Symbolic Simulation of Hybrid Systems

Ralph-Johan Back, Cristina Cerschi Seceleanu, Jan Westerholm
Turku Centre for Computer Science (TUCS),
Lemminkäisenkatu 14, FIN-20520 Turku, Finland
{*backrj, ccerschi, jawester*}*@abo.fi*

Abstract

*Continuous action systems (***CAS***) is a formalism intended for modeling hybrid systems (systems that combine discrete control with continuous behavior), and proving properties about the model within refinement calculus. In this paper we use a symbolic manipulation program to build a tool for simulating CAS models by calculating symbolically the time evolution of the discrete and continuous CAS model functions, as explicit and exact expressions of a continuous time variable. We may then study the time behavior and general properties of the model by plotting these functions with respect to time. For certain models our tool eliminates the need for introducing tolerances into the model structure. The tool is useful for checking that the model behaves correctly, and we can sometimes study the behavior of CAS models with in principle infinite precision.*

1. Introduction

Hybrid systems fall on the borderline between Computer Science and Control Theory, requiring techniques from both areas. These systems can be quite hard to build, due to the many different ways in which the continuous system behavior needs to interact with the discrete controller. Modeling hybrid systems is therefore of great help, allowing one to analyze the properties of the system to be built beforehand, to find out potential trouble spots, and to convince oneself of the correctness of the controller.

A *formal* approach to modeling hybrid systems, especially safety-critical control systems has the advantage of providing a precise model of the system, on which well-established formal verification methods may be applied to prove that any behavior of the system satisfies the properties that are verified.

Discrete concurrent systems can be modeled as *action systems* [4], where a state (described by a collection of state variables) is manipulated by a collection of actions.

Continuous Action Systems (CAS) are an extension of action systems to hybrid systems, being based on a new approach to describe the state of a system. Essentially, the state variables will range over functions over time, rather than just over values. The CAS formalism has been recently introduced by Back, Petre and Porres [5]. This model allows us to describe both control actions and time advancing behavior actions with the same simple mechanism. Besides CAS, there are several other hybrid formalisms developed to support the description and analysis of hybrid systems. Among them, the most popular are *timed automata*, *hybrid automata* and the more general framework of *hybrid input/output automata* [2, 11, 13].

Proving properties about action systems is done within the higher-order logic framework of the refinement calculus developed by Back and von Wright [6]. Because continuous action systems are special cases of ordinary action systems, we can use the proof and refinement techniques developed for ordinary action systems for these as well. This allows us to verify safety and liveness properties of hybrid systems (modeled as continuous action systems), as well as to prove correctness of refinements of hybrid systems [5].

Prior to, or as an alternative to verification, the simulation of a hybrid model brings many benefits to the modeler, increasing the confidence in an error free abstraction. A lot of effort has been devoted to developing simulation tools for hybrid systems, targeting various modeling languages. Such tools include the Hybrid Chi simulator, Dymola, Shift, and Simulink [7, 9, 10, 14].

The contribution of this paper is to show how to *simulate* the behavior of hybrid systems that are modeled as continuous action systems. The simulation technique that we use is symbolic, i.e., given the simulation parameters, we construct the exact analytic functions that describe the behavior of the hybrid system over time (rather than just numeric approximations of the behavior). For this purpose, we use the computer algebra package, *Mathematica* [15]. Besides allowing us to get symbolic solutions to the time varying behavior of the hybrid system, Mathematica also provides good facilities for visualizing the system evolution as graphs. The simulation method is based on calculating

symbolically the next time point when at least one action is enabled, using the minimization capabilities of Mathematica. This means that our simulation method is not dependent on choosing a fixed sampling interval, but that the simulation rather proceeds from one interesting time point to the next. These interesting time points can be very dense in times when the behavior changes rapidly, and be sparse at other times.

We also allow the state variable functions to be described by differential equations. The differential equation solver of Mathematica is then very useful, in particular for the linear case where it is easy to find an exact solution. In cases where we do not get an analytic solution, we can still get a numeric approximation of the time functions, and use these approximations in our simulation. The approximation will introduce an uncertainty into the simulation, but still allows us to carry out the simulation independently of a fixed sampling interval.

We have built a tool that allows us to simulate automatically continuous action systems. This tool is essentially an interpreter with plotting facilities for continuous action systems, written in the Mathematica programming language. Our experiences with this tool have been very promising. It provides a good visualization of the behavior of hybrid systems, and has been also quite efficient in harnessing the power of Mathematica.

