Recent Advanced Languages and Tools for Hybrid Systems

K.L. Man*, T. Krilavičius†, Kaiyu Wan‡

Abstract—**Hybrid systems exhibit both discrete and continuous behavior and thus are notoriously heterogeneous and complex. Over years, there are abundant tools for simulation and verification of hybrid systems. The goal of this paper is to review existing tools as well as presenting recent developed tools for simulation and verification of hybrid systems through classical examples in hybrid academia. Specifically, we use *Bouncing Ball*, *Tank* and *Thermostat* as three examples to illustrate simulation tools of BHPC and Hybrid Chi formalism. In a similar way, classical hybrid system examples (e.g. bouncing ball and tank) are used to illustrate the applicability of the verification tools of KeYmaera, HySAT and iSAT. Afterwards we give a comparative summary for these tools.**

Keywords: hybrid systems, simulation, verification, tools

1 Introduction

Hybrid systems are systems that exhibit both discrete and continuous behavior. Such systems have proved fruitful in a great diversity of engineering application areas including air-traffic control, automated manufacturing, chemical process control and system control. Also, hybrid systems are notoriously heterogeneous and complex. Rapid software/hardware development cycle increased demand for the advanced design and implementation methods. Over the years, formal methods have been put forward as a tool for modeling and analyzing hybrid systems. Usage of formal semantics and syntax allows unambiguous specifications of the systems, and in such a way provides means for rigorous analysis of correctness and performance properties. Several reasons generated much interest in formal techniques.

- Unambiguous models. Formal modeling languages allow defining systems unambiguously, because syntax and semantics are defined formally, and that includes means to define non-deterministic and stochastic behavior precisely, too. Moreover, for the same reasons, unambiguous refinement and code generation techniques can be applied.

- Strict analysis techniques. Because models are defined using languages with strict semantics, rigorous reasoning about models is possible. Model checking, theorem proving and specifically designed algorithms, e.g. for stability analysis [26], can be used.

There are several types of formalisms for specification as shown below.

- Algebraic specifications : Algebraic specification [8] is a formal process of refining specifications to systematically develop more efficient programs. Over the years, through the novel language constructs and well-de fined formal semantics, several hybrid process algebras with efficient algorithmic differentiation equation solvers [12], hybrid Hybrid Chi [7] and Behavioral Hybrid Process Calculus [25] have been developed. They can be effectively used to formally specify hybrid systems. Diverse case studies (e.g. [30], [31],) show that process algebraic formalisms and their tool-sets can be effectively applied for the formal modeling and analysis of the behavior of hybrid systems.

- Automaton based specifications: The automaton based specification is a popular formalism that is used for representing both discrete and continuous processes of hybrid systems within a unified framework. Languages that are tailored for describing models of hybrid systems have been proposed in the past, some of which have become quite popular. Many of these languages come with their own set of analysis tools. Early work on formal models for hybrid systems includes phase transition systems [1] and hybrid automata [5],[15]. These models were further generalized with the introduction of compositionality of parallel hybrid components in hybrid I/O automata [27] and hybrid modules [4], [28].

- Some other specifications : In the past, some researchers investigated the languages and tools of

*Xi'an Jiaotong-Liverpool University (XJTLU), 111 Ren'ai Road, Suzhou, Jiangsu 215123, China. E-mail: ka.man@xjtlu.edu.cn. Tel: +86 512 8816 1509. Fax: +86 512 8816 1899.

†Baltic Institute of Advanced Technologies (BPTI), Vilnius, Lithuania. E-mail: t.krilavicius@gmail.com.

‡Xi'an Jiaotong-Liverpool University (XJTLU), 111 Ren'ai Road, Suzhou, Jiangsu 215123, China. E-mail: kaiyu.wan@xjtlu.edu.cn.

combining process algebras with automata modeling. By this combination, users can not only reap on the expressive power of hybrid automata but also the rigorous proof process provided by the hybrid process algebra [33], [36], [43].

Computer simulation is a powerful tool for analyzing and optimizing real-world systems with a wide range of successful applications. Simulation substitutes extensive testing after manufacturing and, as such, it can reduce design costs and time. It provides an appealing approach for the analysis of dynamic behavior of processes and helps decision makers identify different possible options by analyzing enormous amounts of data. The design of hybrid systems is no exception and the most used and popular tools are indeed simulation based. In this domain, there are strong industrial offerings that are widely used such as Simulink/Stateflow tool-set [34] , Modelica [14, 42], and tools developed in the academic society including HyVisual [19], Scicos [35], Shift [10], [11], and Charon [3]. Since those tools are illustrated and compared thoroughly in [9], we will describe the recently developed tools such as BHPC [25] and Hybrid Chi [29] in this paper.

Formal verification is very appealing as a concept since it avoids the pitfalls of simulation that cannot guarantee design correctness. Formal verification is intended to prove that some properties hold for all admitted modes of operation of the system under analysis. Over the decades, there are abundant verification tools for hybrid systems including HyTech [18], Masaccio [16], CheckMate [41], PHAVer [13], HSolver [40], d/dt [2] etc. This paper will focus on the recent developed verification tools for hybrid systems: KeYmaera [23], HySAT [20] and iSAT [21].

