

# A New Linear Hybrid Automata Reachability Procedure

– Draft of March 23, 2001 –

Steve Vestal  
steve\_vestal@htc.honeywell.com  
Honeywell Technology Center  
Minneapolis, MN 55418\*

## Abstract

*We present a new on-the-fly method for reachability analysis of linear hybrid automata with constant rates. The novelty of our methods lie in the algorithms used to manipulate polyhedra, and in our encoding of domain-specific behavior (real-time scheduling in our case) into the model semantics. Our polyhedra operations can be performed in polynomial time, typically quadratic in the number of continuous variables in a model. Encoding scheduling into the model semantics allows us to significantly reduce the size of the overall system discrete state space. We benchmarked a prototype of our method, HyTech, and Verus using a randomly generated set of classical real-time uniprocessor workloads. We also experimented with two optimization methods, a simplification of model parameters using results from real-time scheduling theory, and a simple form of partial order reduction. When we started our work using HyTech we were able to consistently analyze workloads having 4 concurrent tasks (81 reachable discrete states), using our prototype together with these optimization methods we were able to consistently analyze workloads having 13 concurrent tasks (8192 reachable discrete states).*

## 1 Introduction

Linear hybrid automata are finite state automata augmented with variables whose values change continuously in a way that depends on the current discrete state. The values of the continuous variables can affect, and can be affected by, discrete transitions between discrete states. Linear hybrid automata can be subjected to a reachability analysis to verify that a given set of assertions is true of a system. In general, a semi-decision procedure must be used since the reachability problem for linear hybrid automata is undecidable[14] (as opposed to timed automata, which are decidable[3]). It can be shown, however, that reasonable pragmatic restrictions make models for real-

time allocation and scheduling in distributed heterogeneous systems decidable[23].

Linear hybrid automata can be used to model many kinds of dynamical systems, but the problem of particular interest to us is the modeling and schedulability analysis of real-time systems. The continuous variables of a linear hybrid automaton can be used as timers to control task dispatching and detect missed deadlines, and as so-called integration variables to record accumulated task compute time. Linear hybrid automata are sufficiently powerful to model a number of interesting system features, such as remote procedure calls, rendezvous between tasks, variations in compute time as a function of internal task state, and distributed synchronization and communication protocols. Reasonably detailed models of source code can be written, and linear hybrid automata are also useful for verifying implementations of such things as time-dependent protocols and scheduling kernels[24].

The reachable state space for a linear hybrid automaton is a set of regions, where each region consists of a discrete state plus a polyhedron that defines a set of possible values for the continuous variables. We present a reachability procedure that represents polyhedra as systems of linear inequalities. We present new algorithms for computing the polyhedron that results when time is allowed to pass and variable values change at specified rates; and the polyhedron that results when a variable is unconstrained and removed from the system. These algorithms might be viewed as generalizations of the difference methods used for timed automata[8, 3] and exhibit a vague similarity to the pragmatic algorithm used earlier for quantifier elimination[2]. We present an algorithm to decide if one polyhedron is contained in another. We present a reduction algorithm to simplify the set of constraints that represent a polyhedron. Our prototype is an on-the-fly tool that enumerates regions as they are encountered, rather than first enumerating the complete reachable discrete state space and then enumerating the reachable polyhedra for each discrete

---

\*This work has been supported by the Air Force Office of Scientific Research under contract F49620-97-C-0008.

state. However, our procedure does not provide parametric analysis[15].

We also discuss different ways that real-time scheduling can be incorporated into a linear hybrid automata model, including a novel way of adding scheduling semantics to create an extended resourceful linear hybrid automata model. Extending the semantics of the model rather than trying to write a standard linear hybrid automata model of a scheduling protocol both reduces the size of the region space and allows a much broader range of scheduling protocols to be modeled.

We randomly generated a sequence of uniprocessor workloads consisting of periodic and aperiodic tasks scheduled using preemptive fixed priority. For each of these we generated a linear hybrid automata model whose assertions were satisfied only for schedulable task sets. These models were analyzed using a prototype implementation of our procedure. All these task sets were amenable to analysis using the exact characterization algorithm[18], which we used to double-check our results. We submitted the same models to HyTech[15], a linear hybrid automata tool; and to Verus[7], a discrete timed automata tool.

We also experimented with two optimization methods. First, traditional uniprocessor preemptive priority scheduling theory says that we can replace execution time and event inter-arrival intervals with their worst-case values. Second, we experimented with a simple partial order reduction method.

The earliest reachability tool of which we are aware, HyTech, represented polyhedra as finite sets of linear constraints[2]. The operations performed on these polyhedra used quantifier elimination, a formal way to algebraically manipulate and make decisions about systems of linear inequalities in which some of the variables are existentially quantified. Polka and a later version of HyTech used a pair of representations, the traditional system of linear inequalities together with polyhedra generators consisting of sets of vertices and rays[12, 15]. Different operations required during reachability are more convenient in the different representations, and methods are used to convert between the two as needed. These previous methods are subject to the theoretical risk that some polyhedra operations may require a combinatorial amount of time, although we did not test for this in our experiments. Our polyhedra operations are all doable in polynomial time (although we used the Simplex algorithm in our prototype), typically quadratic in the number of constraints used to represent a polyhedron.

A variety of differences between the tools and certain aspects of our use of them make direct comparisons questionable, and we experimented only with a particular class of problem. Keeping these caveats in mind,

we were able to solve problems with our prototype tool an order of magnitude more quickly than with HyTech, which was perhaps three orders of magnitude faster than Verus without automatic variable reordering. Perhaps as importantly, our prototype tool was more numerically robust and used significantly less memory, it never failed due to numeric overflow or memory exhaustion. We were able to solve problems that HyTech and Verus could not solve. When we began our work using HyTech we were able to consistently solve systems of 4 tasks having 81 reachable discrete system states. Using our prototype tool together with some experimental optimization methods, we were able to consistently solve systems of 13 tasks having 8192 reachable discrete states. In our judgement this is not yet adequate for schedulability analysis but is at the threshold of utility for simple but practical verification problems[24]. Our work suggests that future improvements could result in further significant increases in the size of solvable problem, and we discuss this in our concluding section.

## 2 Resourceful Hybrid Automata

A hybrid automaton is a finite state machine augmented with a set of real-valued variables and a set of propositions about the values of those variables. Figure 1 shows an example of a hybrid automaton whose discrete states are **preempted**, **executing** and **waiting**; and whose real-valued variables are  $c$  and  $t$ . **Waiting** is marked as the initial discrete state, and  $c$  and  $t$  are assumed to be initially zero.

Each of the discrete states has an associated set of differential equations, e.g.  $\dot{c} = 0$  and  $\dot{t} = 1$  for the discrete state **preempted**. While the automaton is in a discrete state, the continuous variables change at the rates specified for that state.

Edges may be labeled with guards involving continuous variables, and a discrete transition can only occur when the values of the continuous variables satisfy the guard. When a discrete transition does occur, designated continuous variables can be set to designated values as specified by assignments labeling that edge.

A discrete state may also be annotated with an invariant constraint to assure progress. Some discrete transition must be taken from a state before that state's invariant becomes false. For example, the hybrid automaton in Figure 1 must transition out of state **computing** before the value of  $c$  exceeds 100.

The hybrid automata of interest to us are called linear hybrid automata because the invariants, guards and assignments are all expressed as sets of linear constraints. The differential equations governing the continuous dynamics in a particular linear hybrid automaton discrete state are restricted to the form  $\dot{x} \in [l, u]$  where  $[l, u]$  is a fixed constant interval.

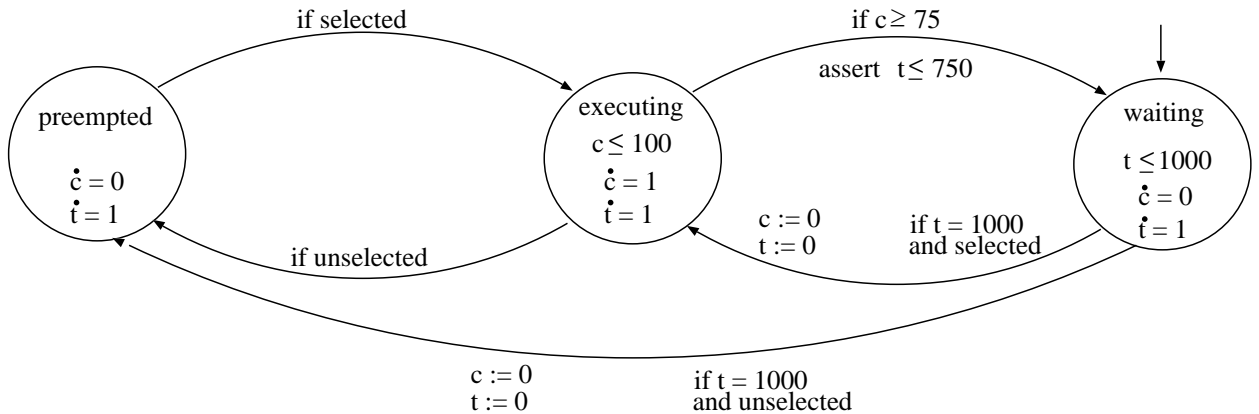


Figure 1: A Hybrid Automata Model of a Preemptively Scheduled Task

We want to verify assertions about the behavior of a hybrid automaton. Although it is possible in general to check temporal logic assertions[2], we make do by annotating discrete states and edges with sets of linear constraints labeled as assertions. These constraints must be true whenever the system is in a discrete state or whenever a transition occurs over an edge.

