---------------
  VIA 0.83 time units
---------------
Time:  2.00
Location:  on
     3x = 8 & y = 2 & 3z = 4
```

Figure 15: Error trajectory

```
unsafe := y=60 & z >= alpha;                              (1)
reachable :=                                              (2)
 reach forward from init_states endreach;                 (3)
bad_alpha_values := omit all locations                    (4)
 hide non_parameters in reachable & unsafe endhide;       (5)
prints "Spec.  violated for parameter values:";           (6)
print omit all locations bad_alpha_values;                (7)
good_alpha_values := ~bad_alpha_values;                   (8)
prints "Spec.  satisfied for parameter values:";          (9)
print omit all locations good_alpha_values;              (10)
```

In line 1, the unsafe states are specified. The states that are reachable from the initial states via iteration of the *Post* operator are computed in line 3. The command on lines 4 and 5 performs existential quantification, in order to obtain a predicate that characterizes the unsafe reachable states as a constraint on the parameters. The symbol ~ that appears in line 8 denotes the negation operation. The statements in lines 6, 7, 9, and 10 produce the following output:

```
Spec. violated for parameter values:
      alpha <= 36
Spec. satisfied for parameter values:
      alpha > 36
```

4.1 Diagnostic information

One of the main benefits of state-space exploration tools lies in their ability to generate error trajectories when a system fails to satisfy a requirement. This information can then be used for debugging the system. If a system fails to satisfy a safety requirement, then an error trajectory leads from an initial state to an unsafe state. For example, for the thermostat of Figure 6, the heater can be active for more than 2/3 of the time during the first few minutes of operation. A debugging trace that demonstrates this fact is generated by the following input commands:

