Beyond HyTech: Hybrid Systems Analysis Using Interval Numerical Methods*

Thomas A. Henzinger, Benjamin Horowitz, and **Rupak Majumdar**
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
{tah, bhorowit, rupak}@eecs.berkeley.edu

Howard Wong-Toi
Cadence Berkeley Laboratories, Berkeley, CA
howard@cadence.com

Abstract

Though the hybrid system model checker HyTech has successfully verified some systems, it restricts the dynamics to linear hybrid automata. We have designed an algorithm capable of verifying systems with more general dynamics. This algorithm uses interval numerical methods to conservatively overapproximate the reachable states of a hybrid automaton. We have implemented our new algorithm in HyTech+. Using three examples, we demonstrate that this algorithm enables both a more *accurate* and a more *direct* analysis of hybrid systems.

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

Hybrid automata (Alur *et al.* 1993) provide a modeling paradigm for hybrid systems. In a hybrid automaton, the discrete state and dynamics are modeled by the vertices 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 hybrid automaton models provides correctness guarantees. HyTech (Henzinger, Ho, & Wong-Toi 1997) is a model checker for hybrid systems that has been successful in analyzing many hybrid systems of practical interest (Ho & Wong-Toi 1995; Ho 1995; Corbett 1996; Henzinger & Wong-Toi 1996; Stauner, Müller, & Fuchs 1997; Villa *et al.* 1998).

One shortcoming of HyTech is that it restricts the dynamical model to linear hybrid automata, in which the continuous dynamics are governed by polyhedral differential inclusions. To make hybrid systems verification practical, one would like to be able to analyze a much wider class of continuous dynamics. Another deficiency of the HyTech is arithmetic overflows in the use of geometrical algorithms (i.e. in polyhedral manipulations): since points are stored as rationals, as the computation progresses these rationals may grow too large.

We have implemented HyTech+, in which these inadequacies are corrected. In particular, we have augmented HyTech so that it allows more general dynamics. We can now directly handle dynamics expressible as a combination of polynomials, exponentials, and trigonometric functions. Further, by restricting our attention to rectangular regions, we avoid overflow problems. Given a hybrid automaton, HyTech+ conservatively overapproximates the set of reachable states of the automaton. In order to obtain validated bounds for reachable sets, we use interval numerical methods (Moore 1966; Rihm 1994).

One of the fundamental steps of verification algorithms for hybrid automata is computing the flow successors of a given region. For certain classes of hybrid automata, e.g. linear hybrid automata, this computation can be performed exactly, either by quantifier elimination in the theory of reals with addition, or by polyhedral manipulation (Alur *et al.* 1993). To compute the flow successors under more general dynamics, algorithms may employ quantifier elimination in other theories (Lafferriere, Pappas, & Yovine 1999). However, such computations may be infeasible in practice, and approximation methods must be used. Some approximation methods bound the dynamics by piecewise constant inclusions, e.g. rate translation (Henzinger, Ho, & Wong-Toi 1998); others use numerical integration (Chutinan & Krogh 1998; Dang & Maler 1998; Greenstreet & Mitchell 1998; Tomlin 1998). Unfortunately, traditional numerical integration methods suffer from roundoff errors, and do not necessarily yield validated results. On the other hand, accurate rate translation suffers from state ex-

*This research was supported in part by the NSF CAREER award CCR-9501708, by the NSF grant CCR-9504469, by the DARPA (NASA Ames) grant NAG2-1214, by the DARPA (Wright-Patterson AFB) grant F33615-98-C-3614, and by the ARO MURI grant DAAH-04-96-1-0341.

plosion.

In contrast to traditional numerical methods, interval numerical methods yield results in which the true solution to an initial value problem is guaranteed to lie within validated bounds. In the worst case, such bounds may be unacceptably wide. However, interval methods never yield false results. Interval methods are based on interval arithmetic, for which several libraries provide support. These libraries have been used successfully in practice (Knüppel 1994). Interval methods are about twice as expensive as traditional methods, a worthwhile price to pay for obtaining validated bounds.

