HyTech : The Cornell HYbrid TECHnology Tool*[†]

Thomas A. Henzinger[‡] Pei-Hsin Ho

Computer Science Department, Cornell University, Ithaca, NY 14853
(tah|ho)@cs.cornell.edu

Abstract. This paper is addressed to potential users of HyTech, the Cornell Hybrid Technology Tool, an automatic tool for analyzing hybrid systems. We review the formal technologies that have been incorporated into HyTech, and we illustrate the use of HyTech with three nontrivial case studies.

1 Introduction

Hybrid systems are digital real-time systems that interact with the physical world through sensors and actuators. Due to the rapid development of digital processor technology, hybrid systems directly control much of what we depend on in our daily lives. Many hybrid systems, ranging from automobiles to aircraft, operate in safety-critical situations, and therefore call for rigorous analysis techniques.

HyTech[1] is a symbolic model checker for linear hybrid systems. The underlying system model is *hybrid automata*, an extension of finite automata with continuous variables that are governed by differential equations [ACHH93]. The requirement specification language is the *integrator computation tree logic* Ictl, a branching-time logic with clocks and stop-watches for specifying timing constraints. Safety, liveness, real-time, and duration requirements of hybrid systems can be specified in Ictl [AHH93]. Given a hybrid automaton describing a system and an Ictl formula describing a requirement, HyTech computes the state predicate that characterizes the set of system states that satisfy the requirement.

In this report we review the formal technologies that have been incorporated into HyTech. In Section 2, we define the syntax and semantics of linear hybrid automata, which were introduced in [ACHH93, NOSY93]. In Section 3, we give an introduction to Ictl model checking and the reachability analysis of linear hybrid automata, which was presented in [AHH93, ACH$^+$95]. We concentrate on the analysis of systems with unknown delay parameters, and use HyTech to derive sufficient and necessary conditions on the parameters such that the system satisfies a given Ictl requirement. We also demonstrate the use of abstract-interpretation operators, which are discussed

*This research was supported in part by the National Science Foundation under grant CCR-9200794, by the Air Force Office of Scientific Research under contract F49620-93-1-0056, by the Office of Naval Research under YIP grant N00014-95-1-0520, and by the Defense Advanced Research Projects Agency under grant NAG2-892.

†This paper will appear in the proceedings of the *Workshop on Hybrid Systems and Autonomous Control* held in Ithaca, NY, in October, 1994.

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

1 HyTech is available by anonymous ftp from ftp.cs.cornell.edu, cd ~pub/tah/HyTech. See also http://www.cs.cornell.edu/Info/People/tah/hytech.html.

in greater detail in [HH95b]. In Section 4, we indicate how nonlinear hybrid systems can be translated into linear hybrid automata, so that linear analysis techniques apply [HH95a]. Throughout, we use a temperature controller for a toy nuclear reactor as a running example to illustrate the use of HyTech. For the practitioners, we present the actual input language for describing linear hybrid automata and verification commands.

In Section 5, we apply HyTech to three nontrivial benchmark problems. All three examples are taken from the literature, rather than devised by us. The first case study is a distributed control system introduced by Corbett [Cor94]. The system consists of a controller and two sensors, and is required to issue control commands to a robot within certain time limits. The two sensor processes are executed on a single processor, as scheduled by a priority scheduler. This scenario is modeled by linear hybrid automata with clocks and stop-watches. HyTech automatically computes the maximum time difference between two consecutive control commands generated by the controller. It follows, for example, that a scheduler that gives higher priority to one sensor may meet the specification requirement, while a scheduler that gives priority to the other sensor may fail the requirement.

The second case study is a two-robot manufacturing system introduced by Puri and Varaiya [PV95]. The system consists of a conveyor belt with two boxes, a service station, and two robots. The boxes will not fall to the floor iff initially the boxes are not positioned closely together on the conveyor belt. HyTech automatically computes the minimum allowable initial distance between the two boxes.

The third case study is the Philips audio control protocol presented by Bosscher, Polak, and Vaandrager [BPV94]. The protocol consists of a sender that converts a bit string into an analog signal using the so-called Manchester encoding, and a receiver that converts the analog signal back into a bit string. The sender and the receiver use clocks that may be drifting apart. In [BPV94], it was shown, by a human proof, that the receiver decodes the signal correctly if and only if the clock drift is bounded by a certain constant. HyTech automatically computes that constant for input strings up to 8 bits. With some extra care in modeling, HyTech can also be used to analyze the general case of input strings with arbitrary length [HW95].

2 Specification of Linear Hybrid Automata in HyTech

The system modeling language of HyTech is linear hybrid automata [AHH93]. Intuitively, a linear hybrid automaton is a labeled multigraph (V, E) with a finite set X of real-valued variables. The edges in E represent discrete system actions and are labeled with guarded assignments to X. The vertices in V represent continuous environment activities and are labeled with constraints on the variables in X and their first derivatives. The state of a hybrid automaton changes either through instantaneous system actions or, while time elapses, through continuous environment activities.

Example: reactor temperature control

We use a variant of the reactor temperature control system from [NOSY93] as a running example. The system consists of a reactor core and two control rods that control the temperature of the reactor core. The reactor core is modeled by the linear hybrid automaton in Figure 1. The temperature of the reactor core is represented by the variable x. Initially the core temperature is 510 degrees and both control rods are not in the reactor core. In this case, the core temperature rises at a rate that varies between 1 and 5 degrees per second. We use $\dot{x}$ to denote the first derivative of the variable x. If the core temperature reaches 550 degrees, one of two control rods can be put

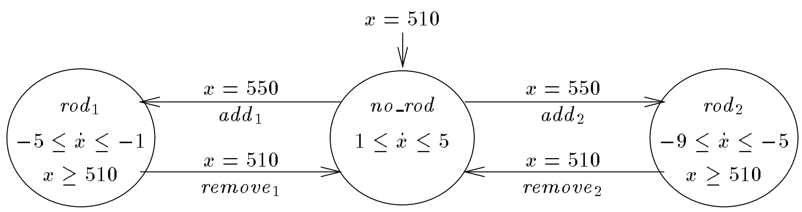

Figure 1: The reactor core automaton

into the reactor core to dampen the reaction. If control rod 1 is put in, the core temperature falls at a rate that may vary between -5 and -1 degrees per second. Control rod 2 has a stronger effect; if it is put in, the core temperature falls at a rate that varies between -9 and -5 degrees per second. Either control rod is removed once the core temperature falls back to 510 degrees.

2.1 Syntax

A *linear term* over a set X of real-valued variables is a linear combination of variables with integer coefficients. A *linear inequality* over X is a nonstrict inequality between linear terms over X.[2] A *linear hybrid automaton* A consists of the following components.

Data variables A finite ordered set $X = \{x_1, x_2, \ldots, x_n\}$ of real-valued *data variables*. For example, the reactor core automaton from Figure 1 has the single data variable x.

A *data state* is a point $(a_1, a_2, \ldots, a_n)$ in the n-dimensional real space $\mathbb{R}^n$ or, equivalently, a function that maps each variable x_i to a real value a_i. A *convex data region* is a convex polyhedron in $\mathbb{R}^n$, and a *data region* is a finite union of convex data regions. A *convex data predicate* is a conjunction of linear inequalities, and a *data predicate* is a disjunction of convex data predicates. Every (convex) data predicate ϕ defines a (convex) data region $[\![\phi]\!]$ of data states that satisfy ϕ.

Control locations A finite set V of vertices called *control locations*. For example, the reactor core automaton has the three control locations *no_rod*, rod_1, and rod_2.

A *state* (v, s) of the hybrid automaton A consists of a control location $v \in V$ and a data state $s \in \mathbb{R}^n$. A *region* $\bigcup_{v \in V}\{(v, S_v)\}$ is a collection of data regions $S_v \subseteq \mathbb{R}^n$, one for each control location $v \in V$. A *state predicate* is a collection $\bigcup_{v \in V}\{(v, \phi_v)\}$ of data predicates ϕ_v, one for each control location $v \in V$. When writing state predicates, we use the *location counter* l, which ranges over the set V of control locations. The location constraint $l = v$ denotes the state predicate $\{(v, \mathit{true})\} \cup \bigcup_{v' \neq v}\{(v', \mathit{false})\}$. Each state predicate $\bigcup_{v \in V}\{(v, \phi_v)\}$ defines the region $\bigcup_{v \in V}\{(v, [\![\phi_v]\!])\}$.

Location invariants A labeling function *inv* that assigns to each control location $v \in V$ a convex data predicate $\mathit{inv}(v)$, the *invariant* of v. The automaton control may reside in location v only as long as the invariant $\mathit{inv}(v)$ is true; so the invariants enforce progress in a hybrid automaton. The state (v, s) is *admissible* if the data state s satisfies the invariant $\mathit{inv}(v)$. We write Σ_A for the region $\bigcup_{v \in V}\{(v, [\![\mathit{inv}(v)]\!])\}$ of all admissible states of A.

[2] The restriction to *nonstrict* inequalities is not essential, but forced upon us by the polyhedron-manipulation library we use in the current implementation of HyTech.

In the graphical representation of a hybrid automaton, we suppress invariants of the form *true*. In the reactor core automaton, we have $inv(no\_rod) = true$, $inv(rod_1) = (x \geq 510)$, and $inv(rod_2) = (510 \leq x)$. In HYTECH, we specify these invariants as follows:

```
inv[l[core] == norod] = True
inv[l[core] == rodone] = 510<=x
inv[l[core] == rodtwo] = 510<=x
```

Continuous activities A labeling function *dif* assigns to each control location $v \in V$ and each data variable $x_i \in X$ a *rate interval* $dif(v, x_i) = [a_i, b_i]$, where a_i and b_i are integer constants. The rate interval $dif(v, x_i) = [a_i, b_i]$ specifies that the first derivative of the data variable x_i may vary within the interval $[a_i, b_i] \subset \mathbb{R}$ while the automaton control resides in location v. If $a_i = b_i$, then $dif(v, x_i)$ is called the *slope* of x_i in location v. A data variable is a *discrete variable* if it has the slope 0 in all locations; a *clock*, if it has the slope 1 in all locations; and a *stop-watch*, if in each location it has either the slope 1 or the slope 0.

