Model Checking Strategies for Linear Hybrid Systems*†

Thomas A. Henzinger††
Pei-Hsin Ho

TR 94-1437
July 1994

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

*This research was supported in part by the National Science Foundation under grant CCR-9200794, by the United States Air Force Office of Scientific Research under contract F49620-93-1-0056, and by the Defense Advanced Research Projects Agency under grant NAG2-892.

†A presentation based on this paper was given at the *Workshop on Formalisms for Representing and Reasoning about Time*, organized by S. B. Seidman and H. Gill as part of the *Seventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* on May 31, 1994, in Austin, Texas. Since then, it has come to the authors' attention that a group in France has been working on approximation strategies for analyzing linear hybrid systems similar to those presented in this paper [HRP94].

††Phone: (607)255-3009. FAX: (607) 255-4428.

Model Checking Strategies for Linear Hybrid Systems*†

Thomas A. Henzinger‡ Pei-Hsin Ho

Computer Science Department
Cornell University
(tah|ho)@cs.cornell.edu

Extended Abstract

Abstract. Linear hybrid systems are dynamical systems whose variables change both discretely and continuously along piecewise linear trajectories; they are useful for modeling digital real-time programs that are embedded in analog environments. Model checking is an algorithmic technique for analyzing finite-state systems that has recently been extended to certain infinite-state systems, including linear hybrid systems. The method has been implemented in HyTech (The Cornell Hybrid Technology Tool), a symbolic model checker for linear hybrid systems. We report on a new implementation and several experiments with HyTech.

The core of HyTech is a semidecision procedure that, given a linear hybrid automaton describing a system and a temporal formula describing a requirement, computes the so-called *target region*—the linear set of system states that satisfy the requirement. Unfortunately, the verification procedure may not return the target region using a reasonable amount of time and space, or it may not terminate in principle. Thus we have reimplemented the model checker using more efficient data structures that represent linear state sets geometrically, as unions of convex polyhedra, and we have experimented with several strategies that are designed to improve the performance of the model checker further: we (1) simultaneously compute the target region from different directions, (2) encode data as finite-state control, (3) approximate the target region by dropping constraints, and (4) iteratively refine the approximation until sufficient precision is obtained. Interestingly, symbolic model checking (fixpoint computation by iteration) and the polyhedral approximation strategies (3) can be viewed as the abstract interpretation of linear hybrid systems.

*This research was supported in part by the National Science Foundation under grant CCR-9200794, by the United States Air Force Office of Scientific Research under contract F49620-93-1-0056, and by the Defense Advanced Research Projects Agency under grant NAG2-892.

†A presentation based on this paper was given at the *Workshop on Formalisms for Representing and Reasoning about Time*, organized by S.B. Seidman and H. Gill as part of the *Seventh International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* on May 31, 1994, in Austin, Texas. Since then, it has come to the authors' attention that a group in France has been working on approximation strategies for analyzing linear hybrid systems similar to those presented in this paper [HRP94].

‡Phone: (607) 255-3009. FAX: (607) 255-4428.

1 Introduction

Hybrid systems extend discrete programs with continuous activities [MMP92]. Consider a communication protocol $\mathcal{A}$ with k propositional variables (values 0 or 1), and a robot $\mathcal{B}$ with k propositional control variables and n real-valued status variables (e.g., the position of the robot). Let $d = k \cdot n$. A state of $\mathcal{A}$ is a point in k-dimensional *boolean* space; a state of $\mathcal{B}$, a point in d-dimensional *euclidean* space. Each transition of $\mathcal{A}$ is a k-dimensional boolean transformation; if $\mathcal{B}$ is *linear*, then each discrete control transition of $\mathcal{B}$ is a d-dimensional linear transformation and each continuous activity of $\mathcal{B}$ is a d-dimensional piece-wise linear motion. It is this geometric intuition that inspired the model-checking strategies developed in this paper.

For the finite-state system $\mathcal{A}$, model-checking algorithms compute the state set (*target region*) that satisfy a given temporal requirement specification, by iterative approximation of the target region as a fixpoint. This is done either enumeratively [CES86] or symbolically, by representing state sets (*regions*) as boolean expressions [BCM+92]. Since the state space of the linear hybrid system $\mathcal{B}$ is infinite, enumerative model checking is no longer possible, and regions must be represented symbolically. A symbolic model-checking algorithm for linear hybrid systems has been developed [ACHH93, NOSY93, ACH+94, HNSY94], and implemented (HyTech) by representing regions as real-valued linear expressions [AHH93]. The linearity condition is key so that if the target region is a finitely approximable fixpoint, then it can be computed in the theory of the reals with addition, at "tolerable" cost. The most serious limitation of finite-state model checking, the state-explosion problem, reoccurs for the infinite state spaces of linear hybrid systems in three guises. The main theoretical limitation of the method is that termination (i.e., finite approximability) is not guaranteed (problem A); the main practical limitation of the implementation is its cost, which is largely due to two factors. First, continuous activities correspond to quantifier-elimination steps on real-valued linear expressions (problem B); second, if expressions are transformed into disjunctive normal form for further manipulation, then the number of disjunctions grows rapidly (problem C).

Having identified the three problems, we improved the performance of HyTech by using algorithms that manipulate geometric objects rather than expressions, and by adding various heuristic strategies, both exact and approximate, to guide the fixpoint computation of the target region.[1] To address problem A, we approach the target region simultaneously from several directions, so that success is guaranteed even if only one direction terminates (Section 3.1). To address problem B, we represent regions as unions of convex polyhedra, so that quantifier elimination is replaced by linear operations on polyhedra. Instead of computing in a single d-dimensional space, it is usually profitable to work with k n-dimensional spaces (Section 3.2). To address problem C, we approximate the union of several convex polyhedra by a single convex polyhedron. We discuss two approximation operators—convex-hull approximation (Section 4.1) and extrapolation (Section 4.2). Each approximation can be iteratively refined to achieve the desired precision (Section 4.3) [DW93].