We have applied our simulation technique to a small collection of hybrid systems. In the paper, we will describe one application in more detail. It models a heat producing nuclear reactor with two cooling rods. The simulation tool has proved to be very useful in this and other cases that we have tried, sometimes revealing quite surprising behavior, and confirming the a priori intuition about the system behavior in other cases.

The rest of the paper is organized as follows. The continuous action system model is briefly described in section 2. Section 3 presents the action system model of a temperature control system (TCS) for a nuclear reactor tank. A particular application of our tool is to simulate the behavior of the TCS model, in Mathematica, which is shown in section 4. The simulation results can be found in section 5. Section 6 offers a general description of the simulator, emphasizing the advantages and disadvantages of our approach, and also the simulation algorithm behind it. Conclusions are presented in section 7.

2. Continuous action systems

A *continuous action system* [5] consists of a finite set of attributes that can range over discrete or continuous valued time functions, forming the state of the system, together with a finite set of actions that act upon the attributes. It is of the form

$$\begin{aligned} \mathcal{C} \triangleq\ & |(\ \mathsf{var}\ x : \mathsf{Real}_+ \to T \bullet S_0; \\ & \quad \mathsf{do}\ g_1 \to S_1 [\!] \ \ldots [\!] \ g_m \to S_m\ \mathsf{od}\ )| : y \end{aligned} \tag{1}$$

Here $x = x_1, \ldots, x_n$ are the *attributes* of the system, S_0 is the initialization statement, while $A_i = g_i \to S_i$, $i = 1, \ldots, m$ are the *actions* of the system. We call g_i the *guard* of the action A_i, and S_i the *body* of the same action. The attributes $y = y_1, \ldots, y_k$ are defined in the environment of the continuous action system (we say that they are *imported*). Real_+ stands for the non-negative reals, and is used as the time domain.

Intuitively, executing a continuous action system proceeds as follows. There is an implicit variable now, that shows the present time. Initially $now = 0$. The guards of the actions may refer the value of now, as may also expressions in the action bodies (but they can not change now). The initialization S_0 assigns initial time functions to the attributes $x_1, \ldots, x_n$. These time functions describe the default future behavior of the attributes. The system will then start evolving according to these functions, with time (as measured by now) moving forward continuously. However, as soon as one of the conditions $g_1, \ldots, g_m$ becomes true, the system chooses one of the *enabled* actions, say $g_i \to S_i$, for execution. The choice is nondeterministic if there is more than one such action. The body S_i of the action is then executed. It will usually change some attributes by changing their future behavior. Attributes that are not changed will behave as before. After the changes stipulated by S_i have been done, the system will evolve to the next time instance when one of the actions is enabled, and the process is repeated. The next time instance when an action is enabled may well be the same as the previous, i.e., time needs not to progress between the execution of two enabled actions. This is usually the case when the system is doing some (discrete, logical) computation to determine how to proceed next. Such computation does not take any time. It is possible that after a certain time instance, none of the actions will be enabled anymore. This just means that, after this time instance, the system will continue to evolve for ever according to the functions last assigned to the attributes.

Note that in our approach actions are selected and executed asynchronously, compared to the hybrid automata formalism [11] where transitions are fired synchronously.

We write $x :- e$ for an assignment rather than $x := e$, to emphasize that only the future behavior of the attribute x is changed to the function e. The past behavior (before now) remains unchanged.

One of the main advantages of this model for hybrid computation is that both discrete and continuous behavior are described in the same way. In particular, if the attributes are only assigned constant functions, then we obtain a discrete computation.

Let $\mathcal{C}$ be the continuous action system described by (1). We explain the meaning of $\mathcal{C}$ by translating it into an ordinary action system. Its semantics is given by the following (discrete) action system $\bar{\mathcal{C}}$:

$$\begin{array}{l}\bar{\mathcal{C}} \triangleq |[\ \mathsf{var}\ now : \mathsf{Real}_+, x : \mathsf{Real}_+ \to T \bullet \\ \quad now := 0\ ;\ S_0\ ;\ N; \\ \quad \mathsf{do}\ g_1 \to S_1\ ;\ N [\!] \ \ldots\ [\!] \ g_m \to S_m\ ;\ N\ \mathsf{od} \\ ]| : y\end{array} \tag{2}$$

Here the attribute now is declared, initialized and updated explicitly. It models the moments of time that are of interest for the system, i.e. the starting time and the succeeding moments when some action is enabled. The value of now is updated by the statement N,

$$N \quad \triangleq \quad now := \mathsf{next}.gg.now$$

Above, $gg = g_1 \vee ... \vee g_m$ denotes the disjunction of all guards of the actions and next is defined by

$$\mathsf{next}.gg.t \triangleq \begin{cases} min\{t' \geq t \mid gg.t'\}, & \text{if exists } t' \geq t \\ & \text{such that } gg.t' \\ t, & \text{otherwise} \end{cases}$$

Thus, the function next gives a moment of time when at least one action is enabled. Only at such a moment can the future behavior of attributes be modified. If no action will be ever enabled, then the second branch of the definition will be followed, and the attribute now will denote the moment of time when the last discrete action was executed. In this case the system terminates with the last assigned values for the attributes. This means in the continuous interpretation that the system will evolve forever according to the functions assigned last. We assume in this paper that the minimum in the definition of next always exists when at least one guard is enabled in the present or future. Continuous action systems that do not satisfy this requirement are considered ill-defined.

