Modeling and Verifying Hybrid Dynamic Systems Using *CheckMate*

B. Izaias Silva,[*1], Keith Richeson[1], Bruce Krogh[1],Alongkrit Chutinan[2]
[1]*Dept. of Electrical and Computer Engineering*
Carnegie Mellon University
Pittsburgh, PA 15213 USA
(krogh|keithr|bsilva)@andrew.cmu.edu

[2]*Powertrain Control Systems Department*
Ford Research Labs
Dearborn, Michigan
achutina@ford.com

Abstract

This article introduces a computational tool called *CheckMate*. *CheckMate* is a tool for modelling, prototyping, simulating specific situation and formally verifying hybrid dynamic systems based on the MATLAB/Simulink and Stateflow Toolbox from The MathWorks, Inc. This paper presents the elements of CheckMate from the user's perspective and illustrates its features using a three dimensional linear system example. The theory underlying CheckMate is also reviewed briefly. More complete presentations of the theory can be found in [1-7].

Keywords: Formal Verification, hybrid systems, simulation of dynamic systems

1 INTRODUCTION

Recently, a tool called *CheckMate* was developed at Carnegie Mellon University to perform simulation and verification of a class of hybrid dynamic systems. This paper introduces this tool and how it works in the modeling and verification of hybrid systems. *CheckMate* deals with a class of hybrid systems called *threshold-event-driven hybrid systems* (TEDHS) for which a verification procedure was proposed in [03] . In a TEDHS the changes in the discrete state can occur only when continuous state variables encounter specified thresholds.

CheckMate models are constructed using a custom graphical user interface (GUI) in the MATLAB Simulink environment. Thresholds in the TEDHS model are hyperplanes. Models are built using the MATLAB Simulink/Stateflow graphical user interface (GUI). Parameters and specifications to be verified are entered using both the Simulink/Stateflow GUI, and the MATLAB command window. The key theoretical concepts used in *CheckMate* are described in [01] .

Hybrid system models in CheckMate have continuous dynamics described by standard differential state equations (possibly nonlinear), planar switching surfaces, and discrete dynamics modeled by finite state machines. Verification is performed using finite-state approximations known in the literature as *quotient transition systems*. The approximations are conservative in the sense that they capture all possible behaviors of the hybrid system. If the verification of an approximating automaton returns a positive result, the user is informed and the program terminates. If a negative result is found, the verification of the original hybrid system is inconclusive and the user is given the option to refine the current approximation and attempt the verification again. Properties of individual trajectories of the system can be verified using the MATLAB simulation engine and the CheckMate validation tool. The ideas presented throughout this paper will be demonstrated using one of the demonstration examples included in the CheckMate package. The example is a three dimensional linear system that can be built from scratch throughout the paper, or found in the `CM~/demo/rgsw` directory (where `CM~` denotes the CheckMate root directory.

The paper is organized as follows. Section 2 provides a brief inroduction to CheckMate and an overview of the demonstration example. Section 3 describes the CheckMate GUI and how it is used to build and simulate models of hybrid dynamic systems. Section 4 describes the basic representation of hybrid dynamic systems, the polyhedral invariant hybrid automaton, used in CheckMate as the formal model for verification. The theory and computations used in the CheckMate verification procedure are reviewed briefly in Section 5. In Section 6, verification results of the example system are presented. Recent improvements to the CheckMate verification procedure are discussed in Section 7 along with current research and development plans.

* Supported by grant from CNPq – proc N^0 20.0079/92

2 INTRODUCTION TO CHECKMATE AND EXAMPLE OVERVIEW

The CheckMate toolbox can be downloaded at http://www.ece.cmu.edu/~krogh/CheckMate/main.htm. This site also provides reference and background information on CheckMate. Once the file has been downloaded, it should be unzipped into a directory which becomes the CheckMate root directory. We will use `CM~` to denote this directory throughout the remainder of this paper. For proper operation, it is necessary to add the CheckMate directories to the MATLAB path. This can be accomplished using the installation routine provided with CheckMate. The m-file `install.m` found in the `CM~` directory provides this functionality. Sjmply type `install('<CM~>')` at the MATLAB command prompt and the appropriate changes will be made. CheckMate is now ready for use.

