Appears in Proc. Conf. on Decision and Control
pp. 1223-1228, Orlando, FL, 2001

Modifying HyTech to Automatically Synthesize Hybrid Controllers

Ranjana G. Deshpande, David J. Musliner, Jorge E. Tierno, Steve G. Pratt, Robert P. Goldman
Automated Reasoning Group
Honeywell Laboratories
3660 Technology Drive
Minneapolis, MN 55418
{rdeshpan,musliner,jtierno,spratt,goldman}@htc.honeywell.com

Abstract

We present HoneyTech, a tool for automatic synthesis of hybrid controllers. HoneyTech is an extension of the HyTech tool, with added features for linear hybrid automata region modeling and analysis. The tool has also been enhanced to improve performance by reducing the number of redundant operations during region computations. The tool has been used to implement the Wong-Toi hybrid controller synthesis algorithm and has been tested on several examples. In this paper, we present an extension of the algorithm to include parametrized controllable transtions. We discuss the HoneyTech implementation of the algorithm and its application to some simple control examples. We also present a brief performance evaluation of the algorithm for increasing scales of magnitude of the control problem.

1 Introduction

Hybrid dynamic systems exhibit both continuous dynamics and discrete event dynamics. Hybrid system models are useful for representing and analyzing a wide variety of real world systems, such as industrial processes and air control systems. In most of these domains, the safe and reliable operation of these systems is an area of primary concern. To ensure the reliability and safety of hybrid systems, we are interested in automatically synthesizing and verifying hybrid controllers. Automatic synthesis of controllers helps in reduction of design and certification time, and can help improve the performance of these systems.

Hybrid systems can be mathematically modeled as a parallel composition of concurrent hybrid automata. Linear hybrid automata (LHA) are a subset of hybrid automata that have been proven computationally tractable for modeling and analysis and can model many useful hybrid systems (or their linear approximations). By generalizing the Ramadge and Wonham regression algorithm [5], Wong-Toi [7] has proposed an algorithm for automatic synthesis of controllers for linear hybrid systems.

In this paper, we describe our extension to the Wong-Toi synthesis algorithm to include parametrized controllable transitions. We also describe several new constructs that we have added to HyTech [2], a tool that provides symbolic model checking (analysis and verification) for LHA models. The HoneyTech[1] tool includes several new analysis functions not available in HyTech, and can automatically synthesize controllers for LHAs. We illustrate the implementation of the synthesis algorithm in HoneyTech, with the help of a simple truck navigation control example. Finally, we present a performance evaluation of the algorithm for increasing levels of complexity in the above example.

2 Definition of Linear Hybrid Automata

A linear hybrid automaton H is the 9-tuple (`X`, `V`, `E`, *inv*, *init*, *jump*, *flow*, Σ, *sync*) with the following components (for a detailed discussion, see [2]):

`X` is a finite set of variables $\{x_1, \ldots, x_n\}$ in $\mathbf{R^n}$ that models the continuous dynamics of the system. The valuation `s` of `X` is a point in $\mathbf{R^n}$, where `s` $= \{s_1, \ldots, s_n\}$ and s_i is the value of the `i-th` value of `X`.

`V` is a finite set of control modes. The state of an automaton is defined as the tuple `(v,s)` consisting of $v \in V$ and valuation `s`.

`E` is a finite multiset of directed edges between a source control mode and a target control mode. The control graph `(V,E)` represents the discrete dynamic component of an automaton.

[1]Our HyTech license agreement requires us to give our modified version a new name.

inv is a mapping from `V` to the predicates over `X`. *inv(v)* is the invariant condition at control mode v.

init is a mapping from `V` to the predicates over `X`. *init(v)* represents the initial conditions at control mode v. At initial state `(v,s)`, *init(v)* and *inv(v)* are true for the valuation `s`.

flow is a mapping from `V` to predicates over $X \cup \dot{X}$. At control mode `v`, the automaton state evolves according to the differentiable functions which satisfy the flow conditions *flow(v)*.

Σ is a finite set of events comprising controllable events Σ_c and uncontrollable events Σ_u. *sync* is the finite set of synchronization labels for each event. Concurrent automata must take transitions with common synchronization labels simultaneously.

jump is a mapping of E to predicates over $X \cup X'$. The jump condition *jump(e)* maps the values of variables `X` before and after a transition due to the control switch e.