We define the *future update* $x :- e$ by

$$x :- e \triangleq x := x/now/e$$

where

$$x/t_0/e \triangleq (\lambda t \cdot \mathsf{if}\ t < t_0\ \mathsf{then}\ x.t\ \mathsf{else}\ e.t\ \ \mathsf{fi})$$

Thus, only the future behavior of x is changed by this assignment.

This means that a hybrid action system is essentially a collection of time functions $x_1, \ldots, x_n$ over the nonnegative reals, defined in a stepwise manner. The steps form a sequence of intervals $I_0, I_1, I_2, \ldots$, where each interval I_k is either a left closed interval of the form $[t_i \ldots t_{i+1})$ or a closed interval of the form $[t_i, t_i]$, i.e., a point. The action system determines a family of functions $x_1, \ldots, x_n$ which are stepwise defined over this sequence of intervals and points. The extremes of these intervals correspond to the control points of the system where a discrete action is performed.

The behavior of a hybrid system is often described using a system of differential equations (DE). CAS allows for this kind of definitions, by introducing the shorthand $\dot{x} :- f(x)$. This will assign to x a time function that satisfies the given differential equation and which is such that the function x will evolve continuously.

As an example, if $f = (\lambda t \bullet c)$, where c is a constant value, then we have that

$$\dot{x} :- c \quad \triangleq \quad x :- (\lambda t \cdot x.now + c * (t - now))$$

We can use clock variables or timers to measure the passage of time and to correlate the execution of an action with the time. A *clock variable* is a variable that measures the time elapsed since it was set to zero. Assume that c is a time variable of type Real. We then use the following definition for resetting the clock c:

$$reset(c) \triangleq c :- (\lambda t \cdot t - now)$$

This definition is just a convenience for correlating the behavior of our system with the passage of time.

Since a clock variable is just a regular variable, we can define as many clocks as we need and reset them independently. It is also possible to do arithmetic operations with clock variables, e.g., to use time intervals in guards. Hence, the formalism is also well suited for modeling real-time systems.

3. Example: the temperature control system

We exemplify our approach on a case taken from [1]. The hybrid system is a temperature control system (TCS) for a heat producing reactor, described by the temperature as a function of time $\theta(t)$. The reactor starts from the initial temperature θ_0 and heats up at a given rate v_r. Whenever it reaches the critical temperature θ_M, it is designed to be cooled down by inserting into the core either of two rods, modeled by the variables $x_1(t)$ and $x_2(t)$, which are in fact clocks measuring the time elapsed between two consecutive insertions of the same rod, respectively. The cooling proceeds at the rate v_1 or v_2 depending on which rod is being used, and the cooling stops when the reactor reaches a given minimum temperature θ_m, by releasing the respective inserted rod. The rod used for cooling is then unavailable for a prescribed time T, after which it is again available for cooling. The object of the modeling is to ascertain that the reactor never reaches the critical temperature θ_M without at

least one of the rods available, otherwise a shutdown will be initiated.

The translated action system model (where time is explicitly advanced) for the TCS consists of a set of initializing statements and a collection of guards and their corresponding action bodies (see Figure 2). We use the convention that an assignment $x :- c$ (where c is a constant) stands for $x :- (\lambda t \cdot c)$ (i.e., the pointwise extended constant function, rather than the constant itself is assigned to x). Observe in Figure 2 that the model contains the analytic solutions of the linear DE that characterize the time evolution of the continuous variables (e.g., in $state0$, the DE expressing the dynamics of the increasing temperature inside the reactor core is $\dot{\theta} = v_r$, and its analytic solution is $\theta(t) = \theta_m + v_r * t$, as it starts increasing from the minimal temperature θ_m).

The last action (action 5) has $abort$ as its body, therefore expressing that the shutdown state is not desired. For a more detailed description of the model, the reader is referred to [3].