In the graphical representation of a hybrid automaton, we write $\dot{x} = a$ short for $\dot{x} \in [a, a]$, and we suppress rate intervals of the form $\dot{x} = 0$. In the reactor core automaton, $dif(no\_rod, x) = [1, 5]$, $dif(rod_1, x) = [-5, -1]$, and $dif(rod_2, x) = [-9, -5]$. In HYTECH, we specify these rate intervals as follows:

```
dif[core,norod,x] = {1,5}
dif[core,rodone,x] = {-5,-1}
dif[core,rodtwo,x] = {-9,-5}
```

Transitions A finite multiset E of edges called *transitions*. Each transition (v, v') identifies a source location $v \in V$ and a target location $v' \in V$. The reactor core automaton has four transitions.

Synchronization letters A finite set L of letters called *synchronization alphabet*, and a labeling function *syn* that assigns to each transition $e \in E$ a letter from L. The synchronization letters are used to define the parallel composition of hybrid automata. In the graphical representation of a hybrid automaton, we suppress synchronization letters that do not occur in the alphabet of any other automata. The reactor core automaton has the four synchronization letters add_1, add_2, $remove_1$, and $remove_2$.

Discrete actions A labeling function *act* that assigns to each transition $e \in E$ a guarded command $act(e) = (\phi \rightarrow \alpha)$. The *guard* ϕ is a convex data predicate. The *command* α is a set of assignments $x_i := t_i$, at most one for each data variable $x_i \in X$, such that each t_i is a linear term over X. We write $dom(\alpha)$ for the set of variables that make up the left-hand sides of the assignments in α, and $t_i(s)$ for the value of the linear term t_i if interpreted in the data state s. The command α defines a function on data states that leaves the variables outside $dom(\alpha)$ unchanged: for all data states s, $\alpha_i(s) = t_i(s)$ if $x_i \in dom(\alpha)$, and $\alpha_i(s) = s_i$ if $x_i \notin dom(\alpha)$, where $\alpha_i(s)$ denotes the i-th component of the data state $\alpha(s)$. A *parameter* is a discrete variable that does not occur in the domain $dom(\alpha)$ of any command α.

In the graphical representation of a hybrid automaton, we write α for the guarded command $true \rightarrow \alpha$, and ϕ for the guarded command $\phi \rightarrow \emptyset$, and we suppress the guarded command $true \rightarrow \emptyset$. In the reactor core automaton, $act(no\_rod, rod_1) = (x = 550 \rightarrow \emptyset)$, etc. In HYTECH, we specify the transitions, synchronization letters, and guarded commands of the reactor core automaton as follows:

```
act[core,1]={l[core]==norod && 550==x, add1, {l[core]->rodone}}
act[core,2]={l[core]==norod && 550==x, add2, {l[core]->rodtwo}}
act[core,3]={l[core]==rodone && 510==x, remove1, {l[core]->norod}}
act[core,4]={l[core]==rodtwo && 510==x, remove2, {l[core]->norod}}
```

Notice that we encode the source and target locations of a transition within a guarded command.

2.2 Semantics

At any time instant, the state of a hybrid automaton specifies a control location and the values of all data variables. The state can change in two ways: (1) by an instantaneous discrete transition that changes both the control location and the values of data variables, or (2) by a time delay that changes only the values of data variables in a continuous manner according to the rate intervals of the corresponding control location. Accordingly, we define the following two binary relations on the admissible states of the given automaton A.

Transition step For all admissible states (v, s) and (v', s') of A, and all synchronization letters σ, let $(v, s) \xrightarrow{\sigma} (v', s')$ iff there exists a transition e from v to v' such that (1) $syn(e) = \sigma$ and (2) $act(e) = (\phi \rightarrow \alpha)$ with $s \in [\![\phi]\!]$ and $s' = \alpha(s)$.

Time step For all admissible states (v, s) and (v, s') of A, and all nonnegative reals $\delta \geq 0$, let $(v, s) \xrightarrow{\delta} (v, s')$ iff there is a differentiable function $\rho : [0, \delta] \rightarrow \mathbb{R}^n$ such that (1) $f(0) = s$, (2) $f(\delta) = s'$, (3) for all reals $t \in [0, \delta]$, $\rho(t) \in [\![inv(v)]\!]$, and (4) for all reals $t \in (0, \delta)$ and each data variable x_i, $d\rho_i(t)/dt \in dif(v, x_i)$, where $\rho_i(t)$ denotes the i-th component of the data state $\rho(t)$.

The linear hybrid automaton A defines the labeled transition system $[\![A]\!] = \langle \Sigma_A, \mathcal{L}, \rightarrow_A \rangle$ that consists of the infinite state space Σ_A, the infinite label set $\mathcal{L} = L \cup \mathbb{R}_{\geq 0}$, and the binary transition relation $\rightarrow_A = \bigcup\{\xrightarrow{\sigma} \mid \sigma \in L\} \cup \bigcup\{\xrightarrow{\delta} \mid \delta \geq 0\}$ on Σ_A.

For a region S, we define $pre(S)$ to be the set of all states σ such that $\sigma \rightarrow_A \sigma'$ for some state $\sigma' \in S$. Similarly, we define $post(S)$ to be the set of all states σ such that $\sigma' \rightarrow_A \sigma$ for some state $\sigma' \in S$. Both $pre(S)$ and $post(S)$ are again regions [AHH93]. We write $pre^*(S)$ for the infinite union $\bigcup_{i \geq 0} pre^i(S)$, and $post^*(S)$ for the infinite union $\bigcup_{i \geq 0} post^i(S)$. In other words, $pre^*(S)$ is the set of all states that can reach a state in S by a finite sequence of transitions of the labeled transition system $[\![A]\!]$; and $post^*(S)$ is the set of all states that can be reached from a state in S by a finite sequence of transitions of $[\![A]\!]$.

2.3 Parallel Composition

A hybrid system typically consists of several components that operate concurrently and communicate with each other. We describe each component as a linear hybrid automaton. The component automata may coordinate either through shared variables or via synchronization letters. The linear hybrid automaton that models the entire system is then constructed from the component automata using a product operation.

Let $A_1 = (X_1, V_1, inv_1, dif_1, E_1, L_1, syn_1, act_1)$ and $A_2 = (X_2, V_2, inv_2, dif_2, E_2, L_2, syn_2, act_2)$ be two linear hybrid automata. The product automaton $A_1 \times A_2$ generally interleaves the transitions of the component automata A_1 and A_2. If, however, a transition e_1 of A_1 is labeled with a synchronization letter σ that is contained also in the alphabet of A_2, then e_1 can be executed only

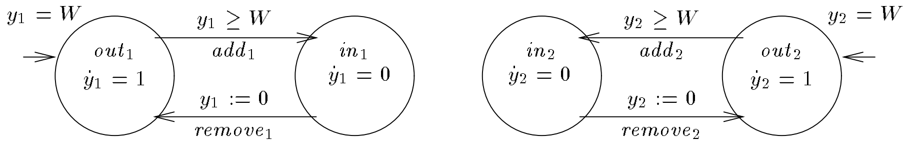

Figure 2: The control rod automata

simultaneously with a σ-labeled transition of A_2. Formally, the *product* $A_1 \times A_2$ is the linear hybrid automaton $A = (X_1 \cup X_2, V_1 \times V_2, inv, dif, E, L_1 \cup L_2, syn, act)$:

- Each location (v, v') in $V_1 \times V_2$ has the invariant $inv(v, v') = (inv_1(v) \wedge inv_2(v'))$. For each variable $x \in X_1 \backslash X_2$, $dif((v, v'), x) = dif_1(v, x)$; for each variable $x \in X_2 \backslash X_1$, $dif((v, v'), x) = dif_2(v', x)$; and for each shared variable $x \in X_1 \cap X_2$, $dif((v, v'), x) = dif_1(v, x) \cap dif_2(v', x)$.

- E contains the transition $e = ((v_1, v_2), (v_1', v_2'))$ iff

  (1) $e_1 = (v_1, v_1') \in E_1$, $v_2 = v_2'$, and $syn_1(e_1) \notin L_2$; or
  (2) $e_2 = (v_2, v_2') \in E_2$, $v_1 = v_1'$, and $syn_2(e_2) \notin L_1$; or
  (3) $e_1 = (v_1, v_1') \in E_1$, $e_2 = (v_2, v_2') \in E_2$, and $syn_1(e_1) = syn_2(e_2)$.

  Suppose that $act_1(e_1) = (\phi_1 \rightarrow \alpha_1)$, and $act_2(e_2) = (\phi_2 \rightarrow \alpha_2)$. In case (1), $syn(e) = syn_1(e_1)$ and $act(e) = act_1(e_1)$. In case (2), $syn(e) = syn_2(e_2)$ and $act(e) = act_2(e_2)$. In case (3), $syn(e) = syn_1(e_1) = syn_2(e_2)$; moreover, $act(e) = (\phi_1 \wedge \phi_2 \rightarrow \alpha_1 \cup \alpha_2)$ if $dom(\alpha_1) \cap dom(\alpha_2) = \emptyset$, and $act(e) = (false \rightarrow \emptyset)$ if $dom(\alpha_1) \cap dom(\alpha_2) \neq \emptyset$,

HyTech automatically constructs the product automaton from a set of input automata.

For the reactor example, we use the two linear hybrid automata of Figure 2 to model the two control rods. Due to the mechanics of moving control rods, after a control rod is removed from the reactor core, it cannot be put back into the core for W seconds, where W is an unknown parameter. This requirement is enforced by the stop-watch y_1 that measures the time that has elapsed since control rod 1 was removed from the reactor core, and the stop-watch y_2 that measures the time that has elapsed since control rod 2 was removed. The rod automata synchronize with the core automaton through synchronization letters such as $remove_1$, which indicates the removal of control rod 1. The entire reactor system, then, is obtained by constructing the product of the core automaton of Figures 1 and the two rod automata of Figure 2.

