Automated Analysis of an Audio Control Protocol*

Pei-Hsin Ho and Howard Wong-Toi

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

Abstract. We show how HyTech, a symbolic model checker for linear hybrid systems, can be used to analyze an audio control protocol. This protocol [BPV94] was first verified by Bosscher et al. without computer support. In this paper, we demonstrate that algorithmic methods can not only verify the protocol, but can also automatically synthesize the bound on the maximum clock drift, and suggest design modification for a more robust protocol. We believe the techniques we used — finite state encodings, automata transformations, strengthening of specifications — provide insight to the practictioner interested in modeling and analyzing similar real-world applications.

1 Introduction

Motivated by the desire to verify real-life reactive systems, Bosscher et al. [BPV94] met with engineers at Philips, Netherlands, and developed a formal description of an audio control protocol. The protocol uses timing-based Manchester encoding to transmit arbitrary length bit sequences between a single sender and receiver whose clocks are subject to a bounded error. Their modeling used an extension to timed I/O automata [LV92], and enabled them to verify its correctness using proof rules. Furthermore, they show that for correct operation, 1/17 is a tight bound on the error tolerance on the sender's and receiver's clocks. Their analysis is entirely mathematical and does not use computer support. They remark that it would be interesting to see how other methods, particularly algorithmic techniques, could handle their example.

Successful automated analysis. We accept their proposal, and demonstrate that HyTech [AHH93], a symbolic model checker for linear hybrid systems, can not only verify the protocol's correctness for Philip's tolerance specification of 1/20, but can also automatically synthesize the critical bound of 1/17. Indeed HyTech even suggests a revision in the protocol to enable wider clock drifts of 1/15. This case study is particularly interesting because it is not immediately clear how to use an automatic tool to analyze this protocol. We show how arbitrary length data streams can be finitely encoded using linear hybrid automata [ACHH93]. The verification analysis not only establishes the correctness of the hybrid automaton, but also justifies our finite encoding of the infinite data streams.

Synthesizing the critical bound is more delicate, and requires a number of steps. It involves introducing a parameter for the clock drift, applying two transformations to the automata, and adding locations to justify the synthesized bound is necessary as well as sufficient.

* This research was supported in part by the NSF under grant CCR-9200794, by the USAFOSR under contract F49620-93-1-0056, and by DARPA under grant NAG2-892.

This study provides evidence that model checkers can give more than a simple answer to the verification problem. We believe they can be used as powerful tools to verify properties, and to provide non-obvious parametric information that can guide system design. We first provide an overview of the more interesting steps taken in our analysis.

Arbitrary length input sequences. The first apparent difficulty is that correctness should be ascertained for arbitrary length transmission sequences, not just all bit streams up to some fixed finite length. However the algorithmic nature of model-checking does not allow for storing unbounded input and output sequences. Instead we nondeterministically generate the input sequence on-the-fly. Rather than adding bits to the receiver's copy of the message, they are immediately checked against the input bits. In this way, we need only store that part of the sequence that has been generated thus far, but not yet acknowledged by the receiver. Indeed we find that at most 3 unacknowledged bits need be stored.

Bosscher et al. also prove the timeliness of the receiver's output as a function of the length of the message. Again, a straightforward encoding of this property in hybrid automata seems impossible because it depends on comparing the time of the output with an arbitrarily large timing constant dependent on the length of the message. We obviate this difficulty by proving instead a stronger property that can be checked without the need to track an arbitrarily large timer.

Synthesizing error tolerances. Automatic parametric analysis is introduced in [CH78], and applied to the analysis of real-time and hybrid systems in [AHH93, AHV93, HH95]. In order to synthesize a parameter for a system's behavior we introduce a new variable, a *parameter*, whose value remains unchanged. HyTech can then detect for which values of the parameter error traces are present. In this study, we must synthesize the permissible *rates* at which the clocks progress. However immediate introduction of a variable into rates of the clocks is impossible, since clock variables in linear hybrid automata must have constants as bounds on their rates (otherwise their behavior is non-linear). To overcome this, we must transform the automaton by translating all skewed clocks into perfect clocks with rate 1 with the information on variable rates transfered into the timing constraints on transitions and location invariants [OSY94, WT94]. A second difficulty arises: the resulting system would still be non-linear, having timing constraints with variables in the denominators. We apply a further clock transformation that eliminates this problem.

There is one further complication: without prior knowledge of the error tolerance, it is impossible to know how many additional input bits may be unacknowledged at any time. Thus if we assume only k unacknowledged bits need be stored, we may encounter two sorts of errors: the tolerance is too large and bits are incorrectly received, or the large tolerance allows a trace with more than k unacknowledged bits. By separately analyzing these two errors, we find that all parameter values giving errors of the second type also give correctness errors, and thus the tolerance we infer is both necessary and sufficient for correctness.

Improved protocol. By analyzing the exact circumstances under which errors arise, we can infer from HyTech's output that a wider error tolerance is permitted if we alter the receiver's protocol to delay slightly longer before finalizing its bit stream.