Let $\Delta\theta = \theta_M - \theta_m$. Obviously, the time that the coolant needs to increase its temperature from θ_m to θ_M is $\tau_r = \Delta\theta / v_r$, and the refrigeration times using $rod1$ and $rod2$ are $\tau_1 = \Delta\theta / v_1$ and $\tau_2 = \Delta\theta / v_2$, respectively.

The sequence of heating and refrigeration times is shown in Figure 1.

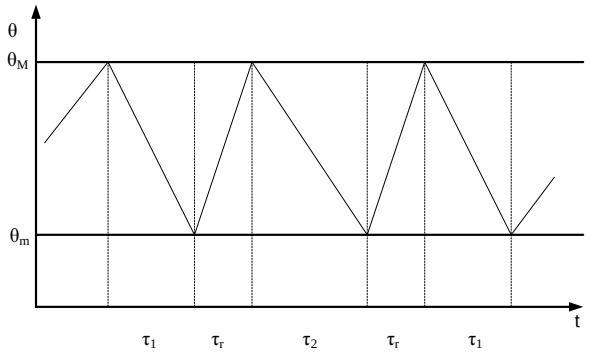

Figure 1. The heating and refrigeration times

Clearly, if $\tau_r \geq T$ (the temperature rises at a rate slower than the time of recovery of the rods), then the *shutdown* state is not reachable. However, this can be a too strong condition for not running into the undesired state. Inspecting Figure 1 we find a weaker condition:

$$2\tau_r + \tau_1 \geq T \wedge 2\tau_r + \tau_2 \geq T \qquad (3)$$

i.e., if the time between two insertions of the same rod is greater than or equal to the time needed for the rod to recover, the shutdown state will never be reached.

$TCS =$
$|($ var $x_1, x_2, c : \mathsf{Real}_+ \to \mathsf{Real}_+;$
$\theta : \mathsf{Real}_+ \to \mathsf{Real};$
$state : \mathsf{Real}_+ \to \{0,1,2,3\};$
$start, now : Real_+ \bullet$
$now :=_{\cdot} 0;$
$state :- (\lambda t \cdot 0)\ ; c :- (\lambda t \cdot t - now);$
$x_1 :- (\lambda t \cdot T_1 + c.t)\ ; x_2 :- (\lambda t \cdot T_2 + c.t);$
$\theta :- (\lambda t \cdot \theta_0 + v_r * c.t);$
$start :=_{\cdot} now;$
$now :=_{\cdot} min\{t' \geq now \mid gg.t'\}$
do $\{action1 : cool\ with\ rod1\}$
$state.now = 0 \wedge \theta.now = \theta_M \wedge x_1.now \geq T \to$
$c :- (\lambda t \cdot t - now);$
$\theta :- (\lambda t \cdot \theta_M - v_1 * (t - now));$
$state :- (\lambda t \cdot 1);$
$start :=_{\cdot} now;$
$now :=_{\cdot} min\{t' \geq now \mid gg.t'\}$
$[\!] \{action2 : release\ rod1\}$
$state.now = 1 \wedge \theta.now = \theta_m \to$
$c :- (\lambda t \cdot t - now);$
$x_1 :- (\lambda t \cdot t - now);$
$\theta :- (\lambda t \cdot \theta_m + v_r * (t - now));$
$state :- (\lambda t \cdot 0);$
$start :=_{\cdot} now;$
$now :=_{\cdot} min\{t' \geq now \mid gg.t'\}$
$[\!] \{action3 : cool\ with\ rod2\}$
$state.now = 0 \wedge \theta.now = \theta_M \wedge x_1.now \geq T \to$
$c :- (\lambda t \cdot t - now);$
$\theta :- (\lambda t \cdot \theta_M - v_2 * (t - now));$
$state :- (\lambda t \cdot 2);$
$start :=_{\cdot} now;$
$now :=_{\cdot} min\{t' \geq now \mid gg.t'\}$
$[\!] \{action4 : release\ rod2\}$
$state.now = 2 \wedge \theta.now = \theta_m \to$
$c :- (\lambda t \cdot t - now);$
$x_2 :- (\lambda t \cdot t - now);$
$\theta :- (\lambda t \cdot \theta_m + v_r * (t - now));$
$state :- (\lambda t \cdot 0);$
$start :=_{\cdot} now;$
$now :=_{\cdot} min\{t' \geq now \mid gg.t'\}$
$[\!] \{action5 : shutdown\}$
$state.now = 0 \wedge \theta.now = \theta_M \wedge$
$x_1.now < T \wedge x_2.now < T \to$
$abort$
od
$)| : \theta_0, \theta_m, \theta_M, v_r, v_1, v_2, T_1, T_2, T$

Figure 2. The TCS action system model

To get a first assurance that condition (3) is indeed sufficient, we proceed with the simulation of the TCS model for two sets of parameters: the first set chosen to satisfy condition (3), the second set chosen not to satisfy the same condition. The simulation results should either confirm or deny our assertion. In the second case, at some point in time, the simulation should run into *abort* by executing the action 5 in the TCS action system model.