The goal of this paper is to review existing tools as well as presenting recent developed tools for simulation and verification of hybrid systems through classical examples in hybrid academia. Specifically, we use *Bouncing Ball*[25, pp. 12, 86], *Tank* [25, pp. 15] and *Thermostat* [25, pp. 13] as three examples to illustrate simulation tools of BHPC and Hybrid Chi formalism. In a similar way, classical hybrid system examples (e.g. bouncing ball and water tank) are used to illustrate the applicability of the verification tools: KeYmaera, HySAT and iSAT.

The structure of this paper is as follows. We present simulation of *Bouncing Ball*, *Tank* and *Thermostat* Bhave and Hybrid Chi simulators in the second section. In the third section verification tools KeYmaera, HySAT and iSAT are used to verify several properties of classical hybrid system examples. We conclude by giving a comparative summary for these tools.

2 Simulation of Hybrid Systems

In this section we present several hybrid system simulation examples.

2.1 Selected Examples

Example 2.1 (Bouncing Ball)**.** Bouncing ball is a common example of hybrid process algebra systems. The system [25, pp. 12, 86] consists of one ball and a ground plane. The ball in the system is defined by its altitude h, vertical speed v and the constant c, which describes the energy that is lost on every bounce. Also, the ball is constantly affected by the gravitational acceleration $g = 9.81$.

Example 2.2 (Tank)**.** The fluid level in a tank is controlled through a monitor, which continuously senses the fluid level and turns a pump on and off. The fluid level changes as a piecewise linear function over time. When the pump is off, the fluid level, denoted by a variable y, falls by 2 units per second; when the pump is on, the fluid level rises by 1 units per second. It is required to keep the fluid level between 1 and 12 units. The pump receives a signal from a monitor delayed by 2 time units. Thus, the signals to turn the pump on and off should be sent before the threshold is reached.

See detailed description in [25, pp. 15]

Example 2.3 (Thermostat)**.** A thermostat is one of the best-known introductory examples of hybrid systems. The room temperature is controlled by a thermostat, which continuously senses the temperature and switches a heater on and off. The temperature changes are defined by the ordinary differential equations. When the heater is off, the temperature decreases according to the exponential function $l(t) = \theta e^{Kt}$, where t is time, l is the temperature in the room, θ is the initial temperature, and K is a constant determined by the room. When the heater is on, the temperature increases according to the function $l(t) = \theta e^{-Kt} + h(1 - e^{-Kt})$, where h is a constant that depends on the power of the heater. The temperature should be maintained between $temp_{\min}$ and $temp_{\max}$. Temperatures $temp_{\text{on}}$ and $temp_{\text{off}}$ are the minimal and maximal thresholds, when the heater can be turned on and off, respectively.

See [25, pp. 13] for more details.

2.2 Bhave Toolset

2.2.1 Examples

Example 2.4 (Bouncing Ball)**.** Formal specification of Bouncing ball from Example2.1 in BHPC is rather simple.

```
BouncingBall(h, v) =
     [  h' = v ;     // dh/dt = v
        v' = -9.81   // dv/dt = -g
      | h ]          // when h = 0, stop
     . bounce{v' = 0;} // bounce action
```

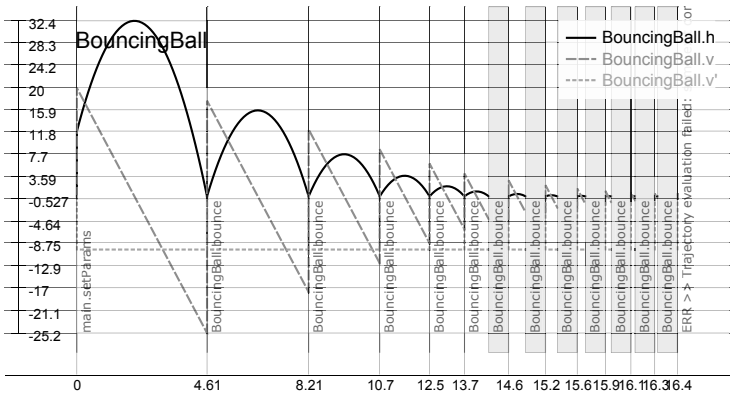

Figure 1: Simulation of the bouncing ball, Example 2.4.

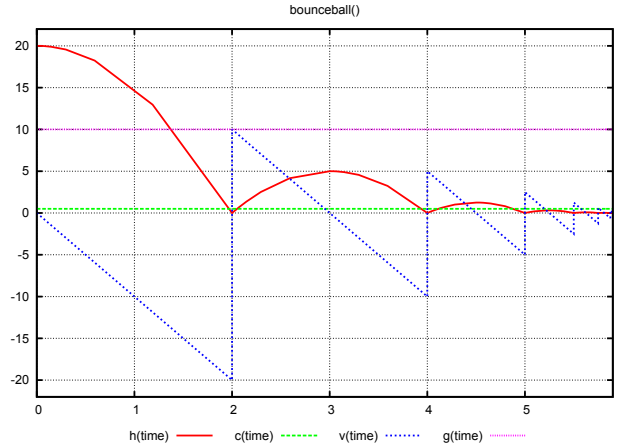

Figure 2: Simulation of the bouncing ball, Example 2.7.