The example used throughout this paper is a three dimensional linear system, RGSW (***ReGion SWitch***). All necessary files can be found in the CM~/demo/rgsw directory, or the reader can choose to build the model and files as they are introduced throughout the paper. In this example, the state space is restricted to the cube ranging from –20 to 20 in each of the three x-yz directions. This cube is the analysis region. It means that the verification will consider this region to answer questions if some specification is met or not. The system is a simple linear switched dynamic system with two different dynamics. One behavior (IN) is enabled when the system trajectory is inside the internal region. In a real application it could represent an error offset that a system has to obey in a steady operation, before leaving the analysis region. When the system goes out of this internal region box, another behavior (OUT) is enforced. The analysis region and the internal region are represented in figure 1.

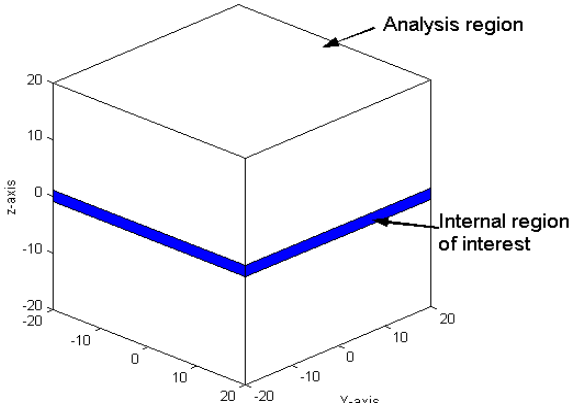


figure 1- Analysis Region and internal region for verification

The differential equations in each case are the following (in both cases vector b is $[0\ 0\ 0\ ]^T$:

$$IN: A_1 = \begin{bmatrix} -1 & 1 & 1 \\ -1 & 0 & 0 \\ -1 & 1 & 0 \end{bmatrix}, OUT: A_2 = \begin{bmatrix} -1 & 1 & 1 \\ 0.5 & -1 & 0 \\ -1 & 1 & 0 \end{bmatrix}$$

If, for some initial condition, we would like to know if the trajectory of the system reaches the internal region before leaving the AR, we could just simulate the system for this initial value. Depending on the time the system trajectory takes to accommodates inside the internal region, the answer could be TRUE or FALSE. In this sense, it would be interesting to model and simulate this system, in a user friendly environment. That is the subject of the next section.

3 MODELING AND SIMULATING HYBRID SYSTEMS IN CHECKMATE

CheckMate models are built with the MATLAB Simulink GUI using two customized (masked) Simulink blocks along with several of Simulink's standard blocks. Figure 1 shows the CheckMate block diagram and its components for our example. To build the model from scratch, the user must enter the command `cmnew` at the MATLAB command prompt. This will open the CheckMate library from which the user can construct the system model. In figure 2 we see the currently set of blocks allowed to be used in *Checkmate*:

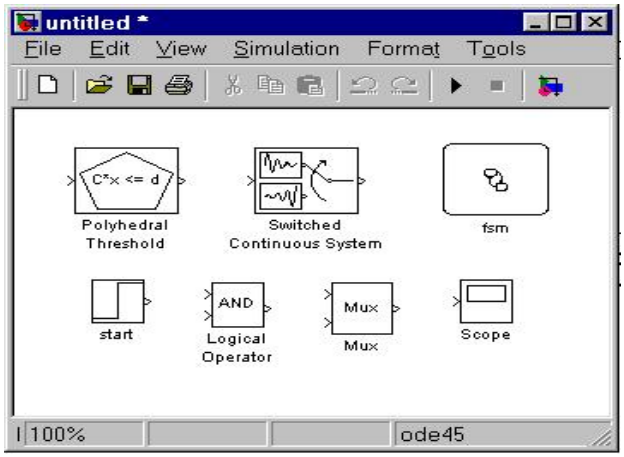


figure 2 – *CheckMate* customized blocks

3.1 Switched Continuous System Block (SCSB)

The custom SCSB represents a continuous dynamic system with state equation $\dot{x} = f_u(x)$, where u is a discrete-valued input vector to the SCSB and the continuous state vector x is the block's output. Currently, three types of dynamics can be specified in an SCSB for each value of the input vector u: clock dynamics ($\dot{x} = c$, where c is a constant vector), linear dynamics ($\dot{x} = Ax + b$, where A is a constant matrix and b is a constant vector), and nonlinear dynamics ($\dot{x} = f(x)$). In the example system one of these blocks in necessary, and should be named `scs`. There are several parameters associated with the SCSB. The figure 3 gives us and idea about them:

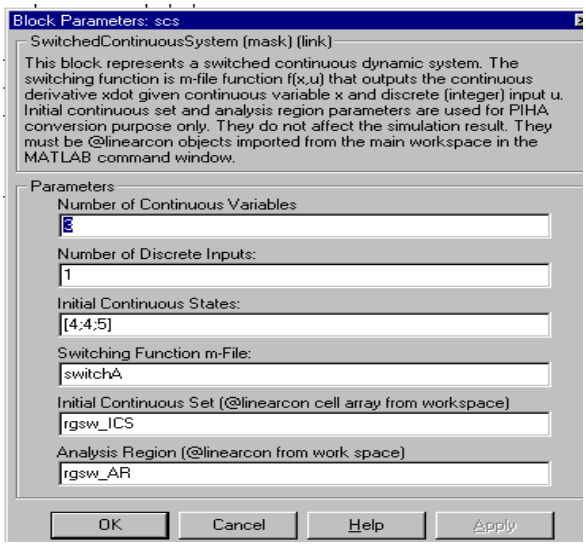


figure 3 – Configuration window for the SCSB block

The switching function is a .m file that gives us the information about the dynamics of the system. For RGSW example we have the file switchA.m below. The variable u selects which dynamics should be used.

```
function [sys,type] = switchA(x,u)
type = 'linear';
switch u
case 1,
   A = [-1  1  1
         -1  0  0
         -1  1  0];
case 2,
   A = [ -1  1  1
        0.5 -1  0
         -1  1  0];
otherwise,
   A = zeros(3,3);
end
sys.A = A; sys.b = [0 0 0]';
return
```

3.2 Polyhedral Threshold Block (PTHB)

The other custom block in CheckMate is the (PTHB), which represents a polyhedral region $Cx \le d$ in the continuous space of the continuous-valued input vector *x*. The PTHB output is a binary signal indicating whether *x* is inside the region or not, i.e. whether or not the condition $Cx \le d$ is true. Only one PTHB is necessary in our example to describe the region between the two limits on the z-axis, as told before.

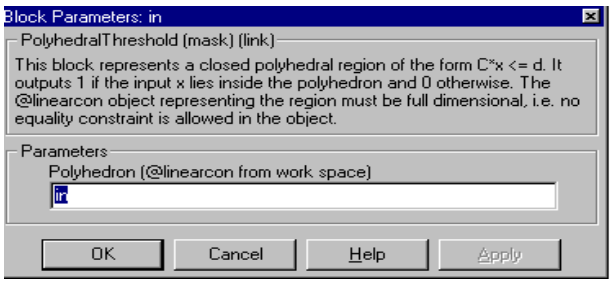


figure 4 - Configuration window for the PTHB block

Once using the simulink environment and selecting a PTHB block, an auxiliary window will pop up as presented in figure . The specification for the internal region for the RGSW example is a linearcon object created from a setup file called setup.m, in figure 4.

```
function setup()

% declare global variables in base work space
evalin('base','global_var');

% declare global variables locally
global_var

% setup parameter for the PTHB block diagram
assignin('base','in',…
    linearcon([],[],[0 0 1; 0 0 -1],[1; 1]));

% Initial condition setup
ICS_CE = [0 0 1];
ICS_dE = 5;
ICS_CI = [1 0 0; -1 0 0; 0 1 0; 0 -1 0];
ICS_dI = [6;-4;6;-4];
assignin('base','rgsw_ICS',. . .
    {linearcon(ICS_CE,ICS_dE,ICS_CI,ICS_dI)});

% Analysis region setup
AR_C = [-1 0 0;0 -1 0;0 0 -1;0 0 1;0 1 0;1 0 0];
AR_d = [20;20;20;20;20;20];
assignin('base','rgsw_AR',. . .
    linearcon([],[],AR_C,AR_d));

% setup verification parameters
GLOBAL_SYSTEM = 'rgsw';
GLOBAL_APARAM = 'rgsw_param';
GLOBAL_SPEC = '(AF AG (fsm == s1)) & (AG
~out_of_bound)';

return
```