Hybrid Automata

To model hybrid systems, we use the simple, semantically precise model of hybrid automata (Alur *et al.* 1993). 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 the 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 *init*, *inv*, and *flow* that assign to each vertex in V a predicate. Each initial condition $init(v)$ and invariant $inv(v)$ has free variables from X. Each initial condition $init(v)$ represents the continuous states in which the hybrid automaton may begin executing, when control starts at location v. Each invariant $inv(v)$ represents a condition that must be satisfied if the automaton is to remain in control location v. Each flow condition $flow(v)$ has free variables from $\{x_1, \ldots, x_n, \dot{x}_1, \ldots, \dot{x}_n\}$. Intuitively, the $\dot{x}_i$s represent the first derivatives of the continuous variables. For a control location v, the flow condition $flow(v)$ constrains the continuous dynamics of the hybrid system at that location.
- A function *pre* that assigns to each edge $e = (v, v') \in E$ a predicate $pre(e)$, with free variables from X, which represents the condition on the continuous state that must hold if control is to pass from v to v'.
- A function *post* that assigns to each edge $e = (v, v') \in E$ a predicate $post(x)$, with free variables from X. Intuitively, the x_is represent the possible values of the variables after the transfer of control from v to v'.
- A function *update* that assigns to each edge $e = (v, v') \in E$ a subset $update(e) \subseteq X$. After traversing e, the variables in $update(e)$ get nondeterministically reset so as to lie in $post(e)$, while the variables in $X \setminus update(e)$ remain unchanged.
- A finite set Σ of events, and a function *event* that assigns to each edge $e \in E$ an event.

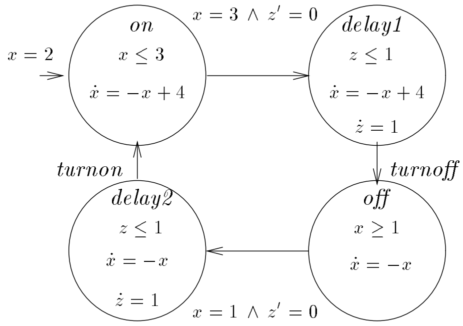

Figure 1: Hybrid automaton for the thermostat

As an example, consider the hybrid automaton of Figure (1), which models a thermostat controller with delays: after the thermometer detects that the temperature is too low or too high, there may be a delay of up to one time unit before the appropriate control action (turn the heater on or off, respectively) is taken. The variable x measures the temperature. Initially, $x = 2$ and the heater is on. The temperature rises according to the differential equation $\dot{x} = -x + 4$. Eventually, the temperature reaches 3; after a delay of one time unit, the thermostat sends a *turnoff* signal to the heater. (The variable z measures the delay.) The temperature then falls according the equation $\dot{x} = -x$ until $x = 1$. One time unit after the temperature reaches 1, the thermostat sends a *turnon* signal to the heater.

We now give a formal definition of the semantics of a hybrid automaton. A *state* of a hybrid automaton is a pair (v, x), with location $v \in V$, continuous state $x \in \mathbb{R}^n$, and x satisfying $inv(v)$. The *state space* of a hybrid automaton is the set of its states. If $X = \{x_1, \ldots, x_n\}$ is a set of variables and $u \in \mathbb{R}^n$ is a vector, we denote by $X := u$ the interpretation for the variables in X in which $x_i = u_i$ 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, x) to state (v', x') if there is an edge $e = (v, v') \in E$ with x satisfying $pre(e)$, x' satisfying $post(e)$, and $x'_i = x_i$ for $x'_i \notin update(e)$.

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* $t > 0$ from state (v, x) to state (v, x') if there is a differentiable function $f : [0, t] \to \mathbb{R}^n$ such that:

1. $f(0) = x$, $f(t) = x'$;
2. for all reals $t' \in (0, t)$, $X := f(t')$ satisfies $inv(v)$; and
3. $X, \dot{X} := f(t'), \dot{f}(t')$ satisfies $flow(v)$.