4. Simulating the behavior of the TCS in Mathematica

The starting point for the formulation of the simulation is to take the initializing expressions and the expressions for the guards and the action bodies from the TCS action system model as such, with as few numerical or logical manipulations as possible. This confirms with our basic strategy of simulating the model as given, thus exposing any possible modeling errors like in the spelling of the model or in the logic of the guarded actions. In the case of TCS the initializing expressions in the language of the symbolic manipulation program are given by

$$\begin{array}{ll} now & = 0 \\ c\,[t\_] & = t - now \\ x_1\,[t\_] & = T_1 + c\,[t] \\ x_2\,[t\_] & = T_2 + c\,[t] \\ \theta\,[t\_] & = \theta_0 + \nu_r * c\,[t] \\ state\,[t\_] & = 0 \end{array}$$

In Mathematica, $t\_$ signifies that t is the variable in the function that is being defined. We assume that we start in $state0$, with the rods 1 and 2 both available for cooling, hence the clocks x_1 and x_2 are initialized to the (constant) values T_1 and T_2 (time units), respectively.

The guards are typically boolean conditions which we test for the value of true. In the TCS model, the first guard has the form

$$\begin{array}{l} guard1solution = InequalitySolve[\\ \qquad\qquad state\,[t] == 0\ \&\& \\ \qquad\qquad \theta\,[t] == \theta_M\ \&\& \\ \qquad\qquad x_1\,[t] >= T\ \&\& \\ \qquad\qquad t >= now, t \\ \qquad\qquad] \end{array}$$

Here we are using the Mathematica built-in function $InequalitySolve$ to determine the next moment or moments in time at or after now, when all the conditions of guard 1 become true, that is, the system is in state 0, it has reached the critical temperature and rod1 is available. As a result of solving the simultaneous inequalities we obtain a list called $guard1solution$, which contains the empty set, or a collection of discrete times and/or finite or infinite ranges of times for which the conditions are true. This list is passed to a subroutine which picks out the earliest time at which guard 1 becomes true.

Similarly, the body of the action 1, should we decide to take that action, is given by the following expressions:

$$\begin{array}{ll} c\,[t\_] & = t - now \\ \theta\,[t\_] & = \theta_M - \nu_1 * c\,[t] \\ state\,[t\_] & = 1 \\ start & = now \end{array}$$

The main task of the simulation is to go through the guards one by one and determine whether they will become true at some point in time in the future. In case there are several solutions to a guard, the minimum of these times is selected, be it a discrete value or the starting value for a closed range. After this, the minimum times for all guards are compared, and the smallest of these with the corresponding action (or actions) body is chosen. In case the next action is one particular action, we will take that action, update the value of now and solve the guards over again. In case several guards become true at the next instance of time, all corresponding action bodies are of course possible, and the user is asked to supply the choice of action to be taken. In addition, a random mode was programmed, in which case a choice between multiple possible actions is made by the simulator.

5. Simulation results

The essential information gained by the above procedure is a list of time moments at which some action has been taken in the model, a corresponding list of actions, and lists with symbolic values for the discrete and continuous functions of the TCS hybrid model: the system state, the temperature of the reactor $\theta(t)$ as a continuous piecewise linear function, and similar functions for the clocks $x_1(t)$ and $x_2(t)$. An artificial upper time limit $t_{max} = 100$ was supplied in case the simulation would go on forever.

Given the parameter values $T_1 = 6, T_2 = 2, T = 6, v_1 = 4, v_2 = 3, v_r = 6, \theta_0 = 0, \theta_m = 3$ and $\theta_M = 15$, which satisfy condition (3), two of the lists mentioned above are the following:

$$\begin{array}{l} now = \{0, 5/2, 11/2, 15/2, 23/2, 27/2, 33/2, 37/2, \\ \qquad 45/2, 49/2, 55/2, 59/2, 67/2, 71/2, 77/2, \\ \qquad 81/2, 89/2, 93/2, 99/2, 103/2, 111/2, 115/2, \\ \qquad 121/2, 125/2, 133/2, 137/2, 143/2, 147/2, \\ \qquad 155/2, 159/2, 165/2, 169/2, 177/2, 181/2, \\ \qquad 187/2, 191/2, 199/2\} \end{array}$$