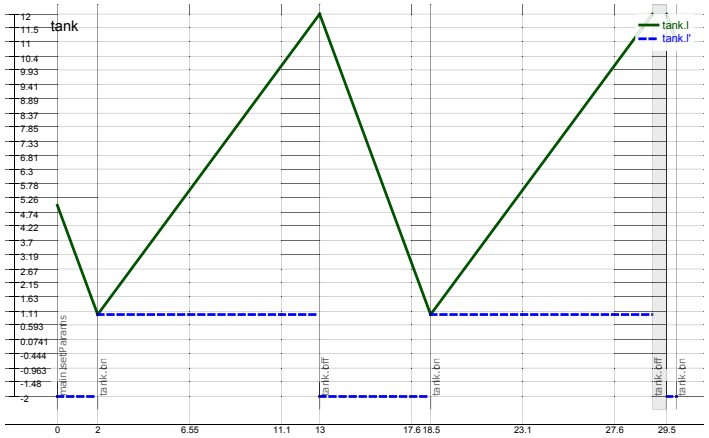

Figure 3: Simulation of the tank, Example 2.5.

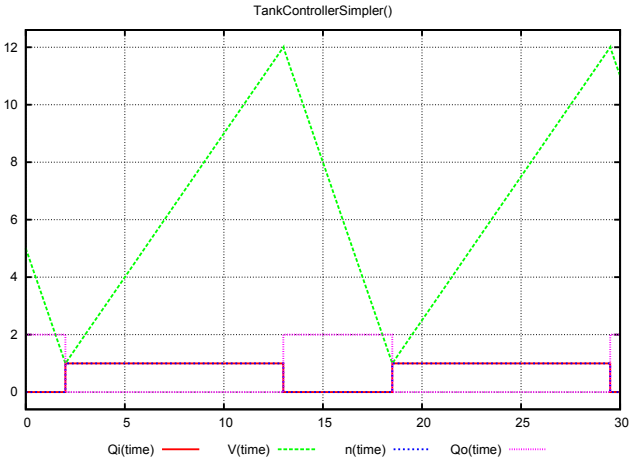

Figure 4: Simulation of the tank, Example 2.8.

```
      // bounce with 0.7 velocity loss
     . BouncingBall(h, -0.7 * v);

// set simulation parameters
main() = setParams{step=1e-1}
      // invoke the bouncing process
     .BouncingBall(12, 20);
```

The system consists of two processes:

- `BouncingBall` process defines the trajectory and the *bounce* action of the ball. The motion is described by the derivative of the altitude, which is the vertical speed v. The speed is affected by the acceleration $\dot{v} = -g$. This motion is executed until the ball touches the ground plane ($h = 0$) and a discrete *bounce* action is executed. As the ball bounces, a fraction $c = 0.7$ of its energy is lost, and the ball changes the direction upwards.
- The process `main` is the simulation entry point. A discrete action *setParams* is invoked, which changes the parameters for the simulation – the parameter *"step"* defines the integration interval length for the DAE solver. Then, a `BouncingBall` process is invoked with the initial parameters $h = 12$ and $v = 20$.

Figure 1 displays the results of bouncing ball simulation. It shows the speed and altitude change over time together with the discrete action `bounce`. Visualization of the model evolution is generated by using a prototype *Message Sequence Plot* (MSP) visualization application [24, 32, 39], proposed in [25, pp. 120-123]. MSP displays the changes of the system's process variables together with the actions that are performed.

The energy of the system is reduced by a fraction in every bounce, which results in a shorter time span with every bounce. This leads to *Zeno* behavior [25, p. 124], where the system tries to execute an infinite amount of bounces in a finite amount of time. The simulator prevents such behavior by forcing a small fraction of the initial integration interval to simulate regardless of the trajectory exit conditions and checks the result after the first step. Such

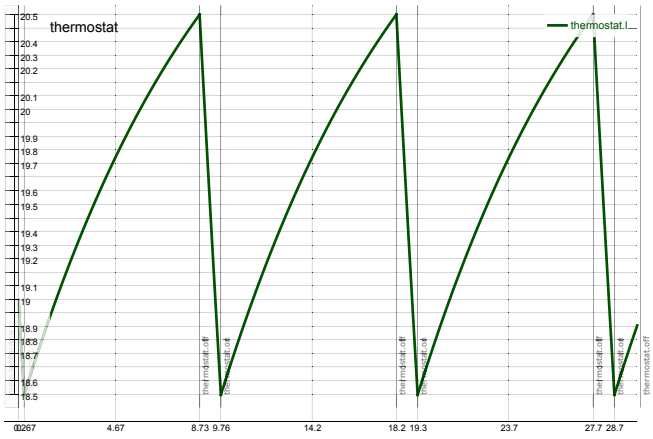

Figure 5: Simulation of the thermostat, Example 2.6.

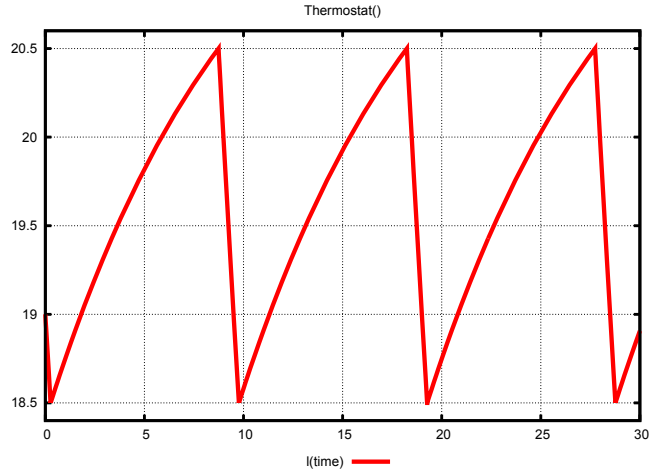

Figure 6: Simulation of the thermostat, Example 2.9.

situation can be seen in the *16.4 second* of the simulation, when the error is printed.