The cross-product construction used to compose concurrent finite state processes can be extended in a fairly straight-forward way to systems of hybrid automata. The invariant and assertion associated with a discrete system state are the conjunction of the invariants and assertions of the individual discrete states. The guards, assertions and assignments of synchronized transitions are the conjunction and union of the guards, assertions and assignments of the individual discrete co-edges. If there is a conflict between the rate assignments of individual discrete states, or a conflict between the variable assignments of co-edges, then the system is considered ill-formed. Note that concurrent hybrid automata may interact through shared real-valued variables as well as by synchronizing their transitions over co-edges.

The application of interest in this paper is the analysis and verification of real-time systems. Figure 1 shows an example of a simple hybrid automata model for a preemptively scheduled, periodically dispatched task. A task is initially waiting for dispatch but may at various times also be executing or preempted. The variable  $t$  is used as a timer to control dispatching and to measure deadlines. The variable  $t$  is set to 0 at each dispatch (each transition out of the waiting state), and a subsequent dispatch will occur when  $t$  reaches 1000. The assertion  $t \leq 750$  each time a task transitions from executing to waiting (each time a task completes) models a task deadline of 750 time units. The variable  $c$  records accumulated compute time, it is reset at each dispatch and increases only when the task is in the com-

puting state. The invariant  $c \leq 100$  in the computing state means the task must complete before it receives more than 100 time units of processor service, the guard  $c \geq 75$  on the completion transition means the task may complete after it has received 75 time units of processor service (i.e. the task compute time is uncertain and/or variable but always falls in the interval [75, 100]).

In this example the edge guards **selected** and **unselected** represent scheduling decisions made at scheduling events (often called scheduling points in the real-time literature). These decisions depend on the available resources (processors, busses, etc.) being shared by the tasks. There are several approaches to introduce scheduling semantics into a model having several concurrent tasks.

Scheduling can be introduced using concepts taken from the theory of discrete event control[20]. A concurrent scheduler automaton can be added to the system of tasks. The scheduling points in the task set become synchronization events at which the scheduler automaton can observe the system state and make control decisions. Many high-level concepts from discrete event control theory carry over into this domain, such as the importance of decentralized control and limited observability in distributed systems.

Discrete event control theory provides an approach to synthesize optimal controllers, which in this domain translates to the automatic construction of application-specific scheduling algorithms. However, classical discrete event control theory does not deal with time. The theory has been extended to synthesize nonpreemptive schedulers for timed automata[4, 1], but this excludes preemptively scheduled systems. It is possible to develop scheduling automata by hand using traditional real-time scheduling policies such as preemptive fixed priority. Examples have been given in the literature, where each distinct ready queue state is modeled as

a distinct discrete state of the scheduler automaton[2]. This would allow a very large class of scheduling algorithms to be modeled, but the size of the scheduler automaton may grow combinatorially with the number of tasks.

It is possible to model preemptive fixed priority scheduling by encoding the ready queue in a variable rather than in a set of discrete states. A queue variable is introduced that will take on only integer values. At each transition where a task  $i$  is dispatched,  $2^i$  is added to this queue variable; at each transition where task  $i$  completes,  $2^i$  is subtracted. The queue variable can be interpreted as a bit vector whose  $i^{\text{th}}$  bit is set whenever task  $i$  is ready to compute. There is no separate scheduler automaton, the scheduling protocol is modeled using additional guards and states in the task automata. This is the approach we took when we started our work using HyTech. This encodes a specific scheduling protocol into each task model, and adds additional discrete states, variables and guards to the model. It is awkward to model any scheduling policy other than simple preemptive fixed priority without inheritance.

In the end, we found it simpler and more general to define a slightly extended linear hybrid automata model that includes resource scheduling semantics[23]. The discrete state composition of the task set is performed before any scheduling decisions are made. A scheduling function is then applied to the composed system discrete state to determine the variable rates to be used for that system state. In essence, the composed system discrete state is the ready queue to which the scheduling function is applied, very much analogous to the way run-time scheduling algorithms are applied in an actual real-time system. It is not necessary to have different discrete states for preempted and computing, since this information is now captured in the variable rates. It is not necessary to model a scheduling algorithm as a finite state control automaton added to the system, it is not necessary to encode a specific scheduling semantics into the task automata. One simply codes up a scheduling algorithm in the usual way and links it with the rest of the reachability analysis code. This approach significantly reduces the number of discrete states in the model and simplifies the modeling of the desired scheduling discipline. The formal details of this model and its semantics are recorded elsewhere[23].

### 3 Reachable Regions

A state of a linear hybrid automaton consists of a discrete part, the discrete state at some time  $t$ ; and a continuous part, the real values of the variables at time  $t$ . It turns out that, although this state space is uncountably infinite, the reachable state space for a given linear hybrid automaton is a subset of the cross-product

of the discrete states with a recursively enumerable set of convex polyhedra in  $\mathbb{R}^n$  (where  $n$  is the number of variables)[2]. A region of a linear hybrid automaton is a pair consisting of a discrete state and a convex polyhedron, where convex polyhedra can be represented using a finite set of linear constraints. Model checking consists of enumerating the reachable regions for a given linear hybrid automaton and checking to see if they satisfy the assertions.

Figure 2 depicts the basic sequence of operations that, given a starting region (a discrete state and a polyhedron defining a set of possible values for the variables), computes the set of values the variables might take on in that discrete state as time passes; and computes a set of regions reachable by subsequent discrete transitions.

The first step is the computation of the time successor polyhedron from the starting polyhedron (often called the post operation). For each point in the starting polyhedron, the time successor of that point is a line segment beginning at that point whose slope is defined by the variable rates specified for the discrete state. This is the set of variable values that can be reached from a starting point by allowing some amount of time to pass. The time successor of the starting polyhedron is the union of the time successor lines for all points in the starting polyhedron. A basic result of linear hybrid automata theory is that the time successor of any convex polyhedron is itself a convex polyhedron (which in general will be unbounded in certain directions)[2].

The second step is the intersection of the time successor polyhedron with the invariant constraint associated with the discrete state. Polyhedra are easily intersected by taking the union of the set of linear constraints that define the two polyhedra. This is the time successor region that is feasible given the invariant specified for the discrete state.

The remaining steps are used to compute new regions reachable from this feasible time successor region by some transition over an edge. For each edge out of the current discrete state, the associated guard is first intersected with the feasible time successor region. This polyhedron, if nonempty, defines the set of all variable values that might exist whenever the discrete transition could occur. Any variable assignments associated with the edge must now be applied to this polyhedron. This is done in two phases. First, a variable to be assigned a new value  $x := l$  is unconstrained (often called the free operation). This operation leaves unchanged the relationships between all other variables, i.e. the polyhedron is projected onto the subspace  $\mathbb{R}^{n-1}$  of the remaining variables. This result is then intersected with the constraint  $x = l$ . This polyhedron, together with the discrete state to which the edge goes, is a new re-

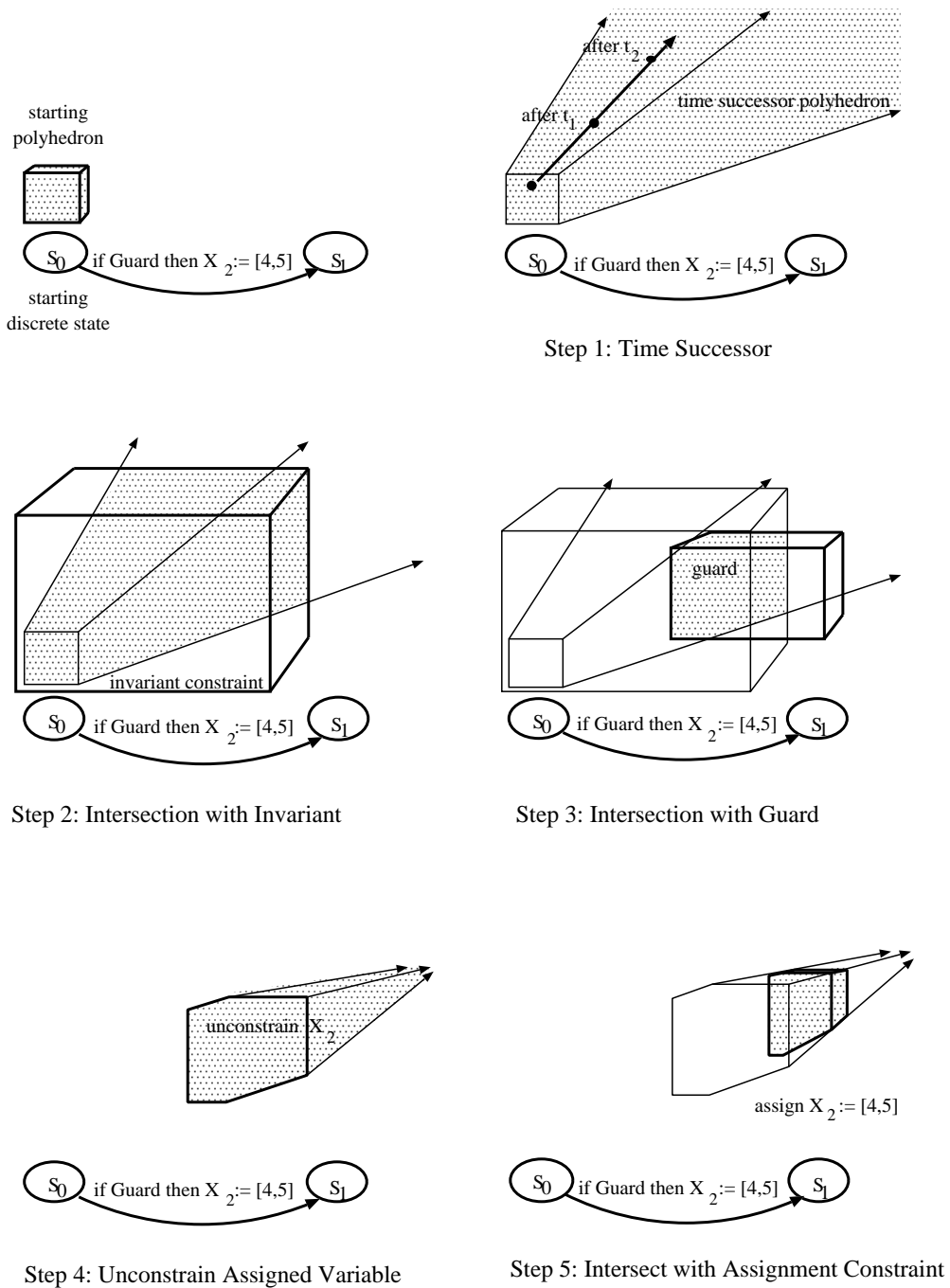


Figure 2: Hybrid Automata Reach Forward Operations

gion for which the above steps may be repeated. In general a set of assignments whose right-hand sides are linear expressions are allowed, with some restrictions. The variables to be assigned are unconstrained and the resulting polyhedra are then intersected with the appropriate linear constraints in some order. With care, fairly complex sequences of assignments can be modeled on a

single edge[24].