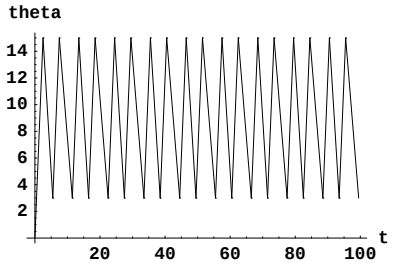

Figure 3. The temperature behavior in time (parameter set 1)

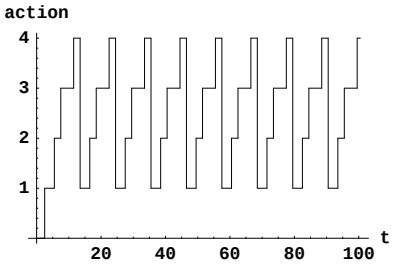

Figure 4. The actions taken at transition time moments (parameter set 1)

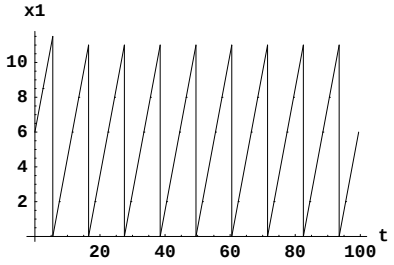

Figure 5. The clock x_1 as a function of time (parameter set 1)

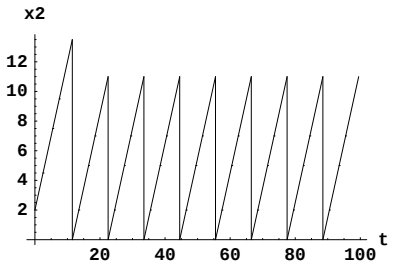

Figure 6. The clock x_2 as a function of time (parameter set 1)

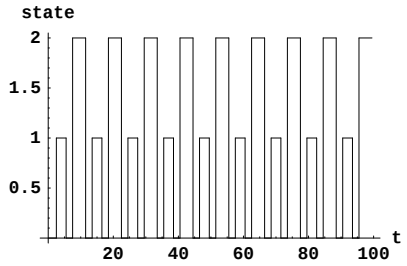

Figure 7. The state as a function of time (parameter set 1)

$$
\begin{aligned}
theta(t) : \{&6t, 25-4t, -30+6t, 75/2-3t, -66+6t,\\
&69-4t, -96+6t, 141/2-3t, -132+6t,\\
&113-4t, -162+6t, 207/2-3t, -198+6t,\\
&157-4t, -228+6t, 273/2-3t, -264+6t,\\
&201-4t, -294+6t, 339/2-3t, -330+6t,\\
&245-4t, -360+6t, 405/2-3t, -396+6t,\\
&289-4t, -426+6t, 471/2-3t, -462+6t,\\
&333-4t, -492+6t, 537/2-3t, -528+6t,\\
&377-4t, -558+6t, 603/2-3t, -594+6t\}
\end{aligned}
$$

Using the first parameter set, the graphical results of the simulation are the plots in Figures 3 to 7. The vertical lines in the graphs $action(t)$ and $state(t)$ are purposely drawn to guide the reader's eye.

In this first case, the simulation did not reveal any unexpected behavior, instead it showed a regular time behavior of the state variables.

For a different set of values that violate the condition (3), e.g. the same set as above except $T = 8$, the simulation shows that the reactor will reach the shutdown state, i.e., action 5 is enabled, since neither of the rods is available at time $t = 37/2$ (see Figure 9). Similar to the first case, here we also get the graphical representation with respect to time, of all the model variables, as seen in Figures 8 to 12.

In consequence, the TCS simulation confirmed our guess for the particular values chosen, that in case the parameters do not satisfy condition (3), the system will eventually reach the undesired shutdown state.

6. The generic simulator

In this section we try to describe the simulator in a more generic setting, independent of the programming language used, of course with its usability certainly benefitting from having as powerful language as possible.

The symbolic simulation of a CAS given by (2) consists of three major steps, as follows:

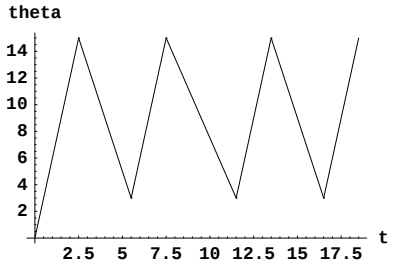

Figure 8. The temperature behavior in time (parameter set 2)