Related work. This paper is the first to apply automated parametric analysis to the audio control protocol. Previous applications of automatic methods performed verification only, using either different models or more limited descriptions. In his thesis,

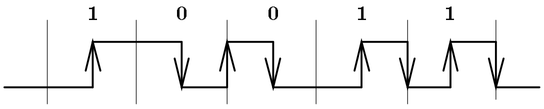

Fig. 1. Timing diagram for Manchester encoding of 10011

Wong-Toi [WT94] verified correctness and timing properties by applying his real-time system verification tool to the timed automaton obtained by transforming a linear hybrid system model. In [HH95], HyTech is used to verify the correctness of the audio control protocol with inputs up to a fixed length only.

2 Audio control protocol

We describe the timing-based communication protocol using Manchester encoding, as presented by Bosscher et al. [BPV94]. The protocol forms a part of a real audio control protocol under development by Philips. Bits are encoded based on timing delays between signals, and the rates of both the sender's and receiver's clocks vary within a given tolerance.

Bit streams are communicated using Manchester encoding. See Figure 1 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 voltage to low, and a 1 bit as an *Up* signal from low to high. The time line is divided into equal length time slots, and the signals are sent in the middle of each time slot. In order to send a repeated bit, there must be an intermediate change in voltage, and this occurs at the edge of the time slot as shown in the diagram for the last two bits.

The audio control protocol has the following complications [BPV94], partly due to the fact that there is a $\pm 5\%$ tolerance in the clock rates of the sending and receiving components:

1. The receiver does not know when the first time slot begins, although it does know the agreed upon width of the slots. The sender and receiver synchronize the start of transmission by requiring a low voltage whenever no bits are being sent, and starting all bit streams with a 1.
2. The receiver is not explicitly told the length of the message being sent. It must infer the bit stream is complete after a suitable lapse in receiving bits.
3. Drops in voltage are not instantaneous, and cannot be reliably detected. Therefore, the receiver must decode the message based solely on upgoing signals. Because the downgoing edge of a final 0 bit is not seen, this creates ambiguity between messages ending in 10 and in 1. This problem is solved by restricting bit streams to be either odd in length, or ending in 00.

The sender and receiver have the same clock error tolerance. The receiver interprets the *Up* signals by rounding the times they are received to the nearest time it expects them to be sent, *i.e.* to the slot edges or to the middle of a time slot, whichever is closer. We verify for arbitrary length bit streams that the receiver correctly receives all bits, and realizes the bit stream has finished in a timely fashion.

3 Analysis by hybrid automata

Our system modeling language is linear hybrid automata. Informally, a linear hybrid automaton [ACHH93] consists of a finite set X of real-valued variables and a labeled multigraph (V, E). The edges E represent discrete system actions and are labeled with nondeterministic guarded assignments to X. The vertices V represent different control modes and are labeled with constraints on the slopes of X. The state of the automaton changes either through instantaneous system actions or, while time elapses, through continuous activities.

Linear hybrid automata

A *linear term* over a set X of real-valued variables is a finite linear combination of variables with integer coefficients. A *linear inequality* over a set X is a nonstrict inequality between linear terms over X. 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.* A *data state* is a point $(a_1, a_2, \ldots, a_n)$ in the n-dimensional space $\mathbb{R}^n$, or equivalently, a function that maps each variable x_i to its value a_i. A *convex data region* is a convex polyhedron in $\mathbb{R}^n$; and a *data region* is a finite collection of convex data regions. A *convex data predicate* is a finite conjunction of linear inequalities; and a *data predicate* is a finite disjunction of convex data predicates. Each (convex) data predicate ϕ defines, then, a (convex) data region $[\![\phi]\!] \subseteq \mathbb{R}^n$ such that $s \in [\![\phi]\!]$ iff $\phi[X := s]$ is true. We often write $s \in \phi$ for $s \in [\![\phi]\!]$

Control locations A finite set V of vertices called *control locations.* 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 $v \in V$. Each state predicate $\varphi = \bigcup_{v \in V}\{(v, \phi_v)\}$ defines, then, the region $[\![\varphi]\!] = \bigcup_{v \in V}\{(v, [\![\phi_v]\!])\}$. A *location predicate* is a state predicate in which each data predicate ϕ_v is either *true* or *false*. 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 $\{(v, true)\} \cup \bigcup_{v' \neq v}\{(v', false)\}$.

Initial condition A state predicate φ^0 called the *initial condition.*

Location invariants A labeling function *inv* that assigns to each control location $v \in V$ a convex data predicate $inv(v)$, the *invariant* of v. The control of A may reside in the location v only while the invariant $inv(v)$ is true, so the invariants enforce progress in the system.