We say that (v', x') is a *flow* (respectively *jump*) *successor* of (v, x) if there is a flow (respectively jump) transition from (v, x) to (v', x'). A *run* of a hybrid automaton is a sequence of states $(v_0, x_0), (v_1, x_1), \ldots$ such that x_0 satisfies $init(v_0)$, and for all $i \geq 0$, (v_{i+1}, x_{i+1}) is a jump or flow successor of (v_i, x_i).

Given a hybrid automaton H and a subset S of its state space, the *reachability* problem asks if there is a run $(v_0, x_0), (v_1, x_1), \ldots$ of H that visits S, i.e. such that for some i, $(v_i, x_i) \in S$. The reachability problem is the fundamental subtask of safety verification.

Though the reachability problem is undecidable even for simple subclasses of hybrid automata (Henzinger *et al.* 1998), semidecision procedures often terminate on specific problems of practical interest. These procedures symbolically traverse the reachable state space of a hybrid automaton. In order to develop a symbolic algorithm, it is necessary to have a compact representation of sets of states. Such an algorithm has been implemented for particular subclasses of hybrid automata in HyTech, which uses unions of convex polyhedra to represent sets of states.

HyTech+ allows a dynamics which is more expressive than HyTech. In HyTech+, for each location $v \in V$, *flow*(v) is an expression generated by:

$$E := id \mid interval \mid E_1 * E_2 \mid f(E)$$

where *id* is a variable, *interval* is an interval; $*$ is one of $+, -, \cdot$, or $/$; and f is an exponential or a trigonometric function. While the dynamical model of HyTech+ is more general than that of HyTech, HyTech+ does place several restrictions on its input automata. These restrictions are more a convenience for building a quick prototype tool than they are fundamental limitations on the algorithmic capabilities of interval numerical methods. We are currently investigating how these conditions may be relaxed.

Our restrictions are as follows. First, for each $v \in V$, $init(v)$ and $inv(v)$ are finite unions of rectangles. (Here, a *rectangle* is a Cartesian product of n intervals, all of whose endpoints are rational.) Second, for each $e \in E$, $post(e)$ is a finite union of rectangles, and $pre(e)$ is a hyperplane. Third, for any control location v, and for any point p in $inv(v)$, there exists a unique edge e such that some flow successor of p satisfies $pre(e)$, and further under the dynamics of v all points in $inv(v)$ move strictly monotonically towards the hyperplane $pre(e)$ and eventually cross $pre(e)$. Fourth, we require that points that have once crossed $pre(e)$ do not return to cross again. Condition three guarantees that the approximations that HyTech+ computes are conservative. The uniqueness clause of condition three ensures that we can stop the computation of flow successors as soon as we have hit and crossed the first such hyperplane. Condition four excludes functions which cross an exit condition multiple times. For a large class of examples, including those analyzed below, these four conditions hold.

Interval Numerical Methods

In numerical computations, for example the numerical solution of ordinary differential equations (ODEs), rounding errors may distort the accuracy of a calculation. Ordinary numerical methods are thus not suitable for providing fully rigorous guarantees about the safety of dynamical systems. Interval numerical methods (Moore 1966) address this problem by computing sets of points that contain the true solutions to a numerical problem. In particular, interval ODE solvers are used to find guaranteed bounds for the solutions to differential equations.

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 $\underline{x} < \overline{x}$ are both real numbers. We can extend to intervals the usual arithmetic operations over reals: if $*$ is an arithmetic operation, then $[\underline{x}, \overline{x}] * [\underline{y}, \overline{y}] = \{x * y \mid x \in [\underline{x}, \overline{x}], y \in [\underline{y}, \overline{y}]\}$. These operations can be implemented as:

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

A computer implementation of these operations controls the processor's rounding mode so that it rounds downwards when computing the lower bound of the result, and it rounds upwards when computing the upper bound. This guarantees that the computed result always encloses the result of exact interval arithmetic calculation. Similarly, we can define interval versions of elementary functions (e.g. sin, cos, e^x, etc.) so that the computed result encloses the exact result. Several implementations of interval operations exist, either as libraries (Knüppel 1994) or as extensions to regular programming languages (Klatte *et al.* 1992). In our implementation, we have used the PROFIL/BIAS library (Knüppel 1994).

