Beyond HyTech: Hybrid Systems Analysis Using Interval Numerical Methods* **

Thomas A. Henzinger[1] Benjamin Horowitz[1] Rupak Majumdar[1]
Howard Wong-Toi[2]

[1] Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
{tah, bhorowit, rupak}@eecs.berkeley.edu
[2] Cadence Berkeley Laboratories, Berkeley, CA
howard@cadence.com

Abstract. Since hybrid embedded systems are pervasive and often safety-critical, guarantees about their correct performance are desirable. The hybrid systems model checker HyTech provides such guarantees and has successfully verified some systems. However, HyTech severely restricts the continuous dynamics of the system being analyzed and, therefore, often forces the use of prohibitively expensive discrete and polyhedral abstractions. We have designed a new algorithm, which is capable of directly verifying hybrid systems with general continuous dynamics, such as linear and nonlinear differential equations. The new algorithm conservatively overapproximates the reachable states of a hybrid automaton by using interval numerical methods. Interval numerical methods return sets of points that enclose the true result of numerical computation and, thus, avoid distortions due to the accumulation of round-off errors. We have implemented the new algorithm in a successor tool to HyTech called HyperTech. We consider three examples: a thermostat with delay, a two-tank water system, and an air-traffic collision avoidance protocol. HyperTech enables the direct, fully automatic analysis of these systems, which is also more accurate than the use of polyhedral abstractions.

1 Introduction

In a hybrid system, digital controllers interact with a continuous environment. Because of the increasing ubiquity of embedded real-time systems, hybrid systems directly control many of the devices in our daily lives. Moreover, hybrid systems are often components of safety- or mission-critical systems. For these

* A preliminary version of this paper appeared in the *Proceedings of the Third International Workshop on Hybrid Systems: Computation and Control* (HSCC 00), *Lecture Notes in Computer Science* **1790**, Springer-Verlag, 2000, pp. 130–144.

** This research was supported in part by the DARPA (NASA) grant NAG2-1214, the DARPA (Wright-Patterson AFB) grant F33615-C-98-3614, the ARO MURI grant DAAH-04-96-1-0341, and the NSF CAREER award CCR-9501708.

reasons, it is necessary to have rigorous guarantees about the correct performance of hybrid systems.

Hybrid automata [1] provide a modeling paradigm for hybrid systems. In a hybrid automaton, the discrete state and dynamics are modeled by the vertices (called *locations*) and edges of a graph, respectively, and the continuous state and dynamics are modeled by points in $\mathbb{R}^n$ and differential equations, respectively. Symbolic model checking on a hybrid automaton provides correctness guarantees. HyTech [10] is a model checker for hybrid systems that has been successful in analyzing many hybrid systems of practical interest [2, 5, 13–16, 23, 26, 27].

Despite its successes, HyTech has several shortcomings. It restricts the dynamical model of the automaton being analyzed to that of linear hybrid automata. In linear hybrid automata, the continuous dynamics are governed by polyhedral differential inclusions, and all trajectories are composed of lines with piecewise constant slopes. These limitations force the verifier to approximate the complex dynamics of a hybrid system in a less expressive dynamical model. This approximation may take the form of *rate translation* [11], in which the first derivative of every continuous variable is bounded above and below by constants. *Location splitting* may be used to make the approximation arbitrarily accurate: each location can be split into many new locations; in these new locations, the dynamics may be bounded more precisely. However, location splitting leads to state explosion, as accuracy in the model comes at the price of a large number of new locations. Thus, the restrictive input language often forces the use of prohibitively large approximate models.

A second deficiency of HyTech is that arithmetic overflows frequently occur in the course of HyTech's computation. To explain this problem, we briefly describe the basic algorithm underlying HyTech. A state s of a hybrid automaton has two types of successors: *flow* successors, which are the states reachable from s by letting time progress; and *jump* successors, which are the states reachable from s if the automaton undergoes a change of location. Call a set of states a *polyhedral region* if its continuous part is a polyhedron. For linear hybrid automata, the flow and jump successors of a polyhedral region form again polyhedral regions. The computation engine of HyTech computes the set of states that can be reached from an initial polyhedral region by any number of flows and jumps. The iterated computation of flow and jump successors continues until either a target state is reached or no new states are generated.[1] The polyhedral manipulations for computing flow and jump successors use exact computation over rationals stored as integer pairs. However, these repeated computations quickly generate rationals with very large representations, leading to arithmetic overflows.

We have implemented the program HyperTech, which addresses both inadequacies of HyTech. First, HyperTech supports the analysis of hybrid automata with much more general dynamics. In particular, HyperTech can analyze automata whose continuous dynamics are given by differential equations of

[1] In general, the computation may fail to terminate, because the reachability problem for linear hybrid automata is undecidable [1].

the form $dx_i/dt = f(x_1, \ldots, x_n)$, where f is a composition of polynomials, exponentials, and trigonometric functions. This class of hybrid systems includes all multi-modal linear systems, i.e., systems whose continuous dynamics are given by matrix differential equations of the form $d\boldsymbol{x}/dt = A\boldsymbol{x} + B\boldsymbol{u}$ (where $\boldsymbol{u}$ represents the control input or disturbance input). HYPERTECH's more permissive input language enables a direct modeling of the continuous dynamics of hybrid systems. Since the need for introducing abstractions (in the form of rate translation and location splitting) is removed, input automaton models can be much more compact than with HYTECH.