The overall method begins at the initial region of a hybrid automaton. The operations described above are applied to enumerate feasible time successor regions and the new regions reachable from these via discrete transitions. As new regions are enumerated, they must be checked to see if they have been visited before (other-

wise the method will not terminate even when there are a finite number of regions). This is done by comparing the discrete states of regions for equality, and by checking to see if the new polyhedron is contained in the polyhedron of a previously visited region. This is summarized in Figure 3.

## 4 Operations on Polyhedra

We use  $X$  to denote a set of real-valued variables  $x_1, x_2, \dots, x_n$ , and  $e = c_1x_1 + c_2x_2 + \dots + c_nx_n$  to denote some linear expression over  $X$  with rational coefficients. We use  $\bar{r} \in \mathfrak{R}^n$  to denote some specific point in  $\mathfrak{R}^n$ , a specific assignment of real values to  $X$ .

We use  $P$  to denote a finite set of constraints of the form  $l \leq e \leq u$ . We use the phrase ‘‘polyhedron  $P$ ’’ to refer to the set of all solutions to the system of constraints  $P$ . We sometimes abuse notation and write  $\bar{r} \in P$  to mean that the point  $\bar{r}$  is in the polyhedron  $P$ , i.e.  $\bar{r}$  satisfies the system of constraints  $P$ .

We use  $\dot{X} = \{[\dot{x}_{1s}, \dot{x}_{1f}], [\dot{x}_{2s}, \dot{x}_{2f}], \dots\}$  to denote a set of finite rate intervals for the variables in  $X$ . Each interval  $[\dot{x}_{is}, \dot{x}_{if}]$  is specified using rational values for the slowest and fastest possible rate for variable  $x_i$ . We use  $\dot{x}$  to denote a specific real-valued vector rate for the variables in  $X$ , where  $\dot{x} \in \dot{X}$ , i.e. the rates  $\dot{x}$  fall within their specified intervals. We use  $\dot{e}$  to denote the rate or derivative of a linear expression  $e$  given some  $\dot{x}$ , i.e.  $\dot{e} = c_1\dot{x}_1 + c_2\dot{x}_2 + \dots + c_n\dot{x}_n$ . The scalar expression rate is a function of the vector variable rate, but for brevity we write  $\dot{e}$  instead of  $\dot{e}(\dot{x})$ . We use  $\dot{E} = [\dot{e}_s, \dot{e}_f]$  to denote the rate interval of a linear expression  $e$  given some  $\dot{X}$ .

We use  $\bar{r} \uparrow \delta$  as a short-hand notation for  $\bar{r} + \dot{x}\delta$ , the point reached from  $\bar{r}$  after allowing  $\delta$  time to pass for some possible variable rate  $\dot{x}$ . Similarly,  $\bar{r} \downarrow \delta$  is the point from which  $\bar{r}$  is reached after allowing  $\delta$  time to pass for a given  $\dot{x}$ .

### 4.1 Time Successor

The time successor of a set of constraints given constant rate interval  $\dot{X}$  is computed by modifying  $P$  in two steps.

1. Let every constraint  $l \leq e \leq u$  where  $\dot{e} \neq [0, 0]$  be written so that  $\dot{e}_f \geq 0$ , which can be achieved by multiplying the constraint by -1 if needed. For every distinct pair of constraints

$$\begin{aligned} l_i &\leq e_i \leq u_i \\ l_j &\leq e_j \leq u_j \end{aligned}$$

where  $\dot{e}_i \neq [0, 0]$  and  $\dot{e}_j \neq [0, 0]$ , add to the set the following constraints under the following conditions.

If  $\dot{e}_{is} \geq 0$  then add the constraint

$$\begin{aligned} &\dot{e}_{jf}(l_i \leq e_i \leq u_i) - \dot{e}_{is}(l_j \leq e_j \leq u_j) \\ \equiv &\dot{e}_{jf}l_i - \dot{e}_{is}u_j \leq \dot{e}_{jf}e_i - \dot{e}_{is}e_j \leq \dot{e}_{jf}u_i - \dot{e}_{is}l_j \end{aligned}$$

else if  $\dot{e}_{is} < 0$  then add the constraint

$$\begin{aligned} &\dot{e}_{jf}(l_i \leq e_i \leq u_i) - \dot{e}_{is}(l_j \leq e_j \leq u_j) \\ \equiv &\dot{e}_{jf}l_i - \dot{e}_{is}l_j \leq \dot{e}_{jf}e_i - \dot{e}_{is}e_j \leq \dot{e}_{jf}u_i - \dot{e}_{is}u_j \end{aligned}$$

If  $\dot{e}_{js} \geq 0$  then add the constraint

$$\begin{aligned} &\dot{e}_{js}(l_i \leq e_i \leq u_i) - \dot{e}_{if}(l_j \leq e_j \leq u_j) \\ \equiv &\dot{e}_{js}l_i - \dot{e}_{if}u_j \leq \dot{e}_{js}e_i - \dot{e}_{if}e_j \leq \dot{e}_{js}u_i - \dot{e}_{if}l_j \end{aligned}$$

else if  $\dot{e}_{js} < 0$  then add the constraint

$$\begin{aligned} &\dot{e}_{js}(l_i \leq e_i \leq u_i) - \dot{e}_{if}(l_j \leq e_j \leq u_j) \\ \equiv &\dot{e}_{js}u_i - \dot{e}_{if}u_j \leq \dot{e}_{js}e_i - \dot{e}_{if}e_j \leq \dot{e}_{js}l_i - \dot{e}_{if}l_j \end{aligned}$$

2. For every constraint  $l \leq e \leq u$ , if  $\dot{e}_f > 0$  then omit the upper bound (replace  $u$  with  $\infty$ ), if  $\dot{e}_s < 0$  then omit the lower bound (replace  $l$  with  $-\infty$ ), if both conditions hold then remove the constraint.

$$\begin{aligned} &\dot{e}_{jf}l_i - \dot{e}_{is}u_j \leq \dot{e}_{jf}e_i - \dot{e}_{is}e_j \\ &\dot{e}_{js}e_i - \dot{e}_{if}e_j \leq \dot{e}_{js}u_i - \dot{e}_{if}l_j \end{aligned}$$

The number of operations required by the algorithm is quadratic in the number of constraints whose expressions have non-zero rate, hence cubic in the size of the constraint matrix. The resulting constraint matrix may have quadratically more rows than the original constraint matrix.

**Theorem:** Let  $P$  be a feasible set of constraints, and let  $P'$  be computed from  $P$  using the above algorithm. Then

- For every  $\bar{r} \in P$  and every  $\delta \geq 0$ ,  $\bar{r} \uparrow \delta \in P'$  (allowing any amount of time to pass for any point that satisfies  $P$  yields a point that satisfies  $P'$ ).
- For every  $\bar{r}' \in P'$  there exists some  $\delta \geq 0$  such that  $\bar{r}' \downarrow \delta \in P$  (every point in  $P'$  is reachable from some point in  $P$  by allowing some amount of time to pass).

