A Real-Time Animator for Hybrid Systems

Tobias Amnell, Alexandre David
Wang Yi
Department of Computer Systems, Uppsala University
{adavid, tobiasa, yi} @docs.uu.se

Abstract

In this paper, we present a real time animator for dynamical systems that can be modeled as hybrid automata i.e. standard finite automata extended with differential equations. We describe its semantic foundation and its implementation in Java and C using CVODE, a software package for solving ordinary differential equations. We show how the animator is interfaced with the UPPAAL tool to demonstrate the real time behavior of dynamical systems under the control of discrete components described as timed automata.

1 Introduction

UPPAAL is a software tool for modeling, simulation and verification of real time systems that can be described as timed automata. In recent years, it has been applied in a number of case studies [4, 5, 6, 7, 8], which demonstrates the potential application areas of the tool. It suits best the class of systems that contain only discrete components with real time clocks. But it can not handle hybrid systems, which has been a serious restriction on many industrial applications. This work is to extend the UPPAAL tool with features for modeling and simulation of hybrid systems.

A hybrid system is a dynamical system that may contain both discrete and continuous components whose behavior follows physical laws [1], e.g. process control and automotive systems. In this paper, we shall adopt hybrid automata as a basic model for such systems. A hybrid automaton is a finite automaton extended with differential equations assigned to control nodes, describing the physical laws. Timed automata [2] can be seen as special class of hybrid automata with the equation $\dot{x} = 1$ for all clocks x. We shall present an operational semantics for hybrid automata with dense time and its discrete version for a given time granularity. The discrete semantics of a hybrid system shall be considered as an approximation of the continuous behavior of the system, corresponding to sampling in control theory.

We have developed a real time animator for hybrid systems based on the discrete semantics. It can be used to simulate the dynamical behavior of a hybrid system in a real time manner. The animator implements the discrete semantics for a given automaton and sampling period, using the differential equation solver CVODE. Currently the engine of the animator has been implemented in Java and C using CVODE. We are aiming at a graphical user interface for editing and showing moving graphical objects and plotting curves. The graphical objects act on the screen according to physical laws described as differential equations and synchronize with controllers described as timed automata in UPPAAL.

The rest of the paper is organized as follows: In the next section we describe the notion of hybrid automata, the syntax and operational semantics. Section 3 is devoted to implementation details of the animator. Section 4 concludes the paper.

2 Hybrid Systems

A hybrid automaton is a finite automata extended with differential equations describing the dynamical behavior of the physical components.

2.1 Syntax

Let X be a set of real-valued variables X ranged over by x, y, z etc including a time variable t.

We use $\dot{x}$ to denote the derivative (rate) of x with respects to the time variable t. Note that in general $\dot{x}$ may be a function over X; but $\dot{t} = 1$. We use $\dot{X}$ to stand for the set of differential equations in the form $\dot{x} = f(X)$ where f is a function over X.

Assume a set of predicates over the values of X; for example, $2^x + 1 \leq 10$ is such a predicate. We use $\mathcal{G}$ ranged over by g, h etc to denote the set of boolean combinations of the predicates, called *guards*.

To manipulate variables, we use concurrent assignments in the form: $x_1 := f_1(X) \ldots x_n := f_n(X)$ that takes the current values of the variables X as parameters for f_i and updates all x_i's with $f_i(X)$'s simultaneously. We use Γ to stand for the set of concurrent assignments.

We shall study networks of hybrid automata in which component automata synchronize with each other via complementary actions. Let $\mathcal{A}$ be a set of action names. We use $\mathcal{A}ct = \{ a? \mid \alpha \in \mathcal{A} \} \cup \{ a! \mid \alpha \in \mathcal{A} \} \cup \{ \tau \}$ to denote the set of actions that processes can perform to synchronize with each other, where τ is a distinct symbol representing internal actions.