We wish to point out that the fixpoint computation by iteration and the approximation strategies of Section 4 are but the old technique of *abstract interpretation* [CC77] applied to a new domain—that of linear hybrid systems. Convex-hull approximation [CH78] and widening [CC77], which is related to our extrapolation operation, has been used for the static analysis of sequential and synchronous programs [Hal93], i.e., for the analysis of *discrete* state spaces. We feel that both

[1] The new version of HyTech, including tactics, is available by anonymous ftp from ftp.cs.cornell.edu, directory pub/tah/HyTech. As far as we know, HyTech is the only implementation of model checking for the full class of linear hybrid systems.

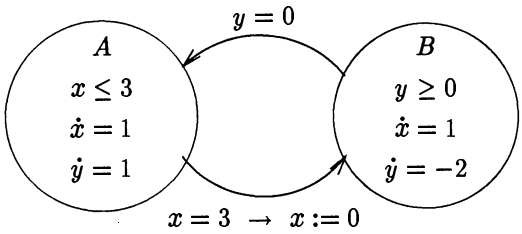

Figure 1: The water-tank automaton

approximations are particularly suited for the analysis of the *continuous* and *linear* state spaces, because a polyhedron more naturally represents all of its interior points rather than just the grid of interior integer points, and because the linear regions are closed under both approximation operations. This feeling is substantiated by the experimental results we report in Section 3.2, where polyhedra approximate continuous regions more efficiently than discrete regions.

This abstract presents only *over*approximation operators. In the full paper we also discuss the sound use of *under*approximation operators, and the iterative application of alternating over- and underapproximation for linear hybrid systems.

2 Linear Hybrid Automata

We model systems as hybrid automata [ACHH93]. A (*linear*) *hybrid automaton* $\mathcal{A}$ consists of a finite set X of real-valued variables and a labeled graph (V, E). The edges E represent discrete system transitions and are labeled with guarded commands on X; the vertices V represent continuous environment activities and are labeled with differential equations and constraints on X. A *state* (v, s) of the hybrid automaton $\mathcal{A}$ consists of a control state $v \in V$ (a vertex) and a data state s: $X \to \mathbb{R}$ (a function that assigns values to all variables). A set of (data) states is called a (*data*) *region*. A *run* of $\mathcal{A}$ is a function from the nonnegative real line $\mathbb{R}^+$ to states. The reachability problem for $\mathcal{A}$ asks, given an *initial region* S and a *final region* T, if there is a run of $\mathcal{A}$ that leads from an initial state in S to a final state in T. If T represents the set of "unsafe" states of a safety requirement, then the requirement can be checked by reachability analysis.

Example: water tank

The hybrid automaton of Figure 1 models a water-level controller that opens and shuts the outflow of a water tank. The automaton has two variables (data), x and y, and two vertices (control), A and B. The variable x represents a clock of the water-level controller and the variable y represents the water level in the tank. Since the clock x measures time, the first derivative of x is always 1 (i.e., $\dot{x} = 1$). In vertex A, the outflow of the water tank is shut and the water level increases 1 inch per sec ($\dot{y} = 1$); in vertex B, the outflow of the water tank is open and the water level decreases 2 inches per sec ($\dot{y} = -2$). The transition from B to A (shut the outflow) is taken as soon as the water tank becomes empty: the guard $y = 0$ on the transition ensures that the transition may be taken only when the water level is 0; the constraint $y \geq 0$ on B ensures that the transition to A must be taken before the water level becomes negative. The transition from A to B (open the

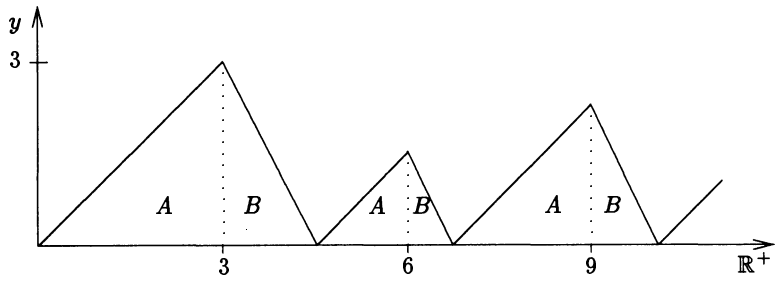

Figure 2: A run of the water-tank automaton

outflow) is taken every 3 sec: the transition is taken whenever $x = 3$, and the transition restarts the clock x at 0. If the automaton is started from the state $A \wedge x = y = 0$,[2] then Figure 2 shows how the water level y changes as a piecewise-linear function of time.

The symbolic model checker HyTech

In this paper, we study the performance of HyTech, a fully automatic model checker for hybrid automata [AHH93], on reachability problems. HyTech is implemented within Mathematica as a collection of procedures that manipulate *linear regions*: if n is the number of variables, then a linear data region is the union of convex polyhedra in n-dimensional space; a linear region $R = \cup_{v \in V}(v, R[v])$ is a collection of linear data regions $R[v]$, one for each control state v. While the original implementation of HyTech represented each region as a boolean combination of linear constraints, the new version of HyTech represents each convex polyhedron by its extreme points (corners) and extreme rays (generators). The polyhedral operations are implemented as C++ subroutines that call on a polyhedral library [Hal93]. The algorithms and implementation details will be given in the full paper. The speedup achieved by the geometric algorithms over the original (pure Mathematica) version of HyTech is about a factor of 4.