The initial condition, the analysis region and the internal region hyperplane are defined as linearcon object. The variable GLOBAL_SPEC shows the specification that will be tested. We will detail this in the following sections.

3.3 Finite State Machine Block (FSMB)

Discrete dynamics are modeled using a (FSMB). FSMBs are regular MATLAB Stateflow blocks that conform to the following restrictions.

- No hierarchy is allowed in the Stateflow diagram.
- Data inputs must be Boolean functions of PTHB and FSMB outputs only.
- Event inputs must be Boolean functions of PTHB outputs only, i.e. events can only be generated by the continuous trajectory leaving or entering the polyhedral regions.
- Only one data output is allowed.
- Every state in the Stateflow diagram is required to have an entry action that sets the data output to a unique value for that state.
- No action other than the entry action discussed above is allowed in the Stateflow diagram.

The discrete dynamics of RGSW can be described with one FSMB having two states. Three events are assigned:

event *coming_in*, *going_out* and *start*. The start event does nothing but starting the execution pointing TO the initial condition in the FSM diagram. The variable *q*, showed inside of the states sets up the output for the FSMB, and corresponds to the value of variable *u* in the setup m-file. One also can specify data input for the FSMB. This could be used to establish guards for some transitions. In figure 5 we have the FSMB for the RGSW example:

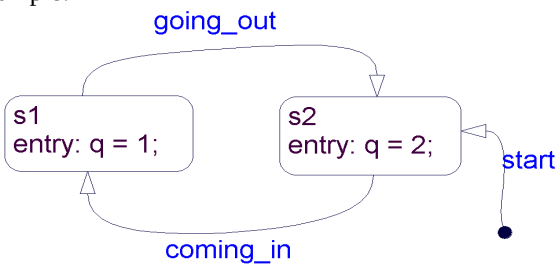


figure 5 – State flow diagram for RGSW example

Other Simulink blocks supported by CheckMate include multiplexers for vectorizing signals and logic blocks (AND, OR, NOT, etc.) for creating logical combinations of PTHB and FSMB outputs. Additionally, scopes, x-y plots, and other sink blocks can be used when taking advantage of the simulation capabilities inherently available through Simulink and CheckMate. A scope has been added to the example system to look at trajectories of the three state variables. In figure we have the complete *CheckMate* model for the RGSW example.
There are some parameters the user must enter, in order to give CheckMate all the details about the verification process. For the RGSW example, these data are stored in the file rgsw_param.m .

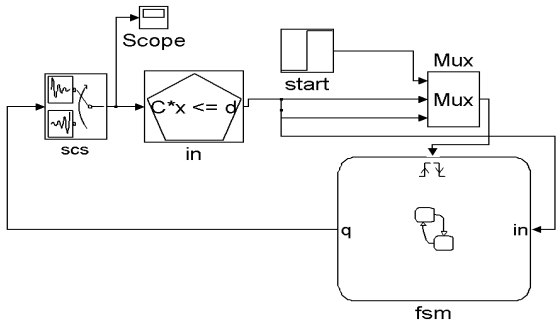


figure 6 - RGSW modeled by *CheckMate*

```
function approx_param = rgsw_param(q)

approx_param = [];
approx_param.dir_tol = [];
approx_param.var_tol = 1;
approx_param.size_tol = 40;
approx_param.W = eye(3);
approx_param.T = 0.1;
approx_param.max_time = Inf;
approx_param.quantization_resolution = 0.1;
approx_param.reachability_depth = Inf;
return
```

The table 1 has the description of each parameter. Several parameters are used throughout the verification process. These parameters as well as any variables used in the Simulink/Stateflow front-end model are defined and stored in the MATLAB workspace. Parameters and variables can be defined manually or through the use of MATLAB m-files.