Continuous activities A labeling function *dif* that assigns to each control location $v \in V$ and each data variable $x_i \in X$ a *rate interval* $dif(v, x_i) = [l_x, u_x]$, where l_x and u_x are finite integer constants with $l_x \leq u_x$. The rate interval $dif(v, x_i) = [l_x, u_x]$, specifies that the rates of change of the data variable x_i may vary within the interval $[l_x, u_x]$, while automaton control resides in location v.
For $\delta \geq 0$, we define the *time-step* relation, $\xrightarrow{\delta}$, such that $(v, s) \xrightarrow{\delta} (v', s')$ if and only if $v = v'$ and there is a function $\rho \colon [0, \delta] \to \mathbb{R}^n$ such that (1) $f(0) = s$, (2) $f(\delta) = s'$, (3) for all $t \in [0, \delta]$, $\rho(t) \in [\![inv(v)]\!]$, and (4) for all time $t \in (0, \delta)$ and for each data variable x_i, $d\rho_i(t)/dt \in dif(v, x_i)$, where $\rho_i(t)$ denotes the value of variable x_i in the data state $\rho(t)$.

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

Discrete actions A labeling function *act* that assigns to each transition $e \in E$ a guarded command $act(e) = \phi \to \alpha$, where the guard ϕ is a convex data predicate, and $\alpha = \{x_i := [t_{1i}, t_{2i}] \mid x_i \in Y \subseteq X\}$ is a set of assignments, where t_{1i} and t_{2i} are linear terms. In the graphical representation, we write $x_i := t_i$ for $x_i := [t_i, t_i]$, and we omit the guard *true*. A data state s maps a linear term t to the real value $s(t)$ of the term t interpreted in the data state s. An assignment $x_i := [t_{1i}, t_{2i}]$ indicates that the value of the variable x_i in data state s is changed nondeterministically to any real number in $[s(t_{1i}), s(t_{2i})]$.

Let (v, s) be a state such that $s \in [\![inv(v)]\!]$, and let (v, v') be a transition with the guarded command $\phi \rightarrow \alpha$. If the data state $s \in [\![\phi]\!]$, and all intervals $[s(t_{1i}), s(t_{2i})]$ are non-empty, then the control of A can proceed from location v to location v', with the data state s nondeterministically changed to a data state $s' \in [\![inv(v')]\!]$ such that $s'(x_i) \in [s(t_{1i}), s(t_{2i})]$ if $x_i := [t_{1i}, t_{2i}]$ is in α, and $s'(x_i) = s(x_i)$, otherwise.

We define the *transition-step* relation, $\xrightarrow{e}$, such that $(v, s)\xrightarrow{e}(v', s')$ if and only if the state (v', s') can be reached from the state (v, s) by taking transition e.

Synchronization labels A finite set L of *synchronization labels* and a labeling function *syn* that assigns to each transition $e \in E$ a synchronization label from L. The synchronization labels are used to define the parallel composition of hybrid automata.

A hybrid automaton is *simple* if all the data predicates in the initial conditions, invariants and guards are of the form $x \leq c$ or $x \geq c$ and all the assignments are of the form $x := [c, d]$ for some integer constants c and d.

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 coordinate through shared variables, and to facilitate message-type coordination, we also use synchronization labels on the transitions. The linear hybrid automaton that models the entire system is then constructed from the component automata using a product operation.

Let $A_1 = (X_1, V_1, \varphi_1^0, inv_1, dif_1, E_1, act_1, L_1, syn_1)$ and $A_2 = (X_2, V_2, \varphi_2^0, inv_2, dif_2, E_2, act_2, L_2, syn_2)$ be two linear hybrid automata. Then in the *product* $A_1 \times A_2$ of A_1 and A_2, two transitions e_1 and e_2 from the two component automata A_1 and A_2 are interleaved, unless $syn_1(e_1) = syn_2(e_2)$. If $syn_1(e_1) = syn_2(e_2)$, transitions e_1 and e_2 are synchronized, and cause the simultaneous traversal of component transitions. Formally, the product $A_1 \times A_2$ of A_1 and A_2 is the linear hybrid automaton $A = (X_1 \cup X_2, V_1 \times V_2, \varphi_1^0 \wedge \varphi_2^0, inv, dif, E, act, L_1 \cup L_2, syn)$:

- 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 $x \in X_1 - X_2$, $dif((v, v'), x) = dif_1(v, x)$; for each variable $x \in X_2 - X_1$, $dif((v, v'), x) = dif_2(v', x)$. 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 $act(e) = act_2(e_2)$; and in case (3), $syn(e) = syn_1(e_1) = syn_2(e_2)$ and $act(e) = \phi_1 \wedge \phi_2 \rightarrow \alpha_1 \cup \alpha_2$ if no variable is assigned to two syntactically different terms in $\alpha_1 \cup \alpha_2$, and $act(e) = false \rightarrow \emptyset$ otherwise.

Trajectories and reachability

At any time instant, the state of a hybrid automaton specifies a control location and the values of all variables. The state can change in two ways: (1) by an instantaneous transition that may change both the control location and the values of variables, or (2) by a time delay that changes only the values of variables in a continuous manner according to the rate interval of the current control location. A *trajectory* τ of A is a finite sequence

$$(v_0, s_0) \;\to\; (v_1, s_1) \;\to\; (v_2, s_2) \;\to\; \cdots \;\to\; (v_k, s_k)$$

of control locations $v_i \in V$, and data states $s_i \in \mathbb{R}^n$ such that: (1) $(v_0, s_0) \in [\![\varphi^0]\!]$, (2) for all $0 \leq i < k$, there exists $\pi_i \in \mathbb{R}_{\geq 0} \cup L$ such that $(v_i, s_i) \xrightarrow{\pi_i} (v_{i+1}, s_{i+1})$, where $\mathbb{R}_{\geq 0}$ denotes the set of nonnegative reals. The *final state* of τ is (v_k, s_k). We write $\mathcal{T}(A)$ for the set of trajectories of A. The *reachable region* $R(A)$ of A is the set of final states of trajectories of A.