Interval methods to solve ODEs provide guaranteed enclosures of the solutions of initial value problems. These methods use as primitives the interval operations $+, -, \cdot$, and $/$ defined above, plus interval implementations of standard functions such as sin and cos. From an initial condition (an interval x_0 at time 0), these methods usually compute a rough enclosure $x_{\Delta t}$ of the solution at time Δt, for some Δt specified by the user. 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 *wrappping 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 the number of steps specified by the user. Several implementations of interval ODE solvers are publicly available, for example (Lohner 1992; Stauning 1997). These typically use Picard iteration to prove the existence and uniqueness of solution, and to find a rough enclosure. This enclosure is then pruned by using a mean value method; by bounding the error term in a truncated Taylor expansion; and by applying local coordinate transformations, to reduce the wrapping effect. For a variety of examples, these implementations find fairly tight solution enclosures.

Conservatively Overapproximating Reachable States

For a complicated 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. A *conservative overapproximation* of the reachable states of H is a set T of states such that any state reachable in H must be in T. For analysis of the safety of a hybrid automaton, a conservative overapproximation T may be useful, since if no unsafe state is in T, then no unsafe state is reachable in H. However, since there may be states in T which are not reachable in H, the presence of an unsafe state in T does not necessarily imply that an unsafe state is reachable in H. In such cases, one could try to refine the automaton under consideration.

For a hybrid automaton with discrete state set V and n real-valued variables, let a *region* be a finite union of sets of the form $\{v\} \times S$, where S is a subset of $\mathbb{R}^n$. HyTech's algorithm works as follows: it maintains two regions, R, the explored region, and R', the to-be-explored region. Initially, $R = \emptyset$, and R' is the initial region. As long as $R' \neq \emptyset$, it does the following:

1. Select and remove one member $\{v\} \times S$ from R'.
2. Add $\{v\} \times S$ to R.
3. Propagate S according to the dynamics of v, and add to R' at least the unexplored jump successors and flow successors of $\{v\} \times S$ as necessary. For exact (nonapproximate) algorithms, the regions added to R' contain exactly the jump successors and flow successors of $\{v\} \times S$. Overapproximate algorithms may add additional areas of the state space; that is, for overapproximate algorithms, the regions added to R' may *strictly* contain the jump successors and the flow successors of $\{v\} \times S$.

In HyTech, the input automaton is a linear hybrid automaton, and S is a convex polyhedron. HyTech uses polyhedral manipulation to compute the flow successors of S. Further, this computation is exact, in the sense mentioned in step (3).

For our overapproximate algorithm, S is a rectangle, and our overapproximation of the reachable flow successors of S is a union of rectangles. We introduce some terminology. For a given control location $v \in V$, an *exit plane* of v is any hyperplane $pre(e)$ for some $e = (v, v') \in E$. In what follows, let $\mathcal{E}$ be the set of points in the exit planes of v. Given a state (v, p), a time successor (v, q) of (v, p) has *crossed* the hyperplane $\mathcal{H}$ if q and p lie in different halfspaces induced by $\mathcal{H}$. We extend this definition to sets of states and unions $\mathcal{E}$ of hyperplanes in the natural way. We say that region S has *partially crossed* $\mathcal{E}$ if some states in S have crossed $\mathcal{E}$ while others have not.

The computation of flow successors is performed as follows. We use an interval ODE solver to compute $S_{\Delta t}$, the flow successor of S after a time step Δt. Note that $S_{\Delta t}$ is a rectangular overapproximation of the flow successors of S at time Δt. The size of Δt must be determined by the user; whereas computations will be faster for larger values, more accurate analysis may require smaller values.