**Proof:** We show the first part by demonstrating that none of the constraints introduced or modified by the algorithm are invalidated by allowing time to pass from any initial value  $\bar{r} \in P$ . Step 1 adds constraints that are linear combinations of existing constraints in  $P$ , so these are all satisfied by every  $\bar{r} \in P$ . After step 2, every constrained expression that can have a positive rate has no upper bound and every constrained expression that can have a negative rate has no lower bound, so that no amount of time passing can make any constraint in  $P'$  infeasible. Note this argument depends only on the signs of the rates of the constrained expressions, not on the specific magnitudes of the individual variable rates. This proves the first part of the theorem.

```

Entry_Region,
New_Region: Region;
Successor_Polyhedron,
Constrained_Polyhedron,
Guarded_Polyhedron,
Unconstrained_Polyhedron,
Assigned_Polyhedron: Polyhedron;
Examined,
To_Be_Examined: set of Region;

Examined := empty;
To_Be_Examined := initial region;
while Not_Empty (To_Be_Examined) loop
  Entry_Region := Choose_And_Remove_One_Of (To_Be_Examined);
  Add_To_Set (Examined, Entry_Region);
  Successor_Polyhedron := Time_Successor (Entry_Region.Polyhedron);
  Constrained_Polyhedron := Intersect (Successor_Polyhedron, Entry_Region.Discrete_State.Invariant);
  Check_Region_Assertion (Entry_Region.Discrete_State, Constrained_Polyhedron);
  for each Transition in Entry_Region.Discrete_State.Transitions_From loop
    Guarded_Polyhedron := Intersect (Constrained_Polyhedron, Transition.Guard);
    if Not_Empty (Guarded_Polyhedron) then
      Check_Transition_Assertion (Transition, Guarded_Polyhedron);
      Unconstrained_Polyhedron := Unconstrain (Guarded_Polyhedron, Transition.Assignments.Variables);
      Assigned_Polyhedron := Intersect (Unconstrained_Polyhedron, Transition.Assignments);
      New_Region.Discrete_State := Transition.To_Discrete_State;
      New_Region.Polyhedron := Assigned_Polyhedron;
      for each Previous_Region in Union (Examined, To_Be_Examined) loop
        if Contained_In (New_Region, Previous_Region) then
          goto next Transition loop;
        end if;
      end loop;
      Add_To_Set (To_Be_Examined, New_Region);
    end if;
  end loop;
end loop;

```

Figure 3: High-Level Region Enumeration Procedure

We show the second part by demonstrating that  $\bar{r}' \in P'$  implies there exists some  $\delta \geq 0$  and some fixed  $\dot{x} \in \dot{X}$  such that  $\bar{r}' \downarrow \delta \in P$ . (The semantics of hybrid automata allow  $\dot{x}_i$  to vary between  $\dot{x}_{is}$  and  $\dot{x}_{if}$  as time passes, but we will show there exists a particular  $\dot{x}$  that satisfies the theorem.)

Observe that any constraint whose expression has zero rate appears in both  $P$  and  $P'$  and remains satisfied independent of time. We only need to find  $\delta$  and  $\dot{x}$  to satisfy constraints originally in  $P$  that may have non-zero rates, and we restrict our attention in the rest of this proof to constraints that are operated on in steps 1 and 2.

We will show there exists a scalar  $\delta \geq 0$  and rate vector  $\dot{x} \in \dot{X}$  such that all of

$$l_i \leq e_i - \dot{e}_i \delta \leq u_i$$

in  $P$  where  $\dot{e}_i \neq [0, 0]$  are feasible for every point  $\bar{r}' \in P'$ . We can rewrite these constraints as

$$\begin{aligned}
& l_i \leq e_i - \dot{e}_i \delta \leq u_i \\
\equiv & l_i - e_i \leq -\dot{e}_i \delta \leq u_i - e_i \\
\equiv & e_i - u_i \leq \dot{e}_i \delta \leq e_i - l_i \\
\equiv & \frac{e_i - u_i}{\dot{e}_i} \leq \delta \leq \frac{e_i - l_i}{\dot{e}_i} \text{ for } \dot{e}_i > 0, \\
& \frac{e_i - l_i}{\dot{e}_i} \leq \delta \leq \frac{e_i - u_i}{\dot{e}_i} \text{ for } \dot{e}_i < 0
\end{aligned} \tag{1}$$

There exists a  $\delta \geq 0$  that satisfies all these inequalities when evaluated at  $\bar{r}'$  if there exists a value for  $\delta$  that simultaneously falls between the upper and lower bounds of all these constraints. Such a value exists when no lower bound exceeds any upper bound, i.e. when the set of constraints

$$\begin{aligned}
& \frac{e_j - u_j}{\dot{e}_j} \leq \frac{e_i - l_i}{\dot{e}_i} \text{ for } \dot{e}_i > 0, \dot{e}_j > 0 \\
& \frac{e_j - u_j}{\dot{e}_j} \leq \frac{e_i - u_i}{\dot{e}_i} \text{ for } \dot{e}_i < 0, \dot{e}_j > 0 \\
& \frac{e_j - l_j}{\dot{e}_j} \leq \frac{e_i - l_i}{\dot{e}_i} \text{ for } \dot{e}_i > 0, \dot{e}_j < 0 \\
& \frac{e_j - l_j}{\dot{e}_j} \leq \frac{e_i - u_i}{\dot{e}_i} \text{ for } \dot{e}_i < 0, \dot{e}_j < 0
\end{aligned}$$

is feasible for all pairs  $i$  and  $j$ . For  $i = j$  the applicable constraints (first and last above) reduce to  $l_i \leq u_i$ ,

which are always true when  $P$  is feasible regardless of the variable rates. For  $i \neq j$  we can rewrite these as

$$\begin{aligned} \dot{e}_j l_i - \dot{e}_i u_j &\leq \dot{e}_j e_i - \dot{e}_i e_j \text{ for } \dot{e}_i > 0, \dot{e}_j > 0 \\ \dot{e}_j e_i - \dot{e}_i e_j &\leq \dot{e}_j u_i - \dot{e}_i u_j \text{ for } \dot{e}_i < 0, \dot{e}_j > 0 \\ \dot{e}_j e_i - \dot{e}_i e_j &\leq \dot{e}_j l_i - \dot{e}_i l_j \text{ for } \dot{e}_i > 0, \dot{e}_j < 0 \\ \dot{e}_j u_i - \dot{e}_i l_j &\leq \dot{e}_j e_i - \dot{e}_i e_j \text{ for } \dot{e}_i < 0, \dot{e}_j < 0 \end{aligned}$$

What we show in the remainder of the proof is that constraints at least as strict as these appear in  $P'$ , which was shown feasible by the first part of the proof, so there exists a solution to these equations.

To do this we must first show the effects of step 2 on the constraints added in step 1. Consider the rate of the expression in the constraint added in step 1 when  $\dot{e}_{is} > 0$ ,  $\dot{e}_{jf}\dot{e}_i - \dot{e}_{is}\dot{e}_j$ . Since all rates are positive in this case, the slowest rate occurs when  $\dot{e}_i$  is slowest and  $\dot{e}_j$  is fastest. This rate is  $\dot{e}_{jf}\dot{e}_{is} - \dot{e}_{is}\dot{e}_{jf} = 0$ , so the lower bound of this constraint will be preserved in step 2. Similar reasoning in the other three cases shows that after step 2,  $P'$  will contain at least the constraints

$$\begin{aligned} \dot{e}_{jf} l_i - \dot{e}_{is} u_j &\leq \dot{e}_{jf} e_i - \dot{e}_{is} e_j \\ \dot{e}_{jf} l_i - \dot{e}_{is} l_j &\leq \dot{e}_{jf} e_i - \dot{e}_{is} e_j \end{aligned}$$

## 4.2 Unconstrain

To unconstrain a variable  $x$  we must remove all constraints that contain that variable. However, there may be constraints between other variables that are transitively implied by a set of removed constraints. For example,  $l_1 \leq y - x \leq u_1$  and  $l_2 \leq x - z \leq u_2$  imply  $l_1 + l_2 \leq y - z \leq u_1 + u_2$ , this information must be preserved before removing the constraints involving  $x$ . We unconstrain a variable  $x$  in a set of constraints  $P$  by constructing a new set of constraints  $P'$  using the following steps.

1. Let each constraint  $l \leq e \leq u$  in  $P$  where  $e$  has an instance of  $x$  be written in the form  $l \leq cx - e' \leq u$ , where  $e'$  involves the remaining variables and their coefficients and  $c > 0$ . For each distinct pair of such constraints in  $P$

$$\begin{aligned} l_i &\leq c_i x - e_i \leq u_i \\ l_j &\leq c_j x - e_j \leq u_j \end{aligned}$$

combine the two in a way that cancels the  $x$  terms, adding to  $P'$  the constraint

$$c_j l_i - c_i u_j \leq c_i e_j - c_j e_i \leq c_j u_i - c_i l_j$$

2. Each constraint  $l \leq e \leq u$  where  $e$  has no instances of variable  $x$  is added to  $P'$ .

Let  $X \setminus x$  refer to the set of variables  $X$  minus the variable  $x$ , and let  $\bar{r} \setminus x$  for  $\bar{r} \in \mathfrak{R}^n$  refer to the  $n - 1$

vector of values that are identical to  $\bar{r}$  with the value for variable  $x$  removed.

**Theorem:** Let  $P$  be a feasible set of difference constraints, and let  $P'$  be computed from  $P$  by applying the above algorithm to unconstrain the variable  $x$ . If  $\bar{r} \in P$  then  $\bar{r} \setminus x \in P'$ , and if  $\bar{r} \setminus x \in P'$  then there exists some value for variable  $x$  such that  $\bar{r} \in P$ .

**Proof:** We will show that  $\bar{r} \in P$  iff  $\bar{r} \setminus x \in P'$  for some value for  $x$ .

Each constraint added to  $P'$  in step 1 is implied by some pair of constraints in  $P$ . Constraints added in step 2 are the same in  $P$  and  $P'$ . Consequently, every constraint in  $P'$  is implied by one or two constraints in  $P$ , and every value  $\bar{r} \in P$  thus satisfies the constraints in  $P'$ . To state this another way, the constraints added to  $P'$  are never any tighter than constraints that occur in  $P$ . Note that this means  $P'$  is feasible since  $P$  is assumed feasible.

For the second part of the proof, consider a value  $\bar{r} \setminus x \in P'$ . We need to show there exists some value for variable  $x$  such that  $\bar{r}$  satisfies all of the constraints

$$l_i \leq c_i x - e_i \leq u_i \quad (2)$$

in  $P$ . There will exist a value for  $x$  that satisfies the above constraints if there is a value for  $x$  that simultaneously satisfies every pair of constraints

$$\begin{aligned} \frac{l_i + e_i}{c_i} &\leq x \leq \frac{u_i + e_i}{c_i} \\ \frac{l_j + e_j}{c_j} &\leq x \leq \frac{u_j + e_j}{c_j} \end{aligned}$$

where  $c_i, c_j > 0$ . For  $i = j$  this reduces to  $l_i \leq u_i$ , which always holds when  $P$  is feasible. For  $i \neq j$  we can rewrite these as

$$\begin{aligned} c_j l_i + c_j e_i &\leq c_i u_j + c_i e_j \\ \equiv c_j l_i - c_i u_j &\leq c_i e_j - c_j e_i \end{aligned}$$

These were all added to  $P'$  in step 1 and are all satisfied for every  $\bar{r} \setminus x \in P'$  ( $P'$  is feasible as noted above). There thus exists a value for  $x$  that satisfies the constraints in (2), and this value is such that  $\bar{r} \in P$ .

