Simulation and Verification of Hybrid Systems using HySon*

Olivier Bouissou†, Samuel Mimram*, Alexandre Chapoutot‡

0.1, 2014-02-07

Abstract

In an industrial setting, control-command systems are usually validated using numerical simulation instead of formal verification as proposed by many academic tools. In this paper, we present a tool named HySon that tries to fill the gap between formal methods and industrial usage. HySon takes as input a dynamical system described by a Simulink model and proposes a new simulation engine that safely computes flowpipes of the system variables by adapting the numerical simulation algorithms to make them safely propagate sets of values instead of floating-point numbers. We show how the tool runs and give some results on small yet challenging examples.

A wide range of academic tools is available in order to ensure the safety of hybrid systems, which are based on various techniques: computation of reachable sets [11, 8], barrier certificates [20, 19, 17], predicate abstraction [1, 23], etc. However, in an industrial setting, control-command systems are usually validated using numerical simulation, and formal techniques are not used much. We see two reasons for this. First, the above mentioned tools are very often based on a representation of hybrid systems as hybrid automata (or an equivalent model), while industrial control-command systems are generally first described in Simulink or Modelica, which offer a rich set of blocks in order to build hybrid systems: the translation from Simulink to hybrid automata is not an easy task [22], and some important features of Simulink such as saturation or lookup tables are difficult to express in the higher-level model of hybrid automata. Second, control-command systems are very peculiar hybrid systems, in which discrete actions can be of two kinds: either event-triggered as in hybrid automata (the *zero-crossing events* in Simulink) or time-triggered when some part of the system has a discrete sampling rate. In this latest case, which often occurs in control-command systems, the translation into the formalism of hybrid automata can be performed, but causes an explosion of the number of discrete transitions, as shown in [3].

*This work was partially supported by the ANR project CAFEIN.

†CEA, LIST, Gif-sur-Yvette, France. {olivier.bouissou,samuel.mimram}@cea.fr

‡ENSTA ParisTech, Palaiseau, France. alexandre.chapoutot@ensta-paristech.fr

In this paper, we present a tool, named HySon, whose aim is to fill the gap between formal methods and industrial usage. It takes as input a dynamical system described by a Simulink model and offers a simulation engine that safely computes flowpipes of the evolution of the variables of the system. The internal simulation engine is based on an adaptation of classical numerical simulation algorithms in order to make them safely propagate sets of values instead of floating-point numbers. We believe that this approach can help to improve the design process of industrial control-command systems because our engine scales well both in dimension and complexity. In Section 1, we briefly present our simulation engine, in Section 2 we describe how our tool can be used and give some examples of systems on which we successfully applied it, and we conclude in Section 3.

1 Set-based simulation in a nutshell

In this section, we briefly present the theoretical foundations of our simulation engine. Its main particularity is that it performs *set-based simulation*: it uses techniques borrowed and adapted from usual numerical simulation engines, but propagates sets of values during the simulation. There are three main design choices that have to be performed in order to elaborate such a simulation system: one has to choose a representation for sets of values, a technique to handle continuous evolutions of the system, and finally a way to handle discrete jumps. We detail how these were implemented in HySon in the rest of this section. For more details on these techniques, we refer the reader to [6, 7].

Representation of sets. In order to represent sets of reals, we use *affine arithmetic* [9]: it is a refinement over interval arithmetic [18] which is able to keep track of linear correlations between variables. A set of values is represented by an *affine form* $\hat{x}$, which is a formal expression of the form

$$\hat{x} \quad = \quad \alpha_0 + \sum_{i=1}^{n} \alpha_i \varepsilon_i$$

where the coefficients α_i are real numbers, and the ε_i are formal variables ranging over the interval $[-1, 1]$ called *noise symbols*. Usual operations on real numbers can be extend to affine arithmetic in a way that over-approximates the theoretical result on sets of values. Linear operations can be implemented in an exact way, but most operations like multiplication have to be over-approximated and create new noise symbols. Some set-theoretic operations can also be performed: for instance, one can compute the *hull* (or *union*) $\text{hull}(\hat{x}, \hat{y})$ of two affine forms $\hat{x}$ and $\hat{y}$, which is the smallest affine form (or a good over-approximation of it) in which both $\hat{x}$ and $\hat{y}$ are included, see [14] for details.

