HYTech

- HYTech is a tool for automated analysis of hybrid systems
- HYTech aids in the design of hybrid systems by:
  - checking system requirements
  - doing parametric analysis
  - Symbolic Calculations
  - Forward
  - Backward
  - Reachability

HYTech

- HYTech’s input consists of two parts:
  - System description
    - HYTech’s system description language allows the user to represent linear hybrid automata textually
  - Analysis commands
    - The analysis command language allows the user to write simple scripts to perform tasks like reachability analysis, parametric analysis, error trace generation etc.
A User Guide to HYTECH†

Thomas A. Henzinger
EECS Department
University of California
Berkeley, CA
tah@eecs.berkeley.edu

Pei-Hsin Ho
Intel Development Labs
Intel Corporation
Hillsboro, OR
pho@ichaips.intel.com

Howard Wong-Toi
Cadence Berkeley Labs
Cadence Design Systems
Berkeley, CA
howard@cadence.com

Abstract

HYTECH is a tool for the automated analysis of embedded systems. This document, designed for the
first-time user of HYTECH, guides the reader through the underlying system model, and through the
input language for describing and analyzing systems. The guide gives installation instructions, several
examples of usage, some hints for gaining maximal computational efficiency from the tool, and the
complete grammar for the input language.

This guide describes version 1.04 of HYTECH. The latest update occurred in October 1996†. HYTECH
is available through the World-Wide Web at http://www.eecs.berkeley.edu/~tah/HYTECH.

1 Introduction

The control of physical systems with embedded hardware and software is a growing application area for
computerized systems. Since many embedded controllers occur in safety-critical situations, it is important
to have reliable design methodologies that ensure that the controllers operate correctly. HYTECH aids in
the design of embedded systems by not only checking systems requirements, but also performing parametric
analysis. Given a parametric system description, HYTECH returns the exact conditions on the parameters
for which the system satisfies its safety and timing requirements.

For completeness, we begin with a brief presentation of the underlying theoretical framework of linear
hybrid automata [ACH93, ACH †95], which we use to describe system specifications and requirement
specifications. These automata model the continuous activities of analog variables (such as temperature, time,
and distance), as well as discrete events (such as interrupts and output signals). Communication is modeled
through event synchronization and shared variables. HYTECH’s input consists of two parts: a system
description and analysis commands. The system-description language allows us to represent linear hybrid
automata textually. The tool forms the parallel composition of a collection of automata, each describing
a modular component of an embedded system. The analysis-command language allows us to write simple
iterative programs for performing tasks such as reachability analysis and error-trace generation.

We illustrate the use of the tool on several examples taken from the literature, and provide hints for a
verification engineer to gain the maximal possible efficiency from HYTECH.

Outline Section 2 reviews linear hybrid automata, their semantics, parallel composition, and associated
analysis techniques. A brief history of HYTECH appears in Section 3. Sections 4 and 5 describe the HYTECH

†This research was supported in part by the Office of Naval Research Young Investigator award N00014-95-1-0520, by the
National Science Foundation CAREER award CCR-9501708, by the National Science Foundation grants CCR-9200794 and
CCR-9504469, by the Air Force Office of Scientific Research contract F49620-93-1-0050, by the Advanced Research Projects
Agency grant NAGZ-892.

†An abbreviated version of this paper appeared in the Proceedings of the First Workshop on Tools and Algorithms for the

†The only changes from version 1.02 of February 1996 are bug fixes, porting to Linux, DEC Ultras, Digital Unix, and HP-UX
in May 1996, and replacement of the closure difference operator cldiff with the exact difference operator diff in October
1996.
Automated Analysis of an Audio Control Protocol*

Poi-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 practitioner 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 1/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 [AHH93]. 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.
Fischer's Mutual Exclusion Protocol (Mex)

\[ k = 0 \rightarrow x := 0 \quad k := 1; x := 0 \quad x \geq b \land k \neq 1 \]

\[ \frac{4}{5} \leq \hat{z} \leq 1 \]

\[ k := 0 \quad \frac{4}{5} \leq \hat{z} \leq 1 \quad x \geq b \land k = 1 \]

\[ \{k = 0\} \]

\[ k = 0 \rightarrow y := 0 \quad k := 2; y := 0 \quad y \geq b \land k \neq 2 \]

\[ y \leq a \quad 1 \leq \hat{y} \leq \frac{11}{10} \]

\[ k := 0 \quad 1 \leq \hat{y} \leq \frac{11}{10} \]

\[ y = b \land k = 2 \]

\[ P_i: \quad \text{repeat} \]

\[ \text{repeat} \]

\[ \text{await} \quad k = 0 \]

\[ k := i; \text{ delay } b \]

\[ \text{until} \quad k = i \]

\[ \text{Critical section} \]

\[ k := 0 \]

\[ \text{forever} \]

\[ 8b > 11a \]

<table>
<thead>
<tr>
<th>Number of locations</th>
<th>Number of transitions</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>81</td>
<td>3.8 sec.</td>
</tr>
</tbody>
</table>

Mutual exclusion (two processes are never in the critical section at the same time).
HyTech Demo Input

You may use our input files, modify our input files, or write your own input files. For editing, use the mouse to cut and paste, or use the mouse to select a position and type in your changes.