Second, HYPERTECH uses interval numerical methods [20, 22] to compute an overapproximation of the set of reachable states of a hybrid automaton. In interval methods, the computed solution to a numerical problem, e.g., an initial value problem, is guaranteed to enclose the true solution. This is in contrast to conventional numerical methods, in which the accumulation of round-off errors may cause a computed solution to deviate from the real solution. The analysis engine of HYPERTECH, like that of HYTECH, starts with an initial region and iteratively adds flow and jump successors. However, HYPERTECH uses an interval ordinary differential equation (ODE) solver, instead of polyhedral manipulations, to compute an overapproximation of the flow successors of a set of states. It is the use of *interval* numerical methods which guarantees that the reachable states of a hybrid automaton H are contained in the set of states computed by HYPERTECH when run on H. All regions resulting from interval methods are *rectangular*, i.e., a product of intervals. Since geometrically manipulating rectangles is simpler than manipulating arbitrary polyhedra, the internal representations of HYPERTECH's rectangles never grow very large. In this way, HYPERTECH avoids the numeric overflow errors of HYTECH.

In essence, while the restrictive dynamics of HYTECH force an approximation in the model (*static approximation*), the permissive dynamics of HYPERTECH allow approximation to occur only during the computation of reachable states (*dynamic approximation*). Despite the fact that the dynamic approximation using rectangular regions seems rough, we demonstrate it to be superior to static approximation using polyhedral differential inclusions, on three examples: a thermostat with delay, a two-tank water system, and an air-traffic collision avoidance protocol.

In traditional numerical integration, the accumulation of round-off errors may cause the computed solution to a numerical problem to differ widely from the real solution. In the context of hybrid systems analysis, the loss of precision caused by the accumulation of round-off errors in the numerical integration process may lead the analyzer, whether human or computer, to overlook potentially hazardous events. In contrast to hybrid systems simulators (see [21] for a survey) and reach-set computation tools [3, 4, 6, 8, 26] which use traditional (not interval-based) numerical methods, HYPERTECH is guaranteed not to miss any events. Thus, if an unsafe (target) state of a hybrid automaton is reachable, HYPERTECH is guaranteed to note its reachability.

The rest of the paper is organized as follows. In Section 2, we describe the syntax and semantics of the hybrid automaton model, and define the reachability problem. In Section 3, we describe in detail the algorithm implemented in HyperTech. In Section 4, we describe the results of running HyperTech on three examples, and compare the results with HyTech.

2 Hybrid automata

To model hybrid systems, we use hybrid automata [1]. Let $\mathbb{R}^n$ be the n-dimensional Euclidean space. A *rectangle of dimension* n is a subset of $\mathbb{R}^n$ that is the Cartesian product of (possibly unbounded) intervals, all of whose finite endpoints are rational. For a positive integer n, let $\mathcal{R}^n$ denote the set of all n-dimensional rectangles. An *axis-parallel hyperplane* $h \subseteq \mathbb{R}^n$ is a set of points $\{\mathbf{x} \mid \mathbf{x}_i = a\}$ for some $i \in \{1, \ldots, n\}$ and some rational number a. Let a *dynamical equation* be an expression A generated by the grammar

$$A := (\dot{x}_1 = B_1 \wedge \dot{x}_2 = B_2 \wedge \cdots \wedge \dot{x}_i = B_i)\,,\ B := x_j \mid [a, b] \mid B_1 \ \mathsf{op}\ B_2 \mid \mathsf{f}(B)$$

where i and j are positive integers, a and b are any rational numbers such that $a \leq b$, op is one of the arithmetic operations $+$ (addition), $-$ (subtraction), $\cdot$ (multiplication), $/$ (division), or $\hat{}$ (exponentiation), and f is one of the functions sin, cos, tan, or exp. We shall use conventional mathematical notation for dynamical equations whenever possible, and if $a = b$ we shall often omit the square braces. For example, $\dot{x}_1 = \frac{1}{2}\sqrt{x_1 - x_2^2} \ \wedge\ \dot{x}_2 = \sqrt{x_1^2 + x_2^2}$ is a dynamical equation. For a positive integer n, let $\mathcal{E}^n$ denote the set of all dynamical equations in which for each subexpression of the form $\dot{x}_i$ or x_j, both $i \leq n$ and $j \leq n$. The above example of a dynamical equation is a member of $\mathcal{E}^2$.

2.1 Syntax. A *hybrid automaton* H consists of the following components:

- A finite set $X = \{x_1, \ldots, x_n\}$ of real-valued variables. A valuation of these variables represents a continuous state of a hybrid system.
- A finite directed multigraph (V, E). The vertices in V (called *control locations*) represent the discrete state of a hybrid system. The edges in E (*control switches*) represent transitions between discrete states.
- Three functions $\mathit{inv} : V \to \mathcal{R}^n$, $\mathit{init} : V \to \mathcal{R}^n$, and $\mathit{flow} : V \to \mathcal{E}^n$. Each invariant $\mathit{inv}(v)$ represents a condition that must be satisfied if the automaton is to remain in location v. Each initial condition $\mathit{init}(v) \subseteq \mathit{inv}(v)$ represents the continuous states in which the hybrid automaton may begin executing, when control starts at location v. Each flow condition $\mathit{flow}(v)$ constrains the continuous dynamics of the hybrid system at location v.
- Two functions $\mathit{pre} : E \to \mathcal{R}^n$ and $\mathit{post} : E \to \mathcal{R}^n$. For each edge $e = (v, v')$ in E, we require that $\mathit{pre}(e) \subseteq \mathit{inv}(v)$ and that $\mathit{post}(e) \subseteq \mathit{inv}(v')$. Intuitively, $\mathit{pre}(e)$ represents the condition on the continuous state that must hold if control is to pass from v to v', and $\mathit{post}(e)$ constrains the possible values of the variables after the transfer of control from v to v'.