A hybrid automaton over X, $\dot{X}$, $\mathcal{G}$, $\mathcal{A}ct$ and Γ is a tuple $\langle L, E, I, T, L_0, X_0 \rangle$ where

- L is a finite set of names standing for control nodes.
- E is the equation assignment function: $E : L \rightarrow 2^{\dot{X}}$.
- I is the invariant assignment function: $I : L \rightarrow \mathcal{G}$ which for each node l, assigns an invariant condition $I(l)$.
- T is the transition relation: $T \subseteq L \times (\mathcal{G} \times \mathcal{A}ct \times \Gamma) \times L$. We denote $(l, g, \alpha, \gamma, l')$ by $l \xrightarrow{g,\alpha,\gamma} l'$. For simplicity, we shall use $l \xrightarrow{g,\gamma} l'$ to stand for $l \xrightarrow{g,\tau,\gamma} l'$.
- $l_0 \in L$ is the initial node.
- X_0 is the initial variable assignment.

To study networks of automata, we introduce a CCS-like parallel composition operator. Assume that $A_1, ..., A_n$ are automata. We use $\overline{A}$ to denote their parallel composition. The intuitive meaning of $\overline{A}$ is similar to the CCS parallel composition of $A_1, ..., A_n$ with *all* actions being restricted, that is, $\overline{A} = (A_1|...|A_n)\backslash\mathcal{A}ct$. Thus only synchronization between the components A_i is possible. We call $\overline{A}$ a *network of automata*. We simply view $\overline{A}$ as a vector and use A_i to denote its ith component.

Example 1 In figure 1 we give a simple example hybrid automaton which describes a bouncing ball and a touch sensitive floor. The left automaton defines three variables, x, the horizontal distance from the starting point, the height y and the speed upwards u. Initially the x-speed is 1, the ball is at 20 m height and the gravitational constant is 9.8. The variables will change according to their equations until the transition becomes enabled when $y <= 0$. The middle automaton is a model of a sensor that will issue a signal when the ball hits the floor. The right automaton is on the UPPAAL side. It synchronizes with the sensor signal and resets a clock z. If the intervals between signals are longer than 5 time units it will return to the initial location, but the first interval that is shorter will lead to the location *low_bounces*.

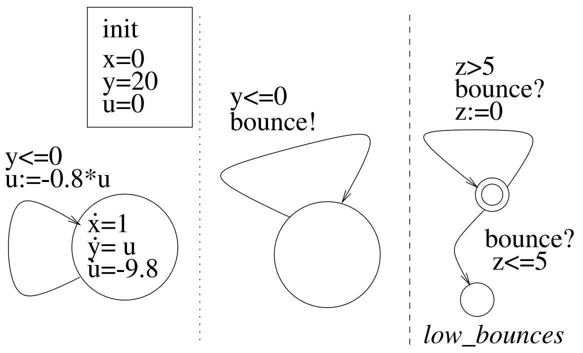

Figure 1: Bouncing ball with touch sensitive floor and control program.

Example 2 As a more complex example we show a model of an industrial robot. In figure 2 a schematic view of robot with a jointed arm is shown. The inner arm can be turned 360 degrees around the z-axis, it can also be raised and lowered between 40 and 60 degrees. The outer arm is positioned at the tip of the inner arm and can be raised and lowered.

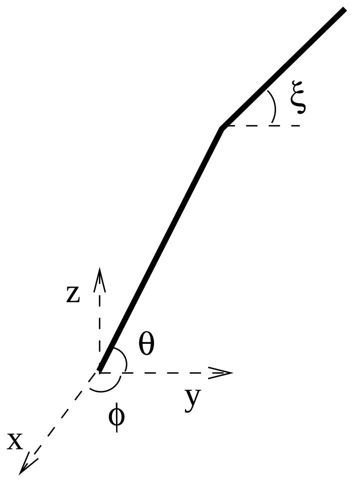

Figure 2: Industrial robot with three degrees of freedom.

In figures 3 and 4 the hybrid automaton controlling the motion of the robot is shown. We do not show the simple control automaton that starts and stops the execution.

When the execution starts the inner arm will stand still until it receives the `move2pickup?` signal from the control automaton. Then the arm will start turning and lowering the arm so that the gripping tool on the outer arm can reach a table where it will pick up an object. When th inner arm reaches a position in front of the table it will try to synchronize with the outer arms automaton on the signal `pickup!` . After the pickup the robot will raise and turn to another table where it will again try to synchronize with the outer arm on `release!` . After the release the robot will return to the original position and issue the `back!` signal to the controller.