System definition: a set of hybrid automata

-- Fischer’s mutual exclusion protocol

\[\begin{align*}
\text{var} & \\
  x, & \quad \text{-- P1's clock} \\
  y, & \quad \text{-- P2's clock} \\
  k, & \quad \text{-- whose turn it is (has values 0,1,2)} \\
  a, & \quad \text{-- max delay time to register intent} \\
  b & \quad \text{-- min time to delay before rechecking} \\
  & \\
\end{align*}\]

automaton p1
synclabs : ;

initially loc_1 & True;

loc loc_1: while True wait {dx in [4/5,1]}
  when k=0 do {x' =0} goto loc_2;

loc loc_2: while x<=a wait {dx in [4/5,1]}
  when True do {k' =1, x' =0} goto loc_3;

loc loc_3: while True wait {dx in [4/5,1]}
  when x=b & k=1 goto cs;
  \text{-- two failed attempts}
  when x=b & k=0 goto loc_1;
  when x=b & k=2 goto loc_1;

loc cs: while True wait {dx in [4/5,1]}
  when True do {k' = 0} goto loc_1;
end

automaton p2
synclabs : ;

initially loc_1 & True;

loc loc_1: while True wait {dy in [1,11/10]}
  when k=0 do {y' =0} goto loc_2;

loc loc_2: while y<=a wait {dy in [1,11/10]}
  when True do {k' =2, y' =0} goto loc_3;

loc loc_3: while True wait {dy in [1,11/10]}
  when y=b & k=2 goto cs;
  \text{-- two failed attempts}
  when y=b & k=0 goto loc_1;
  when y=b & k=1 goto loc_1;
loc cs: while True wait (dy in [1,11/0])
        when True do (k' = 0) goto loc_1;
end

**Requirements analysis: a script of verification commands**

```plaintext
-- analysis commands

var
    init_reg, final_reg, reached : region;

init_reg:= loc[p1] = loc_1 & loc[p2] = loc_1 & k=0;
reached:= reach backward from final_reg endreach;
prints "Condition for faulty system";
print omit all locations hide non_parameters in reached & init_reg endhide;
```

- Back to the HyTech home page
- For questions and comments send mail to hytech@eecs.berkeley.edu

*Last updated in July, 1996, by tah@eecs.berkeley.edu*

---

**RESULTS**

Setting output level to 1
Checking automaton p2
Checking automaton p1
Composing automata *
Iteration: 0
Using this newly reached list ================
Location: cs.cs True
Iteration: 1
Using this newly reached list ================
Location: cs.loc_3
    k = 2
Location: loc_3.cs
    k = 1
Iteration: 2
Using this newly reached list ================
Location: cs.loc_2
    y <= a
Location: loc_2.cs
    x <= a

[OUTPUT DELETED.]

Number of iterations required for reachability: 12

Condition for faulty system
8b <= 11a & a >= 0

================================================================================
Max memory used =  1016 kilobytes = 0.99 MB
Time spent = 0.23u + 0.04s = 0.27 sec total
================================================================================
Railroad Crossing: Train, Controller, and Gate

Train

- Far: \( x \geq 1000 \)
  - \( \dot{x} \in [-50, -40] \)
- Near: \( x \geq 0 \)
  - \( \dot{x} \in [-50, -30] \)
- Past: \( x \leq 100 \)
  - \( \dot{x} \in [30, 50] \)

Gate

- Open: \( g = 90 \)
- Close: \( g = 0 \)

Controller

- About to lower: \( t \leq \alpha \)
- Idle: \( t = 0 \)
- About to raise: \( t \leq \alpha \)

When the train is within 10 meters to the gate, the gate is always fully closed.

<table>
<thead>
<tr>
<th>( \alpha &lt; 49/5 )</th>
<th>Number of locations</th>
<th>Number of transitions</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>36</td>
<td>90</td>
<td>0.2 sec.</td>
</tr>
</tbody>
</table>


HyTech Demo Input

You may use our input files, modify our input files, or write your own input files. For editing, use the mouse to cut and paste, or use the mouse to select a position and type in your changes.