A hybrid system is the parallel composition of concurrent hybrid autmata. The automata communicate with each other via shared variables and event synchronization labels.

3 Controller Synthesis for LHAs

3.1 The Wong-Toi Algorithm

The Wong-Toi algorithm is an iterative, symbolic procedure for synthesizing controllers for hybrid systems modeled as linear hybrid automata. The procedure synthesizes a legal controller by "pruning away bad configurations from the set of potentially controllable configurations to yield a maximal set of controllable configurations" (see [7]).

For an LHA H, let K denote the set of unsafe regions and A denote the extended unsafe regions $K \cup \mathtt{Pre}(K, \Sigma_u)$. In order to compute the controllable regions, the algorithm first computes the following region sets (see Figure 1 for formal definitions):

- $E(K)$ which denotes the *escape* configurations, from where the controller can issue a single control event which avoids the unsafe region.
- $flow\_avoid(A, E)$ which denotes the fixpoint characterization of the set of unsafe regions that are in A or flow into A as a finite sequence of linear witness flows, avoiding the escape region E.
- $\gamma'(K)$ which represents the *unavoidable predecessor* regions, from which it is impossible for the controller to prevent the plant from reaching K in a single control action or in a single control action followed by a single flow step.

Given an unsafe region, *K_init*, the synthesis method iteratively computes the set of controllable configurations for the automata, as defined by the fixpoint formula, $\mu W.K\_init \cup \gamma'(W)$, where W denotes the maximal set of unsafe regions.

3.2 Extending Wong-Toi Algorithm to Handle Parameterized Controllable Transitions

In many control situations, it is sometimes desirable to allow controllable events to depend on random parameters, to model, for example, uncertainties in response times. We have modified the Wong-Toi synthesis algorithm to include parametrized transitions and present the modification for a single variable x and a single parameter r. This may be generalized to allow multi variables and multi parameters.

According to Wong-Toi's formulas the escape set corresponding to unsafe set iteration variable,W, is :

$$\begin{aligned} E \;=\; & \overline{\mathrm{pre}(\mathrm{W}, \Sigma_c)} \cap \\ & \mathrm{pre}(\overline{\mathrm{W}}, \Sigma_c) \cap \\ & \overline{W} \end{aligned}$$

For sake of brevity, we define region sets A[2]and B, such that the escape set computation equation can now be written as $E = \overline{A} \cap B \cap \overline{W}$, where

$A = \{x : \exists \sigma_c \in \Sigma_c, \mathrm{Post(x}, \sigma_c) \in \mathrm{W}\}$.

If σ_c depends on r, we should expand the set A to include all points that map into the set W for at least one value of r, such that $A = \{x : \exists r \exists \sigma_c \in \Sigma_c, \mathrm{Post(x}, \sigma_c) \in \mathrm{W}\}$.

The second term of the escape set definition is the set of regions that map to a safe region via a controllable event, defined as $B = \{x : \exists \sigma_c \in \Sigma_c, \mathrm{Post(x}, \sigma_c) \in \overline{\mathrm{W}}\}$. If the controllable events depend on r, set B is constrained to include only those regions that map into a safe set for *all allowable* values of r. Let R_{σ_c} be the set of allowable values of r. Then $B =$

[2]The region A defined here is merely a region notation and is distinct from the region A defined in Wong-Toi's algorithm

$$\mathrm{Escape(K) = E = \overline{K} \cap \bigcup_{\sigma_c \in \Sigma_c} (Pre(\overline{K}, \sigma_c) \cap \overline{Pre(K, \sigma_c)})}$$

$$\overline{E} = C = \bigcup_{1 \leq i \leq m} C_i$$

$$A = K \cup Pre(K, \Sigma_u) = A_1 \cup \ldots \cup A_r$$

$$\mathrm{flow\_avoid(A_r, E) = \mu W.A \cup \bigcup_{1\leq\ i \leq m} (Pre_t(Pre_t(W) \cap cl(W) \cap cl(C_i) \cap \overline{E}) \cap C_i)}$$

$$\gamma'(K) = A \cup \bigcup_{1 \leq i \leq r} \mathrm{flow\_avoid(A_r, E)}$$

Figure 1: Constructs for Wong-Toi controller synthesis method

$\{x : \exists\sigma_c \in \Sigma_c, \forall r \in R_{\sigma_c}, \mathrm{Post(x, \sigma_c) \in \overline{W}}\}$. If R_{σ_c} is unknown and Pre($\sim W$,σ_c) is not empty and if all values of r are possible for every x, then $R_{\sigma_c} = \{r : \exists x, \mathrm{Post(x, \sigma_c, r) \in \overline{W}}\}$.