## 4.3 Intersection

The intersection of the solution sets for two systems of constraints  $P_1$  and  $P_2$  is the set of values that satisfies both systems of constraints, which is the set of solutions to  $P_1 \cup P_2$  (the union of the sets of constraints has as its solution the intersection of the corresponding polyhedra). This can be done in linear time.

## 4.4 Feasibility

The feasibility of a set of inequalities  $P$  can be determined as a side-effect of solving the associated linear programming problem with some trivial objective function, e.g.  $\max \sum_i x_i$  given  $P$  for the variables  $x_i$  appearing in  $P$ .

We note that feasibility testing seems fundamentally as hard as linear programming[6]. Each linear programming problem has an associated dual problem of the same size, with the property that only optimal solutions are feasible for both. Thus, any feasibility test capable of identifying a feasible solution can be used to solve a linear programming problem by applying that test to the union of the constraints of the original problem and its dual.

We discovered by experiment that guessing a set of variable values (the mid-point of each rectangular constraint) then evaluating the constraints using those values was an effective approximate test. This test can quickly confirm that certain polyhedra are feasible, and our experiments suggest that perhaps half of all feasibility tests could be resolved using this method. However, feasibility testing accounted for a relatively small portion of the overall execution time of our prototype, and this approximation had no significant impact.

#### 4.5 Containment

Given two sets of constraints  $I$  (inner) and  $O$  (outer) we want to determine if every solution to  $I$  is also a solution to  $O$  (whether the polyhedron  $I$  is contained in the polyhedron  $O$ ). We do this using the following algorithm.

1. If it can be (quickly) determined that  $I$  and  $O$  do not intersect, then  $I$  cannot be contained in  $O$ .
2. If  $O$  contains variables that do not appear in  $I$  then terminate with a negative result.
3. For each constraint  $l \leq e \leq u$  in  $O$  solve the linear programming problems  $\bar{x}_l = \min e$  given  $I$  and  $\bar{x}_u = \max e$  given  $I$ . If  $l \leq \bar{x}_l$  and  $u \geq \bar{x}_u$  for every constraint in  $O$  then polyhedron  $I$  is contained in polyhedron  $O$ . Otherwise, the algorithm terminates with a negative result when the first constraint from  $O$  is found that does not pass this test.

The first step is a prefilter to efficiently detect certain common cases where  $I$  obviously cannot be contained in  $O$ . We did not do an exact feasibility test, only an approximate one that quickly checks to see if the intersection is definitely infeasible (discussed later). Our experience suggests that over 80% of all containment tests are resolved by this prefilter. This was important for our prototype, which spent most of its time searching for containing polyhedra (the loop for each `PreviousRegion` in Figure 3).

To establish the correctness of the final step we prove the following.

**Theorem:** For each constraint  $l \leq e \leq u$  in  $O$  let  $\bar{x}_l = \min e$  given  $I$  and  $\bar{x}_u = \max e$  given  $I$  be solutions to linear programming problems. Every feasible value

for  $I$  is also a feasible value for  $O$  iff  $l \leq e(\bar{x}_l)$  and  $u \geq e(\bar{x}_u)$  for every constraint  $l \leq e \leq u$  in  $O$ .

**Proof:** Suppose  $l \leq e(\bar{x}_l)$  and  $u \geq e(\bar{x}_u)$  for every constraint  $l \leq e \leq u$  in  $O$ . It is known that optimal values for the objective function of a linear programming problem are achieved at some boundary vertex or facet of the polyhedron  $I$  (or else the value of the objective function is unbounded). For  $\bar{x}_l = \min e$  given  $I$  the value of  $e$  at every point in the polyhedron is bounded below by  $e(\bar{x}_l)$ , and for  $\bar{x}_u = \max e$  given  $I$  the value of  $e$  at every point in the polyhedron is bounded above by  $e(\bar{x}_u)$  (or else the value of  $e$  is unbounded below or above, respectively). If  $l \leq e(\bar{x}_l)$  and  $u \geq e(\bar{x}_u)$  then every point in the polyhedron  $I$  satisfies the constraint  $l \leq e \leq u$  in  $O$  (where we allow  $l$  or  $u$  to be  $-\infty$  or  $\infty$  respectively). If this holds for all constraints in  $O$  then every feasible value for  $I$  also satisfies all the constraints of  $O$ .

#### 4.6 Assertion Checking

An assertion  $A$  where  $A$  is a system of linear constraints can be evaluated for a given polyhedron  $P$  by seeing if  $P$  is contained in  $A$ . Conjunctions and disjunctions of sets of linear inequalities can be evaluated in the obvious way.

#### 4.7 Reduction

The time successor and unconstrain operations may cause a quadratic increase in the number of constraints. An essential element of our procedure is the use of an algorithm to reduce the number of constraints used to represent a polyhedron by identifying and eliminating redundant constraints. We combine a fast but approximate bounds tightening procedure with a more effective but expensive Simplex-based procedure to detect and eliminate redundant constraints.

An important part of our procedure is the initial use of an efficient bounds tightening procedure to simplify sets of constraints[5]. For each pair of constraints

$$\begin{aligned} l_i &\leq e_i \leq u_i \\ l_j &\leq e_j \leq u_j \end{aligned}$$

if there exist  $c_i, c_j > 0$  such that  $c_i e_i = c_j e_j = e$  then these constraints can be replaced by the single constraint

$$\max(c_i l_i, c_j l_j) \leq e \leq \min(c_i u_i, c_j u_j)$$

There are two such rules, one for  $c_i > 0$  and one for  $c_i < 0$  ( $c_j$  can always be made positive). Our prototype implementation maintains polyhedra as lists of constraints that are lexicographically sorted by variable, so that any two linearly dependent constraints will appear adjacent in the list. This bounds tightening operation is applied during each intersection operation (implemented as a linear-time merge of two sorted lists),

including each time a new constraint is added to a constraint list.

The time successor and unconstrain operations add constraints that are differences of existing constraints, and in practice many of these added constraints are redundant with each other. These two algorithms frequently add triplets of the form

$$\begin{aligned} l_{ij} &\leq e_i - e_j \leq u_{ij} \\ l_{jk} &\leq e_j - e_k \leq u_{jk} \\ l_{ik} &\leq e_i - e_k \leq u_{ik} \end{aligned} \quad (3)$$

where the third constraint may be implied by the sum of the first two (ignoring constant multipliers). Similarly, the first constraint of the triplet

$$\begin{aligned} l_{ij} &\leq e_i - e_j \leq u_{ij} \\ l_j &\leq e_j \leq u_j \\ l_i &\leq e_i \leq u_i \end{aligned} \quad (4)$$

may be implied by the difference of the other two constraints. We can check for these implications and use them to tighten constraints and eliminate redundant constraints. This could be viewed as an approximate generalization of the shortest path algorithm used to simplify bounded difference matrices[8, 3].

More precisely, for each triplet of constraints

$$\begin{aligned} l_i &\leq e_i \leq u_i \\ l_j &\leq e_j \leq u_j \\ l_k &\leq e_k \leq u_k \end{aligned}$$

if there exist  $c_i, c_j, c_k > 0$  such that  $c_i e_i - c_j e_j = c_k e_k$  then constraint  $l_k \leq e_k \leq u_k$  can be replaced by

$$\min(c_k l_k, c_i l_i - c_j l_j) \leq c_k e_k \leq \max(c_k u_k, c_i u_i - c_j l_j)$$

There are four such rules, one for each combination of possible signs for  $c_i$  and  $c_j$  ( $c_k$  can always be made positive). Linear dependence is transitive in the sense that when dependence is detected, each of the three can be tightened using similar formulas involving the other two.

Checking all possible triplets would be  $O(n^3)$  in the number of constraints. Instead, we record with most constraints two references  $T_i$  and  $T_j$  to two other constraints with whose sum it is likely to be linearly dependent. For each constraint added by the time successor operation,  $T_i$  and  $T_j$  are the two constraints differenced to form that constraint. For each constraint added by the unconstrain operation, we search for constraints that have the same variables as the two that were differenced except for the variable being unconstrained. Our experience suggests that reasonable  $T_i$  and  $T_j$  can be identified for most constraints involving two or more variables.

Let  $(e, T_i, T_j)$  be a constraint  $e$  and its associated  $T_i, T_j$  references. We iterate over all triplets of constraints  $(e_1, T_i, T_j), (e_2, T_j, T_k), (e_3, T_i, T_k)$  to produce

candidates likely to have the form shown previously in (3). We also iterate over all triplets  $(e, T_i, T_j), T_i, T_j$ . Each triplet is checked for linear dependence, which can be determined by solving a simple  $2 \times 2$  system of linear equations involving the coefficients of variables common to the three constraints. Where linear dependence is detected, each of the constraints has its bounds tightened using the bounds implied by the appropriate linear combination of the other two constraints.

In our prototype we associate with each constraint in the list its index or numeric position in the list. The pair  $(T_i, T_j)$  is kept in a canonical order where the index of  $T_i$  is less than the index of  $T_j$ . We produce a list of references to the constraints that is lexicographically sorted by the index values for  $(T_i, T_j)$ . Using this sorted list, it is possible to iterate over triplets of constraints in quadratic time.