.dir_tol	tolerance for patch "single-sided-ness". Decides when to split the region to maintain the range of direction variance.
.var_tol	tolerance for patch vector field variation relative to the vector field variation on the parent invariant face
.size_tol	tolerance for patch size. It gives us the maximum slice to split the analysis region.
.W	(diagonal) weighting matrix
.T	time step for flow pipe computation
.max_time	time limit (sec) for mapping computation
.eq_tol	equilibrium termination tolerance for mapping computation
.quantization_resolution	resolution for partition refinement
.reachability_depth"	maximum depth of initial partition reachability analysis

table 1 – internal parameters of CheckMate

In figure 7we can see the result of the simulation of RGSW example for the initial condition X = [4,4,5] and fsm = s2.

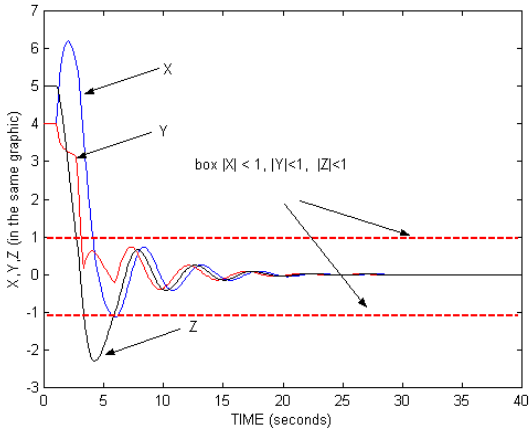


figure 7 – Simulation of the RGSW model

4 REPRESENTATIONS OF HYBRID SYSTEMS IN CHECKMATE

In order to understand how CheckMate works, it is useful to understand the two models of hybrid systems used in CheckMate. Models built with the CheckMate GUI are referred to as the threshold-event-driven hybrid systems (TEDHSs). The formal model used for verification in CheckMate is a class of hybrid automata [6]. A threshold-event-driven hybrid system consists of three types of subsystems, the switched continuous system (SCS), the threshold event generator (TEG), and the finite state machine (FSM). The SCS is the

continuous dynamics system that takes in the discrete-valued input u and produces its continuous state vector x as the output. The continuous dynamics for x evolve according to the differential equation or differential inclusions selected by the discrete input u. The output of the SCS is fed into the TEG, which produces an event when a component of the vector x crosses a corresponding threshold from the specified direction (rising, falling, or both). The event signals from the TEG drive the discrete transitions in the FSM whose output, in turn, drives the continuous dynamics of the SCS.

As illustrated in Figure 7 CheckMate converts the TEDHS into a polyhedral invariant hybrid automaton (PIHA). The PIHA are a subclass of more general hybrid automata. A hybrid automaton is a generalization of the finite automaton that includes continuous dynamics within each discrete state. Each discrete state in the hybrid automaton is called a location. Associated with each location is an invariant, the condition which the continuous state must satisfy while the hybrid automaton resides in that location, and the flow equation representing the continuous dynamics in that location. Transitions between locations are called edges. Each edge is labeled with guard and reset conditions on the continuous state. The edge is enabled when the guard condition is satisfied. Upon the location transition, the values of the continuous state before and after the transition must satisfy the reset condition. In general, the analysis of hybrid automata can be very difficult. In CheckMate, we restrict our attention to the aforementioned PIHA. A PIHA is a hybrid automaton with the following restrictions.

- The continuous dynamics for each location is governed by an ordinary differential equation (ODE).
- Each guard condition is a linear inequality (a hyperplane guard).
- Each reset condition is an identity.
- For the hybrid automaton to remain in any location, all guard conditions must be false. This restriction implies that the invariant condition for any location is the convex
- polyhedron defined by conjunction of the complements of the guards. This gives rise to the name polyhedral-invariant hybrid automaton.