No constructs are currently available in LHA modeling tools for the $\forall$ quantifier. In order to implement this formula, we have to construct the complement of B. Let $W^* = W \cup \{\emptyset\}$. This implies

$$\begin{aligned}
\overline{B} &= \{x : \forall\sigma_c, \exists r \in R_{\sigma_c}, \mathrm{Post(x, \sigma_c) \in W^*}\} \\
&= \bigcap_{\sigma_c} \{x : \exists r \in R_{\sigma_c}, \mathrm{Post(x, \sigma_c) \in W^*}\} \\
&= \bigcap_{\sigma_c} \overline{B_{\sigma_c}}
\end{aligned}$$

where $\overline{B_{\sigma_c}} = \{x : \exists r, (\mathrm{Post(x, \sigma_c), r) \in W^* \times R_{\sigma_c}}\}$ and $x : \mathrm{Post(x, \sigma_c) \in W^*} = \overline{x : \mathrm{Post(x, \sigma_c) \in \overline{\overline{W}}}}$.

Since $\overline{B} = \bigcap_{\sigma_c} \overline{B_{\sigma_c}}$, we will have $B = \bigcup_{\sigma_c} B_{\sigma_c}$.

Currently no implementations of the Wong-Toi algorithm are available in any standard LHA modeling/analysis tool. The next section describes the modifications we made to the HyTech tool to be able to implement this algorithm and the parametrized controllable transition analysis.

4 HoneyTech

HyTech is a symbolic model checker for performing automated analysis of hybrid systems. It can verify the correctness of behavioral constraints such as temporal and reachability (safety) conditions. But with the currently available primitives, it cannot be used to automatically synthesize controllers. We have enhanced HyTech by adding a set of operators and control structures, to form HoneyTech and implement Wong-Toi's synthesis algorithm. We have further improved HoneyTech performance by modifying the implementation of the "complement" operation, to reduce the number of repetitive reductions that are done on regions during each and every set operation.

4.1 Existing HyTech Features

The input to HyTech [1] is a text file that consists of the "system description" and "analysis" sections. In the system description section, the user models the concurrent hybrid automata that make up the hybrid system.

In the analysis section, the user declares the region variables and writes a sequence of analysis commands for manipulating and outputting the regions. HyTech compiles and executes the system description and analysis program to derive results that constitute the analysis of the described system.

The analysis commands and control constructs used by HyTech include:

- Basic boolean operations.
- Region successor/predecessor commands (`pre(W)` and `post(W)`). These commands compute the reachable regions from a region W in one time step or state transition. Safety verification of a system is done with the help of the pre/post and reachability commands for an infinite number of time steps and state transitions.
- Reachability analysis commands for forward and backward reachability.
- Iteration constructs for generating the fixpoint value of an iteration variable
- Basic control constructs: if-then-else, while etc.

For a complete reference of all features and constructs available in HyTech, see the User Guide [1].

4.2 HoneyTech Features

To support the automatic synthesis of LHA controllers, we have added the following new region operators and control constructs to form HoneyTech:

- `pre_t(<reg_exp>)`[3]/ `post_t(<reg_exp>)`
  These commands compute the set of pre- and post-regions for a flow transition (i.e., the passage of time).
  $$\texttt{pre\_t(W)} = \bigcup_{\delta \in \mathcal{R}_{\geq 0}} Pre(W, \delta)$$
  $$\texttt{post\_t(W)} = \bigcup_{\delta \in \mathcal{R}_{\geq 0}} Post(W, \delta)$$

- `pre_single_event(<reg_exp>, <sync>)`/ `post_single_event(<reg_exp>, <sync>)`
  The `pre_single_event(`W`,sync`$_\sigma$`)` operator computes q, the predecessor region of W along a single event σ identified by the HyTech event synchronization label `sync`.
  $$q \mid \exists q' \in W.q \xrightarrow{\sigma} q'$$
  The `post_single_event` operator computes the corresponding successor region.