3 Exact Verification Strategies

We discuss some degrees of freedom offered by the HyTech verifier and show how a particular choice of parameters may influence its performance on a given reachability problem.

3.1 Forward versus Backward Analysis

Suppose that we wish to check if the final region T is reachable from the initial region S. There are two possible approaches for solving the reachability problem by symbolic model checking [ACHH93]: we may compute the target region $post^*(S)$ of states that can be reached from the initial region S and check if $post^*(S) \cap T = \emptyset$ (forward reachability analysis), or we may compute the target region $pre^*(T)$ of states from which the final region T can be reached and check if $pre^*(T) \cap S = \emptyset$ (backward reachability analysis). For any given reachability problem, one approach may perform

[2]We define states and regions by formulas. For example, the formula $A \wedge x > 1$ defines the set of all states whose control component is A and whose data component assigns to the variable x a value greater than 1.

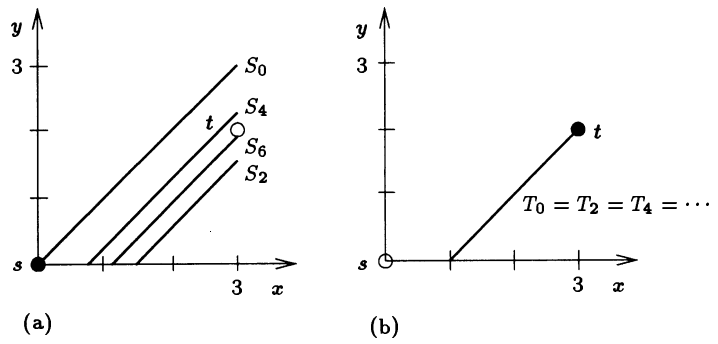

Figure 3: Exact forward and backward reachability

better than the other approach; indeed, it may be that one approach terminates and the other does not.

Recall, for example, the water-tank automaton of Figure 1. We wish to check if the final state $t = (A \wedge x = 3 \wedge y = 2)$ can be reached from the initial state $s = (A \wedge x = y = 0)$. The symbolic model-checking procedure computes the two regions $post^*(s)$ and $pre^*(t)$ as the limits of two sequences of regions. Let $S_{i+1} = post(S_i)$ be the region of states that can be reached from s by $i+1$ discrete transitions, and let $T_{i+1} = pre(T_i)$ be the region of states that can reach t by $i+1$ discrete transitions. Then $post^*(s)$ is the limit $\cup_{i \geq 0} S_i$, and $pre^*(t)$ is the limit $\cup_{i \geq 0} T_i$. Since

$$S_{2i} = (A \wedge y + 1 + \frac{(-1)^i}{2i} = x \wedge x \leq 3)$$

(Figure 3(a)) and

$$S_{2i+1} = (B \wedge 2x + y = 2 + \frac{(-1)^i}{2i}),$$

the forward computation of $post^*(s)$ does not terminate within any finite number of iterations. The backward computation, however, converges in a single iteration with the result

$$T_0 = T_2 = (A \wedge 1 \leq x \leq 3 \wedge x = y + 1)$$

(Figure 3(b)) and

$$T_1 = T_3 = (B \wedge x + 2y = 2).$$

It follows that

$$pre^*(t) = (A \wedge 1 \leq x \leq 3 \wedge x = y + 1) \vee (B \wedge x + 2y = 2).$$

Since $s \notin pre^*(t)$, we conclude that the final state t cannot be reached from the initial state s. For optimal performance, we have implemented a strategy that dovetails both approaches by computing the alternating sequence $S_0, T_0, S_1, T_1, S_2, \ldots$ of regions.

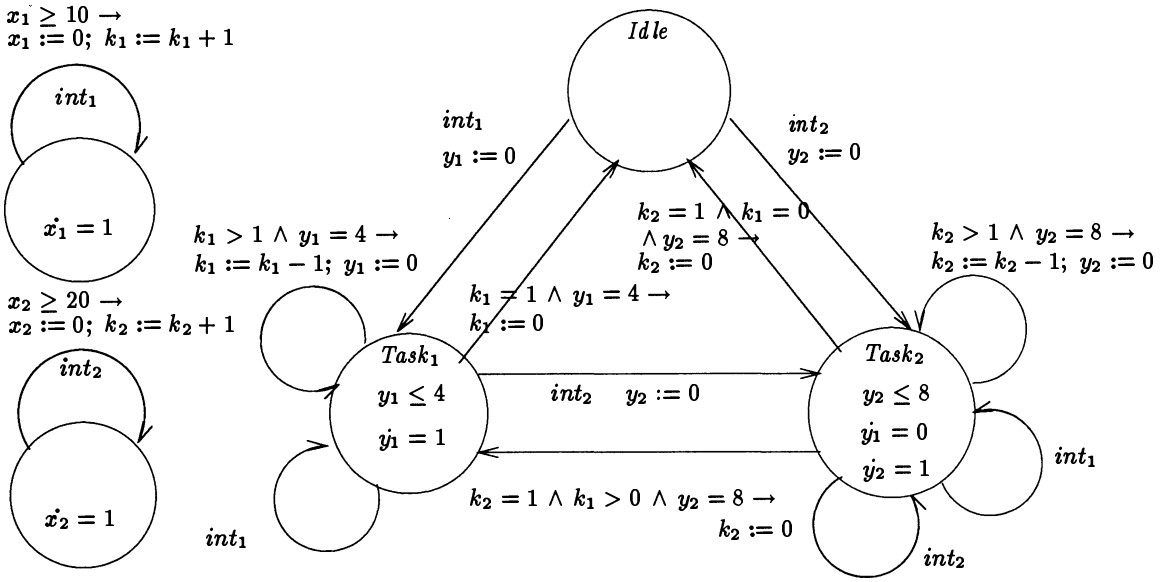

Figure 4: The priority scheduler, data version

3.2 Data versus Control

Data variables that range over a finite domain of values can be encoded within the control of a hybrid automaton. Our experience has been that HyTech performs better if all finite-state variables are encoded in the control. We illustrate this observation with two examples taken from scheduling—a priority scheduler and a round-robin scheduler. Schedulers are examples of hybrid systems with stop watches—variables whose slope is sometimes 0 and sometimes 1—which measure the accumulated amount of execution time for each task.