We now show how the complete reactor temperature control system is specified in HyTech. First we declare the data variables:

```
AnaVariables = {x, y1, y2}
DisVariables = {w}
```

The data variables x, $y1$, and $y2$ are analog variables, and the data variable W is a discrete variable. We have already defined the reactor core automaton. Now we define the two control rod automata:

```
inv[l[rod1] == out] = 0<=y1
inv[l[rod1] == in] = 0<=y1
inv[l[rod2] == out] = 0<=y2
inv[l[rod2] == in] = 0<=y2

dif[rod1,in,y1] = {0,0}
dif[rod1,out,y1] = {1,1}
```

```
dif[rod2,in,y2] = {0,0}
dif[rod2,out,y2] = {1,1}

act[rod1,1] = { l[rod1]==out && w<=y1, add1, {l[rod1] -> in}}
act[rod1,2] = { l[rod1]==in, remove1, {l[rod1] -> out, y1 -> 0}}
act[rod2,1] = { l[rod2]==out && w<=y2, add2, {l[rod2] -> in}}
act[rod2,2] = { l[rod2]==in, remove2, {l[rod2] -> out, y2 -> 0}}
```

The synchronization alphabet of each automaton is defined by declaring a scope for each synchronization letter. The *scope* of the letter σ is the set of automata that contain σ in their synchronization alphabet. For the reactor temperature control system, we specify

```
syn[remove1] = {rod1,core}
syn[remove2] = {rod2,core}
syn[add1] = {rod1,core}
syn[add2] = {rod2,core}
```

For example, the letter $remove_1$ is used by the reactor core automaton and by the first control rod automaton. This means that the core automaton and the rod 1 automaton must synchronize on transitions labeled with $remove_1$.

While we have given symbolic names like *core* and *no_rod* to automata and locations, the analysis procedures of HyTech require that all automaton names and location names are integers starting from 1. To replace the symbolic names with integers, HyTech calls a macro language preprocessor `m4` when it reads an input file. Therefore, we need to define the integer values of the symbolic names at the beginning of the input file. The symbolic names that we use for the reactor temperature control system may be defined as follows:

```
define(rod1,1)
define(rod2,2)
define(core,3)
define(rodone,1)
define(rodtwo,2)
define(norod,3)
define(out,1)
define(in,2)
```

We also must declare the number of input automata, and the number of locations and transitions of each automaton:

```
AutomataNo = 3
locationo = {2,2,3}
transitiono = {2,2,4}
```

The expression `locationo = {2,2,3}` means that the first (control rod 1), second (control rod 2), and third (reactor core) automaton has 2, 2, and 3 locations, respectively. The expression `transitiono = {2,2,4}` specifies the number of transitions in each input automaton.

Global invariants for modeling urgent transitions

Although the product automaton is constructed automatically by HyTech, it is sometimes useful to specify global conjuncts of all invariants of the product automaton. Such global invariants permit,

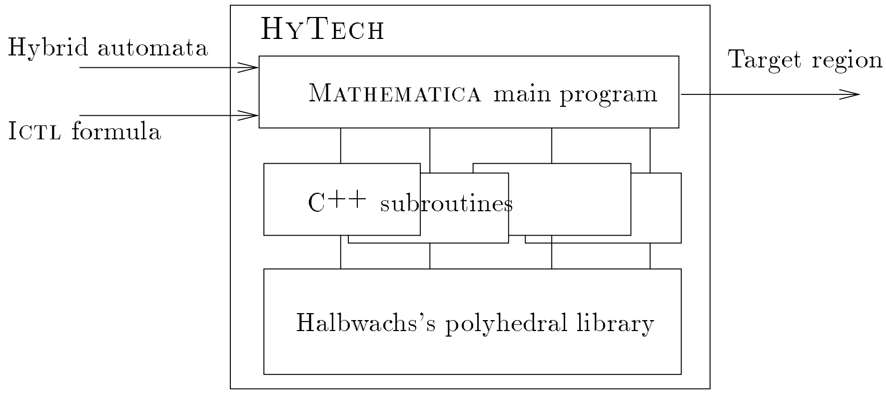

Figure 3: The architecture of HyTech

in particular, the modeling of *urgent transitions*, which are transitions that must be taken as soon as possible. In the graphical representation of hybrid automata, we use boldface synchronization letters to mark urgent transitions. HyTech allows the user to specify location invariants for locations of the product automaton using the command `GlobalInvar`. We will show how urgent transitions can be modeled with global invariants as we analyze the examples of Section 4. The reactor temperature control system does not have any urgent transitions, so we write:

```
GlobalInvar = {}
```

This completes the specification of the reactor temperature control system. Except for initial `define` statements, all HyTech input commands can be written in any order.

3 Symbolic Analysis of Linear Hybrid Automata in HyTech

The core of HyTech is a symbolic model-checking procedure, whose primitives are *pre*, *post*, and boolean operations on regions. The original implementation of HyTech represented regions as state predicates and manipulated regions by syntactic operations on formulas. We have improved the performance of HyTech by representing and manipulating regions geometrically: each data region is represented as a union of convex polyhedra. The current implementation of HyTech consists of a Mathematica main program and a collection of C++ subroutines that make use of a polyhedron-manipulation library by Halbwachs [Hal93, HRP94]. The architecture of HyTech is shown in Figure 3.

3.1 Reachability Analysis

The *reachability problem* $(A, \varphi_I, \varphi_F)$ for a linear hybrid automaton A, an initial state predicate φ_I, and a final state predicate φ_F, asks if the region $post^*([\![\varphi_I]\!]) \cap [\![\varphi_F]\!]$ is empty or, equivalently, if the region $[\![\varphi_I]\!] \cap pre^*([\![\varphi_F]\!])$ is empty. In other words, the reachability problem $(A, \varphi_I, \varphi_F)$ asks if there is no finite path in the underlying transition system $[\![A]\!]$ from some state in $[\![\varphi_I]\!]$ to some state in $[\![\varphi_F]\!]$. If $[\![\varphi_I]\!]$ represents the set of "initial" states of the automaton A, and $[\![\varphi_F]\!]$ represents the set of "unsafe" states specified by a safety requirement, then the safety requirement can be verified by reachability analysis: the automaton satisfies the safety requirement iff the reachability problem has the answer *yes* (i.e., $post^*([\![\varphi_I]\!]) \cap [\![\varphi_F]\!] = \emptyset$).

Unfortunately, the computation of $post^*([\![\varphi_I]\!])$ or $pre^*([\![\varphi_F]\!])$ may not terminate within a finite number of *post* or *pre* operations, because the reachability problem for linear hybrid automata is undecidable [ACHH93]. HyTech, in other words, offers a semidecision procedure for the reachability analysis. It is our experience, however, that for practical examples, including the examples in this paper, the computation does terminate and HyTech solves the corresponding reachability problems. Indeed, as for the practitioner there is little difference between a nonterminating computation and one that runs out of time or space resources, we submit that decidability questions are mostly of theoretical interest.

Suppose that in the reactor temperature control system, the reactor needs to be shut down if the core temperature exceeds 550 degrees. We wish to check the safety requirement that *the reactor never needs to be shut down*; more precisely, whenever the core temperature reaches 550 degrees, then either y_1 or y_2 shows at least W seconds, thus allowing the corresponding control rod to be put into the reactor core. Let A denote the product of the reactor core automaton and the two control rod automata. We define the reachability problem $(A, \varphi_I, \varphi_F)$ as follows. The initial states are characterized by the state predicate

$$\varphi_I \;=\; (l[rod_1] = out \,\wedge\, l[rod_2] = out \,\wedge\, l[core] = no\_rod \,\wedge\, x = 510 \,\wedge\, y_1 = y_2 = W);$$

that is, initially no rod is in the reactor core, the initial temperature is 510 degrees, and $y_1 = y_2 = W$ (we write $l[c]$ for the component of the location counter l that is associated with the component automaton c; so $l[core]$ ranges over the locations of the reactor core automaton, etc.). The unsafe states are characterized by the state predicate

$$\varphi_F \;=\; (l[core] = no\_rod \,\wedge\, x = 550 \,\wedge\, y_1 \leq W \,\wedge\, y_2 \leq W);$$

that is, the unsafe situation is that the core temperature reaches 550 degrees and neither y_1 nor y_2 shows more than W seconds[3] (and, thus, none of the control rods is available). The answer to the reachability problem $(A, \varphi_I, \varphi_F)$ is *yes* iff the reactor temperature control system satisfies the safety requirement.

In HyTech, the reachability problem is specified as follows:

```
InitialState = l[rod1]==out && l[rod2]==out && l[core]==norod &&
    510==x && w==y1 && w==y2
Bad = l[core]==norod && 550==x && y1<=w && y2<=w
```

Forward versus backward analysis

HyTech can attack a reachability problem by forward analysis or by backward analysis. Given the reachability problem $(A, \varphi_I, \varphi_F)$, the *forward analysis* computes the state predicate that defines the region $post^*([\![\varphi_I]\!])$, and then takes the conjunction with the final state predicate φ_F; the *backward analysis* computes the state predicate that defines the region $pre^*([\![\varphi_F]\!])$, and then takes the conjunction with the initial state predicate φ_I. For a given reachability problem, one direction may perform better than the other direction. In fact, it may be that one direction terminates and the other does not. For example, only the backward analysis terminates for the reactor temperature control system.

We ask HyTech to perform a forward or backward analysis, respectively, by writing

```
Go := PrintTime[ Forward ]
```

[3] Remember that we are limited to *nonstrict* inequalities.

or

```
Go := PrintTime[ Backward ]
```

These commands also print the CPU time consumed by the reachability analysis.