- `pre_uncontrollable_events(<reg_exp>)`/ `pre_controllable_events(<reg_exp>)`
  These operators compute the set of predecessor regions, Q, of region W for the set of uncontrollable/ controllable events in the system.
  $$Q_u = \bigcup_{\sigma_u \in \Sigma_u} q_{\sigma_u}$$
  $$q_{\sigma_u} \mid \exists q'_{\sigma_u} \in W.q_{\sigma_u} \xrightarrow{\Sigma_u} q'_{\sigma_u}$$

- `closure(<reg_exp>)`
  The `closure(W)` operator computes the closure for region W. This is used in computing the boundary points for witness flows, for each *flow_avoid* region in the Wong Toi algorithm.

- `escape(<reg_exp>)`
  The `escape(K)` operator computes the escape region for an unsafe region K, as defined by Wong-Toi's algorithm. A similar construct `escape_and_hide(`K`,`R_{σ_c}`,`r`)` is available to compute the escape regions for K given a parameter r and the set of values R_{σ_c} it can take.These primitives simplify the synthesis algorithm representation and allow the user to compute the escape region in a single line of HoneyTech code.

- `loop <var> over <reg_exp> doing <body>`
  The `loop K over L doing <body>` control construct iterates region K over a convex set of regions, L, executing body statements, where $L = \bigcup_i L_i$. For each iteration, K takes the value of L_i. The loop construct is required by the Wong-Toi procedure to compute the *flow_avoid* regions (as defined in the previous section).

[3]`<reg_exp>` denotes an arbitrary region expression.

All controllable events in HoneyTech are identified by the prefix "*c_*" in their synchronization labels. Events without this prefix are analyzed by HoneyTech as uncontrollable events. This naming convention helps in easy readability of the automata model and is useful in computing the *escape* and the *flow_avoid* regions.

Using these new functions, the HoneyTech implementation of WongToi's algorithm is shown in Figure 2.

4.3 Enhancements to Baseline HoneyTech

As implemented, the baseline HoneyTech version of the "complement" function has intrinsic inefficiencies which result directly from the implementations of the region "and" (intersection) and "or" (union) functions. The latter functions are designed to provide "simplified" "minimal" representations of the regions as a union of convex polyhedra which are represented as a list of systems of equations. The union is "simplified" in the sense that none of the polyhedra is wholly contained in any of the others in the union. Each of the polyhedra is "minimal" in the sense that none of the equations can be removed without changing the set. The simplify operation is computationally expensive, and is performed each time a new polyhedron is added to the region in an "or" operation. The minimize operation is even more expensive, and is performed each time a new constraint (equation) is added to a polyhedron in an "intersection" operation.

Since each intermediate result is simplified and minimal, the HoneyTech user need not be concerned with explicit function calls to achieve this representation. Unfortunately, it also means that quite a bit of needless computation might be done in operations which require a large number of union and intersection operations to produce intermediate results. Such is the case with the set complement.

The complement function begins by taking the canonical "union of intersections" form and produces, as an intermediate form, an "intersection of unions". This intermediate form is then reduced by binary intersect/union operations to produce the canonical

```
--  REGION VARIABLE DECLARATION
var
 K_init,     -- initial unsafe region
 K,          -- iterated unsafe region
 E, C,       -- escape region and complement
 A, A_r, flow_avoid_r, C_i,  -- temp vars
 flow_avoid, Gamma_prime_K :  region;
-- WONG-TOI SYNTHESIS ALGORITHM

-- K_init := problem specific K_init

K :=  iterate K from K_init using {
 E := escape(K);
 C := ~E;
 K := K|pre_uncontrollable_events(K);
 A := K;
 flow_avoid:= A;
 loop A_r over A doing {
  flow_avoid_r := iterate flow_avoid_r
                          from A_r using {
    loop C_i over C doing {
      flow_avoid_r := flow_avoid_r |
             pre_t(pre_t(flow_avoid_r) &
                    closure(flow_avoid_r) &
                    closure(C_i) &
                    C) &
             C_i; };};
  flow_avoid := flow_avoid | flow_avoid_r;};
 K:= K | flow_avoid; };
Gamma_K := K;  -- unsafe region
```

Figure 2: The Wong-Toi algorithm implemented in HoneyTech.

form. If the original set had n polyhedra each with m equations, the set complement operation will require on the order of m^n each of the binary intersect and union operations. In the baseline implementation of complement, each pair of operations would require the expensive set minimize and simplify.