```
unsafe := y=2 & z >= 2/3 y;                        (1)
reachable :=                                       (2)
    reach forward from init_states endreach;       (3)
if not empty(reachable & unsafe)                   (4)
    then print trace to unsafe using reachable;    (5)
    else prints "Safety property satisfied.";      (6)
endif;                                             (7)
```

During the reachability analysis invoked in lines 2 and 3, backward pointers are maintained to indicate a predecessor state for each reachable state. This information is utilized by the command on line 5, which constructs an error trajectory. The HyTech output appears in Figure 15. It corresponds to the trajectory

$$(on, 2, 0, 0),\ (on, 3, \tfrac{1}{2}, \tfrac{1}{2}),\ (\mathit{off}, 3, \tfrac{1}{2}, \tfrac{1}{2}),\ (\mathit{off}, 1, 1\tfrac{1}{6}, \tfrac{1}{2}),\ (on, 1, 1\tfrac{1}{6}, \tfrac{1}{2}),\ (on, 2\tfrac{2}{3}, 2, 1\tfrac{1}{3}),$$

where $(on, 2, 0, 0)$ represents the state $(on, x = 2, y = 0, z = 0)$, etc. Notice that the HyTech output also provides the duration of flows. In the generated trajectory, the temperature x increases as slowly as possible in control mode *on*, until it reaches 3 degrees after 0.5 minutes; then it decreases as fast as possible in control mode *off*, until it reaches 1 degree at 1.17 minutes; and then it increases as slowly as possible until the time limit of 2 minutes. From the last state of the trajectory, we infer that the heater has been active for exactly 2/3 of the first 2 minutes of elapsed time.

4.2 Applications

HyTech has been used in a number of case studies—primarily control-based applications—including a distri-buted robot controller [21], a robot system in manufacturing [21], the Philips audio control protocol [33], an active structure controller [23], a generalized railroad controller [23], a nonlinear temperature controller [20], a predator-prey ecology [30], an aircraft landing-gear system [35], a steam-boiler controller [31], and an automotive engine chassis-level controller [38]. Corbett [12] has verified robot controllers written in a subset of Ada by automatically translating them into linear hybrid automata for analysis with HyTech. We are currently experimenting with the modeling and analysis of timed circuits.

Tools are also available for the *simulation* of hybrid automata, such as Shift [14], and for the abstract interpretation of linear hybrid automata, such as Polka [16]. Abstract-interpretation techniques can enforce the convergence of fixpoint computations by relaxing state assertions. For the verification of hybrid systems that are primarily *discrete* but include clocks, we recommend the use of specialized tools for the restricted class of timed automata. Symbolic model checkers for timed automata include Kronos [13], timed Cospan [7], timed Hsis [8], Uppaal [9], and Veriti [15]. These systems use algorithms that are specific to clocks, and therefore are more efficient for clock systems than the more general algorithms of HyTech. For the analysis of hybrid systems whose complexity is primarily in the *continuous* domain, we recommend the use of dynamics theory and numerical tools: in this case, the abstractions from nonlinear hybrid automata to linear hybrid automata are likely to be too crude to reap the full benefits of automated analysis. HyTech has been most successful when applied to systems that involve an intricate interplay between discrete and continuous dynamics.

4.3 Availability

Early versions of HyTech were built using Mathematica [21, 32], and linear predicates were represented and manipulated as symbolic formulas. Based on the observation that a linear predicate over n variables defines a union of polyhedra in $\mathbb{R}^n$, the current, more efficient generation of HyTech [23] manipulates linear predicates via calls to a library for polyhedral operations [16]. HyTech has been ported to the following platforms: Digital workstations running Ultrix V4.5 and Digital Unix V3.2D-1, HP 9000 workstations running HP-UX, Sun workstations running SunOS 4.x and Solaris 5.4, and x86 PCs running Linux. The HyTech home page

`http://www.eecs.berkeley.edu/~tah/HyTech`

includes the source code, executables, an online demo, a user guide, a graphical front end (courtesy of members of the Uppaal project [9]), numerous examples, online versions of papers, and pointers to additional literature. Requests may also be sent to `hytech@eecs.berkeley.edu`.

References

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

[2] R. Alur, C. Courcoubetis, N. Halbwachs, T. 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.

[3] R. Alur, C. Courcoubetis, T. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R. Grossman, A. Nerode, A. Ravn, and H. Rischel, editors, *Hybrid Systems I*, Lecture Notes in Computer Science 736, pages 209–229. Springer-Verlag, 1993.

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

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

[6] R. Alur, T. A. Henzinger, and P. W. Kopke. Real-time system = discrete system + clock variables. *Software Tools for Technology Transfer*, 1(1), 1997.

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

[8] F. Balarin and A. L. Sangiovanni-Vincentelli. Iterative algorithms for formal verification of embedded real-time systems. In *ICCAD 94: International Conference on Computer Aided Design*, pages 450–457, 1994.

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

[10] J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–70, 1992.

[11] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Workshop on Logic of Programs*, Lecture Notes in Computer Science 131. Springer-Verlag, 1981.

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

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

[14] A. Deshpande, A. Göllü, and L. Semenzato. The SHIFT programming language and run-time system for dynamic networks of hybrid automata. *IEEE Transactions on Automatic Control*, 1997. To appear.

[15] D. Dill and H. Wong-Toi. Verification of real-time systems by successive over- and underapproximation. In P. Wolper, editor, *CAV 95: Computer-aided Verification*, Lecture Notes in Computer Science 939, pages 409–422. Springer-Verlag, 1995.

[16] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximation. In B. LeCharlier, editor, *SAS 94: Static Analysis Symposium*, Lecture Notes in Computer Science 864, pages 223–237. Springer-Verlag, 1994.

[17] M. Henzinger, T. Henzinger, and P. 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.

[18] T. 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.

[19] T. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996. Invited tutorial.

[20] T. 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.

[21] T. Henzinger and P.-H. Ho. HyTech: The Cornell Hybrid Technology Tool. In P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors, *Hybrid Systems II*, Lecture Notes in Computer Science 999, pages 265–293. Springer-Verlag, 1995.

[22] T. Henzinger and P.-H. Ho. A note on abstract-interpretation strategies for hybrid automata. In P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors, *Hybrid Systems II*, Lecture Notes in Computer Science 999, pages 252–264. Springer-Verlag, 1995.

[23] T. 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.

[24] T. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In E. Brinksma, W. Cleaveland, K. 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.

[25] T. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 1998. To appear.

[26] T. Henzinger and P. 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.

[27] T. Henzinger and P. 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.

[28] T. Henzinger, P. 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.

[29] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994. Special issue for LICS 92.

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

[31] T. 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.

[32] P.-H. Ho. *Automatic Analysis of Hybrid Systems*. PhD thesis, Cornell University, 1995.

[33] 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.

[34] P. Kopke. *The Theory of Rectangular Hybrid Automata*. PhD thesis, Cornell University, 1996.

[35] 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.

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

[37] J. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In M. Dezani-Ciancaglini and U. Montanari, editors, *Fifth International Symposium on Programming*, Lecture Notes in Computer Science 137, pages 337–351. Springer-Verlag, 1981.

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