Reachability analysis using HyTech

The reachability problem (A, ψ) for a hybrid automaton A and a state predicate ψ asks if the region $R(A) \cap [\![\psi]\!]$ is empty; that is, if there is a trajectory in $\mathcal{T}(A)$ whose final state is in $[\![\psi]\!]$. If $[\![\psi]\!]$ represents the set of "unsafe" states specified by a safety property, then this safety property can be verified by reachability analysis. We say that the reachability problem has answer *yes*, if $R(A) \cap [\![\psi]\!] = \emptyset$; and *no*, otherwise.

HyTech (The Cornell Hybrid Technology Tool)[2] is a symbolic model checker for linear hybrid systems [AHH93]. State sets (*regions*) are represented symbolically, as polyhedra. The existing version is limited to reachability analysis. The core of HyTech is a semidecision procedure, which may not terminate on all inputs. The current stable version consists of a main control program in Mathematica which calls C++ subroutines that make use of Halbwachs' polyhedron manipulation library [Hal93, HRP94]. The tool is undergoing reimplementation entirely in C/C++.

Given a reachability problem (A, ψ), HyTech returns the state predicate that characterizes the region $R(A) \cap [\![\psi]\!]$, which provides the necessary and sufficient condition on the parameters under which the answer to reachability problem (A, ψ) is *yes*. However the returned state predicate may be too complex to infer the condition on the parameters. We implemented additional features in HyTech to simplify it by existentially quantifying out information on locations and control variables.

4 Verification of the audio control protocol

System description

The system to be verified is modeled as the composition of the four processes modeled as the hybrid automata depicted in Figures 2 – 5. The input bit stream is nondeterministically generated on-the-fly by the input process. The sender transmits timely *Down* and *Up* signals. The receiver recognizes only the *Up* signals. Instead of adding bits to its own copy of the message, the receiver has its output bits directly acknowledged by the output process. To achieve this, we need only record the sequence of bits thus far sent as input to the sender, but not yet acknowledged as being received. This information is encoded in the variables *leng* and c: *leng* stores the number of unacknowledged

[2] The current Mathematica version of HyTech, including tactics, is available by anonymous ftp from ftp.cs.cornell.edu, directory pub/tah/HyTech (see also http://www.cs.cornell.edu/Info/People/tah/hytech.html).

bits, and c represents their binary encoding. This enables us to model the transmission of arbitrary length bit sequences. We assume that only 3 bits need be stored at any time, and later see how HyTech justifies this assumption.

The operation of each process is briefly described below. The constant Q denotes 1/4 the length of the time slot.

Sender The sender automaton in Figure 2 generates Manchester encoded signals by reading the value of the next bit, and then determining the time for the next voltage change. The locations in which it is delaying until the right time to change voltage are $transhigh_{1,\epsilon}$, $translow_0$, $transhigh_0$, and $translow_1$. The locations $translow_0$ and $transhigh_{1,\epsilon}$, with invariant $x \leq 2Q$, correspond to waiting to send a signal at the end of the time slot, whereas the other two are for sending *Up* and *Down* signals in the middle of time slots. After sending a signal in the middle of a time slot, the process issues in_1 and in_0 commands. These events are synchronized with the input process, and correspond to consuming an input bit with the appropriate value from the input stream, and causing a new bit value to be chosen. The reading signals $head_1$, $head_0$, and $head_\epsilon$ correspond merely to checking the value of the next bit, and are used to decide whether it is necessary to make an intermediate change in voltage at the end of the time slot. For each location v in this automaton, $dif(v, x) = [1-\epsilon, 1+\epsilon]$.

Input The input process in Figure 3 nondeterministically generates valid bit sequences which are either odd in length or end in two trailing 0 bits. This is achieved through the use of an auxiliary variable k which denotes the parity of the length of the bit sequence created so far. Each time the sender consumes a bit, the value of the next bit is chosen. At this point, the process may also terminate the sequence by entering the locations $endeven_{00}$ or *stop*. The values of c and *leng* are also updated appropriately. This process also provides the sender read-access of the next bit value through the $head_1$, $head_0$, and $head_\epsilon$ signals. For each location $dif(v, c) = dif(v, k) = dif(v, leng) = [0, 0]$.

Receiver The receiver automaton in Figure 4 decodes its incoming *Up* signals by rounding its local time for when it received the signal to the nearest possible time it expects a signal. If no signal is received in due time (within $7Q$ if the last received bit was a 0, and $9Q$ otherwise), the sequence is interpreted as being complete. It also uses an auxiliary variable to record the parity of the received sequence, since the finalization of the bit stream involves adding a trailing 0 unless the input sequence is odd and the last received bit is 1. Whenever an *Up* signal is received the component attempts to emit an output for values it believes were sent. This output must synchronize with the output-acknowledgement process, which checks the attempted output value with the leading bit of the currently unacknowledged input. For each location v in the receiver automaton, $dif(v, y) = [1-\epsilon, 1+\epsilon]$ and $dif(v, m) = [0, 0]$.