In our prototype the constraint list is ordered so that all constraints involving the same variables are adjacent to each other. Each time we check a triplet of constraints  $e_i, e_j, e_k$  for linear dependence, we check all triplets having the same variables as the set  $e_i, e_j, e_k$  and not just those three individual constraints. Our experience suggests this heuristic is worth the additional cost, which tends to be relatively small since subsequences of constraints involving identical sets of variables tend to be relatively short.

In general, multiple such iterations may be needed to find a fixed point at which no constraint bounds are tightened any further. Our experience suggests this can in fact be limited to a small fixed number, such as 3, without any significant impact.

As constraints are tightened, a note is made for each bound as to whether that bound is implied by other nonredundant constraints on the list. For example, if  $l_i \leq e_i \leq u_i$  and  $e_i = e_j + e_k$  and  $l_i = l_j + l_k$  then the lower bound  $l_i$  is implied by the other two constraints and is redundant, providing  $l_j \leq e_j$  and  $l_k \leq e_k$  are not marked as redundant. A final pass is performed to replace all redundant lower and upper bounds by  $-\infty$  and  $\infty$  respectively. All constraints where both the lower and upper bounds are redundant are deleted (our implementation always retains the tightest rectangular constraint for every variable appearing in a polyhedron for book-keeping reasons).

If, during any bounds tightening operation,  $u < l$  for any constraint  $l \leq e \leq u$  then the system is definitely not feasible. This feasibility test is not exact, it is possible for  $l \leq u$  for every constraint in an infeasible system of constraints. However, we use this as a fast test to quickly detect many cases of infeasibility, including the prefilter for our containment test. Our experience suggests that about  $\frac{1}{3}$  of all feasibility tests can be decided in this manner, or about 80% if the containment pre-

filter is also counted.

If the above bounds tightening procedure fails to reduce the number of constraints in a polyhedron to less than half the average number of constraints in all polyhedra, then our prototype tool applies a more expensive but more effective procedure. For each constraint  $l \leq e \leq u$  in  $P$  solve the linear programming problems  $\bar{x}_l = \min e$  given  $P - \{l \leq e \leq u\}$  and  $\bar{x}_u = \max e$  given  $P - \{l \leq e \leq u\}$ . If  $l \leq \bar{x}_l$  and  $u \geq \bar{x}_u$  then  $l \leq e \leq u$  is redundant and is removed. Our experience suggests that over 95% of all reductions are performed using the bounds tightening procedure alone, but it is nevertheless essential to include this more effective procedure. Without this, our prototype was sometimes unable to complete an analysis due to the presence of a few polyhedra for which bounds tightening was ineffective, where these few polyhedra formed a brick wall that prevented complete reachability analysis.

#### 4.8 Integer Model Parameters

We restricted our prototype implementation to models with integer numeric parameters in order to achieve improved numeric stability and efficiency. Linear hybrid automata models may in general have rational numeric parameters, so we need to justify our restriction to integer parameters.

**Theorem:** Let  $A$  be a linear hybrid automata model containing rational numeric parameters. Let  $v$  be an integer value such that multiplying all parameters in  $A$  by  $v$  yields a new model  $A'$  having only integer numeric parameters. The assertions in model  $A$  are true iff the assertions in model  $A'$  are true.

**Proof:** We first note that any finite set of rational numbers has a least common denominator, and multiplying all numbers in the set by this value yields a set of integers, so there always exists some  $v$  that allows the theorem to be applied to any linear hybrid automaton.

*Fill in rest of theorem, or find a citation.*

### 5 Implementation Notes

We implemented our prototype in a compiled language, Ada 95. We represented constraint bounds and coefficients using 64 bit integers. We used a sparse vector representation that only stores non-zero coefficients. Polyhedra were represented as doubly linked lists of constraints. The set of reached regions was stored by hashing the discrete system state, then storing a list of polyhedra for each discrete system state.

Our primal/dual Simplex algorithm used double precision floating point and a sparse matrix representation. The Simplex algorithm is used only as a decision procedure, it does not compute any values that appear in any polyhedra. Nevertheless, this is a notable theoretical shortcoming in our prototype, which would ideally use rational arithmetic for the Simplex procedure. In

this application large numbers of polyhedra have degenerate vertices, and our experience suggests that the Simplex implementation must include methods to deal with degeneracy[9, 17].

Efficient and robust computation of greatest common divisors (GCDs) during polyhedra reduction proved interesting and important enough to merit some comment. If two numbers are represented as products of their prime factors  $X = 2^{x_1}3^{x_2}5^{x_3}\dots$  and  $Y = 2^{y_1}3^{y_2}5^{y_3}\dots$  then the exponents in the prime factorization of their GCD is the min of the exponents of the two values[11],

$$\text{GCD}(X, Y) = 2^{\min(x_1, y_1)} 3^{\min(x_2, y_2)} 5^{\min(x_3, y_3)} \dots$$

When computing the prime factorization of  $Y$  for the purpose of obtaining a GCD, it is only necessary to determine the exponents out to the last non-zero exponent of  $X$ , e.g. when computing  $\text{GCD}(12, Y)$  it is only necessary to compute the prime factors  $y_1$  and  $y_2$ . This is because all the remaining prime factor exponents for 12 are 0, and  $\min(0, y_k) = 0$  for any  $y_k$ . When computing the GCD of all numbers appearing in a constraint we first sort those numbers in ascending order, which greatly reduces the need to determine the exponents of the larger prime factors of the larger numbers. However, we still encounter constraints having very large and relatively prime values, too large to be factored using any reasonably sized table of prime numbers. In this case we apply Euclid's algorithm to adjacent pairs of numbers in the sorted list in a way that reduces the length by half, e.g.  $\text{GCD}(X_1, X_2), \text{GCD}(X_3, X_4), \dots$ . This increases the likelihood the two numbers submitted to Euclid's algorithm have about the same magnitude. This halved list of sorted numbers is then processed recursively.

A new polyhedron is added to the list of regions for a discrete state only when it is not contained in an existing region. However, the new polyhedron might contain a previously visited polyhedron. We check for this condition for all polyhedra that are still on the to-be-examined list and remove any polyhedra that are contained in the newly added one. We access the to-be-examined list in first-in-first-out order (depth-first search), which seems to result in a slightly higher percentage of to-be-examined polyhedra being removed than when using a first-in-first-out order (breadth-first search).

### 6 Benchmark Results

To exercise our prototype we randomly generated a series of 100 linear hybrid automata models for traditional uniprocessor workloads consisting of repetitively dispatched noninteracting harmonic tasks. Each task had minimum and maximum period and compute time values whose magnitudes reflected a reasonable level of

real-world precision (e.g. period of 400 with a compute time range of [61, 73]). Aperiodic tasks were distinguished from periodic tasks by having unequal minimum and maximum periods. Although schedulability in this case is known to be a function of the maximum compute time only, we generated both minimum and maximum compute times because this is significant in distributed systems and verification problems, and this affects the set of reachable regions. Tasks were scheduled using a deadline monotonic preemptive fixed priority discipline.

We wrote a translator to the input specification language for HyTech, a linear hybrid automata reachability tool[2, 15]. Each task had four discrete states: computing, waiting, preempted, and rescheduling. In addition to the timer and accumulated compute time variables for each task, we introduced a single integer-valued queue variable whose  $n^{\text{th}}$  bit is set when the  $n^{\text{th}}$  task is enqueued. Using this queue variable and a reschedule synchronization event, we were able to specify preemptive fixed priority scheduling in a compositional model of the system. Models were analyzed using the `-o2` option to reduce the incidence of numeric overflow.

With reasonable practical restrictions, continuous time hybrid automata models of fairly complex real-time scheduling and allocation problems can be reduced to equivalent and decidable discrete time models[23]. We also wrote a translator to the input specification language for Verus, a discrete timed automata reachability tool[7]. Scheduling was performed by introducing an additional task for this purpose, as described in the literature and in examples that come with the tool. We extended the default number of bits to 20, with a corresponding increase in the number of boolean state variables. We were unable to use the `-r` option to automatically reorder variables, which is cited as being almost essential to achieve good performance. The reason is that this option is normally used as models are developed and grow incrementally, our initial experimentation suggests that the time required to do this from scratch for a full-blown model greatly exceeds the time required to analyze the model without automatic variable reordering.

We generated two variants of this set of models. One set included both feasible and infeasible problems. We used this set to check that all these tools, plus a traditional exact characterization schedulability analysis algorithm[22], agreed. We generated another set that included only feasible problems. This set forced all tools to explore the entire reachable region space and was used for benchmarking purposes.

The ability of these tools to solve the generated feasible models is summarized in Figure 4. This figure shows the percentage of models that were solved as a function of the number of tasks. We imposed a time limit of 1

hour and a memory limit of 300 megabytes all tools. Tool failures also occurred due to numeric overflow and other problems.

Figure 5 shows the solution times in seconds as a function of the number of tasks for those models that were solved by all of the tools at each plotted point, using a logarithmic scale. That is, the figure does not include solution times for models that were solved by our prototype but not by HyTech. We include both individual problem solution times and a line showing the average solution times. The solution times for a fixed number of tasks (a fixed number of variables and reachable discrete states) vary significantly because the number of regions can vary significantly due to even small changes in the values of numeric parameters such as task periods.

We do not believe our results are sufficient to conclude there is any inherent superiority of continuous over discrete time models. BDD techniques are sensitive to variable ordering, and there may be a predictable variable ordering for problems of this particular type that yields significantly better performance. There are generalizations of BDD, such as IDD, that may have better performance[21].