Example: priority scheduler

The two small hybrid automata on the left of Figure 4 model an environment that generates two types of interrupts: an interrupt of type int_1 arrives at most once every 10 sec; an interrupt of type int_2, at most once every 20 sec. For every int_i interrupt, a task of type i needs to be executed: each type-1 task requires 4 sec; each type-2 task, 8 sec. Moreover, only one processor is available for the execution of all tasks, and type-2 tasks have priority over type-1 tasks and interrupt their execution. The resulting priority scheduler is modeled by the hybrid automaton on the right of Figure 4. In vertex $Task_1$, a type-1 task is being executed; in vertex $Task_2$, a type-2 task. The variable k_i represents the number of incomplete (i.e., running and pending) tasks of type i; the variable y_i represents the execution time of the current task of type i. The three automata synchronize on the edge labels int_1 and int_2; that is, whenever an int_i interrupt arrives, the scheduler takes a transition that is labeled with int_i. The entire scheduling system, then, is the product of the three component automata [ACHH93]. Using HyTech, we verified that in any run starting from the region $Idle \wedge k_1 = k_2 = 0$ the number of incomplete type-1 tasks never exceeds 2 and the number of incomplete type-2 tasks never exceeds 1; that is, no state that satisfies $k_1 > 2$ or $k_2 > 1$ is reachable.

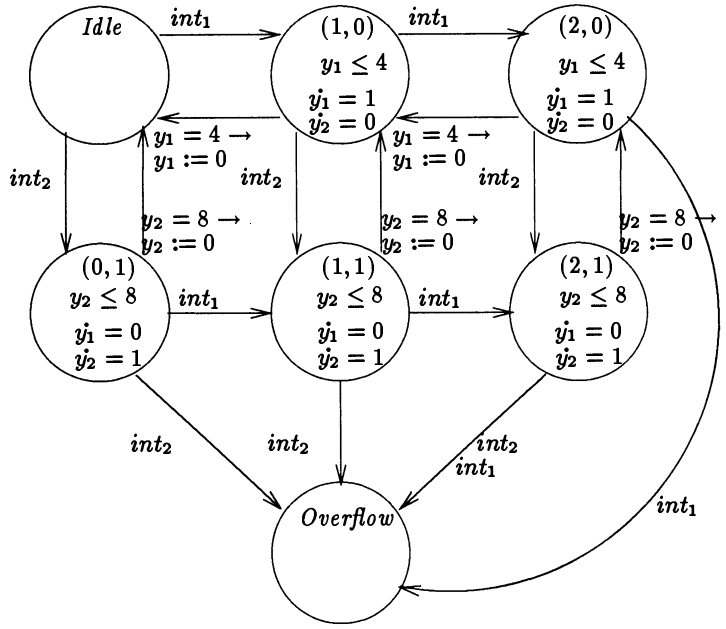

Figure 5: The priority scheduler, control version

Since the variables k_1 and k_2 range over finite domains, they can be encoded in the control of the scheduler. The control version of the priority scheduler (Figure 5), then, contains a vertex (k_1, k_2) for all possible values of the variables k_1 (namely, 0, 1, and 2) and k_2 (0 and 1). The vertex *Overflow* represents any state with $k_1 > 2$ or $k_2 > 1$. We checked that the region *Overflow* cannot be reached from the region *Idle* $= (0,0)$. We also increased the number of interrupt and task types and the results of these experiments, which were performed on a Sparc 670MP station, are shown in Figure 6. Clearly, our verifier handles finite-state control more efficiently than real-valued data.

Example: round-robin scheduler

Figure 7 shows a round-robin scheduler. The two task types are assigned alternating time slices of 2 sec, as measured by the clock z. If all tasks of one type are completed, the scheduler starts a new time slice for the tasks of the other type. Again we check that the number of incomplete type-1 tasks is bounded by 2 and the number of incomplete type-2 tasks is bounded by 1, which allows us to encode the variables k_1 and k_2 in the control of the scheduler. The verification results are also shown in Figure 6.

4 Approximate Verification Strategies

Sometimes the exact computation of a target region is prohibitively expensive or does not terminate, independent of both the system description (data vs. control) and the verification strategy (forward vs. backward). In such cases, we approximate the target region and iteratively refine the approximation until sufficient precision is obtained. We discuss two approximation operators—convex-hull approximation and extrapolation—which can be turned on or off by the user of HyTech.

Priority scheduler, 2 tasks, data version	590 sec
Priority scheduler, 2 tasks, control version	57 sec
Priority scheduler, 3 tasks, data version	33,326 sec
Priority scheduler, 3 tasks, control version	686 sec
Priority scheduler, 4 tasks, data version	out of memory
Priority scheduler, 4 tasks, control version	49,024 sec
Round-robin scheduler, 2 tasks, data version	1,965 sec
Round-robin scheduler, 2 tasks, control version	592 sec

Figure 6: Performance data (CPU time)

Both operators overapproximate the union of convex polyhedra by a single convex polyhedron and thus reduce the time and space requirements of the verifier; indeed, either operator, or the combination of both operators, may cause the termination of an otherwise infinite computation. If we overapproximate, by $\hat{S}$, the target region $post^*(S)$ of states that are reachable from the initial region S (i.e., $post^*(S) \subseteq \hat{S}$), and $\hat{S}$ contains no final state from T, then we can conclude that T is not reachable from S; if, on the other hand, $\hat{S}$ does contain a final state, then we cannot reach a valid conclusion and must refine the approximation.