Parametric analysis

The automatic derivation of parameters was introduced for real-time systems in [AHV93] and applied to hybrid systems in [AHH93]. We can use HyTech to synthesize necessary and sufficient conditions on system parameters such that a hybrid automaton satisfies a requirement.

Recall that the reactor temperature control system contains the parameter w, which specifies the necessary rest time for a control rod. Clearly, the safety requirement will not be satisfied for large values of w. Indeed, the *target region* $[\![\varphi_I]\!] \cap pre^*([\![\varphi_F]\!])$ gives a sufficient and necessary condition on w such that the safety requirement is not satisfied. Typically the state predicate that defines the target region is too complex to see the conditions on the parameters clearly, but these can be isolated in HyTech using *projection operators*. By writing

```
EliminateLocList = {rod1,rod2,core}
EliminateVarList = {x,y1,y2}
```

we eliminate all location information from the state predicate that defines the target region, and we project out all information about the data variables x, y_1, and y_2. Then the resulting projection of the target region, as computed by HyTech using backward analysis, is

```
9w >= 184
```

In other words, the target region is empty if and only if $9W < 184$. It follows that $9W < 184$ is a necessary and sufficient condition on the parameter W that prevents the reactor from shutdown. The verification requires 17.27 seconds of CPU time.[4]

3.2 Abstract Interpretation

To expedite the reachability analysis and to force the termination of the analysis, HyTech provides several abstract-interpretation operators [CC77, HH95b], including the convex-hull operator and the extrapolation operator. Our extrapolation operator is similar to the widening operator of [CH78, Hal93].

An abstract-interpretation operator approximates a set of convex data regions with a single convex data region. The *convex-hull operator* overapproximates a union of convex data regions by its convex hull. The *extrapolation operator* overapproximates a directed chain $S \subset f(S) \subset f^2(S) \subset \cdots$ of convex data regions by a "guess" of the limit region $\bigcup_{i \geq 0} f^i(S)$. Either operator, or the combination of both operators, may cause the termination of a forward or backward reachability analysis that does not terminate otherwise. However, since the use of either operator results in an overapproximation of the target region, the abstract analysis is sound but not complete: if HyTech returns the answer *yes* to a reachability problem, then the approximate target region is empty, and therefore also the exact target region must be empty; but if the answer is *no*, then the exact target region may still be empty, and the correct answer to the reachability problem may be *yes*. In the latter case, we have to refine our approximation, by applying fewer abstract-interpretation operators, or by using two-way iterative approximation (see below).

In HyTech, we write

[4] All performance figures are given for a SPARC 670MP station.

```
TakeConvex = True
```

or

```
TakeConvex = False
```

to turn the convex-hull operator on or off, respectively. The extrapolation operator can be turned on or off selectively for individual control locations. For example, if we want to apply the extrapolation operator only to data regions that correspond to the two locations $l[rod_1] = out \wedge l[rod_2] = in \wedge l[core] = no\_rod$ and $l[rod_1] = in \wedge l[rod_2] = out \wedge l[core] = no\_rod$ of the reactor temperature control system, then we write:

```
ExtraSet[lc_]=((lc === l[rod1]==out && l[rod2]==in && l[core]==norod)
|| (lc === l[rod1]==in && l[rod2]==out && l[core]==norod))
```

The commands

```
ExtraSet[lc_] = True
```

and

```
ExtraSet[lc_] = False
```

ask HyTech to apply the extrapolation operator to all or none of the control locations, respectively. (In our analysis of the reactor temperature control system, it was not necessary to use any abstract-interpretation operators.)

Two-way iterative approximation

If inconclusive, the approximate reachability analysis can be refined by alternating approximate forward and backward analysis [CC92, DW95]. If any abstract-interpretation operators are used, HyTech automatically performs a two-way iterative analysis, beginning with the specified forward or backward pass. The readers should refer to [HH95b] for the details about the two-way iterative analysis of hybrid systems.

3.3 Ictl Model Checking

To check hybrid automata against more general requirements than reachability, we use the requirement specification language Ictl [AHH93]. Ictl is a branching-time logic in the tradition of Ctl [CES86], with additional clock and stop-watch variables for specifying timing constraints. For a formal definition of Ictl, and a discussion of the model-checking algorithm, we refer the reader to [AHH93]; here we present only a couple of typical Ictl requirements for the reactor temperature control system.

First, recall the safety requirement that *the reactor never needs to be shut down*, which was characterized by a reachability problem $(A, \varphi_I, \varphi_F)$ in Section 3.1. In Ictl, the safety requirement is specified by the formula

$$\varphi_I \ \rightarrow \ \forall \Box \neg \varphi_F,$$

which asserts that in the labeled transition system $[\![A]\!]$, along all paths of infinite duration that start from an initial state, no unsafe state is visited.

In addition to safety requirements, in Ictl we can specify also liveness, real-time, and duration requirements of hybrid automata. Consider, for example, the duration requirement that *it is*

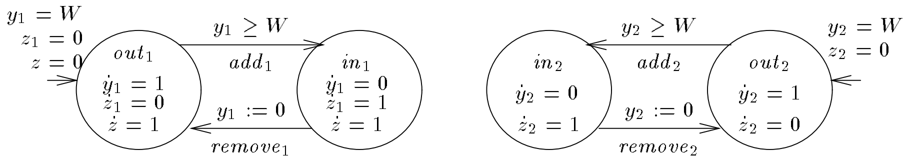

Figure 4: The augmented control rod automata

possible to keep the reactor running without using either control rod more than one third of the time. To specify duration requirements, we use stop-watches. The *type* of a stop-watch z is a set U of control locations: the stop-watch z has the slope 1 whenever the automaton control is in a location of U, and otherwise z has the slope 0. Then z measures the accumulated amount of time that the automaton control spends in locations of U. We specify the type of a stop-watch by a state predicate that constrains the location counter. For example, we write $(z_1 : l[rod_1] = in)$ to declare that the variable z_1 is a stop-watch whose type is the set of all control locations where control rod 1 is in the reactor core. The given duration requirement uses a clock and two stop-watches. The clock z (a stop-watch of type *true*) measures the total elapsed time, the stop-watch z_1 of type $l[rod_1] = in$ measures the accumulated amount of time that control rod 1 spends in the reactor core, and the stop-watch z_2 of type $l[rod_2] = in$ measures the accumulated amount of time that control rod 2 spends in the reactor core. The ICTL formula

$$\varphi_I \rightarrow (z\colon true)(z_1\colon l[rod_1] = in)(z_2\colon l[rod_2] = in)\exists\Box(x \leq 550 \wedge 3z_1 \leq z \wedge 3z_2 \leq z)$$

asserts that in the labeled transition system $[\![A]\!]$, there is a path of infinite duration that starts from an initial state along which the core temperature does not exceed 550 degrees, and the accumulated time that either control rod spends in the reactor core is always at most a third of the total elapsed time.

While the original HyTech prototype accepts ICTL input, we have not yet completed the implementation of ICTL model checking for the current version of HyTech. However, like safety requirements, also many real-time and duration requirements can be reduced to reachability analysis, by moving clocks and stop-watches from the requirement specification to the system model. Consider, for example, the duration requirement of the reactor temperature control system that *independent of the control strategy that is used for deciding which control rod to put into the reactor core, each control rod is used at most one third of the time*:

$$\varphi_I \;\rightarrow\; (z\colon true)(z_1\colon l[rod_1] = in)(z_2\colon l[rod_2] = in)\; \forall\Box(3z_1 \leq z \wedge 3z_2 \leq z).$$

To verify this requirement, we move the clock z and the stop-watches z_1 and z_2 to the control rod automata as shown in Figure 4. Then we use HyTech to check if any state in the unsafe region $[\![3z_1 \geq z \vee 3z_2 \geq z]\!]$ is reachable from an initial state.

4 Analysis of Nonlinear Hybrid Systems

Many physical quantities, such as temperature, exhibit nonconstant derivatives. When using linear hybrid automata for modeling, these quantities need to be either translated into piecewise-linear quantities or approximated by rate intervals. Both options can be formalized as algorithmic translations—the *clock translation* and the *rate translation*—from nonlinear hybrid automata to linear hybrid automata. Both translations are currently being implemented in HyTech, and formal definitions of the translations can be found in [HH95a].

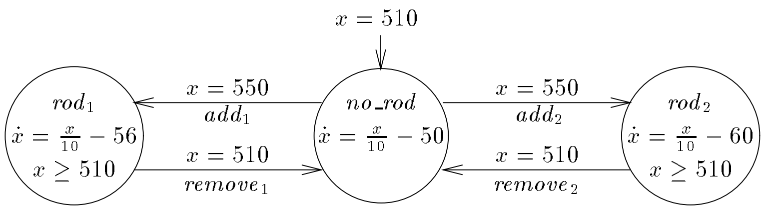

Figure 5: The nonlinear reactor core automaton

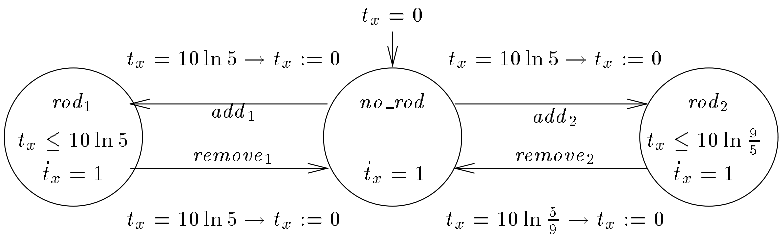

Figure 6: The clock-translated reactor core automaton

4.1 Clock Translation

Suppose that the reactor core of the reactor temperature control example is modeled by the hybrid automaton shown in Figure 5. The core temperature x increases according to the differential equation $\dot{x} = x/10 - 50$ if no control rod is in the reactor core; x decreases according to the differential equation $\dot{x} = x/10 - 56$ if control rod 1 is in the reactor core; and x decreases according to the differential equation $\dot{x} = x/10 - 60$ if control rod 2 is in the reactor core. Since the derivative of x is not governed by a rate interval, we say that x is a *nonlinear variable*, and the automaton containing x is a *nonlinear hybrid automaton*.