HoneyTech has been modified to reduce the computations done during the complement operations, by deferring the reduction operations till the ver end. This has resulted in an increased performance of the tool (as seen in the battle resource allocation example). However, this deferment is not guaranteed to produce the minimal region computation for all problems.The optimal choice of when and how often to minimize and simplify representations probably depends on the structure of the original region.

5 Examples of Controller Synthesis

5.1 Truck Navigation Control

We desire to design a navigation control unit for an autonomous toy truck (initially moving in the SE direction), which is responsible for avoiding a 2 by 1 pit aligned in the E-W and N-S directions. It allows the truck to take 90 degrees left or right turns, which must alternate. The truck has a time lag for response and a finite turn radius. The truck should maneuver early enough to avoid the pit in all instances.

The iterations of the synthesis algorithm, are shown graphically in Figure 4. The letters inside the diagram denote the discrete state of the machine; E indicates vehicles moving South-East and W indicates vehicle moving South-West. The unsafe set is the region inside the innermost line and the escape set is the region outside the outmost line.

In the first iteration, the unsafe set is the rectangular hole in the pavement, in both directions. The escape set is enabled only when traveling South-East. Since we consider the delay an integral part of the command, the escape set includes all those areas that will not fall on the unsafe set after the delay and turn. The second iteration extends the unsafe set to those points that will flow into the unsafe set without touching the previous value of the escape set. This includes an infinite band NE of the pit with the vehicle traveling SW, and a region NW of the pit, inside the previous escape set. On the third iteration, the escape set converges.

The HoneyTech analysis code for this problem is shown in Figure 5.1.

```
automaton truck
 synclabs: c_turnenable;
 initially direction1;
 loc direction1:
     while True wait{dx=xdot1,dy=ydot1}
 when True sync c_turnenable do
     {x'=x+r,y'=y-r} goto direction2;
 loc direction2:
     while True wait {dx=xdot2,dy=ydot2}
end
automaton controller
 synclabs: c_turnenable;
 initially turning & r>=0 & r<=1;
 loc turning: while r>=0 & r <=1 wait {}
 when r>=0 & r<=1 sync c_turnenable
      goto turning;
 end
```

Figure 3: HoneyTech Model of One Pit, One Turn Example.

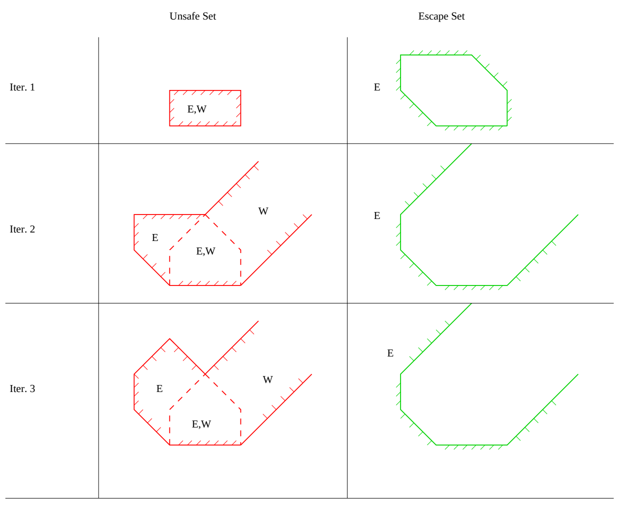

Figure 4: Escape Set and Unsafe Set Fixed Point Computation

```
viol:= x>holexmin & x<holexmax &
       y>holeymin & y<holeymax;
W := iterate W from viol using{
 E:=~(hide r in pre_single_event(W,c_turnenable)
   endhide) & ~W  &
   (hide r in pre_single_event(~W,c_turnenable)
   endhide);
 C:=~E;
 W:=W|(hide r in pre_uncontrolled_events(W)
       endhide);
 flav:=W;
 loop Ar over W doing {
   flavr:=iterate flavr from Ar using{
     loop Ei over C doing {
       flavr:=flavr |
         pre_t(pre_t(flavr) & closure(flavr) &
               closure(Ei) & C)&
         Ei;
       };
    };
    flav:=flav|flavr;
  };
 W:=hide r in W|flav endhide;
};
```

Figure 5: Analysis Code for One-Pit-One-Turn Problem.