We acknowledge the work of Nicolas Halbwachs, who has used a convex-hull operator and a so-called widening operator, which is similar in spirit to our extrapolation operator, for the analysis of synchronous programs [Hal93][3]; also, the current implementation of HyTech makes use of Halbwachs's polyhedral library. The idea of iteratively refining approximations originated with David Dill and Howard Wong-Toi for the analysis of real-time systems [DW93].

4.1 Convex-hull Approximation

An *approximation operator* $+$ is a binary operator on data regions such that for two data regions R and R', (1) $R \cup R' \subseteq R + R'$, (2) $R + R'$ is convex, and (3) if R and R' are linear, then so is $R + R'$. Every approximation operator extends to regions as follows: for two regions $R = \cup_{v \in V}(v, R[v])$ and $R' = \cup_{v \in V}(v, R'[v])$, $R + R' = \cup_{v \in V}(v, R[v] + R'[v])$. Clearly, the *convex-hull operator* $\&$, which maps two data regions R and R' to the convex hull of the union $R \cup R'$, is an approximation operator.

Recall once again the water-tank automaton of Figure 1. Now we wish to check if the final state $t' = (A \wedge x = \frac{1}{2} \wedge y = 0)$ can be reached from the initial state $s = (A \wedge x = y = 0)$. Again, the forward computation of the target region $post^*(s)$ does not terminate. This time, however, we can make the forward approach work with the help of convex-hull approximation. If we naively take the convex hull $\hat{S}_2$ of the two regions S_0 and S_2 (Figure 8(a)), then the forward computation terminates with the overapproximation $\hat{S}_2$ of the A-component $post^*(s)[A]$ of the target region, because $post^2(\hat{S}_2) = \hat{S}_2$. Since $\hat{S}_0$ contains t', however, we cannot conclude that t' is or is not reachable from s. Thus we apply the convex-hull operator only if the resulting region does not contain any final states;[4] that is, we approximate the target region $post^*(S) = \cup_{i \geq 0} S_i$ by the

[3] Halbwachs' widening operator causes a coarser approximation than extrapolation but, unlike extrapolation, guarantees the convergence of iterative application.

[4] This strategy is sensible for all approximation operators.

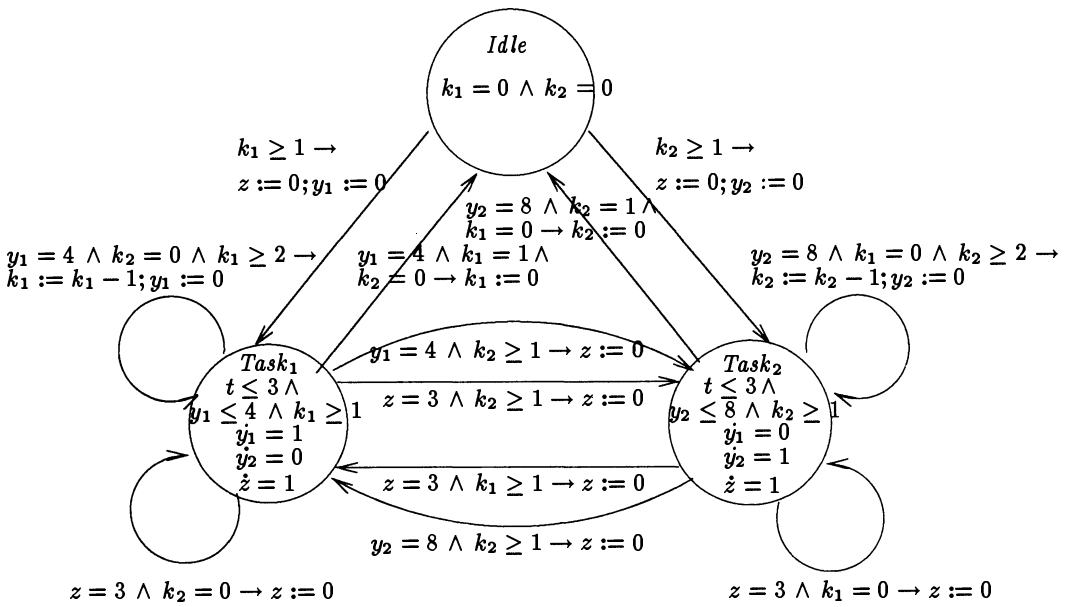

Figure 7: The round-robin scheduler

limit $\hat{S} = \cup_{i \geq 0} \hat{S}_i$, where

$$\hat{S}_{i+1}[v] = \begin{cases} post(\hat{S}_i)[v] \,\&\, \hat{S}_i[v] & \text{if } (post(\hat{S}_i)[v] \,\&\, \hat{S}_i[v]) \cap T = \emptyset, \\ post(\hat{S}_i)[v] & \text{otherwise.} \end{cases}$$

For the water-tank example, this strategy, then, computes $\hat{S}_0 = S_0$, $\hat{S}_2 = S_2$, $\hat{S}_4 = S_4 \& S_2$, and $\hat{S}_6 = \hat{S}_4$ (Figure 8(b)). So we obtain the limit $S_0 \cup \hat{S}_4$ as an overapproximation of the A-component $post^*(s)[A]$ of the target region. Since $S_0 \cup \hat{S}_4$ does not contain t', we conclude that the final state t' is not reachable from the initial state s.