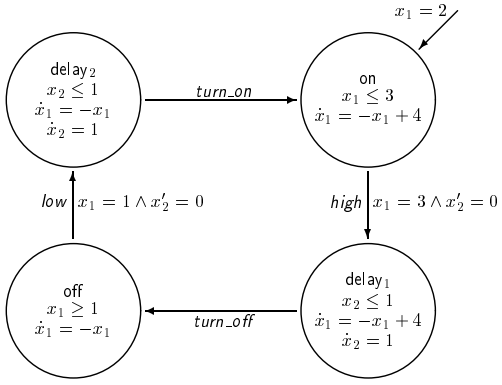

Fig. 1. Thermostat with delay

- A function $update : E \to 2^{\{1,\ldots,n\}}$ that assigns to each edge $e = (v, v') \in E$ a subset $update(e) \subseteq \{1, \ldots, n\}$. After traversing e, if the index i is in $update(e)$, then the variable x_i gets nondeterministically reset so as to lie in the i-th projection of $post(e)$, whereas if $i \notin update(e)$, then x_i remains unchanged.
- A finite set Σ of events, and a function *event* that assigns to each edge $e \in E$ an event.

As an example, consider the hybrid automaton of Figure 1, which models a thermostat system with delays: after the thermometer detects that the temperature is too low or too high, there may be a delay of up to one second before the appropriate control action (turn the heater on or off, respectively) is taken. The variable x_1 measures the temperature. Initially, $x_1 = 2$ and the heater is on. The temperature rises according to the differential equation $\dot{x}_1 = -x_1 + 4$. Eventually, the temperature reaches three degrees; after a delay of one second in location delay$_1$, the thermostat sends a *turn_off* signal to the heater. The variable x_2 measures the delay. The temperature then falls according to the equation $\dot{x}_1 = -x_1$ until $x_1 = 1$. One second after the temperature reaches one degree, the thermostat sends a *turn_on* signal to the heater, and the run of the automaton continues.

2.2 Semantics. We now give a formal definition of the semantics of a hybrid automaton. A *state* of a hybrid automaton is a pair $(v, \mathbf{x})$, with location $v \in V$, continuous state $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{x}$ satisfying $inv(v)$. The *state space* of a hybrid automaton is the set of its states. If $\mathbf{u} \in \mathbb{R}^n$ is a vector, we denote by $X := \mathbf{u}$ the interpretation for the variables in X in which $x_i = \mathbf{u}_i$ for $i = 1, \ldots, n$. A hybrid automaton has two types of transitions:

- *Jump transitions,* which correspond to instantaneous transitions between control locations. Formally, there is a *jump transition* from state $(v, \mathbf{x})$ to

state $(v', \mathbf{x}')$ if there is an edge $e = (v, v') \in E$ with $\mathbf{x}$ satisfying $pre(e)$, and $\mathbf{x}'$ satisfying $post(e)$, and $\mathbf{x}'_i = \mathbf{x}_i$ for $i \notin update(e)$.

- *Flow transitions,* which correspond to the continuous evolution of the system at a single control location v according to the dynamics specified by $flow(v)$. Formally, there is a *flow transition of duration* $t \geq 0$ from state $(v, \mathbf{x})$ to state $(v, \mathbf{x}')$ if there is a differentiable function $f : [0, t] \to \mathbb{R}^n$ such that: (1) $f(0) = \mathbf{x}$, $f(t) = \mathbf{x}'$; (2) for all reals $t' \in [0, t]$, $f(t') \in inv(v)$; and (3) for all reals $t' \in [0, t]$, the interpretation $X, \dot{X} := f(t'), \dot{f}(t')$ satisfies $flow(v)$.

We say that $(v', \mathbf{x}')$ is a *flow* (respectively *jump*) *successor* of $(v, \mathbf{x})$ if there is a flow (respectively jump) transition from $(v, \mathbf{x})$ to $(v', \mathbf{x}')$. A *run* of a hybrid automaton is an infinite sequence of states $(v_0, \mathbf{x}^0), (v_1, \mathbf{x}^1), \ldots$ such that $\mathbf{x}^0 \in init(v_0)$, and for all $i \geq 0$, $(v_{i+1}, \mathbf{x}^{i+1})$ is a jump or flow successor of $(v_i, \mathbf{x}^i)$.

2.3 Reachability problem. The fundamental verification problem for hybrid automata is *safety verification*: given a partition of the state space into "safe" states and "unsafe" states, verify that each execution of the hybrid automaton does not reach the unsafe states. Dually, one may look at the *reachability* question: does any run of the hybrid automaton ever reach an unsafe state? Formally, given a hybrid automaton H and a subset S of its state space, the reachability problem asks if there is a run $(v_0, \mathbf{x}^0), (v_1, \mathbf{x}^1), \ldots$ of H such that $(v_i, \mathbf{x}^i) \in S$ for some i. If there is such a run, we say that the set S is *reachable*. Clearly, a solution to the reachability problem gives a solution to the safety verification problem as well. The reachability problem is undecidable even for simple subclasses of hybrid automata [12]. However, semidecision procedures —for example, the algorithm of HyTech— often terminate on specific problems of practical interest.