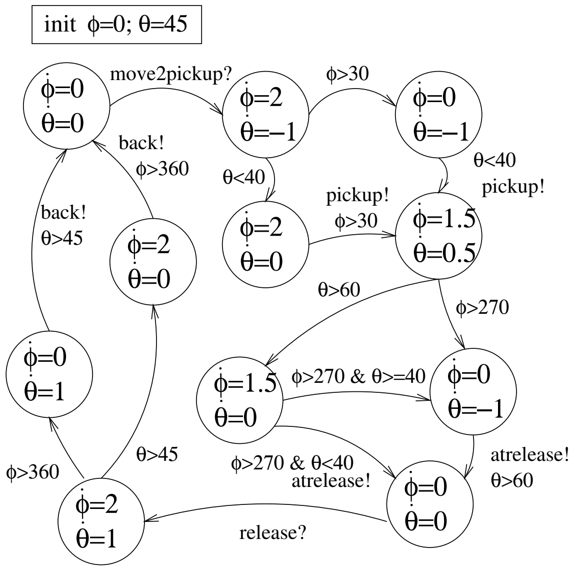

Figure 3: Robot inner arm automaton

The automaton, in figure 4, controlling the outer arm is simpler since the outer arm only has one degree of freedom. It starts with lowering the arm when the controller sends a `move2pickup2!`. When it comes to the right angle it will stop and wait for the inner arm to do the same, then they will synchronize on `pickup`. After the pickup the arm will raise until it reaches its max (30 degrees), then it will wait for the inner arm to reach the release position. When the robot is at the release table the outer arm will descend and then release the object in its grip. On the return to the original position the outer arm will rise again.

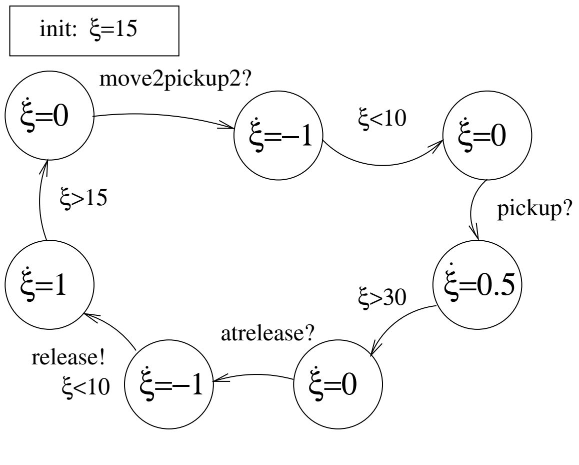

Figure 4: Robot outer arm automaton

2.2 Semantics

To develop a formal semantics for hybrid automata we shall use variable assignments. A *variable assignment* is a mapping which maps variables X to the reals. For a variable assignment σ and a delay Δ (a positive real), $\sigma + \Delta$ denotes the variable assignment such that

$$(\sigma + \Delta)(x) = \sigma(x) + \int_{\Delta} \dot{x} dt$$

For a concurrent assignment γ, we use $\gamma[\sigma]$ to denote the variable assignment σ' with $\sigma'(x) = Val(e, \sigma)$ whenever $(x := e) \in \gamma$ and $\sigma'(x') = \sigma(x')$ otherwise, where $Val(e, \sigma)$ denotes the value of e in σ. Given a guard $g \in \mathcal{G}$ and a variable assignment σ, $g(\sigma)$ is a boolean value describing whether g is satisfied by σ or not.

A *node vector* $\bar{l}$ of a network $\overline{A}$ is a vector of nodes where l_i is a location of A_i. We write $\bar{l}[l'_i/l_i]$ to denote the vector where the ith element l_i of $\bar{l}$ is replaced by l'_i.

A *state* of a network $\overline{A}$ is a configuration $(\bar{l}, \sigma)$ where $\bar{l}$ is a node vector of $\overline{A}$ and σ is a variable assignment.