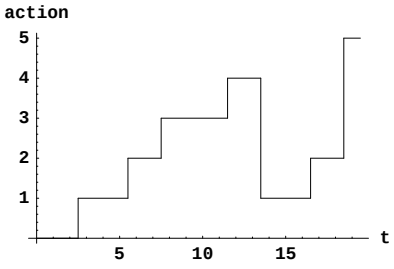

Figure 9. The actions taken at transition time moments (parameter set 2)

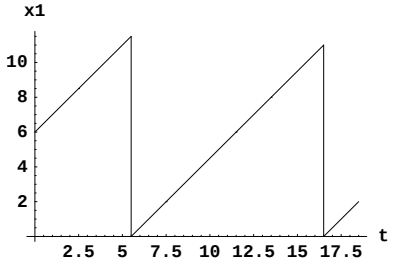

Figure 10. The clock x_1 as a function of time (parameter set 2)

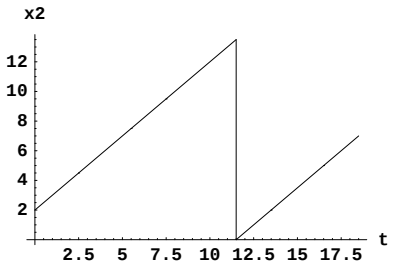

Figure 11. The clock x_2 as a function of time (parameter set 2)

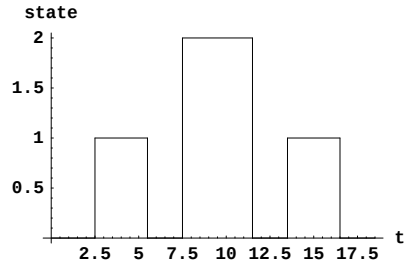

Figure 12. The state as a function of time (parameter set 2)

• The first step is to solve each guard separately and find a list of times in the future when the guard will evaluate to true.

• The second task is to extract the least of the times in the list for each guard.

• The third step is to collect the results from step one and two, from all guards, and determine a globally minimal next time. Having found it, we have simultaneously determined whether we have one or several guards satisfied at the respective time moment. If only one guard is satisfied, the corresponding action body is executed, thus changing some of the program attributes. If there are several guards simultaneously evaluating to true, then the user is asked to supply the choice of action to be taken. It is also possible that the simulator makes a random choice between the possible actions.

In the first step, obviously the determination of the solution list for the guards can be made arbitrarily complicated depending on the structure of the guards. The guard may involve the solution of higher order algebraic equations or nonlinear differential equations or both, in which case analytic solutions to the guards are probably impossible to obtain. In this case we have to resort to a numerical solution of the guards, e.g. integrate differential equations forward in time using some appropriate numerical scheme. Here we can still obtain an approximated continuous solution by interpolating the numerical solution with linear functions between the numerically obtained values.

In case the list of minimum values for the guard from step one is a collection of finite analytic expressions, we will be able to proceed to step two without loss of accuracy. The identification of the minimum value in step two, that is, sorting the list of solutions to a guard, may be numerically cumbersome. The expressions in the list can easily have the tendency of becoming increasingly complicated as time goes on, and in the end we have to resort to evaluating

the minimum values numerically. This immediately makes the comparison of values very close to each other prone to mistakes. The third step is in principle as hard as step two, only now we are comparing the minimum values from each guard with each other.

The usability of the symbolic simulator is thus largely dependent on whether we are able to pass through step one to three using symbolic expressions.

An advantage of the symbolic approach is that as far as possible we tried not to apply any transformations to the guards or the action bodies, when translating them into Mathematica. Instead we have expressed them almost in the same way as they are in the CAS model. This guarantees that the simulated model is indeed as consistent and reliable as the original model. The number of guards and respective action bodies is given as a parameter, hence one can simulate large models that consist of many guarded actions.

However, what we consider to be the most valuable contribution made by this tool for simulating CAS models is the integration of the modeling, simulation and verification of hybrid systems, into the same framework that uses CAS as the modeling language and the refinement calculus as the reasoning environment. This calculus has been already implemented in several theorem provers [8, 12]. The advantage of having a unified design environment might turn continuous action systems into a more attractive modeling language for hybrid systems.

7. Conclusions

In this paper we have presented a simulation tool for hybrid systems modeled as continuous action systems. We have built the tool using Mathematica, a commercial symbolic manipulation program [15]. The tool takes a description of any CAS as input, and provides automatically a symbolic simulation of the system, up to a given maximum time. The restrictions on the simulation are essentially those of Mathematica.