3 The HyperTech algorithm

3.1 Interval numerical methods. In numerical computations, such as the numerical solution of ODEs, rounding errors may distort the accuracy of a sequence of calculations. Thus, ordinary numerical methods cannot provide fully rigorous guarantees about the safety of dynamical systems. Interval numerical methods [20] address this problem by computing sets of points that contain the true solutions to a numerical problem. In particular, interval ODE solvers find guaranteed bounds for the solutions to initial value problems.

In interval methods, the fundamental object of computation is not a floating point number, but rather an interval. An *interval* $[\underline{x}, \overline{x}]$ is a nonempty set of real numbers $\{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}$, where $\underline{x} \leq \overline{x}$ are both real numbers. One can extend to intervals the usual arithmetic operations over reals: if op is an arithmetic operation, then $[\underline{x}, \overline{x}]$ op $[\underline{y}, \overline{y}] = \{x \ \mathsf{op} \ y \mid x \in [\underline{x}, \overline{x}], y \in [\underline{y}, \overline{y}]\}$. The operations $+, -, \cdot$, and $/$ on intervals may be seen to satisfy the following

identities:

$$
\begin{aligned}
[\underline{x},\overline{x}] + [\underline{y},\overline{y}] &= [\underline{x}+\underline{y},\overline{x}+\overline{y}] \\
[\underline{x},\overline{x}] - [\underline{y},\overline{y}] &= [\underline{x}-\overline{y},\overline{x}-\underline{y}] \\
[\underline{x},\overline{x}] \cdot [\underline{y},\overline{y}] &= [\min(\underline{x}\cdot\underline{y},\underline{x}\cdot\overline{y},\overline{x}\cdot\underline{y},\overline{x}\cdot\overline{y}),\ \max(\underline{x}\cdot\underline{y},\underline{x}\cdot\overline{y},\overline{x}\cdot\underline{y},\overline{x}\cdot\overline{y})] \\
1/[\underline{x},\overline{x}] &= [1/\overline{x}, 1/\underline{x}] \quad \text{if } 0 \notin [\underline{x},\overline{x}]
\end{aligned}
$$

A computer implementation of these operations sets the processor's rounding mode to round down when computing the lower bound of the result, and round up when computing the upper bound. This guarantees that the computed result always encloses the result that would have been obtained using exact arithmetic calculation. In a similar fashion, one can implement interval versions of standard functions (e.g., $\sin x$, e^x, etc.) so that the computed result contains the exact result. Several interval arithmetic packages exist, either as libraries [18] or as extensions to regular programming languages [17].

Interval methods to solve initial value problems use as primitives the interval operations $+$, $-$, $\cdot$, and $/$ defined above, plus interval implementations of standard functions such as sine and cosine. From an initial condition (a rectangle r_0 at time 0), these methods usually compute a rough enclosure $r_{\Delta t}$ of the solution at time Δt, where Δt is an input parameter to the program. This rough enclosure, which is a rectangle, is usually narrowed by a pruning procedure that reduces the accumulation of numerical errors, and mitigates the wrapping effect. (The *wrapping effect* is the error resulting from enclosing a nonrectangular region by a rectangle.) This iteration —computing $x_{i\Delta t}$ using $x_{(i-1)\Delta t}$ by finding, and then pruning, a rough enclosure at time $i\Delta t$— continues for a number of steps which is specified by another input parameter.

Several implementations of interval ODE solvers are publicly available, for example [19, 24]. These typically use Picard iteration to prove the existence and uniqueness of a solution, and to find a rough enclosure. This enclosure is then pruned both by using a mean value method and by bounding the error term in a truncated Taylor expansion. To reduce the wrapping effect, local coordinate transforms may be applied. For a variety of examples, these implementations find fairly tight solution enclosures. In our implementation, we have used the ADIODES library [24]. Our choice of this library is independent of the other parts of HyperTech; thus, any other interval ODE solver, e.g., AWA [19], may be used in place of ADIODES.

3.2 Overapproximating reachable states. For a complex hybrid automaton H, precise analytic or closed-form descriptions of the reachable states of H may not exist or may be extremely difficult to find. In such cases, one must seek feasibly computable approximations of the reachable states. An *overapproximation* of the reachable states of H is a superset T of the reachable states of H. For analysis of the safety of a hybrid automaton, such an approximation may be useful, since if no unsafe state is in T, then no unsafe state is reachable. However, since there may be states in T which are not reachable states of H, the

presence of an unsafe state in T does not necessarily imply that an unsafe state of H is reachable. In such cases, one could try to refine the automaton under consideration.

Alternatively, one could compute overapproximations of the states which are backward reachable from the intersection of T and the unsafe states. If no initial state is contained in this backward approximation, then no unsafe state is reachable. This process may be iterated to find closer approximations of the reachable unsafe states [7, 9]. It is an interesting question in its own right to determine whether an error run produced by an overapproximative algorithm is an actual error run.