Example 2.5 (Tank)**.** Formal specification of Tank from Example 2.2 in BHPC.

```
const L_min     = 1;
const L_max       = 12;
const Inflow       = 1;
const Outflow       = -2;

tank(l) = [ l' = Outflow | l - L_min ]
          . on
          [ l' = Inflow | L_max - l ]
          . off
          . tank(l);

main() = setParams{step=1e-2;tStop=30}.
         tank(5);
```

See simulation results in Figure 3

Example 2.6 (Thermostat)**.** BHPC specification of Thermostat from Example 2.3. Temperature is maintained between $temp_{\text{on}}$ (`tempOn`) and $temp_{\text{off}}$ (`tempOff`).

```
const K = 0.1;
const H = 22;
const tempOff = 20.5;
const tempOn  = 18.5;

thermostat(l)
  = [ l' = 0 - K * l | l - tempOn ]
  . on
  . [ l' = K  * ( H - l ) | ( tempOff - l ) ]
  . off
  . thermostat(l);

main() = setParams{step=1e-1;tStop=30}
       . thermostat(19);
```

See simulation results in Figure 5

2.3 Simulation of Hybrid Chi: Hybrid Chi Python Simulator

2.3.1 Examples

Example 2.7 (Bouncing Ball)**.** Formal specification of Bouncing ball (Example 2.1) from `http://se.wtb.tue.nl/sewiki/chi/hybrid_examples/bouncing_ball` in Hybrid Chi is the following.

```
model bounceball()=
|[ cont h: real= 20.0
 , cont v: real= 0.0
 , var c : real= 0.5
 , var g : real= 10.0
:: eqn dot h = v
     , dot v = -g
||*( h <= 0.0 -> v:= -c*v; v <= 0.0 -> skip)
]|
```

Simulation results are depicted in Figure 2.

Example 2.8 (Tank)**.** Formal specification of Tank from Example 2.2 in Hybrid Chi specified in CIF [6].

```
model TankControllerSimpler() =
|[ extern var V: cont real = 5
            ; Qi, Qo: alg real
            ; n: disc nat = 0
:: |( mode physics = inv dot V = Qi - Qo
                       &    Qi = n
                       &    Qo = -2 * (n-1)
   :: physics
   )|
||
   |( mode closed = when V <= 1
```

```
                    now do n := 1 goto opened
        , opened = when V >= 12
                    now do n := 0 goto closed
   :: closed
   )|
]|
```

See simulation results in Figure 4

Example 2.9 (Thermostat)**.** Formal specification of Tank from Example 2.3 in CIF [6].

```
model Thermostat() =
|[ extern var l: cont real = 19
           ; n: disc nat  = 0
:: |( mode physics = inv
                dot l = 0.1 * (22  * n - l)
   :: physics
   )|
||
   |( mode off = when l <= 18.5
                now do n := 1 goto on
         , on  = when l >= 20.5
                now do n := 0  goto off
   :: off
   )|
]|
```

See simulation results in Figure 6

3 Verification of Hybrid Systems

As mentioned previously, modeling and verification of hybrid systems are complicated by their heterogeneous nature as well as their sheer complexity. Existing modeling techniques for hybrid systems rely upon semantics to represent the relationship between the discrete and continuous features of a hybrid system.

However, modeling and analyzing a hybrid system with all of its details always results in state explosion. Nevertheless, over the years, various techniques, algorithms, specification logics and software tools have been developed for simplifying hybrid system models to achieve certain verification goals.

For illustration purpose, in this section, we present three recent developed verification tools for hybrid systems: KeYmaera, HySAT and iSAT along with some classical hybrid system examples from literature. It is worth mentioning that some materials presented in this section are taken from [23, 20, 21].

3.1 KeYmaera

KeYmaera is an automated and interactive theorem prover for a natural specification and verification logic for hybrid systems. KeYmaera supports differential dynamic logic (dL) and program notation for hybrid automata. KeYmaera is particularly suitable for verifying parametric hybrid systems and for verifying collision avoidance in case studies from train control [38] and air traffic management [37].

More precisely, KeYmaera is a verification tool for hybrid systems and built as a hybrid theorem prover for hybrid systems. KeYmaera separates the overall verification work flow into two phase. In the first phase one specifies the hybrid system that one would like to verify along with its correctness properties. In the second phase, one can use KeYmaera and its automatic proof strategies to verify the specified property of the hybrid system.