4.2 Extrapolation

Convex-hull approximation may be found useful for systems with variables whose values are bounded, such as the water-tank automaton. By contrast, if the regions in the sequence $\cup_{i \geq 0} S_i$ grow without bound, we may want an extrapolation operator that "guesses" an overapproximation of the limit $\cup_{i \geq 0} S_i$. Consider, for example, the hybrid automaton of Figure 9, which models the movement of a robot in the (x, y)-plane. The robot can be in one of two modes—heading roughly northeast (vertex A) or heading roughly southeast (vertex B): in mode A, the derivatives of the coordinates x and y vary within the closed interval $[1, 2]$; in mode B, the derivative of x remains between 1 and 2, while the derivative of y changes its sign and varies between -1 and -2. The robot changes its mode every 1 to 2 min, according to its clock z. Moreover, there is a wall at the $x = 0$ line, causing the system invariant $x \geq 0$. Figure 10 shows how the position of the robot in the (x, y)-plane may change with time, assuming the robot is started in the initial state $s = (A \wedge x = y = z = 0)$.

We wish to check if the robot can reach, from s, the final position $T = (x = 9 \wedge y = 12)$. At this point, we invite the ambitious reader to prove that T cannot be reached from s. Using HyTech,

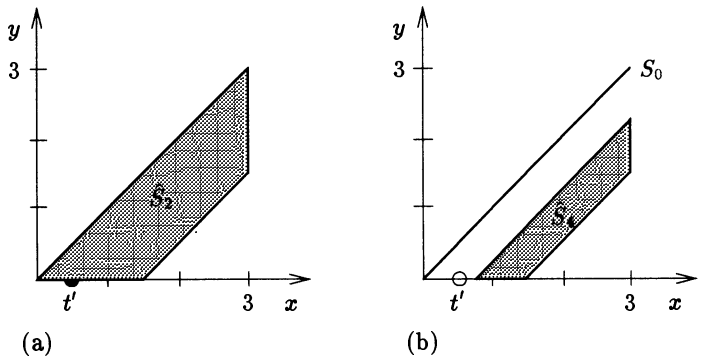

Figure 8: Approximate forward reachability

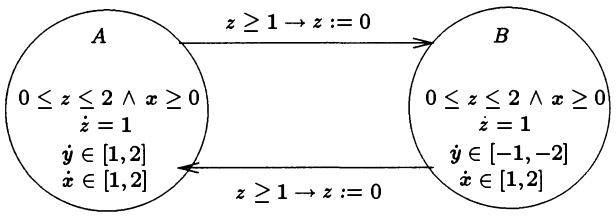

Figure 9: The robot automaton

the backward computation of the region $pre^*(T)$ terminates within 74 sec of CPU time after seven iterations, and $s \notin pre^*(T)$. The forward computation of the region $post^*(s)$, by contrast, does not terminate, because the robot keeps zig-zagging to the east and the limit $\cup_{i \ge 0} post^i(s)$ cannot be computed exactly.[5] As convex-hull approximation does not help with this example, we define an extrapolation operator that approximates the unbounded target region $post^*(s)$; using HyTech, the forward computation with extrapolation, then, terminates within 8 sec of CPU time.

We first give an intuitive motivation for the extrapolation operator. Suppose that the iterative computation of the target region leads, for a given control state, to the data region (polyhedron) R and, in the subsequent iteration, to the polyhedron R'. Suppose, furthermore, that there is a function f such that f maps R to R', mapping extreme points to extreme points. A reasonable guess for the target region, then, would be $f^\infty(R)$.

To formally define an operator $\propto$ that extrapolates R and $f(R)$ to $f^\infty(R)$, we need to review some facts about convex polyhedra [NW88]. By Minkowski's theorem, every nonempty convex polyhedron R in $\mathbb{R}^n$ has a unique (within scalar multiplication) representation by its extreme

[5] True, if the target region T is reachable, then it can be reached within a finite number of discrete transitions, and if not, then the invariant $x > 9$ becomes true after a finite number of transitions; extrapolation, however, avoids the ad-hoc "guessing" of suitable invariants.

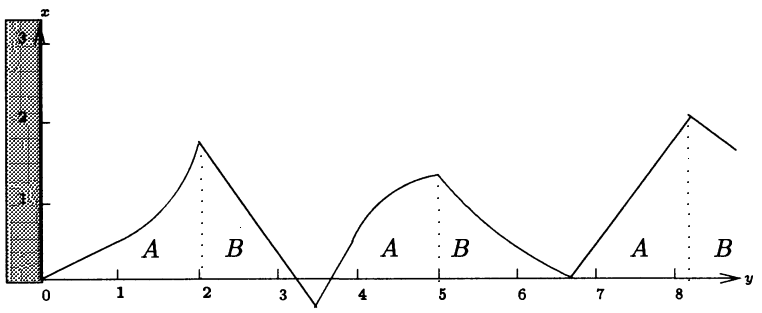

Figure 10: A run of the robot automaton

points $\{x_i \mid i \in I\}$ and extreme rays $\{r_j \mid j \in J\}$ such that

$$R = \{x \in \mathbb{R}^n \mid x = \sum_{i \in I} \lambda_i x_i + \sum_{j \in J} \mu_j r_j, \text{ where } \lambda_i, \mu_j \in \mathbb{R}^+ \text{ and } \sum_{i \in I} \lambda_i = 1\}.$$

Notice that if the point x and the ray r are in R, then so are all points of the form $x + \lambda r$, for $\lambda \in \mathbb{R}^+$. The nonempty convex polyhedron R has *dimension* k, denoted $dim(R) = k$, if there are $k+1$ points $x_1, x_2, \ldots, x_{k+1}$ in R such that the k differences $x_2 - x_1, x_3 - x_1, \ldots, x_{k+1} - x_1$ are linearly independent. Notice that if $dim(R) = 0$, then R is a point, and if $dim(R) = 1$, then R is a straight line, a ray, or a line segment. The set $F = \{x \in R \mid ax = b\}$, for two vectors a and b, is a *face* of R if every point x in R satisfies the inequality $ax \leq b$. Notice that every face of R is also a convex polyhedron. The face F is a *facet* of R if $dim(F) = dim(R) - 1$. The set F is a 1-dimensional face of R iff F is the intersection of $dim(R) - 1$ facets of R (†).