- If $S_{\Delta t}$ has completely crossed the exit planes $\mathcal{E}$, then we compute the intersection of $\mathcal{E}$ with the rectangular hull of $S_{\Delta t}$ and S, and add the unexplored jump successors of this intersection to R'. (Our numerical method ensures that all flow successors of S, for times in $[0, \Delta t]$, lie in the convex hull. Taking the rectangular hull, as opposed to the more natural convex hull, maintains rectangularity of the reached regions at the cost of an overapproximation, and also ensures numerical robustness.)
- If some portion P of $S_{\Delta t}$ has crossed $\mathcal{E}$, we compute the intersection of $\mathcal{E}$ with the rectangular hull of S and P, add the unexplored jump successors of this intersection to R', and continue to propagate the set $S_{\Delta t} \setminus P$.
- Otherwise none of $S_{\Delta t}$ has crossed $\mathcal{E}$. We compute $S'_{2\Delta t}$ and continue, using $S'_{2\Delta t}$ in place of $S'_{\Delta t}$.

Note that, unlike the algorithm of HyTech, our algorithm does not find all the flow successors in one step. But for dynamics that require the use of numerical ODE solvers, such a price seems unavoidable.

In our implementation, we use the ADIODES library (Stauning 1997) to compute the flow successors of a region. ADIODES uses a Picard iteration to find a rough enclosure of the solution, then computes correct enclosures of the solution using a mean value method, by bounding the error term of a truncated Taylor series expansion of the dynamical equations. Automatic differentiation techniques are used to compute the coefficients of the Taylor series expansion. Our choice of this method is independent of the other parts of HyTech+; thus, any other interval ODE solver, e.g. AWA (Lohner 1992), may be used in place of ADIODES.

Our use of an interval ODE solver gives a conservative overestimate of the region reachable from $\{v\} \times S$. With the use of enclosure methods, we obtain *both* a more direct model of the target system (i.e. no rate translation needed) *and* tighter bounds on the reach sets.

Three Examples

We now describe the results of running HyTech+ on three examples.

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 directly modeled in HyTech. Instead, the dynamics of x is enclosed by piecewise-constant bounds. This approximation, described in (Henzinger, Ho, & Wong-Toi 1998), is called *rate translation.* Using this method, the best obtainable bounds are $0 \leq x \leq 4$. 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 obtained the bounds $0.28 \leq x \leq 3.76$.

We ran our algorithm *directly* on the automaton of Figure (1). Initially, $x = 2$, and the automaton is in location *on.* Our algorithm propagates the values of x according to the differential equation $\dot{x} = -x + 4$, until the interval containing the true value of x entirely crosses the exit condition $x = 3$. At this point, there is a discrete jump to location *delay1.* Now our algorithm propagates the interval $[3, 3]$ for one unit. At the end of one time unit, $x \leq 3.64$, and the automaton jumps to location *off.* Continuing this process, our algorithm reports that the minimum value of x (which is reached in location *delay2*) is 0.367. Therefore, using HyTech+, the bounds are $0.367 \leq x \leq 3.64$. The bounds found by analytically solving this system are $1/e \leq x \leq 4 - 1/e$. Comparing our results with the analytic solution shows that HyTech+ computes a close approximation to the actual reach set.

Railroad gate controller. As a second example, we consider the nonlinear railroad gate controller from (Ho 1995) (see Figure (2)). The three hybrid automata of Figure (2) model a train, a railroad gate, and the gate's controller. These automata can be composed as in (Alur *et al.* 1993). The real-valued variable x represents the distance of the train from the gate. Initially, in location *far*, the train is 1,000 meters from the gate, traveling at 50 meters per second towards the gate. At 500 meters, a sensor on the tracks detects the train, sending a signal *app* to the controller. The train slows down, obeying the differential equation $\dot{x} = -x/25 - 30$. After a delay of five seconds, which is modeled by the variable t, the controller sends the signal *lower* to the gate, which begins to descend from 90 degrees to 0 degrees at a rate of -20 degrees per second. After crossing the gate, the train accelerates according to the differential equation $\dot{x} = x/5 + 30$. A second sensor placed 100 meters past the crossing detects the leaving train, sending a signal *exit* to the controller. After five seconds, the controller raises the gate. At least 1,100 meters separate consecutive trains.