3.3 Overapproximation using interval numerical methods. For a hybrid automaton with discrete state set V and n real-valued variables, let a *region* be a set of states of the form $\{v\} \times U$, where U is a rectangle in $\mathbb{R}^n$. For any control location $v \in V$, let $\mathcal{H}_v$ be the set $\bigcup\{\{v\} \times pre(v, v') \mid (v, v') \in E\}$. For a rectangle U and a location v, let $U_{\Delta t,v}$ be the points $\mathbf{x}' \in \mathbb{R}^n$ such that there is a flow transition from $(v, \mathbf{x}) \in U$ to $(v, \mathbf{x}')$ of duration Δt. The procedure of HyperTech, which is presented in Figure 2, works as follows. It maintains two sets of regions: *Reached*, the explored set of regions, and *Frontier*, the set of regions that still need to be explored. As long as *Frontier* $\neq \emptyset$, one member $\{v\} \times U$ of *Frontier* is selected and removed from *Frontier*. The rectangle U is propagated according to the dynamics of v. An overapproximation of the set of reachable states $(v, \mathbf{x})$ is added to *Reached*, and an overapproximation of the set of reachable states $(v', \mathbf{x})$ (with $(v, v') \in E$) is added to *Frontier*.

The subroutine $Propagate_1$ first computes Y, a rectangular overapproximation of $U_{\Delta t,v}$. HyperTech uses an interval ODE solver to compute this overapproximation. The size of Δt must be determined by the user. Let S be the set of points reachable for some $\Delta\tau \leq \Delta t$, i.e., $S = \bigcup_{0 \leq \Delta\tau \leq \Delta t} U_{\Delta\tau,v}$. In addition to Y, the interval numerical method generates a rectangle T that contains the set S. In the procedure, *New-Reached* gets set to an overapproximation of the set of states in $\{v\} \times S$. Moreover, *New-Frontier* gets set to an overapproximation of the jump successors of states in $\{v\} \times S$. Notice that for large values of Δt, this bound on S may be quite coarse, and may not suffice to prove the safety property of interest. In that case, we have to reduce Δt and run the procedure again. Thus, whereas computations will be faster for larger values of Δt, more accurate analysis may require smaller values. (This speed/accuracy tradeoff is illustrated in Figure 6.) We wish to emphasize that our procedure is sound regardless of which $\Delta t > 0$ is chosen.

Theorem 1. *Let H be a hybrid automaton, and let $(v, \mathbf{x})$ be a reachable state of H. If the procedure Reachable-States (using subroutine $Propagate_1$) terminates on H, then $(v, \mathbf{x}) \in \bigcup Reached$.*

While $Propagate_1$ performs only one time step computation, under additional assumptions it is possible to group together multiple time step computations. The resulting procedure, called $Propagate_2$, is shown in Figure 3. In order for

Reachable-States(H : hybrid automaton)
Initialization: $\mathit{Frontier} := \{\{v\} \times \mathit{init}(v) \mid v \in V\}$;
$\mathit{Reached} := \mathit{Frontier}$;
while $\mathit{Frontier} \neq \emptyset$ do
pick $(\{v\} \times U) :\in \mathit{Frontier}$;
$\mathit{Frontier} := \mathit{Frontier} \setminus (\{v\} \times U)$;
$(\mathit{New\text{-}Reached}, \mathit{New\text{-}Frontier}) := \mathit{Propagate}\,(v, U)$;
$\mathit{Reached} := \mathit{Reached} \cup \mathit{New\text{-}Reached}$;
$\mathit{Frontier} := \mathit{Frontier} \cup \mathit{New\text{-}Frontier}$;
endwhile;

Propagate$_1$(v : location, U : rectangle)
$Y :=$ a rectangular overapproximation of $U_{\Delta t, v}$;
$T :=$ a rectangle which contains $\mathit{inv}(v) \cap \left(\bigcup_{0 \leq \Delta\tau \leq \Delta t} U_{\Delta\tau, v}\right)$;
$\mathit{New\text{-}Reached} := \{\{v\} \times T\}$;
$\mathit{New\text{-}Frontier} :=$
$(\mathit{Unexplored\text{-}Jump\text{-}Successors}(v, T)) \cup (\{v\} \times (\mathit{inv}(v) \cap Y))$;
return $(\mathit{New\text{-}Reached}, \mathit{New\text{-}Frontier})$;

Unexplored-Jump-Successors(v : location, T : rectangle)
return $\{\{v'\} \times Z \mid (v, v') \in E, Z = \mathit{Update}(T \cap \mathit{pre}(v, v')),$
$Z \neq \emptyset, (\{v'\} \times Z) \not\subseteq \bigcup \mathit{Reached}\}$;

Fig. 2. HyperTech's procedure for reach-set computation

the subroutine $\mathit{Propagate}_2$ to function correctly, the hybrid automaton H must satisfy the following conditions: (1) for each edge $(v, v') \in E$, the rectangle $\mathit{pre}(v, v')$ is a boundary of the invariant $\mathit{inv}(v)$; and (2) for each control location v and each point $\mathbf{x} \in \mathit{inv}(v)$, there exists a unique edge $e = (v, v')$ such that, under the dynamics $\mathit{flow}(v)$, the point $\mathbf{x}$ moves strictly monotonically towards the hyperplane $\mathit{pre}(v, v')$, and $\mathbf{x}$ eventually crosses $\mathit{pre}(v, v')$. For a large class of examples, including the hybrid automata in this paper, these two conditions hold.