In KeYmaera, the behavior of hybrid systems can be specified in a program notation called hybrid program with the following syntax:

- `a ::= a; b` Sequential composition that does `a` first and then `b`, where `a` and `b` are typical names.
- `x:=t` Discrete assignment/jump assigning the value of `t` to variable `x`.
- `x:=*` Random assignment assigns any real value to variable `x` non-deterministically.
- `if(F) then a fi` If-then statement, performs `a` if `F` holds and does nothing otherwise, where `F` is a formula of (possibly non-linear) real arithmetic (possibly including quantifiers).
- `if(F) then a else b fi` If-then-else statement, performs `a` if `F` holds and performs `b` otherwise.
- `?F` State assertion testing whether formula `F` is true in current state (otherwise abort).
- `a ++ b` Non-deterministic choice following either `a` or `b`.
- `while(F) a end` While loop, repeats `a` as long as `F` holds, stops before doing `a` only if `F` is false (`a` will not be stopped in the middle just because `F` becomes false at some intermediate state during `a`).
- `a*` Non-deterministic repetition, repeating `a` arbitrarily often including 0 times.
- `{x'=t,y'=s, F}` Continuous evolution along differential equation system with terms `t,s`, optionally with domain of maximum evolution or invariant region `F`, where `x` and `y` are variables; and `x'` denotes the time time-derivative of `x`. This domain constraint `F` needs to be true at every time during the evolution, otherwise the system needs to stop following this continuous mode and move on. One

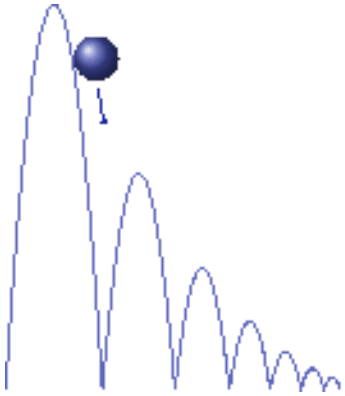

Figure 7: Bouncing ball.

can use systems of differential equations, differential-algebraic equations, differential inequalities, and differential equations with disturbances.

- `{x'=t,y'<=s,y'>=r, F}` Continuous evolution along system of differential equations and differential inequalities with terms `t,s,r`, optionally with domain of maximum evolution or invariant region `F`. The time-derivative of `y` needs to stay within `r` and `s` all the time.
- `{\exists R u; (x'=t & y'=s & F)}` Continuous evolution along system of differential-algebraic equations with disturbance `u` (which may occur in the terms `t,s` and formula `F`), optionally with domain of maximum evolution or invariant region `F`.
- `R x` Variable declaration, declaring `x` as a real variable (either a state variable or auxiliary variable).
- `R x, y, z` Variable declaration, declaring `x, y` and `z` as real variables.

The rest of this section presents two hybrid system examples that are simple enough to be handled and complex enough to expose the strength of the verification tool KeYmaera.

3.1.1 Bouncing Ball

A slight variant of the Bouncing ball example presented in Section 2.1 - a ball falls from height h and bounces back from the ground ($h = 0$) after an elastic deformation (see Figure 7). The current speed of the ball is denoted by v, and t is a clock measuring the falling time. We assume an arbitrary positive gravity force g and that the ball loses energy according to a damping factor $0 \leq c < 1$.

The ball loses energy at every bounce, thus the ball never bounces higher than the initial height. This can be expressed by the safety property $0 \leq h \leq H$, where H denotes the initial energy level, i.e., the initial height if $v = 0$. The above problem specification states that the ball never bounces higher than that, i.e., it always remains within the region $0 \leq h \leq H$ when it starts jumping in an appropriate state.

The hybrid program of the bouncing ball is given below (note that the block `\problem{...}` is used to introduce a problem specification for verification in KeYmaera):

```
\problem {

/*
 * h = height
 * v = velocity
 * H = height limit
 * g = gravitation
 * c = elastic dampening factor
       at floor (h=0)
*/

/* state variable declarations */
\[ R h,v,t; R c,g,H \] (
    /* initial state characterization */
    g>0 & h>=0&t>=0 & 0<=c&c<1 &
    v^2<=2*g*(H-h) & H>=0
  ->     /* implication */
  \[    /* system dynamics */
    (
      /* falling/jumping */
      {h'=v,v'=-g,t'=1, h>=0};
      if (t>0 & h=0) then /* if on ground */
        v := -c*v;        /* bounce back  */
            t := 0
      fi.
     )*       /* repeat these transitions */
     /* safety / postcondition */
  \] (0<=h & h<=H)
)
}
```

KeYmaera was applied successfully to verify the safety property (i.e. `0<=h & h<=H`) of the hybrid program describing the behavior of the bouncing ball. This also concludes that the bounces of the ball always remain within the expected region.

3.1.2 Water Tank

A slight variant of the Tank example presented in Section 2.1 - a water tank (see Figure 8) regulates water level y between 1 and 12 by filling or emptying the water tank. The hybrid automaton based specification of the water tank is given in Figure 9 (note that x denotes the outgoing water level).

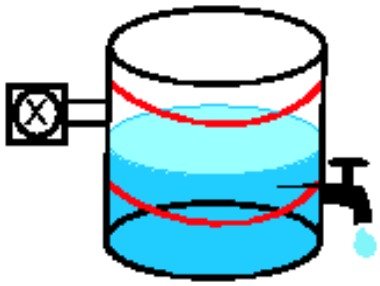

Figure 8: Water tank.