5 Quotient Transition Systems and Flowpipe Approximations

The verification portion of CheckMate is based on the theory of quotient transition systems [14]. A quotient transition system (QTS) is a finite state transition system that is a conservative approximation of the hybrid system. A QTS is defined from a partition of the state space of the hybrid system with each state in the QTS corresponding to a member of the partition. In the QTS, a transition is defined from a state (a set) *P1* to another state *P2* if and only if there is a state *p2* in *P2* that is reachable from a state *p1* in *P1* in the original hybrid system. A QTS is a conservative approximation in the sense that for every trajectory in the original hybrid system, there is a trajectory in the QTS corresponding to the set of states that the trajectory in the hybrid system goes through. Thus, if we can verify that all trajectories in the QTS satisfy some property, we can conclude that all trajectories in the hybrid system also satisfy the same property. CheckMate only pays attention to the behavior of the hybrid system at the switching instants. Thus, CheckMate approximates the QTS for the hybrid system from the partition of the switching surfaces, which are the boundaries of the location invariants in the PIHA, and the set of initial continuous states. The reachability analysis used to define the transitions in the QTS is performed using an approximation method called flow pipe approximations [3].

The flow pipe approximation is used to define

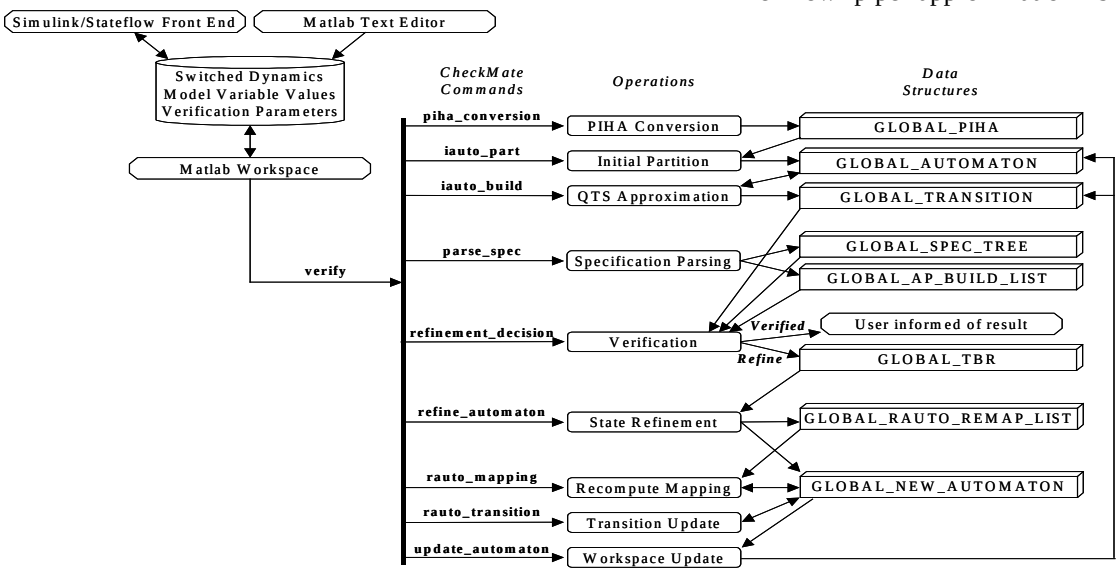


Figure 7– CheckMate verification procedure

transitions in the quotient transition system for the PIHA as follows. A state in the quotient transition system is a triple (π,p,q) where π is a polytope in the location (p,q) of the PIHA. For each state in the quotient transition system, the flow pipe is computed for the associated polytope under the associated continuous dynamics. The mapping set, the set of states on the invariant boundary that can be reached from π is computed. A transition is then defined from (π,p,q) to any other state whose polytope overlaps with the mapping from π.

As shown in Figure 7, CheckMate then performs model checking [7] on this transition system to obtain a verification result for the desired specification. If the verification returns a positive result, then the program informs the user and terminates. If a negative result is returned, the user is informed and given the option to quit or allow CheckMate to refine the approximation and repeat the verification. This process continues until a positive verification result is obtained, or the user decides to quit.