It is difficult to do a direct comparison between the methods we employ and those found in HyTech because there is no one single difference.

Our polyhedra operations are restricted to constant fixed rates. HyTech allows rate ranges to be specified and supports parametric analysis.

On this particular set of benchmarks, our prototype used less memory and was more numerically robust. However, these symptoms may not be due to fundamental differences, they might both be local and easily fixed artifacts of the current HyTech implementation. Improved numeric robustness might also be due to our use of a floating point Simplex implementation, a theoretically questionable aspect of our current prototype.

HyTech first enumerates the reachable discrete state space then enumerates the polyhedra. We do on-the-fly reachability, which may suppress enumeration of some discrete system states for some problems because edge guards may prevent transitions that would otherwise occur in the purely discrete model. However, in our set of benchmarks all of the discrete states were reached anyway.

By making scheduling a part of the model semantics rather than a part of the model itself, we reduced the size of the discrete state space from  $3^t$  to  $2^t$  where  $t$  is the number of tasks.<sup>1</sup> We cannot say with certainty

---

<sup>1</sup>Although each task in the HyTech model had four discrete states, transitions to rescheduling states were forced to be simultaneous in many cases, and the size of the reachable discrete state space was only  $3^t$  instead of  $4^t$ .

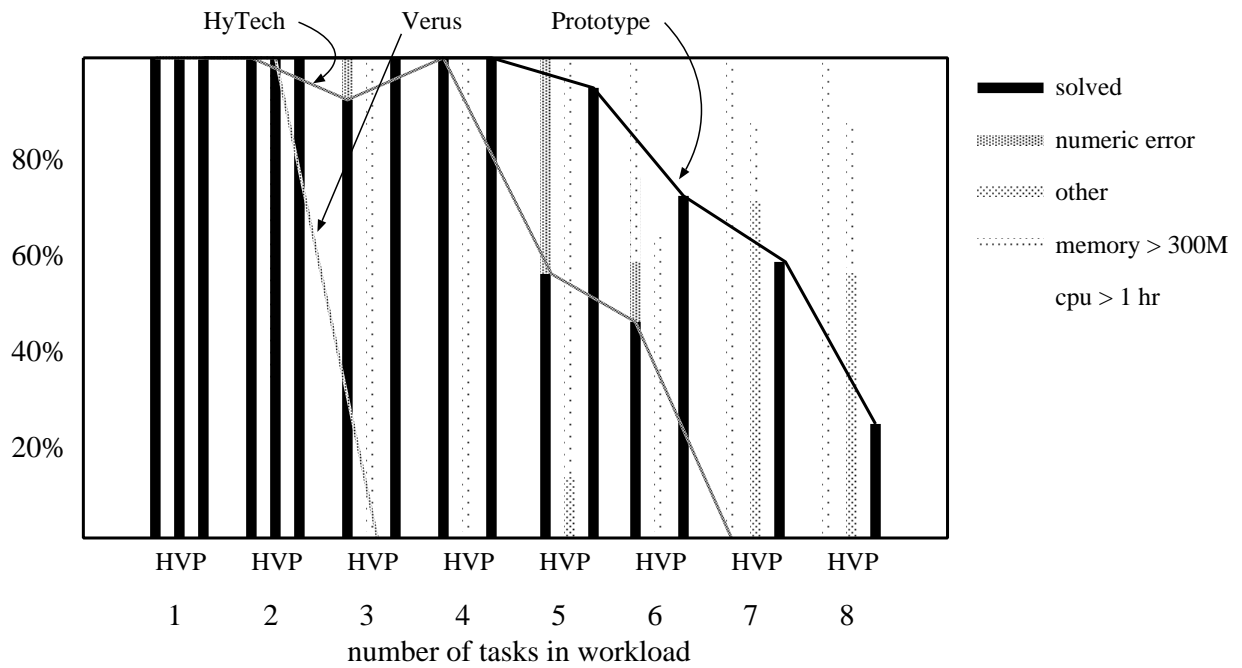


Figure 4: Percentage of Generated Problems That Were Solved

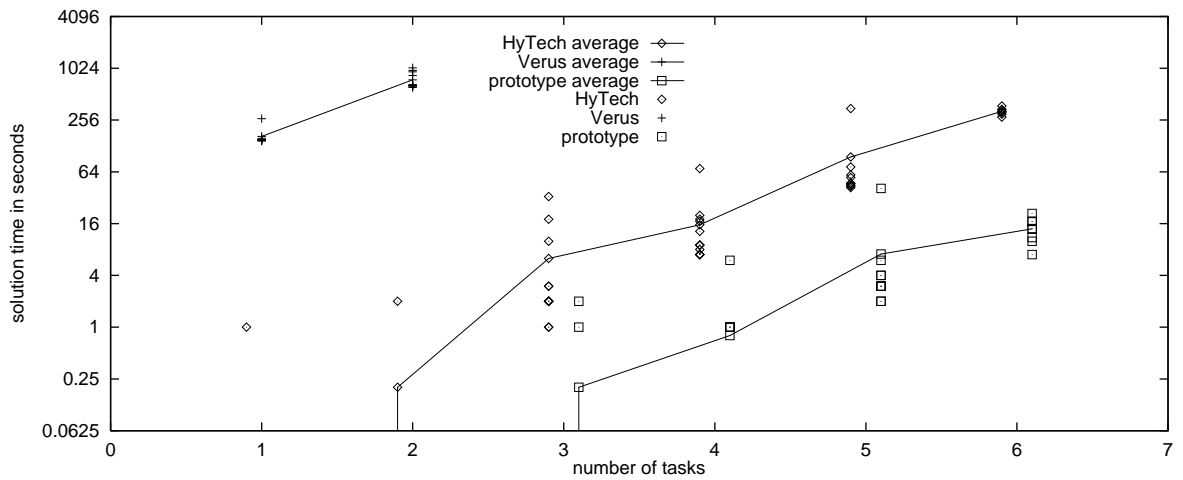


Figure 5: Solution Times for Problems That Were Solved

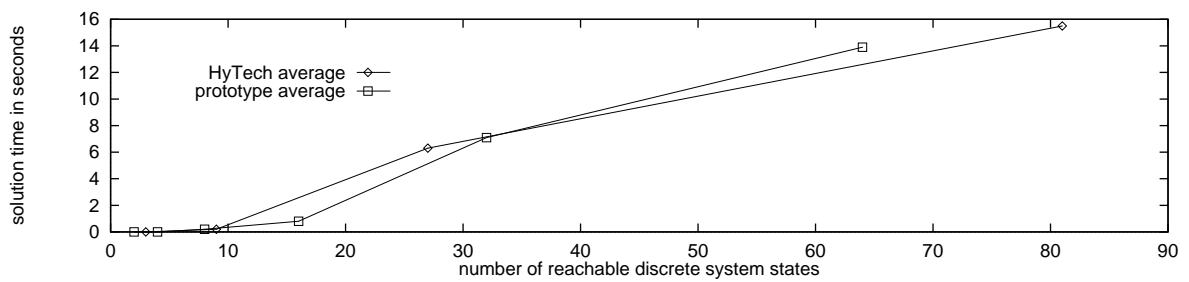


Figure 6: Average Solution Times as a Function of the Number of Discrete System States