The *semantics of a network* of automata $\overline{A}$ is given in terms of a labelled transition system with the set of states being the configurations. The transition relation is defined by the following three rules:

- $(\bar{l}, \sigma) \overset{\alpha}{\rightsquigarrow} (\bar{l}[l'_i/l_i], \gamma_i[\sigma])$ if $l_i \xrightarrow{g_i\, \alpha, \gamma_i} l'_i$ and $g_i(\sigma)$ for some l_i, g_i,α,γ_i.
- $(\bar{l}, \sigma) \overset{\tau}{\rightsquigarrow} (\bar{l}[l'_i/l_i, l'_j/l_j], (\gamma_j \cup \gamma_i)[\sigma])$ if $l_i \xrightarrow{g_i\, a!\, \gamma_i} l'_i$, $l_j \xrightarrow{g_j\, a?\, \gamma_j} l'_j$, $g_i(\sigma)$, $g_j(\sigma)$, and $i \neq j$, for some l_i, l_j, g_i, g_j, a, γ_i, γ_j.
- $(\bar{l}, \sigma) \overset{\Delta}{\rightsquigarrow} (\bar{l}, \sigma + \Delta)$ if $I(\bar{l})(\sigma)$ and $I(\bar{l})(\sigma + \Delta)$ for all positive real numbers Δ.

where $I(\bar{l}) = \bigwedge_i I(l_i)$.

The execution of a hybrid automata then becomes an alternating sequence of delay and action transitions in the form:
$s_0 \overset{\Delta_0}{\rightsquigarrow} (\bar{l}_0, \sigma_0 + \Delta_0) \overset{\alpha_0}{\rightsquigarrow} (\bar{l}_1, \sigma_1) \overset{\Delta_1}{\rightsquigarrow} (\bar{l}_1, \sigma_1 + \Delta_1) \overset{\alpha_1}{\rightsquigarrow} (\bar{l}_2, \sigma_2) \ldots (\bar{l}_i, \sigma_i) \overset{\Delta_i}{\rightsquigarrow} (\bar{l}_i, \sigma_i + \Delta_i) \overset{\alpha_i}{\rightsquigarrow} (\bar{l}_{i+1}, \sigma_{i+1})$

2.3 Tick semantics

The operational semantics above defines how an automaton will behave at every real-valued time point with arbitrarily fine precision. In fact, it describes all the possible runnings of a hybrid automata.

In practice, a "sampling" technique is often needed to analyze a system. Instead of examining the system at *every* time point, which often is impossible, only a finite number of time points are chosen to approximate the full system behavior. Based on this idea, we shall adopt a time–step semantics called δ–semantics relativized by the granularity δ, which describes how a hybrid system shall behave in every δ time units. In practical applications, the time granularity δ is chosen according to the nature of the differential equations involved. In a manner similar to sampling of measured signals the sampling interval should be short for rapidly changing functions. To achieve finer precision, we can choose a smaller granularity.

We use the distinct symbol χ to denote the sampled time steps. Now we have a discrete semantics for hybrid automata.

- $(\bar{l},\sigma)\overset{\chi}{\mapsto}(\bar{l},\sigma+\delta)$ if $(\bar{l},\sigma)\overset{\delta}{\leadsto}(\bar{l},\sigma+\delta)$ and
- $(\bar{l},\sigma)\overset{\alpha}{\mapsto}(\bar{l}',\sigma')$ if $(\bar{l},\sigma)\overset{\alpha}{\leadsto}(\bar{l}',\sigma')$

We use β_i to range over $\{\chi,\tau\}$ representing the discrete transitions. The "sampled" runnings of a hybrid automaton will be in the form:

$(\bar{l}_0,\sigma_0)\overset{\beta_1}{\mapsto}(\bar{l}_1,\sigma_1)\ldots(\bar{l}_i,\sigma_i)\overset{\beta_{i+1}}{\mapsto}(\bar{l}_{i+1},\sigma_{i+1})\ldots$

In the following section, we shall present a real time animator based on the δ-semantics. For a given hybrid automaton, the animator works as an interpretor computing the δ-transitions step by step using CVODE, a differential equations solver.

3 Implementation

Our goal is to extend the Uppaal tool to deal with hybrid systems. The Uppaal GUI is written in Java and the differential equation solver that we have adopted, CVODE, is written in C. This gives the natural architecture of the animator: the animator itself with the objects is written in Java and the engine of the animator in C, connected through the Java native interface (JNI). The two main layers of the implementation are the animation system and the CVODE layers.