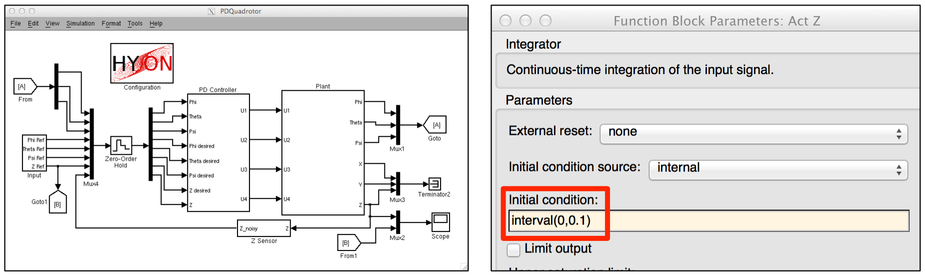
Continuous evolution. Given a differential equation of the form $\dot{\mathbf{x}} = f(\mathbf{x}, u)$, where $\mathbf{x}$ is a vector-valued function of time and u is an input which can be given as an interval. We use numerical schemes such as Euler, RK4 or Bogaki-Shampine to compute numerical approximations of the solution [21]. When the initial condition $\mathbf{x}(0)$ is given by an affine form $\hat{x}_0$, the evaluation of such a scheme using affine arithmetic provides a precise and efficient way of computing *approximations* of the set of possible solutions of the differential equation depending on the precise value of the initial parameters. If one wants instead to compute an *over-approximation* of this set, it is possible to perturb the chosen numerical scheme by over-approximating the error induced by the method using the well-known Picard operator and automatic differentiation, as explained in [5].

Discrete jumps. In Simulink as in other formalisms, discrete jumps can be represented as equations of the form "`on` $(g(\mathbf{x}) = 0)$ `do` $\mathbf{x} = r(\mathbf{x})$" which means that the value of the variables $\mathbf{x}$ has to be reset to $r(\mathbf{x})$ as soon as the condition $g(\mathbf{x}) = 0$ is satisfied. Note that the operation of changing state in an hybrid automaton can be encoded in this way using a discrete variable in $\mathbf{x}$ which encodes the current state. The main difficulty in handling such events consists in detecting the exact time τ and affine form $\hat{x}_\tau$ such that $g(\hat{x}_\tau) = 0$. In order to do so, the technique used in many tools is to continue the simulation in order to obtain two states $\hat{x}_l$ and $\hat{x}_r$ such that $g(\hat{x}_l) < 0$ (i.e. $\forall x \in \hat{x}_l,\ g(x) < 0$) and $g(\hat{x}_r) > 0$, and then use an intersection to obtain a tight enclosure of $\hat{x}_\tau$. In HySon, instead of performing intersection with the hyperplane $g(\mathbf{x}) = 0$ which can be complicated when g is non-linear, we use a guaranteed version of the algorithm used in Simulink [4]: we bisect on the time interval $[t_l, t_r]$ and use an extrapolation polynomial to obtain over-approximation of $\mathbf{x}$ at any time $t \in [t_l, t_r]$.

2 Simulation using HySon

The above described techniques have been implemented in a tool named HySon and applied to hybrid systems written in Simulink. We developed a parser for Simulink models that translate them into an intermediate language of simple equations, as described in [4]. As shown in the following table, our tool supports many Simulink blocks, making it already usable for complex models:

Block type	*Block name*
Continuous	`Integrator`
Discrete	`DiscreteIntegrator`, `UnitDelay`, `ZeroOrderHold`
Signal routing	`Mux`, `Demux`, `From`, `Goto`, `Subsystems`
Mathematics	`Abs`, `Gain`, `MinMax`, `Sum`, `Product`, `Saturate` `Sin`, `Trigonometry`
Inputs	`Clock`, `Step`, `Constant`, `Ground`, `InitialCondition`
Logic	`Logic`, `Compare To Zero`, `Compare To Constant` `Switch`, `RelationalOperator`



Function Block Parameters: Act Z

Integrator

Continuous-time integration of the input signal.

Parameters

External reset: none

Initial condition source: internal

Initial condition: interval(0,0.1)

Limit output

Figure 1: Use of HySon on a Simulink model of a quadrotor system. On the left, the high-level Simulink model and on the right the interval parameter given to the initial position of the quadrotor.