The clock translation replaces nonlinear variables by clocks. The nonlinear variable x can be replaced by a clock if its value is uniquely determined by the last assignment to x and the time that has expired since that assignment. For example, the core temperature x from Figure 5 can be replaced by a clock t_x. The resulting automaton, which is shown in Figure 6, is not a linear hybrid automaton, because real numbers occur in invariants and guards. In a second step, we overapproximate the automaton of Figure 6 by the linear hybrid automaton of Figure 7. For instance, since $16 \leq 10 \ln 5 \leq 16.1$, we overapproximate the the invariant $t_x \leq 10 \ln 5$ of the location *no_rod* by the invariant $10t_x \leq 161$, and we overapproximate the guard $t_x = 10 \ln 5$ of the transition from *no_rod* to rod_1 by the guard $160 \leq 10t_x \leq 161$.

Now suppose we wish to check if a final state in $[\![\varphi_F]\!]$ is reachable from an initial state in $[\![\varphi_I]\!]$ according to the nonlinear hybrid automaton of Figure 5. For simplicity, assume that neither φ_I nor φ_F contain the nonlinear variable x. If no final state in $[\![\varphi_F]\!]$ is reachable from an initial state in $[\![\varphi_I]\!]$ according to the linear hybrid automaton of Figure 6, which can be checked using HyTech, then this is also the case for the original nonlinear hybrid automaton of Figure 5. The converse, however, is not necessarily true, because of our overapproximation of invariants and guards. So if the reachability problem for the overapproximated automaton has a positive answer, we cannot conclude anything about the original reachability problem and must refine our approximation.

In summary, the clock translation of nonlinear variables is exact (i.e., it preserves all Ictl

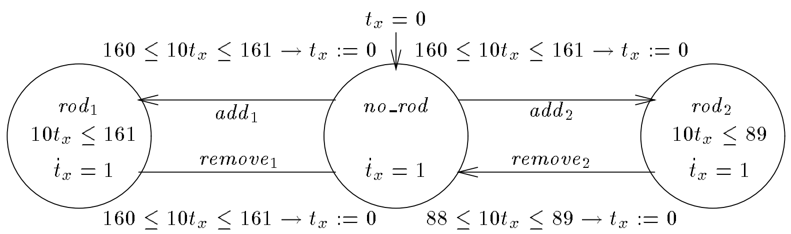

Figure 7: The overapproximated clock-translated core automaton

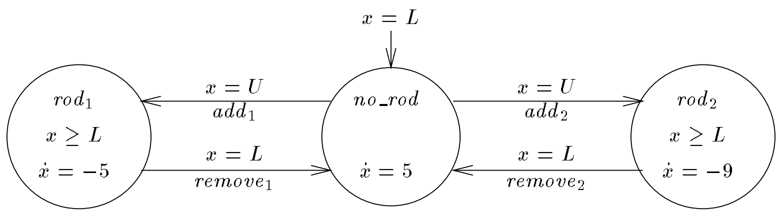

Figure 8: The reactor core automaton with linear worst-case assumptions

requirements) up to the representation of real numbers. The applicability of the clock translation depends on the solvability of differential equations, and we have begun to characterize sufficient conditions for the applicability of the clock translation [HH95a].

4.2 Rate Translation

The rate translation overapproximates nonlinear variables using linear variables that are governed by rate intervals. For the nonlinear variable x, and for each control location v, we compute the minimal and maximal derivative of x in v that observes the invariant of v. Consider, for example, the location rod_1 of the reactor core automaton from Figure 5, with the differential equation $\dot{x} = x/10 - 56$. Since we can strengthen the invariant to $510 \leq x \leq 550$, the derivative of x is bounded below by -5 and above by -1. Thus we can overapproximate the behavior of the nonlinear variable x in the location rod_1 by replacing the differential equation with the rate interval $dif(no\_rod, x) = [-5, -1]$. By treating the other locations similarly, we obtain the linear hybrid automaton of Figure 1.

Recall that HyTech guarantees that the reactor core automaton of Figure 1 meets its safety requirement iff the parameter W satisfies the condition $9W < 184$. For the clock-translated reactor core automaton of Figure 7, HyTech computes the weaker condition $5W < 189$. This condition is weaker, because the clock translation of Figure 7 gives a better approximation of the nonlinear system of Figure 5 than does the rate translation of Figure 1. Thus better approximations allow the design engineers to use slower mechanisms for moving the control rods. If desired, the approximation by rate translation can be refined by splitting control locations [HH95a].

Finally, suppose we replace the differential equations for the nonlinear variable x of the reactor core automaton from Figure 5 by worst-case constant-slope assumptions. By experimenting, we find that under these simplifying assumptions, additional parameters can be synthesized by HyTech. In particular, we replace the lower bound of 510 and the upper bound of 550 for the core temperature by the parameters L and U, respectively. The resulting linear hybrid automaton is shown in

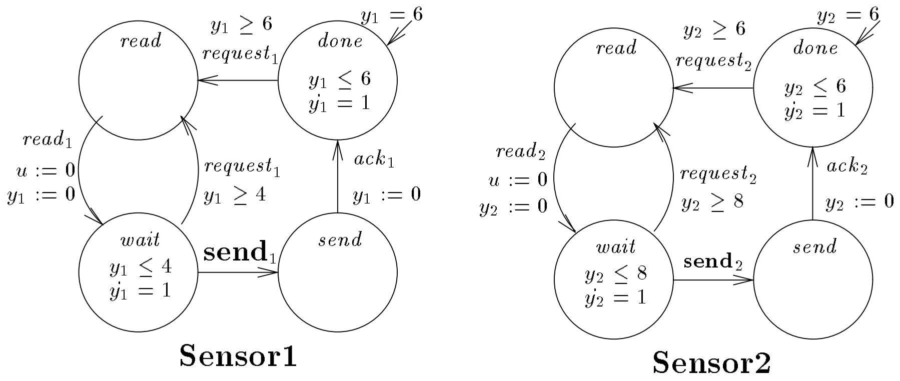

Figure 9: The two sensors

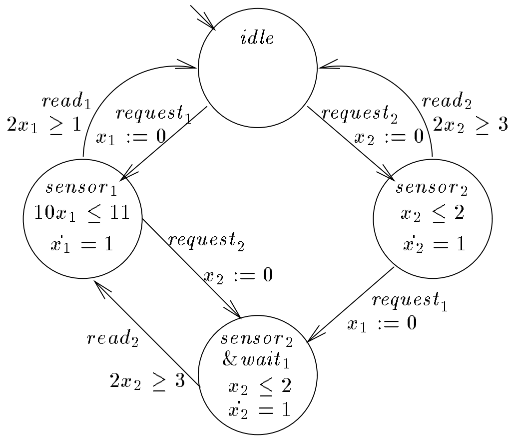

Figure 10: The scheduler

Figure 8. HyTech automatically synthesizes a necessary and sufficient condition on the three parameters W, L, and U such that the reactor meets the safety requirement of never being shut down:

$$45W < 23(U - L)$$

(the computation uses 17.38 seconds of CPU time). Notice that if we replace the parameters L and U by the constants 510 and 550, respectively, we obtain exactly the condition $9W < 184$ that results from analyzing the rate-translated reactor core automaton of Figure 1.

5 Three Case Studies

We report on the application of HyTech to three nontrivial benchmark problems.

5.1 A Distributed Control System with Time-outs

The distributed control system of [Cor94] consists of two sensors and a controller that generates control commands to a robot according to the sensor readings. The programs for the two sensors

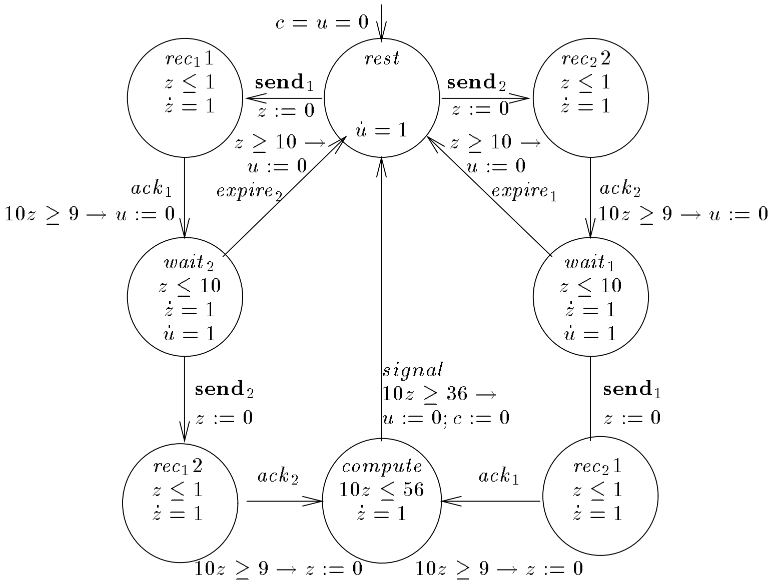

Figure 11: The controller

and the controller are written in ADA. The two sensors share a single processor, and the priority of sensor 2 for using the processor is higher than the priority of sensor 1. In other words, if both sensor 1 and sensor 2 want to use the processor to construct a reading, only sensor 2 obtains the processor, and sensor 1 has to wait. The two sensors are modeled by the two linear hybrid automata in Figure 9 and the priorities for using the shared processor are modeled by the scheduler automaton in Figure 10.

Each sensor can be constructing a reading (location *read*), waiting for sending the reading (location *wait*), sending the reading (location *send*), or sleeping (location *done*). The processor can be scheduled idle (location *idle*), serving sensor 1 (location $sensor_1$), serving sensor 2 while sensor 1 is waiting (location $sensor_2\&wait_1$), or serving sensor 2 while sensor 1 is not waiting (location $sensor_2$).