**System definition:** a set of hybrid automata

```plaintext
-- train-gate-controller

var
    x, -- distance
    y, -- angle of gate
    t -- dcontrollers timer
    : analog:
    alpha -- cutoff point for controller to issue commands
    : parameter

automaton train
    synclabs : app, -- (send) approach signal for train
               exit;  -- (send) signal that train is leaving

    initially far & x>=2000;

    loc far: while x>=1000 wait {dx in [-50,-40]}
              when x=1000 sync app goto near;

    loc near: while x>=0 wait {dx in [-50,-30]}
               when x=0 goto past;

    loc past: while x<=100 wait {dx in [30,50]}
               when x=100 do {x' >=2000} sync exit goto far;

end -- train

automaton controller
    synclabs: app,
               exit,
               lower, -- lower command sent to the gate
               raise; -- raise command sent to the gate

    initially idle & True;

    loc idle: while True wait {dt = 0}  -- wait for a signal from train
              when True sync app do {t' = 0} goto about_to_lower;
              when True sync exit do {t' = 0} goto about_to_raise;

    loc about_to_lower: while t<=alpha wait {dt = 1}
                          when True sync app goto about_to_lower;
                          when True sync exit do {t' = 0} goto about_to_raise;
                          -- send lower signal any time before t<=alpha;
                          when True sync lower goto idle;

    loc about_to_raise: while t<=alpha wait {dt in [1,1]}
                          when True sync app do {t' = 0} goto about_to_lower;
                          when True sync exit goto about_to_raise;
                          -- send raise signal any time before t<=alpha
                          when True sync raise goto idle;
```
end -- controller

automaton gate
synclabs: raise, lower;
initially open & y=90;

loc up: while y<=90 wait {dy in [9,9]} -- gate is being raised
   -- gate is fully raised
   when y=90 goto open;
   -- selfloops for receptiveness
   when True sync raise goto up;
   when True sync lower goto down;
   when x<=10 goto error;

loc open: while True wait {dy in [0,0]} -- wait for command
   when True sync raise goto open;
   when True sync lower goto down;
   when x<=10 goto error;

loc down: while y>=0 wait {dy in [-9,-9]} -- gate is being lowered
   -- gate is fully down
   when y=0 goto closed;
   when True sync lower goto down;
   when True sync raise goto up;
   when x<=10 goto error;

loc closed: while True wait {dy in [0,0]} -- wait for command
   when True sync raise goto up;
   when True sync lower goto closed;

loc error: while True wait {dy in [0,0]}

end -- gate

Requirements analysis: a script of verification commands

-- analysis commands

var init_reg, final_reg, reached: region;

init_reg := loc[train]=far & x>=2000 &
   loc[controller]=idle &
   loc[gate]=open & y=90;
final_reg := loc[gate] = up & x<=10 | loc[gate]=open & x<=10 |
   loc[gate] = down & x<=10;
reached := reach forward from init_reg endreach;

prints 'Conditions under which system violates safety requirement';
print omit all locations
   hide non_parameters in reached & final_reg endhide;

- Back to the HyTech home page
- For questions and comments send mail to hytech@eecs.berkeley.edu

Last updated in July, 1996, by tah@eecs.berkeley.edu
HyTech Demo Output

on a Digital AlphaServer 1000A 4/266 with 128Mb of memory

=================================================================

HyTech: symbolic model checker for embedded systems
Version 1.04 10/15/96
For more info:
  email: hytech@eecs.berkeley.edu
  http://www.eecs.berkeley.edu/~tah/HyTech
Warning: Input has changed from version 1.00(a). Use -i for more info
=================================================================

Setting output level to 1
Checking automaton gate
  WARNING: locn error of automaton gate has no outgoing transitions
Checking automaton controller
Checking automaton train
Composing automata **
Iteration: 0
Using this newly reached list ================================
  Location: far.idle.open
  y = 90  &  x >= 1000
Iteration: 1
Using this newly reached list ================================
  Location: near.about_to_lower.open
  y = 90  &  t <= alpha  &  x + 30t <= 1000  &  x + 50t >= 1000  &  x >= 0
Iteration: 2
Using this newly reached list ================================
  Location: past.about_to_lower.open
  y = 90  &  t <= alpha  &  x >= 0  &  50t >= x + 1000  &  30t <= x + 1000  &  x
Location: near.about_to_lower.error
  y = 90  &  t <= alpha  &  x <= 10  &  x + 50t >= 1000  &  x >= 0  &  x + 30t <=
Location: near.idle.down
  t <= alpha  &  x >= 0  &  9x + 450t >= 50y + 4500  &  y <= 90  &  3x + 90t <= 1

*[ 1 1/2 PAGES DELETED]*

y = 0  &  x + 60t <= 30alpha + 400  &  t >= 0  &  alpha >= 12  &  x + 60t <= 10
Number of iterations required for reachability: 10

Conditions under which system violates safety requirement
5alpha >= 49

=================================================================

Max memory used = 920 kilobytes = 0.90 MB
Time spent = 0.25u + 0.03s = 0.28 sec total
=================================================================

Use the "BACK" button of your browser to modify the input file.

- Back to the HyTech home page
- For questions and comments send mail to hytech@eecs.berkeley.edu

Last updated in July, 1996, by tah@eecs.berkeley.edu