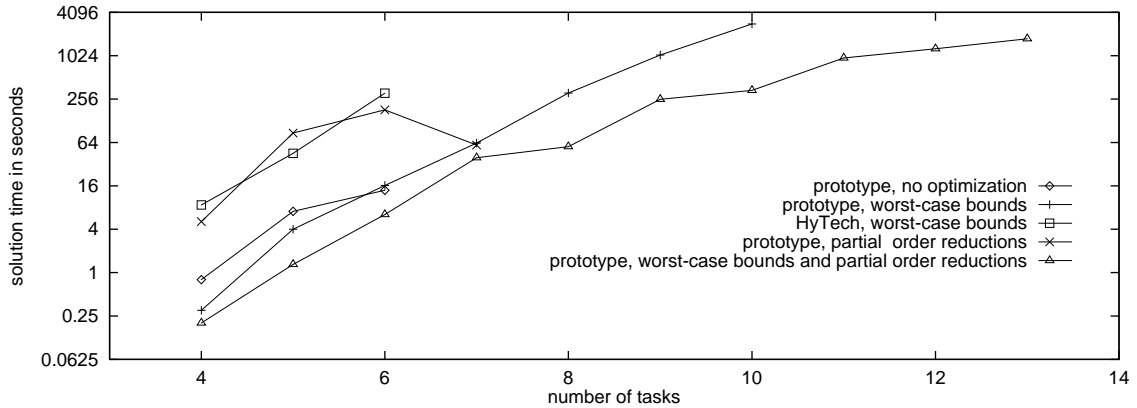


Figure 7: Prototype Average Solution Times with Optimization Methods

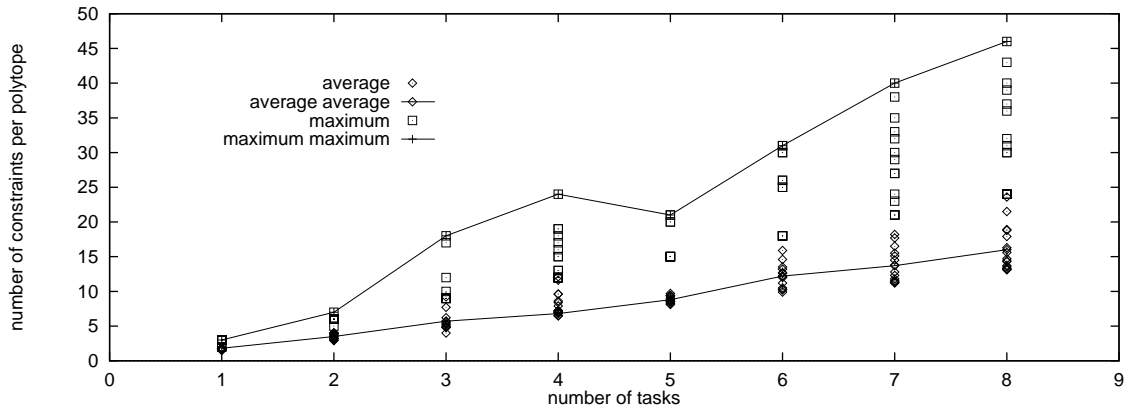


Figure 8: Average and Maximum Number of Constraints per Polyhedra

how much of the performance improvement might be due to our different polyhedra operations versus the incorporation of scheduling semantics into the model. Figure 6 shows a plot of solution time versus number of reachable discrete system states rather than number of tasks, which seems to suggest the two different sets of polyhedra operations are about equally efficient. However, our prototype may be operating on more complex polyhedra, since information encoded in discrete states in HyTech must now be encoded in the polyhedra in our prototype. Also, with fewer discrete states, the set of polyhedra that must be checked for containment for each discrete state is larger for our prototype (the loop that searches for containing polyhedra accounts for most of the execution time). It may or may not be the case that some of HyTech's numeric problems occur when attempting to operate on polyhedra that are difficult for those algorithms in some fundamental way.

Classical uniprocessor schedulability analysis methods rely on the fact that analysis can be performed using only minimum periods and maximum compute times[19]. We simplified our models in this way. In

our models there were frequently cases where periodic tasks were simultaneously dispatched. We added a test for these cases and performed such transitions concurrently, which is a partial order reduction method. Figure 7 compares the average solution times with and without these methods using a logarithmic scale. All solved problems were shown for all tools and methods at each point for which more than 75% of the problems were solved. The partial order method alone is actually slower than the unoptimized version for the task sets that could be analyzed in less than 1 hour, presumably because the time required to operate on polyhedra and the number of regions dominated the growth in the number of discrete system states to the extent that the additional testing required for simultaneous transitions (which requires among other things extra feasibility tests) is not worth the reduction in discrete state space size. As the figure suggests, partial order reduction did result in a somewhat higher percentage of problems being solved within one hour. The use of worst case values is significantly beneficial, this significantly reduces the number of enumerated regions. What

this figure does not show is that this also significantly reduces the variability in number of regions and solution times. The figure shows that the two optimization methods are complementary and synergistic.

Figure 8 shows the average and maximum number of constraints required to represent polyhedra as a function of the number of tasks in a model. These experiments suggest that in practice the size of constraint sets grows roughly linearly with the size of models of this type.

## 7 Remarks

Our prototype tool and benchmarking exercises used only forward reachability analysis. It should be possible to perform a backward reachability analysis by using negated rates in the time successor operation, which would essentially run time backwards to obtain the time predecessor of a polyhedra[2, 3].

We attempted to generate problems that intuitively resembled real-world schedulability problems, but randomly generated problems may not be reflective of problems that would be encountered in practice. Moreover, the models we generated were precisely what would not be analyzed using linear hybrid automata techniques, since these models were amenable to analysis using traditional methods. Experience is needed with models that include features such as distributed execution, remote procedure calls, rendezvous, etc., the kinds of models for which we intended this analysis technique. Our preliminary experience with some simple distributed scheduling problems and with a software verification exercise suggests that our method works as well on these problems as on the uniprocessor benchmarks discussed in this paper[24].

We observed an increased ability to solve problems when only maximum compute times and minimum periods were used, which classical preemptive fixed priority theory tells us is sufficient for noninteracting tasks on uniprocessors. It is known that this is not true in certain multi-processor situations[10], and this is likely inadvisable when applying these methods to verification problems. It would be useful to identify more general conditions under which selected intervals in a model could be replaced by a scalar, or be expanded to a containing interval, in a way that significantly reduces the number of regions.

We added scheduling semantics to a linear hybrid automata model rather than explicitly modeling scheduling behavior using standard hybrid automata. The basic idea, which is to determine the variable rates (or edge guards or assignments) using a computable function of the composed system discrete states rather than a simple union of rates (or edge guards or assignments) specified in separate automata, might be applicable in other problem domains.

Our results suggest that the computational complexity of performing a reachability analysis does not lie in the complexity of the individual polyhedra operations but in the possible combinatorial explosion in the number of reachable regions, at least in practice in the problem domain we studied. Our work illustrates how the incorporation of domain semantics into the model and the use of a partial order reduction method can be effective in reducing the growth in the size of the discrete state space. An on-the-fly method might reduce the discrete state space size in some problem domains, but not in the one we studied. Several other researchers have explored methods that automatically approximate sets of polyhedra using a containing polyhedron to reduce the growth in the region space size[12, 16, 13], although we found during some preliminary experiments with some simple methods that it was difficult to simultaneously achieve acceptable accuracy and significant performance improvements. Our work illustrates another approach, conservative modifications to model parameters to better condition a model, that also holds some promise.

## References

- [1] K. Altisen, G. Göbler, A. Pnueli, J. Sifakis, S. Tripakis and S. Yovine, "A Framework for Scheduler Synthesis," *Real-Time Systems Symposium*, December 1999.
- [2] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho, "Automatic Symbolic Verification of Embedded Systems," *IEEE Transactions on Software Engineering*, vol. 22, no. 3, March 1996, pp 181-201.
- [3] Johan Bengtsson and Fredrik Larsson, *UPPAAL, A Tool for Automatic Verification of Real-Time Systems*, DoCS 96/97, Department of Computer Science, Uppsala University, January 15, 1996.
- [4] B. A. Brandin and W. M. Wonham, "Supervisory Control of Timed Discrete-Event Systems," *IEEE Transactions on Automatic Control*, v39, n2, February 1994.
- [5] A. L. Brearly, G. Mitra and H. P. Williams, "Analysis of Mathematical Programming Problems Prior to Applying the Simplex Algorithm," *Mathematical Programming*, 8, p54-83.
- [6] David Bremner, personal communication, University of Washington Department of Mathematics, 1999.
- [7] S. Campos, E. Clarke, W. Marrero, M. Minea and H. Hiraishi, "Computing Quantitative Characteristics of Finite-State Real-Time Systems," *Real-Time Systems Symposium*, December 1994.

- [8] David L. Dill, "Timing Assumptions and Verification of Finite-State Concurrent Systems," *International Workshop on Automatic Verification Methods for Finite State Systems*, Grenoble, France, June 12-14, 1989, also in *Lecture Notes in Computer Science 407*, J. Sifakis (Ed.), Springer-Verlag, pp 197-212.
- [9] Saul I. Gass, *Linear Programming*, McGraw-Hill Book Company, New York.
- [10] R. L. Graham, "Bounds on Multiprocessing Timing Anomalies," *SIAM Journal of Applied Mathematics*, Vol. 17, No. 2, March 1969.
- [11] Ronald L. Graham, Donald E. Knuth and Oren Patashnik, *Concrete Mathematics, A Foundation for Computer Science*, Addison-Wesley, 1989.
- [12] Nicolas Halbwachs, Yann-Erik Proy and Patrick Roumanoff, "Verification of Real-Time Systems using Linear Relation Analysis," *Formal Methods in System Design*, 11(2):157-185, August 1997.
- [13] Nicolas Halbwachs, Pascal Raymond and Yann-Erik Proy, "Verification of Linear Hybrid Systems by Means of Convex Approximations," *Workshop on Verification and Control of Hybrid Systems*, Piscataway, NJ, October 1995.
- [14] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri and Pravin Varaiya, "What's Decidable About Hybrid Automata?" *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, 1995.
- [15] Thomas A. Henzinger, Pei-Hsin Ho and Howard Wong-Toi, "HyTech: The Next Generation," *Real-Time Systems Symposium*, December 1995.
- [16] Thomas A. Henzinger and Pei-Hsin Ho, "A Note On Abstract Interpretation Strategies for Hybrid Automata," *Hybrid Systems II*, also *Lecture Notes in Computer Science 999*, Springer-Verlag, 1995.
- [17] James P. Ignizio, *Linear Programming in Single- and Multiple- Objective Systems*, Prentice-Hall.
- [18] J. Lehoczky, L. Sha and Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior," *IEEE Real-Time Systems Symposium*, 1989, pp 166-171.
- [19] C. L. Liu and James W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *Journal of the ACM*, v20, n1, January 1973, pp 46-61.
- [20] Peter J. G. Ramadge and W. Murray Wonham, "The Control of Discrete Event Systems," *Proceedings of the IEEE*, v77, n1, January 1989.
- [21] Karsten Strehl, Lothar Thiele, Dirk Ziegenbein, Rolf Ernst and Jürgen Teich, "Scheduling Hardware/Software Systems Using Symbolic Techniques," *Proceedings of the 7<sup>th</sup> International Workshop on Software/Hardware Codesign*, Rome, Italy, May 3-5, 1999, pp 173-177.
- [22] Steve Vestal, "Fixed Priority Sensitivity Analysis for Linear Compute Time Models," *IEEE Transactions on Software Engineering*, April 1994.
- [23] Steve Vestal, "Linear Hybrid Automata Models of Real-Time Scheduling and Allocation in Distributed Heterogeneous Systems," Honeywell Technology Center, 3660 Technology Drive, Minneapolis, MN 55418, 1999.
- [24] Steve Vestal, "Formal Verification of the MetaH Executive Using Linear Hybrid Automata," Honeywell Technology Center, Minneapolis, MN 55418, December 1999.