5.1.1 Performance Evaluation

To evaluate the scalability of the Wong-Toi algorithm as implemented in HoneyTech, we tested it on increasingly large versions of the truck domain. The size of the problem was determined by the number of distinct obstacles. Figure 6 shows the run time for the computation of both unsafe and escape sets on a SparcStation. A least-squares linear fit to the logarithmic data is $t = 87.4 \times 2.8^n$. As expected, the controller synthesis process is exponential and even small problems can overwhelm available computing resources. As discussed below, we are currently exploring alternative techniques that may prove to have more tractable nominal performance.

5.2 Steam Boiler Plant Control

HoneyTech was used to replicate the controller synthesis experiment for water level control in a steam boiler plant by Wong-Toi [7]. Results for the synthesis experiments with only pump control modes are discussed.

The algorithm converges in 2 iterations. The plant

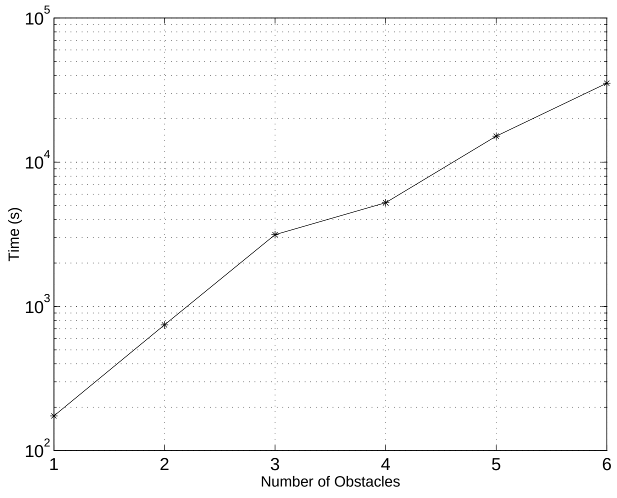

Figure 6: Computation time as a function of number of obstacles

is shown to be unsafe at all times for water levels $w \leq 5$ and $w \geq 220$. Control action must be enabled at the intersection of the escape set and the unsafe configuration set at pump control modes [going_on, on], [on, off] and [going_off, off]. An increase in the number of control modes for the problem- by introducing water drains or steam vents, increases the complexity of the problem , leading to non convergence of the algorithm.

5.3 Resource Allocation in Combat Operations

A control system was designed for resource allocation in a combat scenario. In this scenario, three battles are occurring simultaneoulsy on three fronts- Red, South and North. Progress along each line is sequential, but some tasks may need to be synchronized across more than one front.

It is desired to design a campaign controller that will assign resources to the battles at the end of each round to ensure maximum probability of success and safety. At the end of each round, each battle front suffers a variable number of casualties and in order to maintain a minimum effective force, it sends a request for resources. The campaign controller will either assign resources from one front to another or will allocate resources from the available reserves. Since allocation of resources is done at the end of a round and each round proceeds at varying rates, the events need to be synchronized in order to allocate resources from one battle front to the other. For a more complete description of the problem, see [6].

This problem was initially modeled in the base version of HoneyTech. The synthesis algorithm however could not run to completion. With the improved version of HoneyTech, the algorithm was able to synthesize a controller in 120 seconds on a SunSparc, 296 MHz station.

Two variants of the problem were solved using this algorithm. In the first scenario, the battle was modeled for only two controllable actions, allocation of resources to the north front and to the south front. In the second example, a third controllable action is added, where resources are allocated to the south front as well. Allocation of resources to a front can be done either from one of the other two fronts or from the available reserves. The algorithm in both cases, converges in two iterations. The results show that the battle is unsafe (from a resource allocation point of view) for red resources $red\_res < 4$ and $red\_res > 4$. The HoneyTech model for the red front of the problem is shown in Figures 5.3 and 5.3. The automata for the other fronts are modeled similarly.

6 Future Directions

Wong-Toi's algorithm explores the entire space of virtual failures to derive safety-preserving controllers. Any controller that avoids that virtual failure region is considered acceptable. However, as seen by our preliminary performance evaluation, this algorithm is highly exponential.