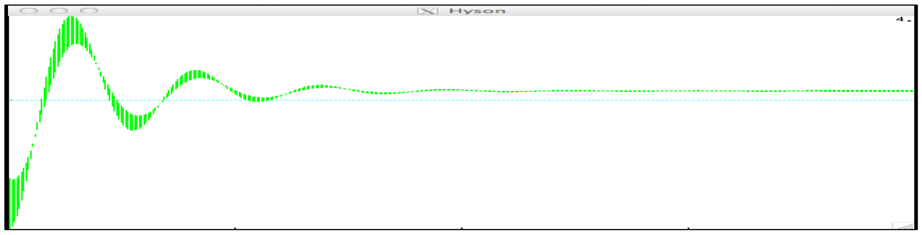


Figure 2: Result of HySon on the quadrotor system.

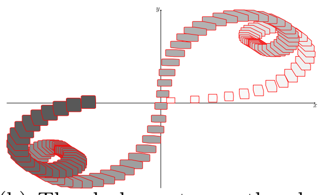
In order to ease the use of HySon for industrial models, we integrated it smoothly into Simulink: we propose a new block, named the `HySon` block, that can be inserted into an existing diagram. As show on Figure 1, the user can then modify some parameters of the model with the keyword `interval(a,b)`: this specifies that the given parameter is unknown but remains in the interval $[a, b]$. For example, initial conditions of `Integrator` blocks can be specified in this way. The simulation of the whole model is then performed by HySon engine instead of Simulink's one. Depending on the chosen mode, the output is a *simulation* of the system (when set-based simulation is used) or an *over-approximation* of the dynamics (when guaranteed set-based simulation is used).

Note that HySon can handle systems with both discrete-time (i.e. sampled) blocks and continuous-time blocks. We use the same step-size selection mechanism as Simulink in order to ensure that we hit each sampling time during our simulation. For example, in the system of Figure 1, the `PD Controller` subsystem is discrete, with a sampling time of 0.01 seconds while the `Plant` subsystem is continuous. Note also that continuous blocks are allowed to define zero-crossing events that are handled as described in Section 1.

Being based on numerical simulation methods, our tool handles systems with both non-linear dynamics and non-linear guards. It also scales well to high-dimensional problems, at least in the *simulation* mode. For example, the system shown in Figure 1 is a model of a proportional-integral controller for

$$\begin{array}{rcl}\dot{x} = & v \cdot \cos(\delta) \cdot \cos(\theta), & x(0) \in [0,1] \\ \dot{y} = & v \cdot \cos(\delta) \cdot \sin(\theta), & y(0) \in [0,1] \\ \dot{\theta} = & 0.2 \cdot v \cdot \sin(\delta), & \theta(0) \in [0,0.01] \\ \dot{v} = & 0, & v(0) = 7 \\ \dot{\delta} = & 0.2, & \delta(0) = 0\end{array}$$

(a) Car model



(b) The darker sets are the closest to the simulation end.

Figure 3: Non-linear system simulation

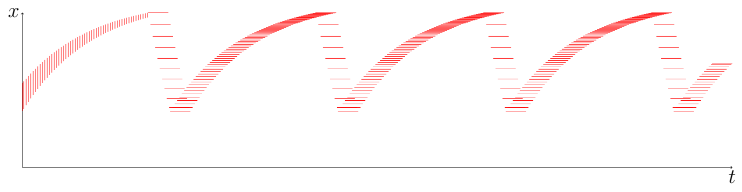


Figure 4: Results of HySon on the classical thermostat hybrid system (up to $t = 30s$).

a quadrotor system whose dynamics is highly non-linear with 12 continuous variables. A complete description of the system can be found in [2] and the equations of the dynamics are given in [7]. We want to perform a simulation of this system up to $t = 4s$, HySon achieves this in $4.4s$.

Let us also remark that although we are using affine forms, the bounds we obtain for the system variables can be stable even for non-linear dynamics. For example, consider the system given at Figure 3(a). Using HySon, we obtain the evolution shown in Figure 3(b) in the (x, y) plane (this was computed in 12.3s) for a simulation time up to $t = 30s$. This shows that the over-approximations we perform for non-linear computations do not perturb our simulation engine.

Clearly, HySon was primarily designed to handle control-command systems, i.e. hybrid systems in which the discrete evolution is time-triggered rather than event-triggered. However, as shown in Section 1, it can also handle guards and jumps, that we call zero-crossing events, and can thus perform guaranteed bounded-time reachability for hybrid systems. Consider for example the classical *room-heating system* with one room [10], in which the dynamics of the temperature x switches between two functions depending on the value of x. The result of HySon on this system is shown in Figure 4: as can be seen, the width of the enclosure for the temperature x remains stable as time goes by.