Each sensor constructs a reading and sends the reading to the controller. The shared processor for constructing sensor readings is requested via *request* transitions, the completion of a reading is signaled via *read* transitions, and the reading is delivered to the controller via **send** transitions. Sensor 1 takes 0.5 to 1.1 milliseconds and sensor 2 takes 1.5 to 2 milliseconds of CPU time to construct a reading. These times are measured by the stop-watches x_1 and x_2 of the scheduler automaton. Notice that at most one of the two stop-watches x_1 and x_2 runs in a location of the scheduler automaton, which reflects the fact that only one sensor can use the shared processor at a time. If sensor 1 loses the processor because of preemption by sensor 2, it can continue the construction of its reading after the processor is released by sensor 2.

Once constructed, the reading of sensor 1 expires if it is not delivered within 4 milliseconds, and the reading of sensor 2 expires if it is not delivered within 8 milliseconds. These times are measured by the clocks y_1 and y_2 of the sensor automata. If a reading expires, then a new reading must be constructed. After successfully delivering a reading, a sensor sleeps for 6 milliseconds (measured again by the clocks y_1 and y_2), and then constructs the next reading.

The controller is modeled by the automaton in Figure 11. The controller is executed on a dedicated processor, so it does not compete with the sensors for CPU time. We use the clock z

to measure the delays and time-outs of the controller. The controller accepts and acknowledges a reading from each sensor, in either order, and then computes and sends a command to the robot. The sensor readings are acknowledged via *ack* transitions, and the robot command is delivered via a *signal* transition. It takes 0.9 to 1 milliseconds to receive and acknowledge a sensor reading. The two sensor readings that are used to construct a robot command must be received within 10 milliseconds. If the controller receives a reading from one sensor but does not receive the reading from the other sensor within 10 milliseconds, then the first sensor reading expires (via an *expire* transition). Once both reading are received, the controller takes 3.6 to 5.6 milliseconds to synthesize a robot command.

We want to know how often a robot command can be generated by the controller. For this purpose, we add a clock c to the controller automaton such that c measures the elapsed time since the last robot command was sent. The slope of the clock c is 1 in all locations of the controller automaton (this is omitted from Figure 11), and c is reset to 0 whenever a robot command is sent. We want to compute the maximum value of the clock c in all states that are reachable in the product of all four automata.

However, the product of the four automata does not model the system exactly according to Corbett's specification. This is because the **send** transitions should be urgent, that is, they should be taken as soon as they are enabled. We model the urgency of the **send** transitions by adding an additional clock, u, and global invariants. The clock u is reset whenever a sensor is ready to send a reading to the controller, and whenever the controller is ready to receive a sensor reading. Then we use the global invariant that $u = 0$ if both a sensor and the controller are ready for a transmission; that is,

$$
\begin{array}{l}
(l[\mathit{sensor}_1] = \mathit{wait} \wedge l[\mathit{controller}] = \mathit{rest} \;\rightarrow\; u = 0) \;\wedge \\
(l[\mathit{sensor}_2] = \mathit{wait} \wedge l[\mathit{controller}] = \mathit{rest} \;\rightarrow\; u = 0) \;\wedge \\
(l[\mathit{sensor}_1] = \mathit{wait} \wedge l[\mathit{controller}] = \mathit{wait}_1 \;\rightarrow\; u = 0) \;\wedge \\
(l[\mathit{sensor}_2] = \mathit{wait} \wedge l[\mathit{controller}] = \mathit{wait}_2 \;\rightarrow\; u = 0).
\end{array}
$$

This invariant enforces whenever a transmission of a sensor reading is enabled, the transmission happens immediately.

In HyTech, the global invariant is defined as follows:

```
GlobalInvar = {{l[sensor1]==wait && l[controller]==rest, 0==u},
    {l[sensor2]==wait && l[controller]==rest, 0==u},
    {l[sensor1]==wait && l[controller]==wait1, 0==u},
    {l[sensor2]==wait && l[controller]==wait2, 0==u)}}
```

To compute the range of possible values for the clock c in the reachable states, we write:

```
InitialState = l[sensor1]==done && l[sensor2]==done && l[scheduler]==
    idle && l[controller]==rest && 0==c && 6==y1 && 6==y2 && 0==u
Bad = True

EliminateLocList = {sensor1,sensor2,sched,gen}
EliminateVarList = {x1,x2,y1,y2,z,u}
```

Notice that, using the two projection operators, we ask HyTech to print only information about the clock c. Using forward analysis without approximation, HyTech returns, in 89.53 seconds of CPU time, the following answer:

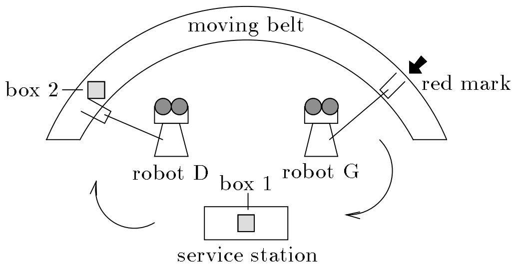

Figure 12: The two-robot manufacturing system

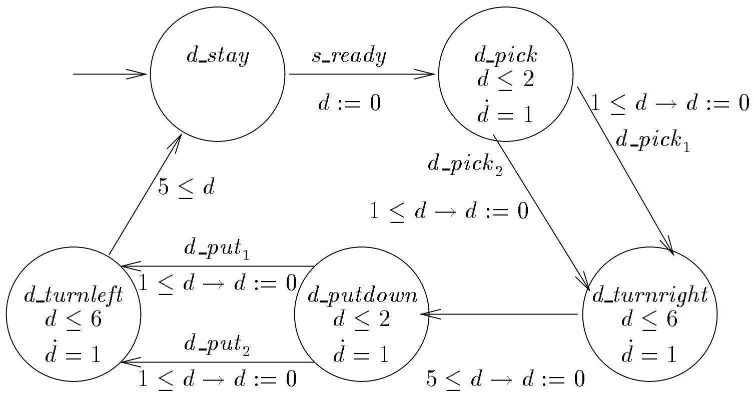

Figure 13: Robot D

```
0 <= c && -12 <= -5*c || -7 <= -2*c && 9 <= 10*c ||
-3 <= -c && 7 <= 10*c || -9 <= -2*c && 3 <= 2*c ||
12 <= 5*c && -28 <= -5*c || 5 <= 2*c && -18 <= -2*c ||
33 <= 10*c && -105 <= -10*c || 42 <= 5*c && -56 <= -5*c
```

From this result (the last disjunct is $42 \leq 5c \wedge -56 \leq -5c$), it follows that the maximum value of the clock c is 11.2; that is, a robot command is generated by the controller at least once every 11.2 milliseconds. We can also apply HyTech to analyze the same system except that the priority of sensor 1 for using the shared processor is higher than the priority of sensor 2. In that case, a robot command is generated at least once every 11.0 milliseconds.

5.2 A Two-robot Manufacturing System

Puri and Varaiya [PV95] designed a manufacturing system that consists of a conveyor belt with two boxes, a service station, and two robots. The system is illustrated in Figure 12. This system has been also modeled and analyzed in [DY95].

Robot D, one of the two robots, is modeled by the linear hybrid automaton of Figure 13. The clock d is used to measure the time needed for the actions performed by robot D. Initially robot D is looking at the service station (location *d_stay*). When it sees an unprocessed box in the service station, it picks up that box from the service station in 1 to 2 seconds (location *d_pick*), makes a right turn in 5 to 6 seconds (location *d_turnright*), puts the box at one end of the conveyor belt in 1 to 2 seconds (location *d_putdown*), makes a left turn back to the service station in 5 to 6 seconds (location *d_turnleft*), and stays at there waiting for the next unprocessed box (location *d_stay*).

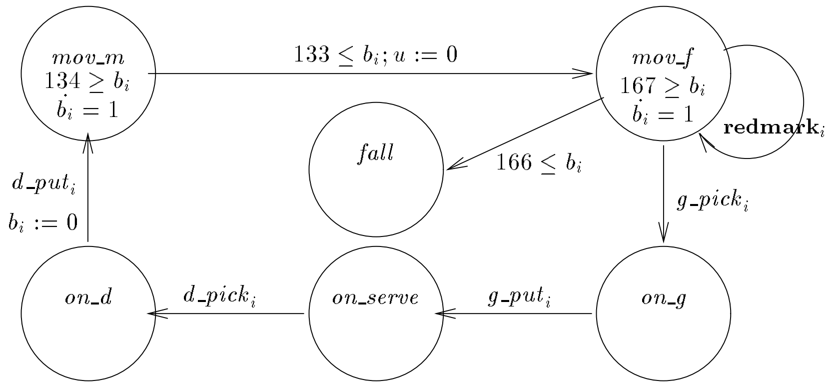

Figure 14: Box i

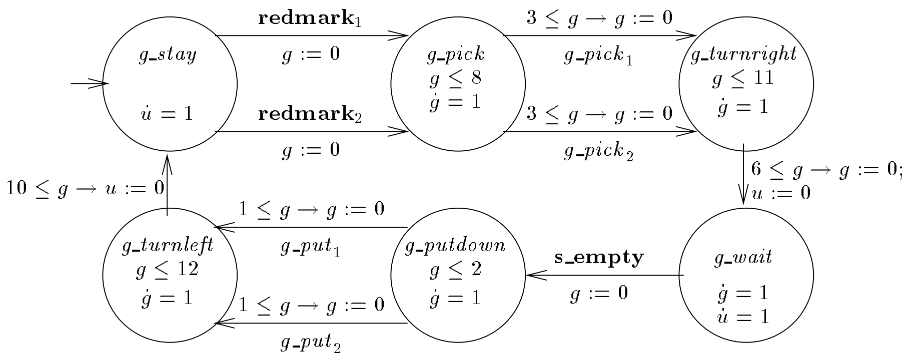

Figure 15: Robot G

The two boxes, box 1 and box 2, are modeled by the indexed linear hybrid automaton in Figure 14, where the index i is either 1 (for box 1) or 2 (for box 2). A box may be in the service station (location *on_serve*), held by robot D (location *on_d*), moving on the conveyor belt before a red mark (location *mov_m*), moving on the conveyor belt beyond the red mark (location *mov_f*), held by robot G (location *on_g*), or falling off the end of the conveyor belt (location *fall*). A box on the conveyor belt is processed by the manufacturing system. The conveyor belt is moving at a certain speed from one end to the other. The clock b_i measures the total time that box i spends on the conveyor belt, and thus determines the position of box i on the belt. A box requires 133 to 134 seconds to reach the red mark after it is placed on the belt by robot D. If a box is not picked up by robot G before the end of the belt, then the box falls off the belt 166 to 167 seconds after it is placed on the belt.