Output-acknowledgement The role of the output automaton in Figure 5 is to monitor attempts by the receiver to output bits. It looks at the number of currently unacknowledged bits, and their binary encoding to infer whether the leading bit is a 0, 1, or non-existent (in the case of no remaining bits). Whenever bits are correctly acknowledged, the values of c and *leng* are updated.

We note that HyTech's limitation to closed intervals for timing constraints forces us to model slightly inaccurately the strict inequalities of the original protocol [BPV94].

Specification

Correct reception of bits. Our paradigm for specifying correctness is reachability analysis, in which we label certain states as violating. The system is correct if no violating states are reachable.

Correctness of the received bit stream is verified by comparing each output bit with the value of the currently unacknowledged bits. If a bit cannot be correctly matched with the input bits, the receiver enters an error location. We also verify the entire bit stream is received by adding a transition from the receiver's *out* location to the *error* location conditional on any input bits remaining unmatched. Thus the violating states are specified as $l[receiver] = error$.

Time-bounded output. Bosscher et al. prove the receiver's output occurs within a certain time bound dependent on the length of the message, namely within $(4m + 5)Q/(1-\epsilon)$ where m is the length of the bit stream. Direct encoding of this property in hybrid automata would involve tracking the exact length of the bit stream in a separate variable. However, this would lead to a nonterminating reachability analysis where the value of m increases arbitrarily. We avoid this problem by proving instead a stronger property that can be checked without the need to record the message length. We use an additional timer that is reset every time a new input bit is generated. If there are m input bits, then the timer may only be reset m times. If the value of the timer is no more than $4Q/(1-\epsilon)$ every time it is reset except for the last time when it may reach $9Q/(1-\epsilon)$ in the input's *idle* location, the total accumulated time does not exceed $(4(m-1)+9)Q/(1-\epsilon) = (4m+5)Q/(1-\epsilon)$.

One technicality remains: the current version of HyTech allows only nonstrict inequalities in its constraints, so we cannot directly specify violation states with expressions such as $z > 4Q/(1-\epsilon)$. We overcome this by specifying instead a violation if $z \geq 4Q/(1-\epsilon)$, and then visually inspecting HyTech's output to check that the only possible violations in this location occur at $z = 4Q/(1-\epsilon)$.

Analysis

We verify the protocol using Philip's error tolerance of 1/20. We arbitrarily choose a value of 1 for Q, since the protocol's correctness is independent of its value. HyTech successfully discovers that the violating states are not reachable. Notice that this justifies our assumption that at most three unacknowledged bits need be stored, since the output process flags errors whenever there are four or more bits unacknowledged. The correctness of the transmitted sequence is verified in 4.9 hours. The verification of the timing property also takes 4.9 hours[3]. All performance data in this paper was measured on a Sun 670MP workstation.

5 Synthesis

We show how algorithmic techniques can automatically discover the critical 1/17 error tolerance.

Methodology

To synthesize the permissible clock drifts under which the protocol is correct, the first natural step would be to add to the product automaton A a new parameter ϵ for the

[3] Preliminary results from our new implementation show marked improvement, with the correctness of bits verified in 48 seconds and the timing property in 61 seconds.

error tolerance. The rates of change of each clock would now vary between $[1-\epsilon, 1+\epsilon]$ instead of the fixed bounds $[19/20, 21/20]$. However the resulting automaton is no longer a linear hybrid automaton. Linear hybrid automata restrict the rates of change to be bounded by constants in order to guarantee reachable regions are describable by linear inequalities. We overcome this problem by considering ϵ as a constant and applying two transformations which are trace-preserving for every non-zero constant value of ϵ.

Recall that our modeling of the system allows two kinds of errors: correctness errors due to faulty reception of bit values, and modeling "errors" due to our assumption of an *a priori* bound on the number of unacknowledged bits. By analyzing these two categories of errors separately, HyTech shows that any parameter values causing modeling errors also cause correctness errors, so that the bound is indeed necessary and sufficient.

The synthesis procedure starts from the automaton A, and is summarized as follows:

1. introducing a syntactic constant ϵ for the clock drift, yielding $A(\epsilon)$,
2. transforming $A(\epsilon)$ into $A'(\epsilon)$, by moving the constant from the rates into constraints,
3. transforming $A'(\epsilon)$ into $A''(\epsilon)$, by making constraints linear in the constant,
4. interpreting the constant ϵ as a parameter ranging over all values for the constant,
5. synthesizing the bound of $1/17$, and,
6. checking $1/17$ is both necessary and sufficient.

Clock transformations

We assume the symbolic constant ϵ has been introduced into the rate intervals of both the sender's and receiver's clocks. Let $A(\epsilon)$ denote the resulting automaton for each particular value of $\epsilon \in \mathbb{R}$[4]. The following two transformations are applied to each $A(\epsilon)$.