We wish to verify that if there is a train within 10 meters of the gate, then the gate is closed. The nonlinear dynamics of the train could not be modeled directly in HyTech. Instead, rate translations and additional approximations were used. Using HyTech+, we can model and verify the entire system directly from its specifications.

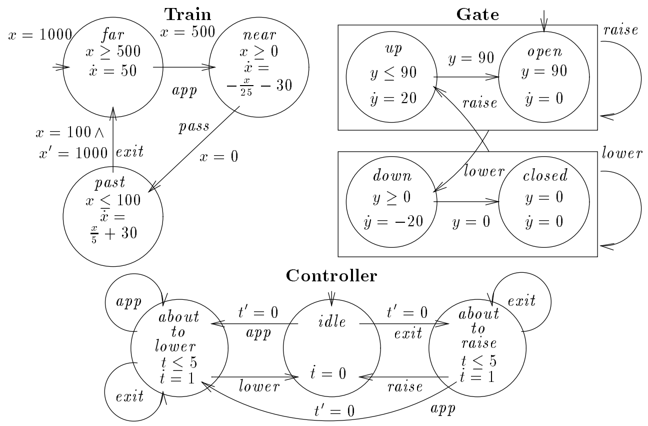

Figure 2: Automata for the train, gate, and controller

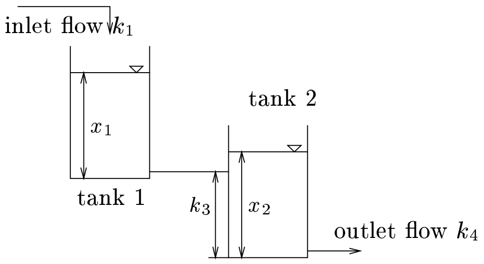

Figure 3: Two-tank system

Two-tank system. As a final example, we consider the two-tank system of (Stursberg *et al.* 1997) (see Figure (3)). The plant consists of two identical interconnected tanks. Into tank 1 flows a stream characterized by the loss parameter k_1.[1] Tank 1's outlet stream, characterized by the loss parameter k_2, flows into tank 2. Tank 1 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:

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

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

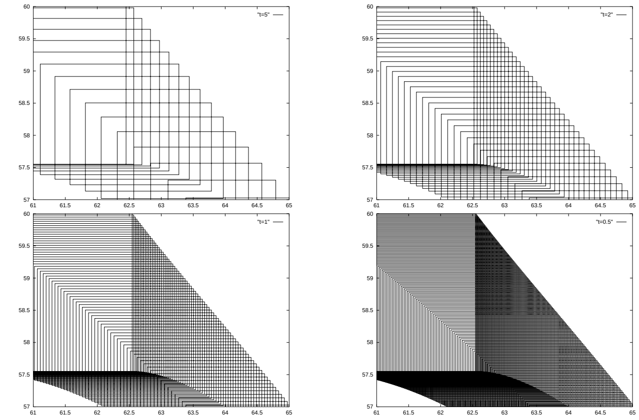

Figure 4: A portion of the reachable space for the two tank system. Clockwise from upper-left: $\Delta t = 5$, run time: 24.27 s.; $\Delta t = 2$, run time: 53.39 s.; $\Delta t = 1$, run time: 98.60 s.; and $\Delta t = 0.5$, run time: 190.64 s.(x_1 and x_2 are in centimeters.)

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 (Stursberg *et al.* 1997), rate approximation was used to model this dynamical system as a 12-location hybrid automaton; HyTech was then used to overapproximate which states were reachable. With HyTech+, 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. As may be seen using MATLAB, these states are actually not reachable.

In Figure (4), 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.