Robot G at the end of the conveyor belt is modeled by the automaton in Figure 15. The clock g measures the time needed to perform the actions of robot G. Initially robot G is looking at the red mark next to the conveyor belt (location *g_stay*). When it sees a processed box moving beyond the red mark, it picks up that box from the belt in 3 to 8 seconds (location *g_pick*), makes a right turn in 6 to 11 seconds (location *g_turnright*), waits for the service station to be empty (location *g_wait*), puts the box into the service station in 6 to 11 seconds (location *g_putdown*), makes a left turn back to the conveyor belt in 1 to 2 seconds (location *g_turnleft*), and stays there watching the red mark (location *g_stay*).

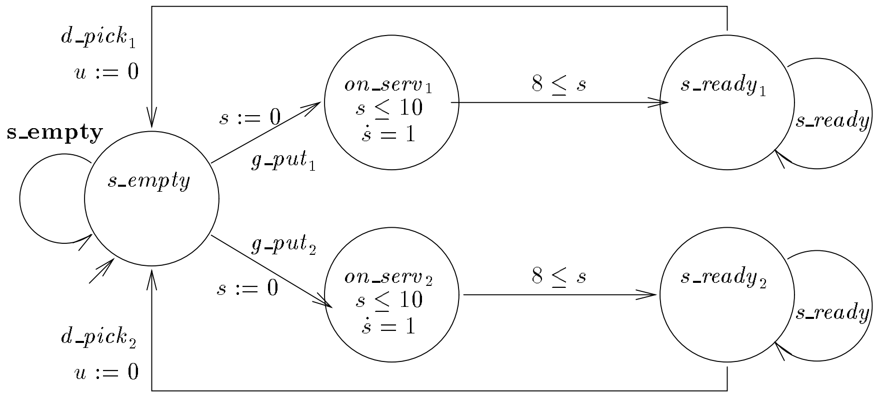

Figure 16: The service station

The service station is modeled by the automaton in Figure 16. Whenever the service station receives a processed box, it pops up an unprocessed box for robot D to pick up. The service station takes 8 to 10 seconds to switch the processed and unprocessed boxes, which is measured by the clock s. Initially both boxes are on the conveyor belt before the red mark. There are at most two boxes on the belt at any time, because the service station pops up a new box only when it receives a processed box from robot G.

According to Puri and Varaiya's specification, the transitions with the synchronization letters *s_ready*, $\mathbf{redmark}_1$, $\mathbf{redmark}_2$, and **s_empty**, are urgent; that is, robot D picks up a box from the service station as soon as it is ready and sees a box in the service station, etc. We treat the *s_ready* transitions as ordinary transitions, because this assumption will not affect our analysis. We use the clock u and the following global invariants to model the urgent transitions:

```
GlobalInvar = {{l[grobot]==gstay && l[box1]==movf, 0==u},
    {l[grobot]==gstay && l[box2]==movf, 0==u},
    {l[grobot]==gwait && l[station]==sempty, 0==u}}
```

We want to check the safety requirement that no box will ever fall off the conveyor belt. This requirement clearly depends on the initial positions of the two boxes on the belt. Then we use HyTech to analyze the reachability problem $(A, \varphi_I, \varphi_F)$, where A is the product of all five automata and

$\varphi_I \;=\; (l[box_1] = mov\_m \wedge l[box_2] = mov\_m \wedge l[robot_G] = g\_stay \wedge l[robot_D] = d\_stay \wedge l[servicestation] = s\_empty \wedge u = 0)$,
$\varphi_F \;=\; (l[box_1] = fall \vee l[box_2] = fall)$.

After we simplified the product automaton by eliminating unreachable locations and identifying locations in which a box is fallen, HyTech is able to return, in 163.41 minutes of CPU time, the following target region:

```
-1 <= -b1+b2 && -9 <= b1-b2 || -1 <= b1-b2 && -9 <= -b1+b2
```

using backward computation without approximation. It follows that

$$b2 - b1 > 9 \vee b1 - b2 > 9$$

is a necessary and sufficient condition on the initial condition of the system so that neither box will fall off the conveyor belt; that is, $|b_1 - b_2| > 9$.

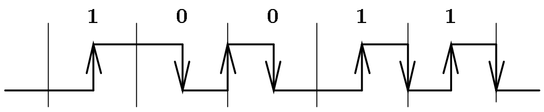

Figure 17: The Manchester encoding

5.3 The Philips Audio Control Protocol

In [BPV94], the timing-based Philips audio control protocol is modeled by an extension of the timed I/O automata model [LV92, LV93], and verified mathematically without computer support. We model the same protocol using linear hybrid automata, and verify its correctness for input strings up to length 8 using HyTech.

The protocol consists of a sender and a receiver. The sender uses the Manchester encoding to encode an input string of bits into a continuous signal (see Figure 17 for the encoding of 10011). The voltage on the communication bus is either high or low. A 0 bit is sent as a *down* signal from high to low voltage; a 1 bit is sent as a *up* signal from low to high voltage. The time line is divided into time slots of equal length, and the signals are sent in the middle of each time slot. The receiver decodes the continuous signal into an output string of bits. The protocol is correct iff the input and output strings match.

The time slots are measured by local clocks of the sender and receiver. These local clocks, however, may not be accurate, and their derivatives may vary within the rate interval $[\frac{19}{20}, \frac{21}{20}]$. Besides this potential 5% (or 1/20) timing error, the protocol faces also the following complications:

- The receiver does not know when the first time slot begins. The sender and the receiver can synchronize at the beginning of the transmission by knowing that (1) before the transmission, the voltage is low, and (2) the transmitted string starts with the bit 1.
- The receiver does not know the length of the bit string that is transmitted.
- The receiver sees only up signals and no down signals (because down signals are difficult to detect).

Using HyTech, we verify that whenever the sender encodes and sends a string of up to 8 bits, the receiver correctly decodes all bits in string. HyTech also shows that the protocol is incorrect in the case that the local clocks are subject to timing errors up to 1/15.

We use four linear hybrid automata to model the input, the sender, the receiver, and the output. The input automaton of Figure 18 generates all the possible bit strings up to a certain length. The length of the input string is decided by the initial value of the integer variable k. Whenever a bit is nondeterministically generated by the input automaton, the value of k is decremented by 1. When k becomes 0, the input automaton nondeterministically generates a suffix of one or two bits. So the input automaton generates all possible input strings of $k+1$ or $k+2$ bits. The integer variable c stores the message that is sent. If the bit 0 is sent, then c is updated to $2c$; if the bit 1 is sent, then c is updated to $2c+1$. The input automaton synchronizes with the sender through the synchronization letters $head_i$ and $input_i$, which correspond to looking at the next bit of the input string and sending the next bit of the input string, respectively.

The sender is modeled by the automaton of Figure 19. The variable x represents the drifting local clock of the sender, and its rate interval is $[\frac{19}{20}, \frac{21}{20}]$ for all locations of the sender automaton.

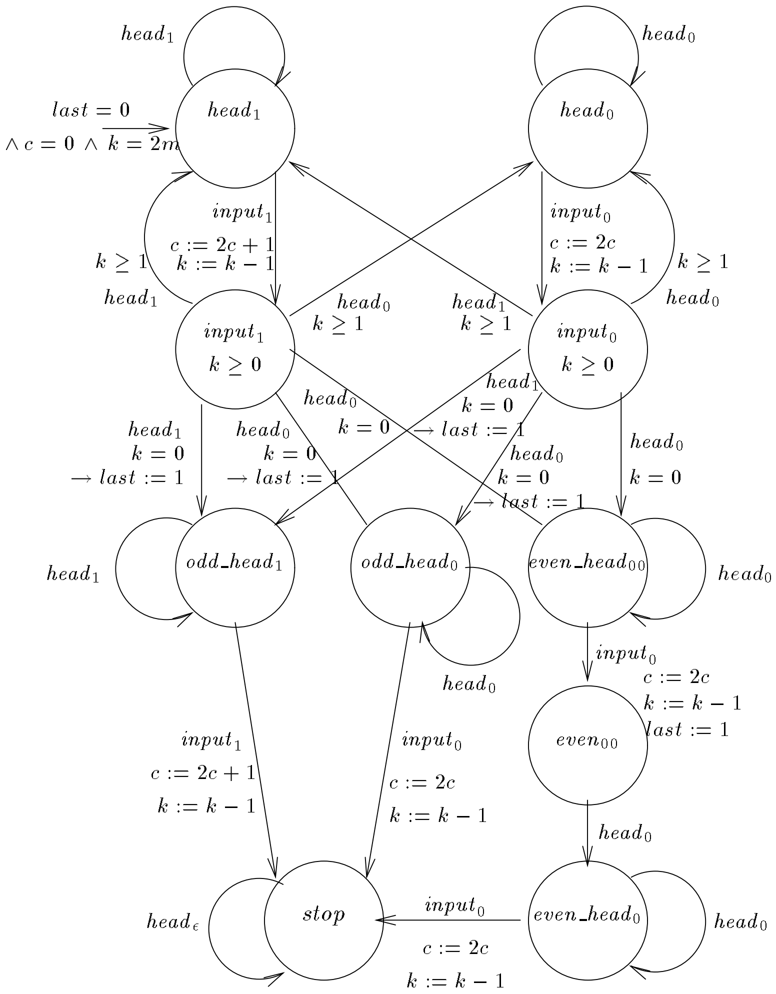

Figure 18: The input automaton

The sender synchronizes with the receiver through the synchronization letter *up*, which represents up signals.