Transformation I: eliminating parameterized slopes. We first move the constant ϵ from the slopes of the clock variables into constraints on the clocks' values. The automaton transformation, for constant bounded slopes, was reported in [OSY94]. It applies to automata where there are no constraints explicitly enforced on the relative differences between variables.

The clock transformation K applied to a simple hybrid automaton A yields the simple hybrid automaton $K(A)$ where each variable x is replaced by the primed variable x' with the rate interval $[1, 1]$. For simplicity we assume that for each variable v the rate intervals are fixed throughout the automaton as $[l_v, u_v]$, with all bounds strictly positive. The locations and transitions of $K(A)$ as the same as those of A. The clock transformation K replaces (1) the atomic data predicates of the form $x \leq c$ ($x \geq c$) in the initial condition, invariants, and guards by the data predicate $x' \leq c/l_x$ ($x' \geq c/u_x$), and (2) the assignments of the form $x := [c, d]$ by the assignments $x' := [c/u_x, d/l_x]$.

For example, in our automata from Figures 2 and 4, atomic data predicates of the form $3Q \leq x \leq 5Q$ are replaced by $3Q/(1+\epsilon) \leq x' \leq 5Q/(1-\epsilon)$, and $x = 2Q$ by $2Q/(1+\epsilon) \leq x' \leq 2Q/(1-\epsilon)$, provided $\epsilon \notin \{-1, 1\}$.

The clock transformation K is sound but not complete for reachability problems of simple hybrid automata. Fortunately, our automata fall into the subclass of linear hybrid automata known as the *reset skewed clock automata (RSCA)* introduced in [WT94]. It is proven there that for RSCAA and location predicates ψ, the reachability problems $(K(A), \psi)$ and (A, ψ) have the same answer.

[4] Technically we need to extend our definition of automata to admit real-valued constants. Details are omitted.

Definition 1. A reset skewed clock automaton (RSCA) is a simple linear hybrid automaton whose constraints satisfy the following properties:

- all slopes of variables are positive,
- the values of two variables (clocks) are never compared in any constraints,
- all the assignments set variables to $[0, 0]$, and
- along any path in the automaton structure, there are never two constraints on a variable value without there being an intervening reset of the variable to a fixed value, except that an upper bound constraint may occur as long as it is preceded by a looser upper bound constraint since the last reset.

Theorem 2. *[WT94] If A is a reset skewed clock automata and ψ is a location predicate, then the reachability problems $(K(A), \psi)$ and (A, ψ) have the same answer.*

The sender automaton is then first converted into a RSCA before applying K, while the receiver automaton is already a RSCA and can have the transformation applied directly.

Transformation II: eliminating non-linear constraints. The resulting automaton $K(A(\epsilon))$ has non-linear constraints where ϵ appears in the *denominator.* We eliminate these sources of non-linearity by applying a clock-transformation that multiplies every timing constant by $(1-\epsilon)(1+\epsilon)$. This operation corresponds to changing the time scale by a constant factor. We denote the transformation by value c as T_c.

Theorem 3. *If A is a hybrid automaton and ψ is a location predicate, then for any non-zero c, the reachability problems $(T_c(A), \psi)$ and (A, ψ) have the same answer.*

Therefore, given a RSCA A, a constant ϵ such that $\epsilon \notin \{-1, 1\}$ and a location predicate ψ, the reachability problems $(T_{(1-\epsilon)(1+\epsilon)}(K(A(\epsilon))), \psi)$ and (A, ψ) have the same answer. The automata for the sender and receiver after these two transformations are shown in Figures 6 and 7.

Monitoring overflow

The final remaining difficulty has to do with modeling the system under uncertainty of the value of the parameter ϵ. Recall that our finite state modeling assumed that there could be at most 3 unacknowledged bits in the system. This assumption was shown true for the fixed parameter $\epsilon = 1/20$. However, for wider tolerances it may be possible for the number of unacknowledged bits to grow arbitrarily large as the sender may process bits much faster than the receiver.

These "errors" should be separated from errors in faulty reception, since they are due to modeling simplifications only, and may not correspond to truly faulty systems. We therefore create a new location in the receiver to catch all instances where the number of unacknowledged bits exceeds 3.

Analysis

HyTech provides the synthesized bound by showing that correctness errors occur whenever $1/17 \leq \epsilon \leq 1$ and the finiteness assumption is only violated if $1/15 \leq \epsilon \leq 1$. Therefore the bound of $1/17$ is both necessary and sufficient for correctness. Total computation time was 27.9 hours[5].

[5] The new implementation being developed takes 14 minutes.

6 Improved design: a more tolerant receiver

With the help of HyTech we are able to modify slightly the receiver's protocol to enable a wider clock drift. A casual perusal of HyTech's detailed output of the reachable violating states showed that the critical bound of $1/17$ is not reached in many situations. Since the absolute clock drifts are maximal when the clocks are compared to large values, we suspected the bound to be generated by the receiver's final output of the message at time $9Q$ after its last received *Up* signal. We reran HyTech with all premature termination errors caught in a separate location. HyTech's output then shows that such termination errors occur when $1/17 \leq \epsilon \leq 1$, other correctness errors occur only when $1/15 \leq \epsilon \leq 1$, and modeling errors occur when $1/15 \leq \epsilon \leq 1$. Thus the bound of $1/17$ is only necessary for generating errors when the receiver believes the message is terminated before acknowledging all bits. We therefore altered the receiver to wait longer before assuming transmission was over. When it delays until $10Q$ instead of $9Q$ out of its location *last_is_1*, the error tolerance should be widened to $1/15$. HyTech verified this in 27.5 hours[6].