Note that the above conditions imply that transitions are *urgent* —they must be taken as soon as they are enabled. Thus, $\mathit{Propagate}_2$ needs only to consider the first time a region hits one or more exit hyperplanes. The subroutine $\mathit{Propagate}_2$ functions like multiple iterations of $\mathit{Propagate}_1$, except that at each iteration those trajectories which have crossed an exit hyperplane are not further explored. By the conditions above, this optimization does not compromise soundness —the procedure of HyperTech still explores all reachable states.

Theorem 2. *Let H be a hybrid automaton satisfying conditions (1) and (2) above, and let $(v, \mathbf{x})$ be a reachable state of H. If the procedure Reachable-States (using subroutine $\mathit{Propagate}_2$) terminates on H, then $(v, \mathbf{x}) \in \bigcup \mathit{Reached}$.*

$Propagate_2(v$: location, U : rectangle)
    $W := U$; $New\text{-}Reached := New\text{-}Frontier := W_{prev} := T := P := \emptyset$;
    while $W \neq \emptyset$ do
        $W_{prev} := W$;
        $W :=$ a rectangular overapproximation of $W_{\Delta t,v}$;
        $T :=$ a rectangle which contains $inv(v) \cap \left(\bigcup_{0 \leq \Delta\tau \leq \Delta t} U_{\Delta\tau,v}\right)$;
        $New\text{-}Reached := New\text{-}Reached \cup \{\{v\} \times T\}$;
        $P :=$ the subset of W that has crossed $\mathcal{H}_v$;
        $W := W \setminus P$;
        if $P \neq \emptyset$ then $New\text{-}Frontier :=$
            $New\text{-}Frontier \cup Unexplored\text{-}Jump\text{-}Successors(v,T)$ endif;
    endwhile;
    return $(New\text{-}Reached, New\text{-}Frontier)$;

Fig. 3. Grouping together multiple time step computations

4 Three examples

With the use of interval methods, we obtain *both* a more direct model of the target system (i.e., no rate translation needed) *and* tighter bounds on the sets of reachable states. We substantiate this claim by describing the results of running HyperTech on three examples.

4.1 Thermostat with delay. Consider again the hybrid automaton of Figure 1. We wish to determine the range within which the temperature always lies. The nonlinear dynamics cannot be modeled directly in HyTech. Instead, the dynamics of the temperature x_1 are approximated using rate translation [11]. Using this method, the bounds obtained by HyTech are $0 \leq x_1 \leq 4$. This approximation may be made arbitrarily accurate by splitting each control location and using better bounds on the derivatives in the new locations. By combining rate translation with location splitting, and using a 20-location approximation of the system, HyTech obtains the bounds $0.28 \leq x_1 \leq 3.76$. This 20-location automaton is pictured in Figure 4.

We can run our algorithm directly on the automaton of Figure 1, with a step size of $\Delta t = 0.1$. Initially, $x_1 = 2$, and the automaton is in location on. Our algorithm propagates the values of x_1 according to the differential equation $\dot{x}_1 = -x_1 + 4$, until the interval containing the true value of x_1 entirely crosses the exit condition $x_1 = 3$. At this point, there is a discrete jump to location delay_1. Now our algorithm propagates the interval $[3,3]$ for one time unit. At the end of one time unit, $x_1 \leq 3.64$, and the automaton jumps to location off. Continuing this process, our algorithm reports that the minimum value of x_1 (which is reached in location delay_2) is 0.367. Therefore, using HyperTech, the bounds are $0.367 \leq x_1 \leq 3.64$. The bounds found by analytically solving this system are $\frac{1}{e} \leq x_1 \leq 4 - \frac{1}{e}$. Note that $\frac{1}{e} \approx 0.3679$ and $4 - \frac{1}{3} \approx 3.632$. Comparing

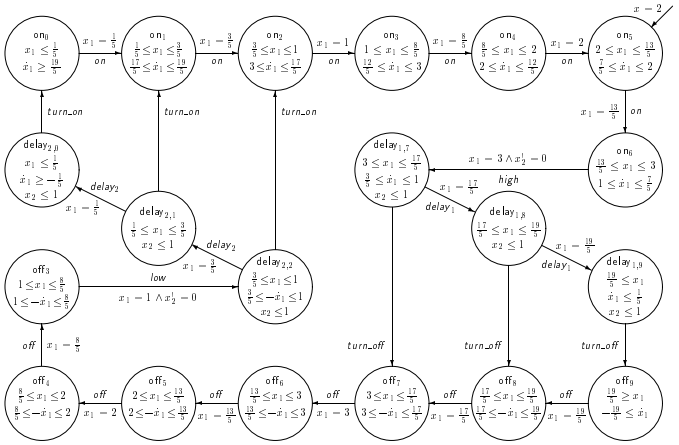

Fig. 4. Rate translation of thermostat automaton, with each location split into five locations

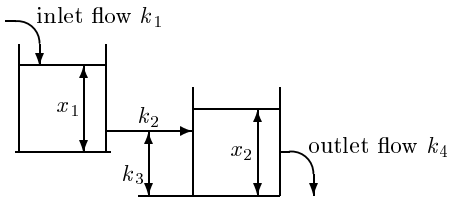

Fig. 5. Two-tank system

our results with the analytic solution shows that HyperTech computes a close approximation to the actual set of reachable states.