The receiver is modeled by the automaton of Figure 20. The variable y represents the drifting local clock of the receiver, and its rate interval is $[\frac{19}{20}, \frac{21}{20}]$ for all locations of the receiver automaton. The receiver sees each up signal of the sender and decides if it encodes a 0 or a 1. If no up signal is received for a certain amount of time, the receiver times out and concludes that the transmission is completed. The receiver synchronizes with the output automaton through the synchronization letters $output_i$, which represent the bits of the decoded string.

The output automaton of Figure 21 stores the decoded bit string in the integer variable d. Then $c = d$ signals a correct decoding, and $c \neq d$ signals an error in the protocol. We check if it is possible to reach a state that satisfies $c \neq d$ when the transmission is completed. A forward reachability analysis with HyTech successfully verifies the protocol for $k = 6$. The performance of HyTech is summarized in Table 22.

Remark. Since the submission of this paper, we have reimplemented HyTech independent of Mathematica [HHWT95]. That implementation is significantly more efficient, and achieves

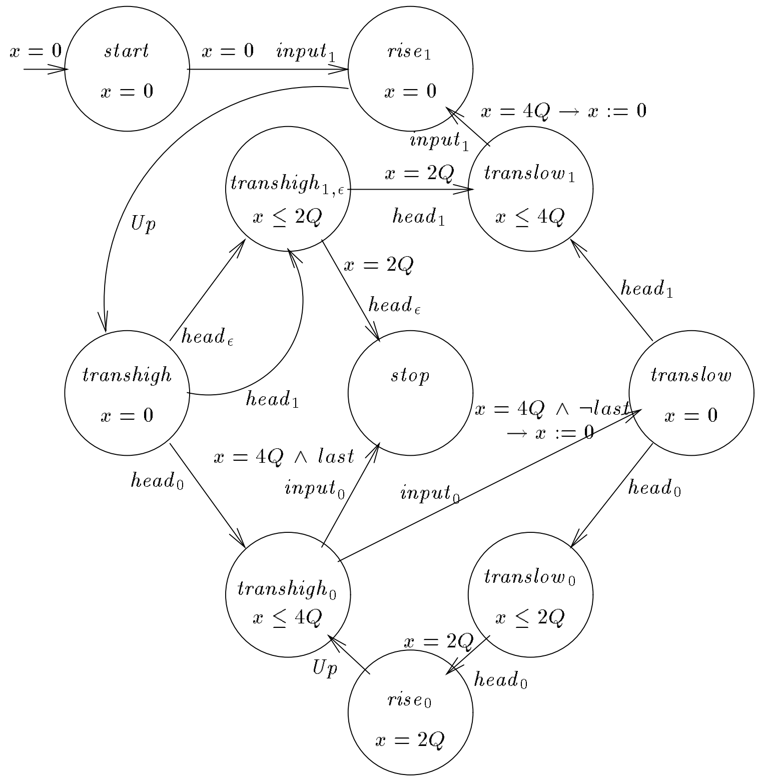

Figure 19: The sender automaton

speedup of one to three orders of magnitude. For example, the distributed control system of Section 5.1 can be checked using 12 seconds of CPU time, and the manufacturing system of Section 5.2 can be checked using 353 seconds of CPU time.

Acknowledgement. We thank Howard Wong-Toi for the performance figures of the new implementation of HyTech and a careful reading of this paper.

References

[ACH+95] R. Alur, C. Coucoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[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.

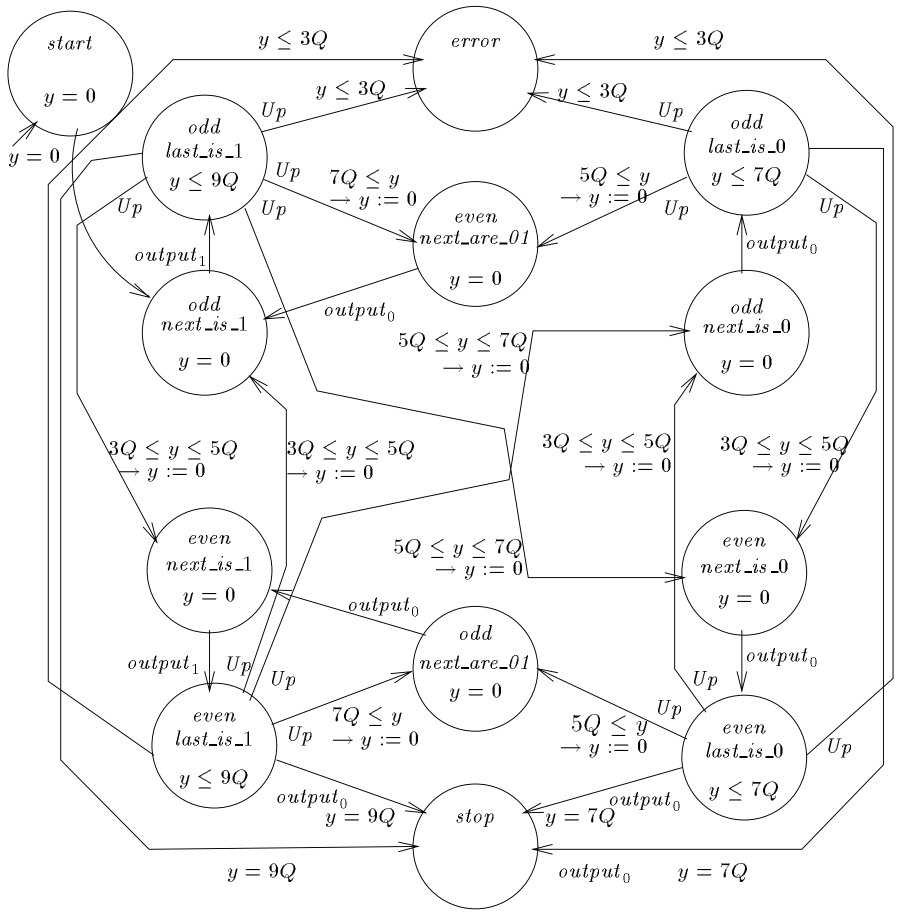

Figure 20: The receiver automaton

[AHV93] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, pages 592–601. ACM Press, 1993.

[BPV94] D. Bosscher, I. Polak, and F. Vaandrager. Verification of an audio-control protocol. In H. Langmaack, W.-P. de Roever, and J. Vytopil, editors, *FTRTFT 94: Formal Techniques in Real-time and Fault-tolerant Systems*, Lecture Notes in Computer Science 863, pages 170–192. Springer-Verlag, 1994.

[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*, pages 238–252, Los Angeles, California, 1977. ACM Press.

[CC92] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In *PLILP*, Lecture Notes in Computer Science 631, pages 269–295. Springer-Verlag, 1992.

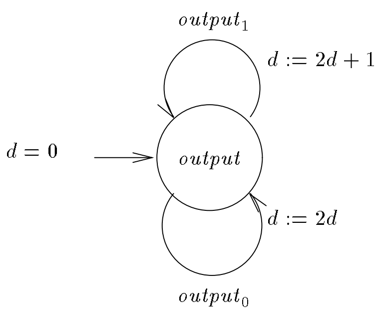

Figure 21: The output automaton

Clock error	Input length	Location number	Transition number	CPU time	
1/20	5 or 6	1300	2795	681.8 sec.	Proved
	7 or 8			4275 sec.	
1/15	5 or 6			2018 sec.	Disproved

Figure 22: Verification of the audio control protocol

[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.

[Cor94] J.C. Corbett. Modeling and analysis of real-time Ada tasking programs. In *Proceedings of the 15th Annual Real-time Systems Symposium*. IEEE Computer Society Press, 1994.

[DW95] D.L. Dill and H. Wong-Toi. Verification of real-time systems by successive over- and underapproximation. In *CAV 95: Computer-aided Verification*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[DY95] C. Daws and S. Yovine. Verification of multirate timed automata with KRONOS: two examples. Technical Report Spectre-95-06, VERIMAG, April 1995.

[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.

[HH95a] T.A. Henzinger and P.-H. Ho. Algorithmic analysis of nonlinear hybrid systems. In *CAV 95: Computer-aided Verification*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[HH95b] T.A. Henzinger and P.-H. Ho. A note on abstract-interpretation strategies for hybrid automata. To appear at Proceedings of Hybrid System Workshop, 1995.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: The next generation. Submitted, 1995.

[HRP94] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximation. In B. LeCharlier, editor, *International Symposium on Static Analysis, SAS'94*, Lecture Notes in Computer Science 864, Namur (belgium), September 1994. Springer-Verlag.

[HW95] P.-H. Ho and H. Wong-Toi. Automated analysis of an audio control protocol. In *CAV 95: Computer-aided Verification*, Lecture Notes in Computer Science. Springer-Verlag, 1995.

[LV92] N.A. Lynch and F. Vaandrager. Action transducers and timed automata. In R.J. Cleaveland, editor, *CONCUR 92: Theories of Concurrency*, Lecture Notes in Computer Science 630, pages 436–455. Springer-Verlag, 1992.

[LV93] N.A. Lynch and F. Vaandrager. Forward and backward simulations, part ii: timing-based systems. Technical Report CS-R9314, CWI, Amsterdam, 1993.

[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.

[PV95] A. Puri and P. Varaiya. Verification of hybrid systems using abstractions. To appear at Proceedings of Hybrid System Workshop, 1995.