Our approach relies on the fact that symbolic manipulation is an efficient way of simulating a model execution. Plotting the discrete and also continuous model variables as functions of time, with infinite precision, makes the simulation available even without knowing the sampling period to be used for the actual implementation, thus in many cases our tool eliminates the need of introducing tolerances in the model. This is true especially when the physical phenomena of the hybrid system is described by linear differential equations. In case the hybrid model is non-linear, Mathematica solves the respective non-linear DE either symbolically or numerically. It then follows that, in case we get a numerical solution, we need to introduce tolerances in our action system model and rely on an approximation of the behavior of the variables.

The experiences with this tool have been very promising. It provides a good visualization of the behavior of hybrid systems, and has been also quite efficient in harnessing the power of Mathematica. We have applied our simulation technique to a small collection of hybrid systems. Here, we have exemplified the tool on the temperature control system inside a nuclear reactor core, which uses two independent rods for cooling. Given a certain set of parameters, the objective of the simulation was to make sure that the reactor never reaches a critical temperature without at least one of the cooling rods being available, to avoid a shutdown of the reactor. The simulation results helped in correlating the model with the actual system behavior.

One of the main advantages of using continuous action systems for modeling hybrid systems is that we now have both a solid proof technique for proving properties of the systems, as well as a powerful simulation technique that we can use to analyze and explore the systems. Simulation can either be used as a precursor to more comprehensive proofs, to iron out bugs in the model, or as an alternative to a complete correctness proof.

Future work includes simulating more complex hybrid systems, e.g. non-linear, modeled as continuous action systems, and also the design of some graphical user interface to the simulator.

References

[1] R. Alur, C. Courcourbetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine, “The Algorithmic Analysis of Hybrid Systems”, *Theoretical Computer Science*, 1995, vol. 138, pp. 3-34.

[2] R. Alur, D.L. Dill, “A Theory of Timed Automata”, *Theoretical Computer Science*, April 1994, vol. 126(2), pp. 183-235.

[3] R.J.R. Back, C. Cerschi, “Modeling and Verifying a Temperature Control System using Continuous Action Systems”, In proceedings of the 5th International ERCIM Workshop on Formal Methods for Industrial Critical Systems (FMICS’2000), GMD Report 91, *ERCIM and GMD*, April 2000, pp. 265-286.

[4] R.J.R. Back and R. Kurki-Suonio, “Decentralization of Process Nets with Centralized Control”, In the 2nd Symposium on Principles of Distributed Computing, Lecture Notes in Computer Science, vol. 873, *ACM SIGACT-SIGOPS*, 1983, pp. 131-142.

[5] R.J. Back, L. Petre, and I. Porres-Paltor, “Continuous Action Systems as a Model for Hybrid Systems”, *Nordic Journal of Computing*, 2001, vol. 8, pp. 2-21.

[6] Back R.J.R. and J. von Wright, *Refinement Calculus, A Systematic Introduction*, Springer Verlag, 1998.

[7] D.A. van Beek, J.E. Rooda, "Languages and Applications in Hybrid Modelling and Simulation: Positioning of Chi", *Control Engineering Practice*, 2000, vol. 8, nr. 1, pp. 81-91.

[8] M.J. Butler, J. Grundy, T. Långbacka, R. Rukšenas, and J. von Wright, "The refinement calculator: Proof support for program refinement", In Proceedings FMP'97 - Formal Methods Pacific, Discrete Mathematics and Theoretical Computer Science, Wellington, New Zealand, *Springer-Verlag*, July 1997.

[9] H. Elmqvist, "Object-Oriented Modeling and Automatic Formula Manipulation in Dymola", SIMS'93, *Scandinavian Simulation Society*, 1993.

[10] A. Göllü, M. Kourjanski, P. Varaiya, "The SHIFT Simulation Framework: Language, Model and Implementation (Extended Abstract)", In proceedings of the 5th International Hybrid Systems Workshop, Notre Dame, Indiana, Sept. 1997.

[11] T.A. Henzinger, "The Theory of Hybrid Automata", In proceedings of the 11th Annual Symposium on Logic in Computer Science (LICS), IEEE Computer Society Press, 1996, pp. 278-292.

[12] J. Knappmann, "A PVS based Tool for Developing Programs in the Refinement Calculus", *http://www.informatik.uni-kiel.de/inf/deRoever/DiplJKm.html*, October 1996.

[13] N. Lynch, R. Segala, F. Vaandrager, "Hybrid I/O Automata Revisited", In proceedings of the 4th Hybrid Systems Computation and Control (HSCC 2001), Rome, Italy, Lecture Notes in Computer Science, vol. 2034, pp. 403-417.

[14] "The MathWorks: Developers of MATLAB and Simulink for Technical Computing", *http://www.mathworks.com/*.

[15] Wolfram S., *The Mathematica Book*, Fourth Edition, Wolfram Media/Cambridge University Press, 1999.