4.2 Two-tank system. As a second example, we consider the two-tank system of [25] (see Figure 5). The plant consists of two identical interconnected tanks. Into tank 1 flows a stream characterized by the loss parameter k_1.[2] Tank 1's outlet stream, characterized by the loss parameter k_2, flows into tank 2. Tank 1's outlet stream is k_3 meters above tank 2. The outlet stream of tank 2 is characterized by loss parameter k_4. Let x_1 and x_2 denote the heights of the liquid columns in tank 1 and tank 2. Applying Toricelli's law, the dynamics of this system may be seen to be:

$$\begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{cases} \begin{pmatrix} k_1 - k_2\sqrt{x_1 - x_2 + k_3} \\ k_2\sqrt{x_1 - x_2 + k_3} - k_4\sqrt{x_2} \end{pmatrix} & \text{if } x_2 > k_3 \\ \begin{pmatrix} k_1 - k_2\sqrt{x_1} \\ k_2\sqrt{x_1} - k_4\sqrt{x_2} \end{pmatrix} & \text{if } x_2 \leq k_3 \end{cases} \tag{1}$$

[2] This loss parameter may be thought of as a friction loss term.

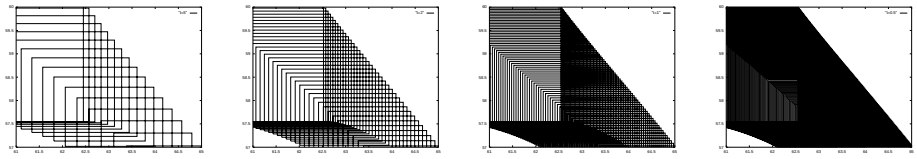

Fig. 6. A portion of the generated rectangles for the two-tank system at times $i\Delta t$, for $i = 0, 1, 2, \ldots$. HyperTech's actual computed overapproximation is the union of all rectangular hulls of pairs of consecutive rectangles. The horizontal (resp. vertical) axis shows the values of x_1 (resp. x_2). From the left: $\Delta t = 5$, running time: 24.27 s.; $\Delta t = 2$, running time: 53.39 s.; $\Delta t = 1$, running time: 98.60 s.; and $\Delta t = 0.5$, running time: 190.64 s.

The dynamical equations change when the liquid level in tank 2 is equal to the height of the connecting pipe. Under this dynamics, the system moves towards an equilibrium point for all $x_i > 0$ and for all $k_i > 0$. For example, for the parameter values $k_2 = k_4 = 1$ $\sqrt{\text{meters}}$ per second, $k_3 = 0.5$ meters, and $k_1 = 0.75$ meters per second, the system moves towards the equilibrium point $x_1 = 0.625\ldots$, $x_2 = 0.563\ldots$. In [25], rate approximation is used to model this dynamical system as a 12-location hybrid automaton; HyTech is then used to overapproximate which states were reachable. With HyperTech, we directly model the system as a hybrid automaton with two states, corresponding to whether $x_2 > k_3$ or not. Further, the analysis is more accurate. For example, HyTech's analysis of the 12-location rate approximation finds that starting from $0.70 \leq x_1 \leq 0.80$ and $0.45 \leq x_2 \leq 0.50$, some states in which both $0.60 \leq x_1 \leq 0.80$ and $0.60 \leq x_2 \leq 0.65$ are reachable, whereas our algorithm shows that these states are unreachable. In Figure 6, we show a part of the overapproximation of the reachable states of the two-tank system, for four different choices of the time step Δt, with the corresponding running times. The running times are obtained on a Sun SPARCstation-20.

4.3 Air-traffic conflict resolution. As a final example, consider an air-traffic

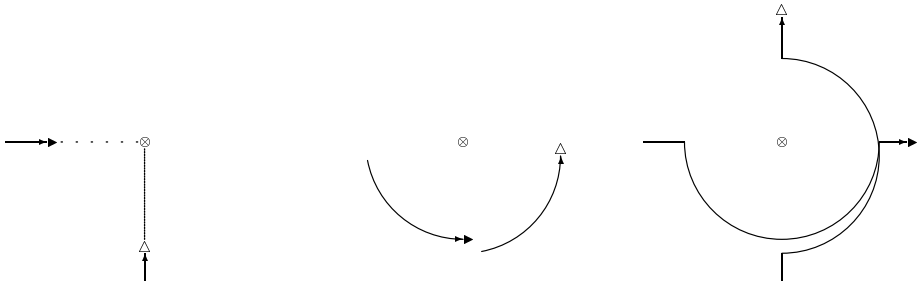

Fig. 7. Aircraft collision avoidance protocol

conflict resolution system from [26] (see Figure 7). Two aircraft fly towards each other at a fixed altitude and 90 degree relative orientation. When the distance between the aircraft decreases to seven miles, they initiate an avoidance maneuver: each turns 90 degrees to its right, and starts following a half circle. After the half circle is complete, each again turns 90 degrees to its right to continue on the original heading along a straight path.