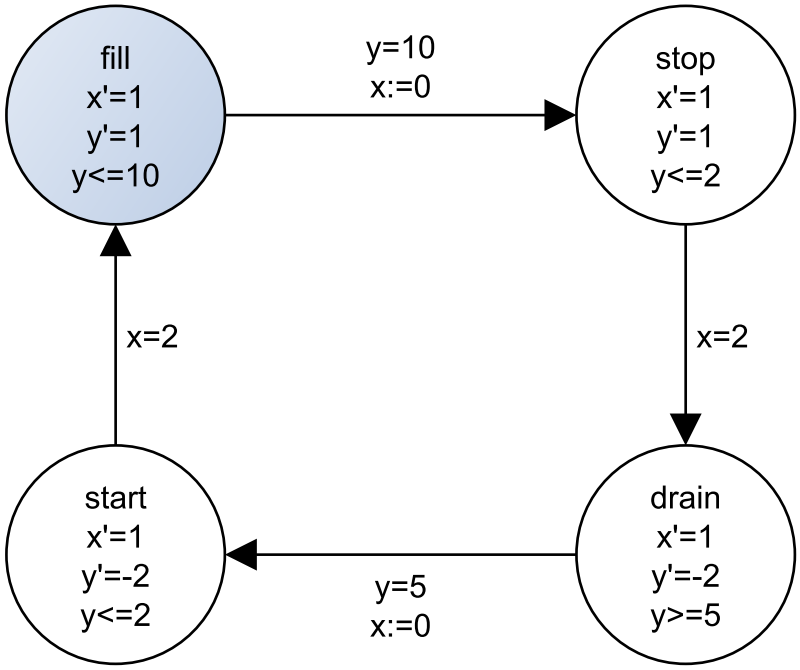

Figure 9: Hybrid automaton based specification of the water tank.

The hybrid program of the water tank is given below:

```
\problem {
  /* state variable declarations */
  \[ R y, x, st; x:=0; y:=1; st:=0 \] (
  /* initial state characterization */
  st = 0
  ->    /* implication */
    \[  /* system dynamics */
      ( /* repeat the following discrete or
           continuous transitions */
        (?(st=0);
              (?(y=10); x:=0; st:=1)
           ++ {x'=1,y'=1, y<=10}
        )
      ++ (?(st=1);
              (?(x=2); st:=2)
           ++ {x'=1,y'=1, x<=2}
        )
      ++ (?(st=2);
              (?(y=5); x:=0; st:=3)
           ++ {x'=1,y'=-2, y>=5}
        )
      ++ (?(st=3);
              (?(x=2); st:=0)
           ++ {x'=1,y'=-2, x<=2}
        )
      )*@invariant(1<=y & y<=12
          & (st=3-> (y>=5-2*x))
          & (st=1 -> (y<=10+x)))
       /* safety / postcondition */
    \] (1<=y & y<=12)
    )
}
```

KeYmaera was applied successfully to verify the safety property (i.e. `1<=y & y<=12`) of the hybrid program describing the behavior of the water tank. This also concludes that the water level y is always between 1 and 12 by filling or emptying the water tank.

3.2 HySAT and iSAT

This section briefly presents two bounded model checkers for hybrid systems.

3.2.1 Bounded Model Checking for Hybrid Systems

Bounded model checking (BMC) aims at checking whether a hybrid system has a run of bounded length k which

- starts in an initial state of the hybrid system;
- obeys the hybrid system transition relation;
- ends in a state in which a certain (desired or undesired) property holds.

The goal of BMC is to construct a formula which is satisfiable if and only if a trace with above properties exists. In case of satisfiability, any satisfying valuation of this formula corresponds to such a trace.

3.2.2 HySAT

HySAT is a satisfiability checker (SAT) for Boolean combinations of arithmetic constraints over real and integer valued variables. A peculiarity of HySAT, which sets it apart from many other solvers, is that it is not limited to linear arithmetic, but can also deal with nonlinear constraints involving transcendental functions. The algorithmic core of HySAT is a tight integration of recent SAT solving techniques with interval-based arithmetic constraint solving. For technical details, see [20].

3.2.3 iSAT

iSAT is the successor tool of HySAT which was developed to facilitate automated reasoning about large Boolean combinations of non-linear arithmetic constraints involving transcendental functions. The algorithmic core of iSAT is based on a tight integration of recent DPLL-style SAT solving techniques (like lazy clause evaluation, conflict-driven learning, non-chronological backtracking, and restarts) with interval-based arithmetic constraint solving. For technical details, see [21].

3.2.4 Outline of Verified Examples

A series of benchmark examples of hybrid systems including the bouncing ball and railroad crossing was carried out to evaluate the performances of HySAT and iSAT. In addition to provide counterexamples which are a salient feature of model checking that help users to understand the problem in a faulty design, HySAT and iSAT can be used to search for a counterexample in executions whose length is bounded by some integer k. If no bug is found then k is increased until either a bug is found, the problem becomes intractable or some pre-known upper bound is reached.

For instance, in the bouncing ball example, HySAT and iSAT can be applied to find an initial hight and an initial velocity so that the ball hits the ground which is located at a certain distance from the starting point. They can also be used in the railroad crossing example to verify the property: "*When the train is within 10 meters to the gate, the gate is always fully closed*" and to determine in quantity that the violations are possible for certain time delay of controller and the trace length of the verification/execution.