6 VERIFYING SYSTEMS

Checkmate can perform two kind of verification: a quick and a complete verification. By *quick verification* (command *validate)* we mean just testing the vertices of the polyhedral defined by the initial conditions. Checkmate calls the simulation environment from matlab, and build the transition system following the simulation trajectory. After the simulation entered in a null state, or after a time out situation, CheckMate check if the transition system built satisfies the specification, written in ACTL. For the RGSW system, we have a square the initial condition 4≤x≤6, 4≤y≤6, z=5. In figure 8we can see the result of the validate command for the vertices and in figure 9 the trajectories are shown, as well as a partial view of the inside box |x|≤1, |y|≤1, |z|≤1. Note that after some time, the trajectories concentrates inside of the box. The advantage of the quick verification is that the user can have a gross idea about the result of the complete verification. If some vertices already violates some constraints, probably the verification wouldn't give any better result. In some cases, for linear time invariant system, we can even rely on the results of the quick verification, because if the initial set is convex, all the reachable set from there will be also convex. If some property is satisfied for the vertices of the initial condition set, it will be by any inner point.

The complete verification, on the other hand, performs the operations already mentioned in section 5. in we see the states of the approximate quotient transition system that lead to the null event (represented by the spiral with no further state transition). In we have two different 3D partial views of the flowpipe, used to generate the approximate quotient transition system.

```
Validate –vertices
block orders: x = [scs], q = [fsm], pth = [in]
```

```
(1/4) init: t = 1, x = [6 6 5], q = 2, pth = 0
--->...-->null_event / specification satisfied

(2/4) init: t = 1, x = [6 4 5], q = 2, pth = 0
--->...--> null_event / specification satisfied

(3/4) init: t = 1, x = [4 6 5], q = 2, pth = 0
--->...--> null_event / specification satisfied

(4/4) init: t = 1, x = [4 4 5], q = 2, pth = 0
--->...--> null_event / specification satisfied
```

figure 8 – Command *validate* for quick verification

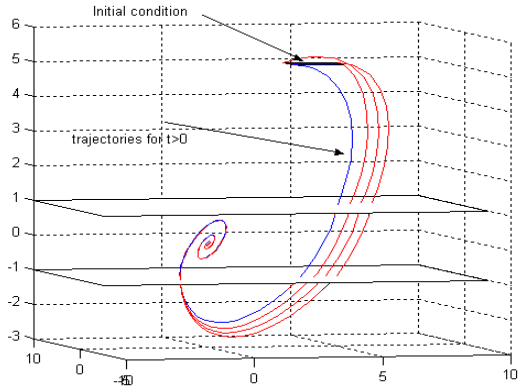


figure 9 – Trajectories for the initial condition vertices

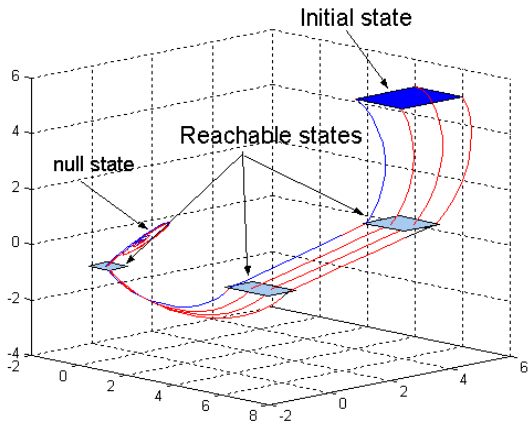
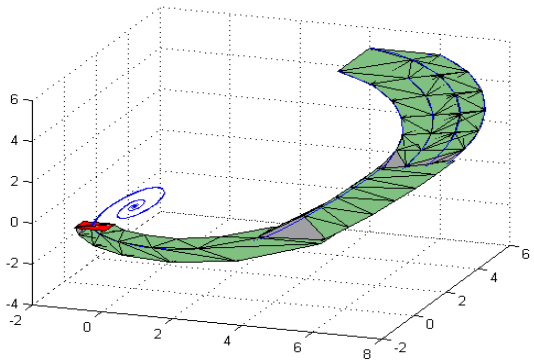