Future Work

These examples support our claim that HyTech+ both enables a more direct modeling of hybrid systems, and finds tighter bounds on the set of reachable states. We will apply our algorithm to more complex examples, to determine whether these may be successfully analyzed. We also wish to investigate whether interval numerical methods may be used with more complex sets in $\mathbb{R}^n$, for example unions of convex polyhedra, or unions of ellipsoids.

References

Alur, R.; Courcoubetis, C.; Henzinger, T.; and Ho, P.-H. 1993. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems I*, LNCS 736. Springer-Verlag. 209–229.

Chutinan, A., and Krogh, B. 1998. Computing polyhedral approximations to flow pipes for dynamic systems. In *Proc. 37th Conference on Decision and Control*. IEEE Press. 2089–2094.

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

Dang, T., and Maler, O. 1998. Reachability analysis via face lifting. In *HSCC 98: Hybrid Systems—Computation and Control*, LNCS 1386. Springer-Verlag. 96–109.

Greenstreet, M. R., and Mitchell, I. 1998. Integrating projections. In *HSCC 98: Hybrid Systems—Computation and Control*, LNCS 1386. Springer-Verlag. 159–174.

Henzinger, T., and Wong-Toi, H. 1996. 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. Springer-Verlag. 265–282.

Henzinger, T.; Kopke, P.; Puri, A.; and Varaiya, P. 1998. What's decidable about hybrid automata? *Journal of Computer and System Sciences* 57:94–124.

Henzinger, T.; Ho, P.-H.; and Wong-Toi, H. 1997. HyTech: a model checker for hybrid systems. *Software Tools for Technology Transfer* 1:110–122.

Henzinger, T.; Ho, P.-H.; and Wong-Toi, H. 1998. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control* 43(4):540–554.

Ho, P.-H., and Wong-Toi, H. 1995. Automated analysis of an audio control protocol. In *CAV 95: Computer-aided Verification*, LNCS 939. Springer-Verlag. 381–394.

Ho, P.-H. 1995. *Automatic Analysis of Hybrid Systems.* Ph.D. Dissertation, Cornell University.

Klatte, R.; Kulisch, U.; Neage, M.; Ratz, D.; and Ullrich, C. 1992. *Pascal-XSC: Language Reference and Examples.* Springer.

Knüppel, O. 1994. PROFIL/BIAS — a fast interval library. *COMPUTING 94* 53(3-4):277–287.

Lafferriere, G.; Pappas, G. J.; and Yovine, S. 1999. A new class of decidable hybrid systems. In *HSCC 99: Hybrid Systems: Computation and Control.* Springer-Verlag. To Appear.

Lohner, R. 1992. Computation of guaranteed enclosures for the solutions of ordinary initial and boundary value problems. In *Computational Ordinary Differential Equations*. Oxford.

Moore, R. E. 1966. *Interval Analysis*. Englewood Cliffs, N.J.: Prentice-Hall.

Rihm, R. 1994. Interval methods for initial value problems in ODEs. In Herzberger, J., ed., *Topics in Validated Computations*. North-Holland.

Stauner, T.; Müller, O.; and Fuchs, M. 1997. Using HyTech to verify an automotive control system. In *HART 97: Hybrid and Real-time Systems*, LNCS 1201. Springer-Verlag. 139–153.

Stauning, O. 1997. *Automatic Validation of Numerical Solutions*. Ph.D. Dissertation, Technical University of Denmark.

Stursberg, O.; Kowaleski, S.; Hoffmann, I.; and Preussig, J. 1997. Comparing timed and hybrid automata as approximations of continuous systems. In *Hybrid Systems IV*, LNCS 1273. Springer-Verlag. 361–377.

Tomlin, C. 1998. *Hybrid Control of Air Traffic Management Systems*. Ph.D. Dissertation, University of California at Berkeley.

Villa, T.; Wong-Toi, H.; Balluchi, A.; Preussig, J.; Sangiovanni-Vincentelli, A.; and Watanabe, Y. 1998. Formal verification of an automotive engine controller in cutoff mode. In *Proc. of the 37th Conference on Decision and Control*. IEEE Press. 4271–4276.