To improve performance, we are designing a modified controller synthesis algorithm that focuses only on finding a single acceptable (safe) controller through search. We plan to combine the forward-projection and search-based controller synthesis techniques from CIRCA [3, 4] with the HoneyTech representational power, to form an automatic hybrid controller synthesis system that explores a smaller state space. By exploiting abstraction and the state space structure, we hope to show that automatic controller synthesis can be applied to large-scale, real world domains.

References

[1] T. Henzinger, P.-H. Ho, and H. Wong-Toi, "A User Guide to HyTech," in *Proceedings of the 1st International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95).*, Lecture Notes in Computer Science 1019, pp. 41–71. Springer-Verlag, 1995.

[2] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "HyTech: A Model Checker for Hybrid Systems,"

```
automaton Northfront
synclabs:end1232,c_allocnorth;
initially NorthFirst & t1=0;

loc NorthFirst:
  while t1 <= FirstMax  wait {dt1=1}
    when t1>=FirstMin & t1<=FirstMax
 do {northres'<=northres-minLoss,
     northres'>=northres-maxLoss,t1'=0}
         goto NorthSecond;
    when reserves>1 sync c_allocnorth
 do {northres'=northres+1,
     reserves'=reserves-1}
 goto NorthFirst;
    when redres>1 sync c_allocnorth
 do {redres'=redres-1,
     northres'=northres+1}
 goto NorthFirst;

loc NorthSecond:
  while t1 <=SecondMax  wait {dt1=1}
    when t1>=SecondMin & t1<=SecondMax
                      &  northres>= 6
 do {northres'<=northres-minLoss,
             northres'>=northres-maxLoss}
         goto NorthLasta;

    when t1>=SecondMin & t1<=SecondMax
              &  northres <= 5
 do {northres'<=northres-minLoss,
     northres'>=northres-maxLoss}
 goto NorthLastb;
    when reserves>1 sync c_allocnorth
 do {northres'=northres+1,
     reserves'=reserves-1}
         goto NorthSecond;
    when redres>1 sync c_allocnorth
 do {redres'=redres-1,
     northres'=northres+1}
 goto NorthSecond;
```

Figure 7: HONEYTECH Model of Red Front.

```
loc NorthLasta:
  while t1<=1  wait {dt1=9/10}
    when True
      do {northres'<=northres-minLoss,
  northres'>=northres-maxLoss}
      goto done1;
    when True sync end1232
      goto NorthLasta;
    when reserves>1 sync c_allocnorth
      do {northres'=northres+1,
  reserves'=reserves-1}
      goto NorthLasta;
    when redres>1 sync c_allocnorth
      do {redres'= redres-1,
  northres'=northres+1}
      goto NorthLasta;

loc NorthLastb:
  while t1<=1  wait {dt1=1/10}
    when True
      do {northres'<=northres-minLoss,
  northres'>=northres-maxLoss}
      goto done1;
    when True sync end1232
      goto NorthLastb;
    when reserves>1 sync c_allocnorth
      do {northres'=northres+1,
  reserves'=reserves-1}
      goto NorthLastb;
    when redres>1 sync c_allocnorth
      do {redres'=redres-1,
  northres'=northres+1}
      goto NorthLastb;

loc done1:
  while True wait {dt1=0}
    when True sync end1232
      goto done1;

end -- NorthFront
```

Figure 8: HONEYTECH Model of Red Front (contd).

Software Tools for Technology Transfer, vol. 1, pp. 110–122, 1997.

[3] D. J. Musliner, E. H. Durfee, and K. G. Shin, "World Modeling for the Dynamic Construction of Real-Time Control Plans," *Artificial Intelligence*, vol. 74, no. 1, pp. 83–127, March 1995.

[4] D. J. Musliner, R. P. Goldman, and M. J. Pelican, "Using Model Checking to Guarantee Safety in Automatically-Synthesized Real-Time Controllers," in *Proc. IEEE Int'l Conf. on Robotics and Automation*, 2000.

[5] P. Ramadge and W.M.Wonham, "Supervisory Control of a Class of Discrete Event processes," *SIAM J. on Control and Optimization*, vol. 25, no. 1, pp. 206–230, January 1987.

[6] J. Tierno and R.P.Goldman, "Systematic Verification of Military Operations," Technical report, Honeywell Laboratories, October 2000.

[7] H. Wong-Toi, "The Synthesis of Controllers for Linear Hybrid Automata," in *Proceedings of IEEE Conference on Decision and Control*, December 1997.