figure 10 – Reachable states for the RGSW system



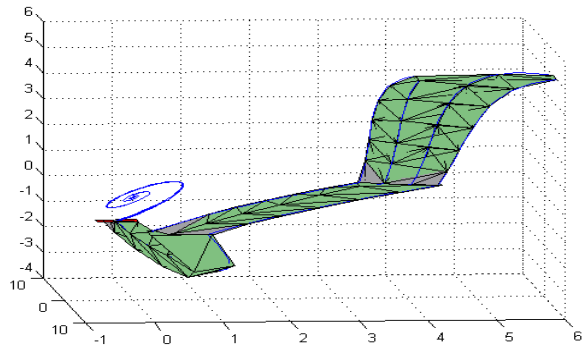


figure 11 – Flowpipe computation - 3D view

7 DISCUSSION AND RESEARCH DIRECTIONS

CheckMate has been developed to take advantage of the characteristics of clock and linear dynamics [3]. Recently, revisions have been made to further reduce the computations involved in the approximation of the QTS. In particular, the flowpipe approximation procedure has been amended for the linear (affine) case. The new procedure considerably reduces the conservativeness of the QTS approximations and reduces total computation time. Future improvements are expected to deal with sampled data systems and cases where the flow reduces to an extremely small region (thus causing numerical problems in the calculation of the flowpipes).

8 REFERENCES

[01] Alongkrit Chutinan. Hybrid System Verification Using Discrete Model Approximations. PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, May 1999.

[02] Chutinan and B.H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In Hybrid systems: Computation and Control, Second International Workshop, Lecture Notes in Computer Science. Springer-Verlag, 1999. Copyright Springer-Verlag.

[03] Chutinan and B.H. Krogh. Computing polyhedral approximations to flow pipes for dynamic systems. In The 37th IEEE Conference on Decision and Control: Session on Synthesis and Verification of Hybrid Control Laws (TM-01), 1998.

[04] Chutinan and B.H. Krogh. Computing approximating automata for a class of hybrid systems. Mathematical and Computer Modeling of Dynamical Systems: Special Issue on Discrete Event Models of Continuous Systems, 1998. To appear.

[05] Marian Grobelny. Study of Flourosilicic Acid Reactions With Variety of Aluminum Compounds. Translated by Oktawian Sobieraj from Badania reakcji dwasu fluorokrzemowego z roznymi substratami glinowymi. Przemysl Chemiczny, 57/12 1978.

[06] T.A. Henzinger. The theory of hybrid automata. In Proceedings of the 11th Annual symposium on Logic in Computer Science, pages 278-292. IEEE Computer Society Press, 1996. Invited tutorial.

[07] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. Model Checking. The MIT Press, ISBN 0-262-03270-8, January 2000.

[08] S. Kowalewski, S. Engell, J. Preußig and O. Stursberg: Verification of logic controllers for continuous plants using timed condition/event-system models. Automatica 35 (1999), 505-518.

[09] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A Model Checker for Hybrid Systems. Software Tools for Technology Transfer 1:110-122, 1997.

[10] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool kronos. In R. Alur, T. A. Henzinger, and E.D. Sontag, editors, Hybrid Systems III: Verification and Control, pages 208-219. Springer-Verlag, 1996.

[11] UPPAAL a Tool for Automatic Verification of Real-Time Systems. Johan Bengtsson and Fredrik Larsson. DoCS Technical Report Nr 96/67, Uppsala University, ISSN 0283-0574, January 1996.

[12] The Shift Programming Language and Run-time System for Dynamic Networks of Hybrid Automata. Akash Deshpande, Aleks Göllü and Luigi Semenzato. California PATH Research Report UCB-ITS-PRR-97-7, January1997. 22 pages.

[13] Zohar Manna and the STeP group. STeP: The Stanford Temporal Prover. Technical report STAN-CS-TR-94-1518, Computer Science Department, Stanford University, July 1994. 44 pages.

[14] G. Lafferriere, G.J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In F.W. Vaandrager and J.H. Van Schuppen, editors, Hybrid systems: Computation and Control, 2nd International Workshop, HSCC'99, pages 137-151, Berg en Dal, The Netherlands, 1999. Springer-Verlag.