7 Conclusions

Our analysis demonstrates some techniques that may well prove useful in examining other systems: for example, the use of finite state modeling assumptions which are justified by the verifier itself, the transformations on constants to enable rate information to be synthesized, and the separation of errors to gain greater insight into the protocol.

It would be interesting to see whether there are better choices of timing constants in the sender and receiver that would allow even greater clock drifts. Also, there may be better protocols that do not rely on Manchester encoding. It may be possible to use HyTech to help synthesize these.

Acknowledgements. We thank Peter Kopke for a careful reading, and Prof. Thomas A. Henzinger for helpful comments.

References

[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.

[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.

[6] The new implementation being developed takes 23 minutes.

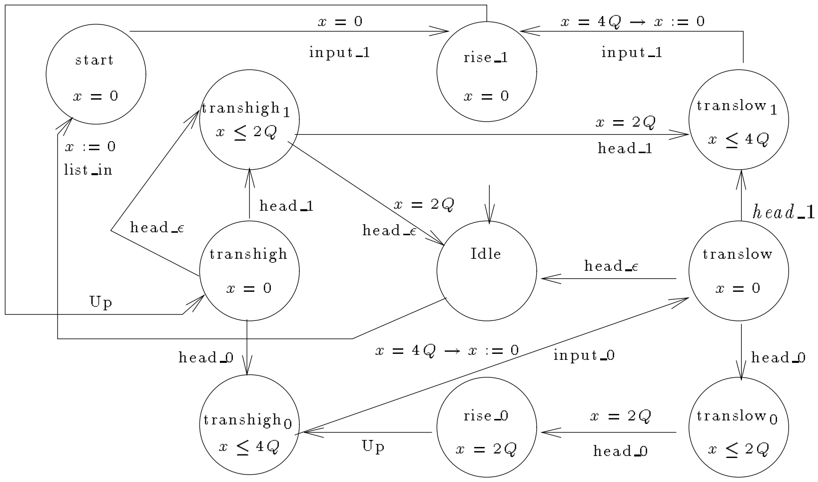

Fig. 2. The sender automaton

[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.

[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.

[HH95] T.A. Henzinger and P.-H. Ho. HyTech: The Cornell Hybrid Technology Tool. To appear, 1995.

[HRP94] N. Halbwachs, P. Raymond, and Y.-E. Proy. Verification of linear hybrid systems by means of convex approximation. In *Proceedings of the First Static Analysis Symposium*, 1994.

[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.

[OSY94] A. Olivero, J. Sifakis, and S. Yovine. Using abstractions for the verification of linear hybrid systems. In D.L. Dill, editor, *CAV 94: Computer-aided Verification*, Lecture Notes in Computer Science 818, pages 81–94. Springer-Verlag, 1994.

[WT94] Howard Wong-Toi. *Symbolic Approximations for Verifying Real-Time Systems.* PhD thesis, Department of Computer Science, Stanford University, CA, December 1994.