Now consider two polyhedra R and R'. We first compute the convex hull $R\&R'$ of $R \cup R'$. All extreme points and extreme rays of the convex hull $R\&R'$ are extreme points and extreme rays of R or R'. Let $\{x_k \mid k \in I\}$ be the extreme points of $R\&R'$ and R, let $\{x_l \mid l \in I'\}$ be the remaining extreme points of $R\&R'$, and let $\{r_j \mid j \in J\}$ be the extreme rays of $R\&R'$. Suppose that $k \in I$ and $l \in I'$, and x_k and x_l are the endpoints of a 1-dimensional face of $R\&R'$, which can be checked by Criterion (†). Then it is possible that $f(x_k) = x_l$ for the imaginary function f from R to R'. Moreover, if f is applied infinitely many times, then all points of the form $x_k + \lambda(x_l - x_k)$, for $\lambda \in \mathbb{R}^+$, are included in $f^\infty(R)$. We therefore add the ray $x_l - x_k$ into the generators of $R \propto R'$: the *extrapolation operator* $\propto$ maps the two polyhedra R and R' to $R \propto R' =$

$$\{x \in \mathbb{R}_n \mid x = \sum_{i \in I \cup I'} \lambda_i x_i + \sum_{j \in J} \mu_j r_j + \sum_{(k,l) \in K} \mu_{k,l}(x_l - x_k), \text{ where } \lambda_i, \mu_j, \mu_{k,l} \in \mathbb{R}^+ \text{ and } \sum_{i \in I \cup I'} \lambda_i = 1\},$$

where $(k,l) \in K$ iff $k \in I$ and $l \in I'$ and both x_k and x_l are in the intersection of $dim(R\&R') - 1$ facets of $R\&R'$. Notice that $\propto$ is an approximation operator, that $\propto$ is not symmetric (i.e., $R \propto R'$ and $R' \propto R$ may be different), and that $R\&R' \subseteq R \propto R'$. Figure 11 shows two convex 2-dimensional polyhedra R and R' and the extrapolation result $R \propto R'$.

4.3 Two-way Iterative Approximation

Suppose that an approximate forward reachability analysis is inconclusive; that is, the overapproximation $\hat{S}$ of the target region $post^*(S)$ contains some final states from T. If the intersection $\hat{S} \cap T$

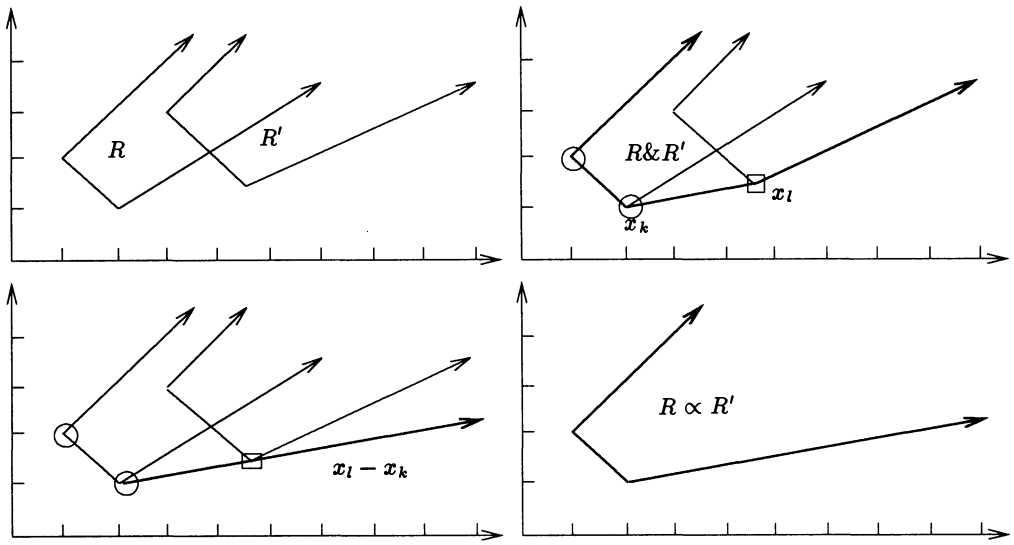

Figure 11: Extrapolation

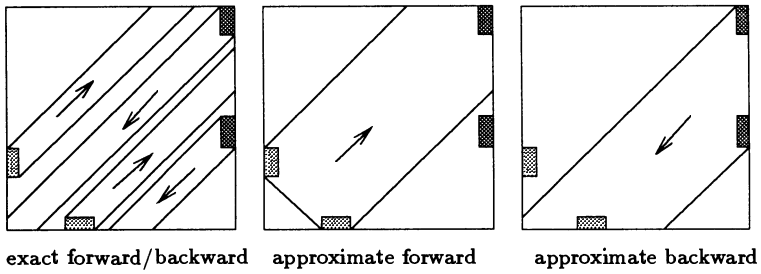

Figure 12: Single-pass exact and approximate analysis

is a proper subset of T, then we have nonetheless obtained new information—namely, that the states in $T - \hat{S}$ are not reachable from S—and we may proceed computing backward the new target region $pre^*(T - \hat{S})$, or an overapproximation $\hat{T}$ of $pre^*(T - \hat{S})$. If $\hat{T} \cap S = \emptyset$, then T is not reachable from S; if, on the other hand, $\hat{T}$ does contain some initial states from S, then we may continue the two-way iterative approximation process, this time computing forward from $\hat{T} \cap S$ [DW93].