We model this protocol directly as a three-location hybrid automaton with the original kinematics. In contrast, the protocol would need to be approximated in HyTech in order to be verified. Our model works in a relative coordinate system, so that x_r and y_r give the position of airplane 2 relative to airplane 1, and ψ_r gives the angular orientation of airplane 2 relative to airplane 1. In relative coordinates, the kinematic equations of this system are

$$\dot{x}_r = -v_1 + v_2 \cos\psi_r + \omega_1 y_r \ , \quad \dot{y}_r = v_2 \sin\psi_r - \omega_1 x_r \ , \quad \dot{\psi}_r = \omega_1 - \omega_2 \ , \qquad (2)$$

where v_1 (respectively v_2) is the airspeed of airplane 1 (respectively airplane 2) and ω_1 (respectively ω_2) is the angular velocity of airplane 1 (respectively airplane 2). Our automaton has three locations: cruise_1, avoid, and cruise_2. In location cruise_1 the airplanes follow straight-line trajectories, with airspeeds v_1 and v_2 in the range $[.8, 1]$. When the distance between the airplanes decreases to seven miles, the control location changes to avoid. On changing to location avoid, the heading of each aircraft decreases instantaneously by $\frac{\pi}{2}$ radians. In location avoid, $\omega_1 = \omega_2 = 1$ and $v_1 = v_2 = 1$, so that both airplanes follow circular trajectories of the same radius at the same airspeed. When the airplanes have completed their half-circles, the location changes to cruise_2. Again the heading of each aircraft decreases instantaneously by $\frac{\pi}{2}$ radians, and the airplanes continue in straight-line trajectories, with airspeeds v_1 and v_2 as in location cruise_1. Using this model, we are able to verify in HyperTech that the two airplanes never come within five nautical miles of each other.

References

1. R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems I*, LNCS 736, pages 209–229. Springer-Verlag, 1993.
2. B. Bérard and L. Fribourg. Automated verification of a parametric real-time program: the ABR conformance protocol. In *CAV 99: Computer-aided Verification*, LNCS 1633, pages 95–107. Springer-Verlag, 1999.
3. O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *HSCC 2000: Hybrid Systems: Computation and Control*, LNCS. Springer-Verlag, 2000.
4. A. Chutinan and B. Krogh. Computing polyhedral approximations to flow pipes for dynamic systems. In *Proceedings of the 37th Conference on Decision and Control*, pages 2089–2094. IEEE Press, 1998.
5. J.C. Corbett. Timing analysis of ADA tasking programs. *IEEE Transactions on Software Engineering*, 22(7):461–483, 1996.

6. T. Dang and O. Maler. Reachability analysis via face lifting. In *HSCC 98: Hybrid Systems: Computation and Control*, LNCS 1386, pages 96–109. Springer-Verlag, 1998.
7. D.L. Dill and H. Wong-Toi. Verification of real-time systems by successive over- and underapproximation. In *CAV 95: Computer-aided Verification*, LNCS 939, pages 409–422. Springer-Verlag, 1995.
8. M.R. Greenstreet and I. Mitchell. Integrating projections. In *HSCC 98: Hybrid Systems: Computation and Control*, LNCS 1386, pages 159–174. Springer-Verlag, 1998.
9. T.A. Henzinger and P.-H. Ho. A note on abstract-interpretation strategies for hybrid automata. In *Hybrid Systems II*, LNCS 999, pages 252–264. Springer-Verlag, 1995.
10. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.
11. T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):540–554, 1998.
12. T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.
13. T.A. Henzinger and H. Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, LNCS 1165, pages 265–282. Springer-Verlag, 1996.
14. P.-H. Ho. *Automatic Analysis of Hybrid Systems.* PhD thesis, Cornell University, 1995.
15. P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In *CAV 95: Computer-aided Verification*, LNCS 939, pages 381–394. Springer-Verlag, 1995.
16. P.-A. Hsiung, F. Wang, , and Y.-S. Kuo. Scheduling system verification. In *TACAS 99: Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1579, pages 19–33. Springer-Verlag, 1999.
17. R. Klatte, U. Kulisch, M. Neage, D. Ratz, and C. Ullrich. *Pascal-XSC: Language Reference and Examples.* Springer, 1992.
18. O. Knüppel. PROFIL/BIAS: A fast interval library. *Computing*, 53(3–4):277–287, 1994.
19. R. Lohner. Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems. In *Computational Ordinary Differential Equations.* Oxford University Press, 1992.
20. R.E. Moore. *Interval Analysis.* Prentice-Hall, 1966.
21. P.J. Mosterman. An overview of hybrid simulation phenomena and their support by simulation packages. In *HSCC 99: Hybrid Systems Computation and Control*, LNCS 1569, pages 165–177. Springer-Verlag, 1999.
22. R. Rihm. Interval methods for initial value problems in ODEs. In *Topics in Validated Computations.* North-Holland, 1994.
23. T. Stauner, O. Müller, and M. Fuchs. Using HyTech to verify an automotive control system. In *HART 97: Hybrid and Real-time Systems*, LNCS 1201, pages 139–153. Springer-Verlag, 1997.
24. O. Stauning. *Automatic Validation of Numerical Solutions.* PhD thesis, Technical University of Denmark, 1997.
25. O. Stursberg, S. Kowaleski, I. Hoffmann, and J. Preußig. Comparing timed and hybrid automata as approximations of continuous systems. In *Hybrid Systems IV*, LNCS 1273, pages 361–377. Springer-Verlag, 1997.

26. C.J. Tomlin. *Hybrid Control of Air Traffic Management Systems.* PhD thesis, University of California at Berkeley, 1998.
27. T. Villa, H. Wong-Toi, A. Balluchi, J. Preußig, A. Sangiovanni-Vincentelli, and Y. Watanabe. Formal verification of an automotive engine controller in cutoff mode. In *Proceedings of the 37th Conference on Decision and Control.* IEEE Press, 1998.