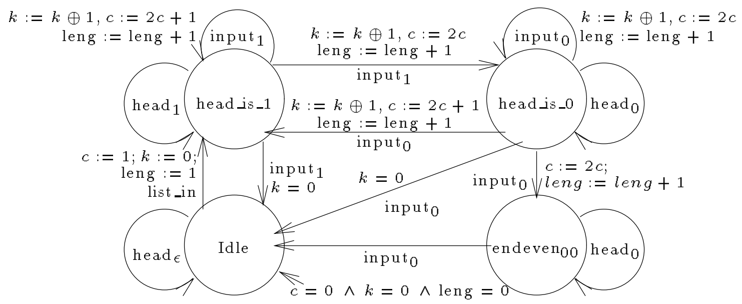

Fig. 3. The input automaton

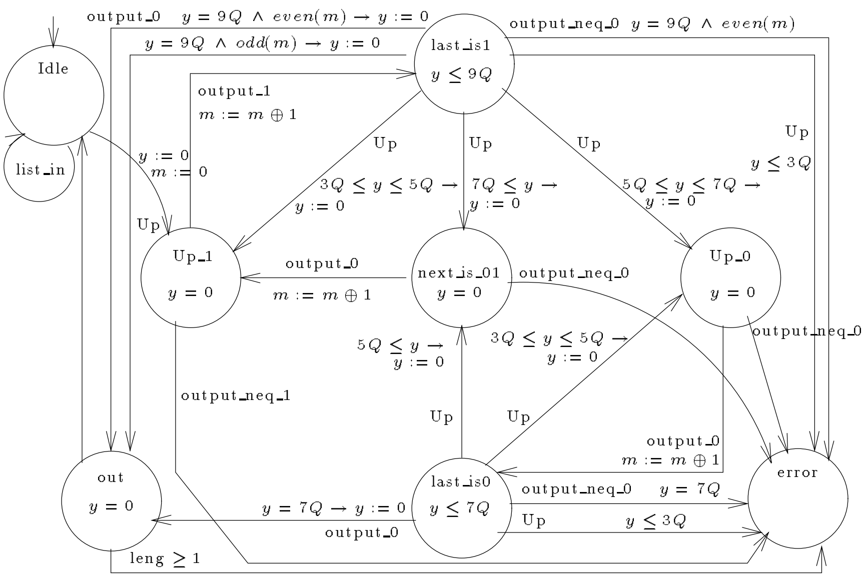

Fig. 4. The receiver automaton

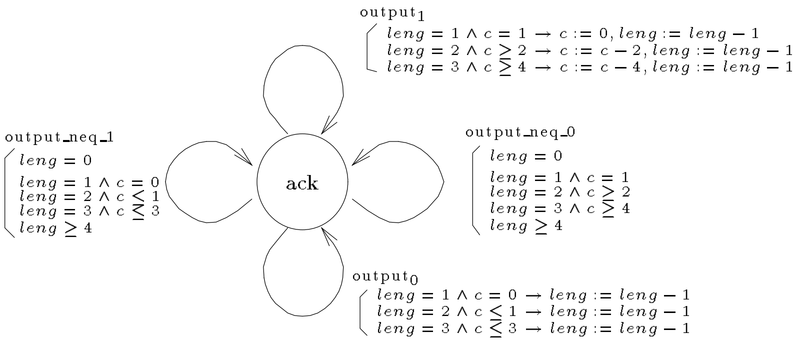

Fig. 5. The output automaton

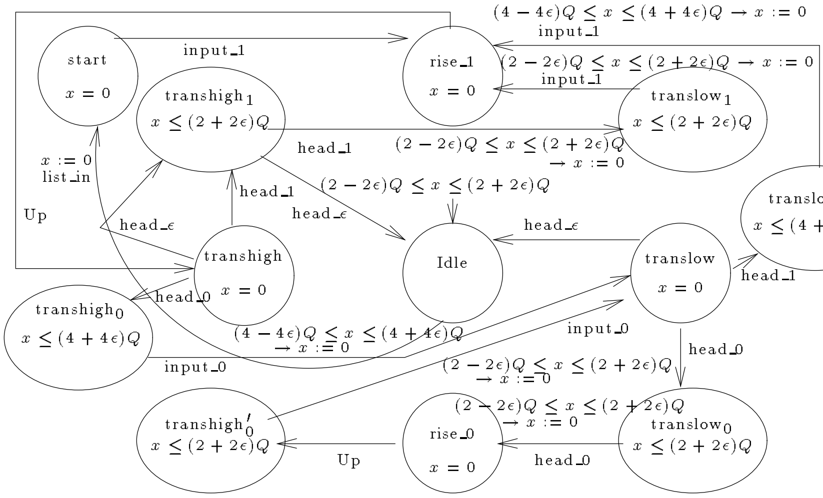

Fig. 6. The transformed sender automaton

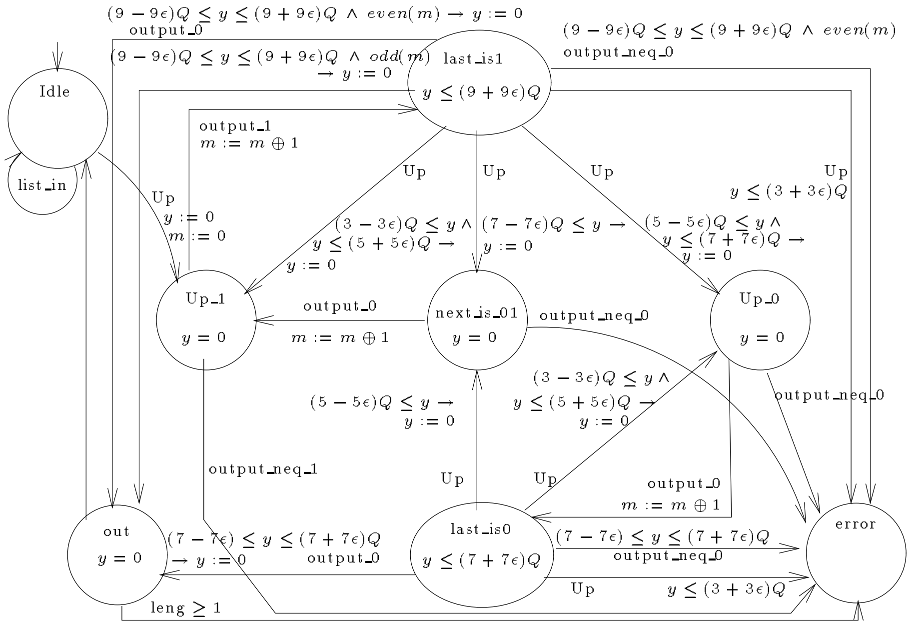

Fig. 7. The transformed receiver automaton