3.1 The Animation System Layer

The system to be modeled is defined as a collection of *objects*. Each object is described by a hybrid automaton with its corresponding variables. Every state of the hybrid automaton has a set of equations and transitions. The equations, conditions (guards) and assignments are given as logical/arithmetic expressions with ordinary mathematical functions such as sine, cosine ..., and also user defined functions.

The evaluation of the object equations, conditions and assignments is written in C. Each animator object, i.e. a hybrid process, is associated with one Uppaal process that is an abstraction of the hybrid part and a bridge to Uppaal. The abstraction is modeled as a stub process that performs the same synchronizations as the hybrid counterpart. This choice of implementation is motivated by the desire to model-check the rest of the Uppaal processes as a closed system. Figure 5 shows the association of animator objects with Uppaal automata.

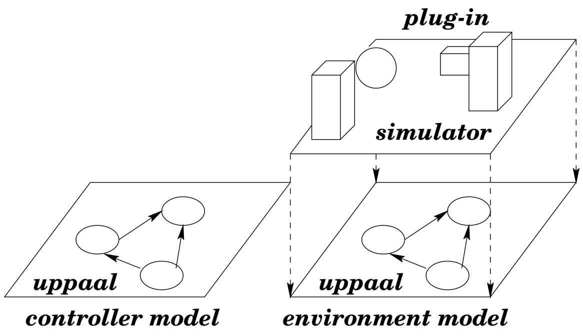

Figure 5: Association between animator objects and Uppaal automata.

3.2 The CVODE Layer

At the heart of the animator we have used the CVODE [9] solver for ordinary differential equations (ODE's). This is a freely available ODE solver written in C, but based on two older solvers in Fortran.

The mathematical formulation of an initial value ODE problem is

$$\dot{x} = f(t,x),\ x(t_0)=x_0,\ x\in \boldsymbol{R}^N. \qquad (1)$$

Note that the derivative is only first order. Problems containing higher order differential equations can be transformed to a system of first order. When using CVODE one gets a numerical solution to (1) as discrete values x_n at time points t_n.

CVODE provides several methods for solving ODE's, suitable for different types of problem. But since we aim at general usage of the animator engine we cannot assume any certain properties of the system to solve. Therefore we only use the full dense solver and assume that the system is well behaved (non-stiff in numerical analysis terminology). This will give neither the most memory efficient nor the best solution, but the most general.

We use one CVODE solver for the whole system. This is set up and started with new initial values at the beginning of each delay transition. The calculations are performed stepwise, one "tick" (δ–transition) at a time. After each tick all the conditions of the current state are checked, if any is evaluated to true one has to be taken. If an assignment on the transition changes a variable the solver must be reinitialized before the calculations can continue.

It is worth pointing out that the tick length δ is independent of the internal step size used by the ODE solver, the solver will automatically choose an appropriate step size according to the function calculated and acceptable local error of the computation. From the solvers point of view the tick intervals can be seen as observation or sampling points.

After each tick the system variables are returned to the Java side of the animator where they are used either to update a graph or as an input to move graphical objects.

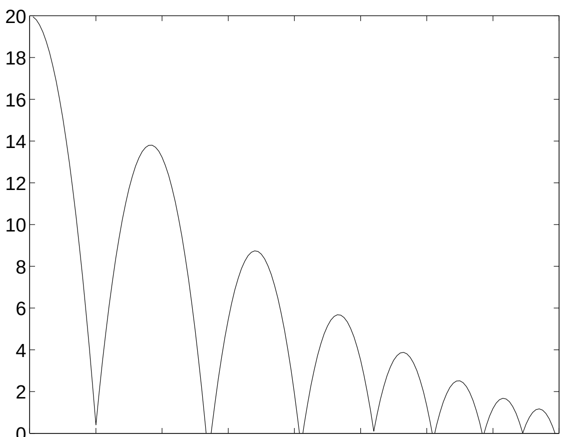

Figure 6: Bouncing ball on touch sensitive floor that continues until bounces are shorter than 1 second

Example 1, continued In figure 6 a plot of the system described in figure 1 is shown. The plot shows the height and the distance of the bouncing ball. Not shown in the figure is that the touch sensitive floor will create a signal every time the ball hits the floor, and that the system will continue running until the time between bounces is less than 1 second.