4 Summary and Future Work

In this paper we review existing tools as well as presenting recent developed tools for simulation and verification of hybrid systems through classical examples in hybrid academia. Below is a comparative summary for these tools we present in this paper.

- *Behavioural Hybrid Process Calculus (BHPC)* is a hybrid process algebra which was specifically designed for the description of the dynamic behavior of hybrid systems along with a powerful simulator called Bhave toolset. Currently, simulation results obtained by means of the BHPC simulator can also be visualized and analyzed via Message Sequence Plots (MSP).
- The *Hybrid Chi* formalism is suited to modeling, simulation and verification of hybrid systems. The semantics of Hybrid Chi is defined by means of deduction rules (in SOS style) that associate a hybrid transition system with a Hybrid Chi process. The Hybrid Chi formalism integrates concepts from dynamics and control theory with concepts from computer science, in particular from process algebra and hybrid automata. Its 'consistent equation semantics' enforces state changes to be consistent with delay predicates, that combine the invariant and ow clauses of hybrid automata.
- *KeYmaera* is an automated and interactive theorem prover for a natural specification and verification logic for hybrid systems. KeYmaera supports differential dynamic logic (dL), and is particularly suitable for verifying parametric hybrid systems and for verifying collision avoidance in case studies from train control [38] and air traffic management [37].
- *HySAT* is a satisfiability checker for Boolean combinations of arithmetic constraints over real and integer valued variables. One of important features of HySAT is that it is not limited to linear arithmetic, but can also deal with nonlinear constraints involving transcendental functions [17].
- *iSAT* is the successor tool of HySAT which was developed to facilitate automated reasoning about large Boolean combinations of non-linear arithmetic constraints involving transcendental functions.

Hybrid systems are notoriously heterogeneous, complex and with different characteristics. These characteristics necessitate the analyze and compare the available tools. Therefore users may choose the appropriate tool for their specific investigated system.

References

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H, X. Nicollin, J. Sifakis, and S. Yovine, The algorithmic analysis of hybrid systems, Theoretical Computer Science, February 1995, vol. 138, pp. 3-34.

[2] E. Asarin, T. Dang, and O. Maler, d/dt: A verification tool for hybrid systems, in Proc. of the 40th IEEE Conf. on Decision and Control, 2001, pp. 2893-2898.

[3] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee, Modular specification of hybrid systems in Charon, in Proc. of the Third Intl. Work. on Hybrid Systems: Computation and Control, 2000, pp. 6-19.

[4] R. Alur and T. A. Henzinger, Modularity for timed and hybrid systems, in CONCUR 97: Eight International Conference on Concurrency Theory, 1997, pp. 74-88.

[5] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. IEEE Transactions on Software Engineering, 22(3):pp. 81-201, 1996.

[6] D.A. van Beek, P. Collins, D.E. Nadales Agut, J.E. Rooda, and R.R.H. Schiffelers, New concepts in the abstract format of the compositional interchange format, 3rd IFAC Conference on Analysis and Design of Hybrid Systems, 2009, Zaragoza, Spain, pp. 250-255.

[7] D. A. van Beek, K. L. Man, M. A. Reniers, J. E. Rooda and R. R. H. Schiffelers. Syntax and Consistent Equation Semantics of Hybrid Chi, Journal of Logic and Algebraic Programming, 2006, vol. 68, number 1-2, pp. 129-210.

[8] Bergstra, J. A., and B. Mahr , Algebraic Specification, Academic Press, 1989, ISBN 0-201-41635-2.

[9] L.P. Carloni, R. Passerone, A. Pinto, and A.L. Sangiovanni-Vincentelli, Languages and Tools for Hybrid Systems Design, Foundations and Trends in Electronic Design Automation Vol. 1, No 1/2 (2006) pp. 1-193

[10] A. Deshpande, A. Gollu, and P. Varaiya, Shift: A formalism and a programming language for dynamic networks of hybrid automata, in Hybrid Systems IV, 1997, pp. 113-134.

[11] A. Deshpande, A. Gollu, and P. Varaiya, The SHIFT programming language for dynamic networks of hybrid automata, IEEE Transactions on Automatic Control, April 1998, vol. 43, pp. 584-597.

[12] Emmanuel M. Tadjouddine, On the Application of Algorithmic Differentiation to Newton Solvers, Lecture Notes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists 2010, IMECS 2010, 17-19 March, 2010, Hong Kong, pp. 1342-1347.

[13] G. Frehse, PHAVer: Algorithmic verification of hybrid systems past HyTech, in HSCC, 2005, pp. 258-273.

[14] P. Fritzson, Principles of Object-oriented Modeling and Simulation with Modelica 2.1, J. Wiley & Sons, 2004.

[15] T.A. Henzinger, The theory of hybrid automata, In the Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science (LICS 96), 1996, pp. 278-292.

[16] T. A. Henzinger, Masaccio: A formal model for embedded components, in TCS 00: Theoretical Computer Science, 2000, pp. 549-563.

[17] Christian Herde, HySAT Quick Start Guide, December 20, 2009, available at http://hysat.informatik.uni-oldenburg.de/.

[18] T. Henzinger and P. H. Ho, HyTeCh: The Cornell hybrid technology tool, in Hybrid Systems II, 1995, pp. 265-293.