The two-way iterative approximation strategy is illustrated in Figures 12 and 13, assuming a 2-dimensional state space and convex-hull approximation. The two shaded boxes in the lower left corner of the state space represent the initial region S; the two shaded boxes in the upper right corner represent the final region T. The first derivatives of both variables x and y are 1. While an exact forward or backward analysis shows that T cannot be reached from S, both approximate forward analysis and approximate backward analysis are inconclusive (Figure 12). An approximate forward analysis followed by an approximate backward analysis is successful (Figure 13).

We now apply the two-way iterative approximation strategy to analyze the bouncing-ball automaton

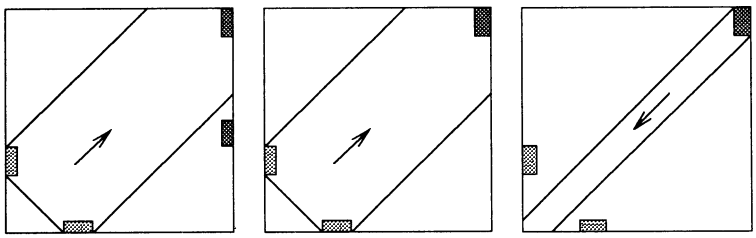

Figure 13: Two-pass approximative analysis

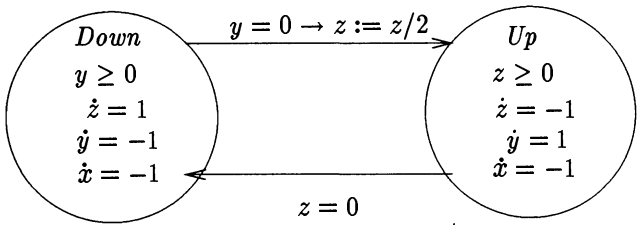

Figure 14: The bouncing-ball automaton

of Figure 14. The variable x represents the horizontal distance of the ball from a reference point, the variable y represents the distance of the ball from the floor, and the variable z represents the "energy" of the ball. Suppose that the ball is dropped at the position $x = 14 \wedge y = 4$ in the direction of the reference point; that is, $s = (\mathit{Down} \wedge x = 14 \wedge y = 4 \wedge z = 0)$. While the ball is falling (vertex *Down*), its energy is increasing ($\dot{z} = 1$); when the ball hits the floor ($y = 0$), it loses half of its energy and bounces back up (vertex *Up*); on the way up, the energy decreases ($\dot{z} = -1$) until it becomes 0 and the ball starts to fall again. The trajectory of the ball is shown in Figure 15.

We wish to prove that the ball never reaches the region $T = (x \leq 2 \wedge y = 0)$ of the floor. First notice that the exact forward computation of the target region $\mathit{post}^*(s)$ does not terminate, and convex-hull approximation is of no help. If we use extrapolation, then we obtain the overapproximation $\hat{S}$ of the target region $\mathit{post}^*(s)$, and $\hat{S}$ contains the final states $\hat{S} \cap T = (x = 2 \wedge y = 0 \wedge z = 0)$. Since $\mathit{pre}(\hat{S} \cap T) = \hat{S} \cap T$, a second, backward, pass terminates immediately without reaching the initial state s. HyTech requires 12 sec CPU time for both passes. (Notice that if we were to begin with a backward pass, attempting to compute the target region $\mathit{pre}^*(T)$, neither the exact computation, nor the approximate computation with convex-hull approximation, nor the approximate computation with extrapolation would terminate.)

Acknowledgments. We thank Rajeev Alur, Limor Fix, and Nicolas Halbwachs for many stimulating discussions, and Nicolas for his polyhedral library.

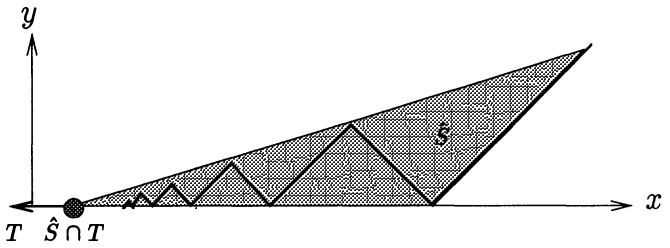

Figure 15: A run of the bouncing-ball automaton

References

[ACH+94] R. Alur, C. Coucoubetis, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. In *Analysis and Optimization of Discrete Event Systems*, Lecture Notes in Control and Information Sciences 199, pages 331–351. Springer-Verlag, 1994.

[ACHH93] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 209–229. Springer-Verlag, 1993.

[AHH93] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the 14th Annual Real-time Systems Symposium*, pages 2–11. IEEE Computer Society Press, 1993.

[BCM+92] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[CC77] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for the static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the Fourth Annual Symposium on Principles of Programming Languages*. ACM Press, 1977.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[CH78] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the Fifth Annual Symposium on Principles of Programming Languages*. ACM Press, 1978.

[DW93] D.L. Dill and H. Wong-Toi. Using iterative symbolic approximation for timing verification. In T. Rus, editor, *Proceedings of the First AMAST Workshop on Real-time Systems*, 1993.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In C. Courcoubetis, editor, *CAV 93: Computer-aided Verification*, Lecture Notes in Computer Science 697, pages 333–346. Springer-Verlag, 1993.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[HRP94] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximation. Unpublished manuscript, 1994.

[MMP92] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de Bakker, K. Huizing, W.-P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 447–484. Springer-Verlag, 1992.

[NOSY93] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 149–178. Springer-Verlag, 1993.

[NW88] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, 1988.