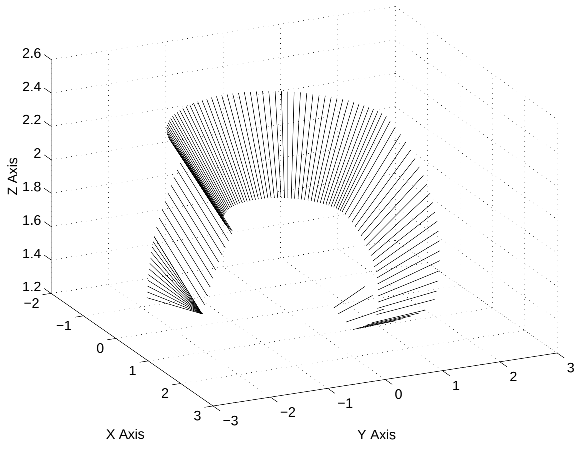

Figure 7: The movement of the robots outer arm.

Example 2, continued For the robot example we only show, in figure 7, how the outer arm will raise and turn during the execution of the system. In order to get a better view, the plot only shows the movement from the initial position to the release. The robot starts at the right and turns counter clock-wise.

4 Conclusion

We have presented a real time animator for hybrid automata. For a given hybrid automaton modeling a dynamical system and a given time granularity representing sampling frequency, the animator demonstrates a possible running of the system in real time, which is a sequence of sampled transitions. The animator has been implemented in Java and C using CVODE, a software package for solving differential equations. As future work, we aim at a graphical user interface for editing and showing moving graphical objects and plotting curves. The graphical objects act on the screen according to the differential equations and synchronize with controllers described as timed automata in UPPAAL.

References

[1] Thomas A. Henzinger. *The Theory of Hybrid Automata.* Proceedings of the 11th Annual IEEE Symposium on Logic on Computer Science (LICS 96), pp. 278-292.

[2] R. Alur and D.L. Dill. *A Theory of Timed Automata.* Theoretical Computer Science, 125:183-235,1994.

[3] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and W. Yi. UPPAAL*: a Tool-Suite for Automatic Verification of Real-time Systems.* In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, Hybrid Systems III, Lecture Notes in Computer Science 1066, pages 232-243. Springer-Verlag 1996.

[4] Kåre J. Kristoffersen, Kim G. Larsen, Paul Pettersson and Carsten Weise. *Experimental Batch Plant - VHS Case Study 1 Using Timed Automata and UPPAAL.* Deliverable of EPRIT-LTR Project 26270 VHS (Verification of Hybird Systems).

[5] Magnus Lindahl, Paul Pettersson and Wang Yi *Formal Design and Analysis of a Gear Controller.* In Proceedings of the 4th International Workshop on Tools and Algorithms for the Construction and Analysis of Systems. Gulbenkian Foundation, Lisbon, Portugal, 31 March - 2 April, 1998. LNCS 1384, pages 281-297, Bernhard Steffen (Ed.).

[6] Henrik Lönn and Paul Pettersson. *Formal Verification of a TDMA Protocol Start-Up Mechanism.* In Proceedings of 1997 IEEE Pacific Rim International Symposium on Fault-Tolerant Systems, pages 235-242. Taipei, Taiwan, 15-16 December, 1997.

[7] Klaus Havelund, Arne Skou, Kim G. Larsen and Kristian Lund. *Formal Modelling and Analysis of an Audio/Video Protocol: An Industrial Case Study Using UPPAAL.* In Proceedings of the 18th IEEE Real-Time Systems Symposium, pages 2-13. San Francisco, California, USA, 3-5 December 1997.

[8] P.R. D'Argenio, J.-P. Katoen, T.C. Ruys, and J. Tretmans. *The bounded retransmission protocol must be on time!* In Proceedings of the 3rd International Workshop on Tools and Algorithms for the Construction and Analysis of Systems. Enschede, The Netherlands, April 1997. LNCS 1217, pages 416-431.

[9] S. Cohen and A. Hindmarsh. *CVODE, a Stiff/Nonstiff ODE Solver in C.* Computers in Physics, 10(2):138-43, March-April 1996.