[19] C. Hylands, E. A. Lee, J. Liu, X. Liu, S. Neuendorffer, and H. Zheng, HyVisual: A hybrid system visual modeler, Tech. Rep. UCB/ERL M03/1, UC Berkeley, 2003, available at http://ptolemy.eecs.berkeley.edu/hyvisual/.

[20] Martin Franzle and Christian Herde, HySAT: An efficient proof engine for bounded model checking of hybrid systems, Formal Methods in System Design, 2006.

[21] Martin Franzle, Christian Herde, Tino Teige, Stefan Ratschan, and Tobias Schubert, Efficient Solving of Large Non-Linear Arithmetic Constraint Systems with Complex Boolean Structure. Journal on Satisfiability, Boolean Modeling, and Computation, Volume 1 (2007), pp. 209-236.

[22] KeYmaera: A Hybrid Theorem Prover for Hybrid Systems, available at http://symbolaris.com/info /KeYmaera.html/.

[23] Andre Platzer and Jan-David Quesel, KeYmaera: A hybrid theorem prover for hybrid systems, Lecture Notes in Engineering and Computer Science : Proceedings of Automated Reasoning, Third International Joint Conference, IJCAR 2008, Sydney, Australia, pp. 171-178.

[24] T. Krilavičius and K.L. Man, Intelligent Automation and Computer Engineering, chapter Behavioural Hybrid Process Calculus for Modelling and Analysis of Hybrid and Electronic Systems, 2009.

[25] T. Krilavčius, Hybrid Techniques for Hybrid Systems, PhD thesis, Univ. of Twente, 2006.

[26] Langerak, R., Polderman, J. W., and Krilavičius, Stability analysis for hybrid automata using conservative gains, In Proceedings of Conference on Analysis and Design of Hybrid Systems (ADHS 03), 2003, pp. 377-382.

[27] N. Lynch, R. Segala, F. Vaandrager, and H. Weinberg, Hybrid I/O automata, in Hybrid Systems III: Verification and Control, 1996, pp. 496-510.

[28] J. Lygeros, C. Tomlin, and S. Sastry, Controllers for reachability specifications for hybrid systems, in Automatica, Special Issue on Hybrid Systems, 1999.

[29] K.L. Man and R.R.H. Schiffelers, Formal Specification and Analysis of Hybrid Systems, Phd thesis, Univ. of Eindhoven, 2006.

[30] K. L. Man and M. P. Schellekens, Interoperability of Performance and Functional Analysis for Electronic System Designs in Behavioural Hybrid Process Calculus (BHPC), Lecture Notes in Electrical Engineering: Current Trends in Intelligent Systems and Computer Engineering, 2008, Volume 6, pp. 375-394.

[31] K.L. Man, M.P. Schellekens and M. Boubekeur, Formal Specification and Analysis of Analog and Mixed-Signal Circuits Using Process Algebras for Hybrid Systems, the 3rd IEEE International SoC Conference, 2006, Seoul, South Korea.

[32] K.L. Man, T. Krilavičius, C. Chen, and H.L. Leung, Application of Bhave Toolset for System Control and Electronic System Design, Lecture Notes in Engineering and Computer Science: In Proceedings of the International MultiConference of Engineers and Computer Scientists 2010, IMECS 2010, March, 2010, Hongkong, pp. 1336-1341.

[33] K.L. Man, H. Leung, M. Mercaldi, and J. Huang, Performance and functional analysis of tlm models in the SHE methodology, in the IEEE International Conference on Computer Science and Software Engineering, Wuhan, China, 2008.

[34] S. Neema, Analysis of Matlab Simulink and Stateflow data model, Tech. Rep. ISIS 01-204, March 2001, Vanderbilt University, Nashville.

[35] R. Nikoukhah and S. Steer, SCICOS - a dynamic system builder and simulator users guide - version 1.0, Tech. Rep. 0207, June 1997, INRIA, Rocquencourt, France.

[36] PAT: Process analysis toolkit, 2009, available at http://www.comp.nus.edu.sg/pat/.

[37] Andre Platzer and Edmund M. Clarke, Formal verification of curved flight collision avoidance maneuvers: A case study, Lecture Notes in Computer Science, volume 5850, In 16th International Symposium on Formal Methods, FM, Eindhoven, Netherlands, 2009, pp. 547-562.

[38] Andre Platzer and Jan-David Quesel, European Train Control System: A case study in formal verification, Lecture Notes in Computer Science volume 5885, In the 11th International Conference on Formal Engineering Methods, ICFEM, Rio de Janeiro, Brasil, 2009, pp. 246-265.

[39] Prototype application for Message Sequence Plot visualisation, March 2010.

[40] S. Ratschan and Z. She, Safety verification of hybrid systems by constraint propagation based abstracition refinement, in HSCC, 2005, pp. 573-589.

[41] B. I. Silva, K. Richeson, B. Krogh, and A. Chutinan, Modeling and verifying hybrid dynamic systems using CheckMate, in Proceedings of 4th International Conference on Automation of Mixed Processes, September 2000, pp. 323-328.

[42] M. M. Tiller, Introduction to physical modeling with Modelica, Kluwer Academic Publishers, 2001.

[43] Tim A.C. Willemse, Embeddings of Hybrid Automata in Process Algebra, Lecture Notes in Engineering and Computer Science volume 2999, IFM 2004, pp. 343-362.