3 Conclusion

We presented a tool named HySon that adapts numerical simulation algorithms to perform set-based simulation and guaranteed flowpipe construction of Simulink models. It can handle many feature of the Simulink language and, as shown by our examples, it scales well in dimension and complexity. We believe that this tool can be very useful both as a verification tool but also in an early design process to evaluate the performance of a control-command system in the presence of uncertainties.

We plan to increase the number of Simulink blocks that our tool can handle. The main challenges we see come from the `Saturate` and `Lookup Tables` blocks. For the `Saturate` block, the difficulty is to keep relations between the input and output of the block, and for the the look-up tables, we will need to define new extrapolation algorithms working on affine forms.

Note that simulation has recently gained interest from hybrid systems community [13, 15, 16]. We believe that understanding and using numerical simulation algorithms is very important as they are used every day by engineers creating hybrid systems, and our experience showed that it they can be adapted to produce guaranteed results rather than approximations. We thus plan to continue in this direction by applying the same ideas for DAE solvers as the ones used in Modelica [12].

References

[1] R. Alur, T. Dang, and F. Ivančić. Counter-example guided predicate abstraction of hybrid systems. In *TACAS*, volume 2619 of *LNCS*, pages 208–223. Springer, 2003.

[2] S. Bouabdallah. *Design and control of quadrotors with application to autonomous flying.* PhD thesis, 2007.

[3] O. Bouissou. From control-command synchronous programs to hybrid automata. In *ADHS*, 2012.

[4] O. Bouissou and A. Chapoutot. An operational semantics for Simulink's simulation engine. In *LCTES*. ACM, 2012.

[5] O. Bouissou, A. Chapoutot, and A. Djoudi. Enclosing Temporal Evolution of Dynamical Systems Using Numerical Methods. In *NASA Formal Methods*, number 7871 in LNCS, pages 108–123. Springer, 2013.

[6] O. Bouissou, A. Chapoutot, and S. Mimram. HySon: Precise Simulation of Hybrid Systems with Imprecise Inputs. In *RSP*. IEEE, 2012.

[7] O. Bouissou, A. Chapoutot, S. Mimram, and B. Strazzulla. Set-based simulation for design and verification of Simulink models. In *ERTS2*, 2014.

[8] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *CAV*, volume 8044 of *LNCS*, pages 258–263. Springer, 2013.

[9] L. H. de Figueiredo and J. Stolfi. *Self-Validated Numerical Methods and Applications.* Brazilian Math. Colloquium monographs. IMPA/CNPq, 1997.

[10] A. Fehnker and F. Ivancic. Benchmarks for hybrid systems verification. In *HSCC*, volume 2993 of *LNCS*, pages 326–341. Springer, 2004.

[11] G. Frehse, C. Le Guernic, A. Donzé, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *CAV*, volume 6806 of *LNCS*, pages 379–395. Springer, 2011.

[12] P. Fritzson. *Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica*. Wiley, 2011.

[13] A. Girard and G. J. Pappas. Verification using simulation. In *HSCC*, volume 3927 of *LNCS*, pages 272–286. Springer, 2006.

[14] E. Goubault and S. Putot. A zonotopic framework for functional abstractions. *arXiv:0910.1763 preprint*, 2009.

[15] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. In *HSCC*, pages 43–52. ACM, 2013.

[16] J. Kapinski, J. V. Deshmukh, S. Sankaranarayanan, and N. Aŕechiga. Simulation-guided Lyapunov analysis for hybrid dynamical systems. In *HSCC*, 2014.

[17] H. Kong, X. Song, D. Han, M. Gu, and J. Sun. A new barrier certificate for safety verification of hybrid systems. *The Computer Journal*, 2013.

[18] R. E. Moore. *Interval analysis*. Prentice-Hall, 1966.

[19] S. Prajna. Barrier certificates for nonlinear model validation. *Automatica*, 42(1):117–126, 2006.

[20] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *HSCC*, volume 2993 of *LNCS*, pages 477–492. Springer, 2004.

[21] L. Shampine and M. Reichelt. The MATLAB ODE Suite. *Journal on Sci. Comput.*, 18(1):1–22, 1997.

[22] K. Sukumar. Translation of Simulink/Stateflow models to hybrid automata, 2011.

[23] A. Tiwari. Abstractions for hybrid systems. *Formal Methods in Systems Design*, 32:57–83, 2008.