Analysis of Hybrid Systems using HySAT

Christian Herde, Andreas Eggers, Martin Fränzle, and Tino Teige
Carl von Ossietzky Universität, Oldenburg, Germany
{herde|eggers|fraenzle|teige}@informatik.uni-oldenburg.de

Abstract

In this paper we describe the complete workflow of analyzing the dynamic behavior of safety-critical embedded systems with HySAT. HySAT is an arithmetic constraint solver with a tightly integrated bounded model checker for hybrid discrete-continuous systems which — in contrast to many other solvers — is not confined to linear arithmetic, but can also deal with nonlinear constraints involving transcendental functions. Based on a controller for train separation implementing a “moving block” interlocking scheme in the forthcoming European Train Control System Level 3, we exemplify the usage of the tool over the whole cycle from encoding a hybrid system to interpreting the results.

1. Introduction

Automatic verification of hybrid discrete-continuous systems, even in the restricted form of reachability analysis, still is a challenging research topic and not as established as other, less comprehensive, analysis methods like, e.g., simulation. Given the enormous practical importance of hybrid discrete-continuous models, with fields of application ranging from technical systems (discrete control + continuous environment, multi-objective scheduling, etc.) over biological systems (signaling in cell differentiation, blood clotting, etc.) to economical models, this sad state of affairs has sparked intensive research worldwide. Aiming at complete coverage of the infinite set of possible behaviors of these inherently open systems, state-exploratory verification techniques have gained considerable interest, starting as early as in the mid-nineties with the tool HyTech [14]. Since then, these tools have advanced wrt. performance and the classes of hybrid behavior they can handle. Nevertheless, even recently published tools for *unbounded* hybrid verification like PHAVer [12] and HSolver [18] lack a fully symbolic treatment of the complete discrete-continuous state space, thus being confined to moderately sized systems due to explicit state representations for the embedded discrete state spaces.

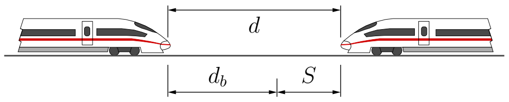

Figure 1. The absolute braking distance d equals the sum of the braking distance d_b of the following train and an additional safety distance S.

A technique offering good prospects for such a fully symbolic, homogeneous treatment of hybrid state spaces is *bounded model checking* (BMC) for *bounded* reachability analysis of systems, as originally proposed for the discrete-state case in [13, 4] to overcome scalability problems wrt. very large discrete state spaces. The idea of BMC is to encode the next-state relation of a system as a propositional formula, unroll this to some given finite depth k, and to augment it with a corresponding finite unravelling of the tableau of (the negation of) a temporal formula in order to obtain a propositional SAT problem which is satisfiable iff an error trace of length k exists. Enabled by the impressive gains in performance of propositional SAT checkers in recent years, BMC can now be applied to very large finite-state designs.

Though originally formulated for discrete transition systems only, the basic idea of BMC also applies to hybrid discrete-continuous systems. However, the BMC formulae arising from such systems are no longer purely propositional, but comprise complex Boolean combinations of arithmetic constraints over real-valued variables. Constraint and BMC solvers for pure linear arithmetic are, e.g., MathSAT [1] and HySAT-I [8]. More recently, we changed the algorithmic basis and thereby extended the scope of HySAT to non-linear arithmetic involving transcendental functions.[1] The algorithmic core of HySAT now is the iSAT algorithm [9], a tight integration of the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [6, 5] and interval constraint propagation (ICP, cf. [3] for an extensive survey),

[1] A HySAT executable, the tool documentation, and benchmarks can be found on `http://hysat.informatik.uni-oldenburg.de`.

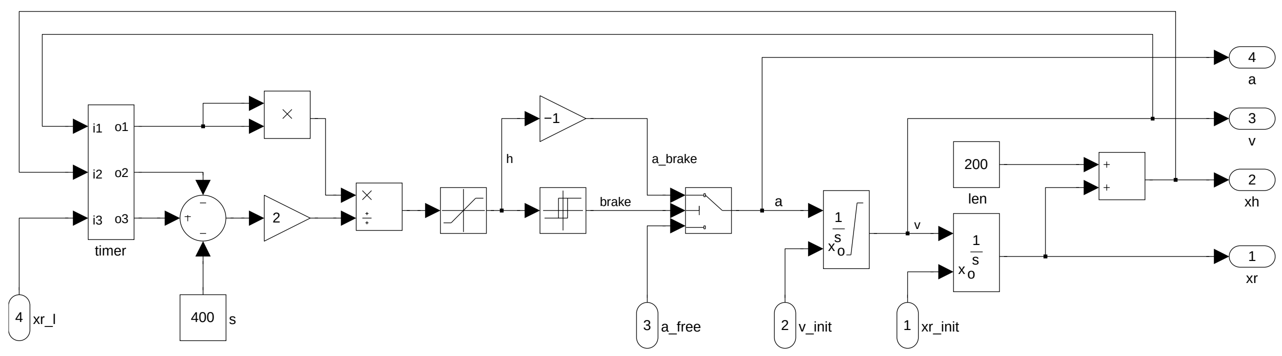

Figure 2. Implementation of the controller and train dynamics.

enriched by enhancements like *conflict-driven clause learning* and *non-chronological backtracking*. We showed that iSAT outperforms the approach of ABSOLVER [2], which — to the best of our knowledge — is the only other tool addressing large Boolean combinations of mixed discrete and continuous, non-linear arithmetic constraints. The new version of HySAT also exploits the BMC optimizations described in [8], such as *reusing and shifting of learned clauses* and *forward/ backward decision strategies*.

Structure of the paper. In Section 2 we explain the use of HySAT by means of modelling and analyzing a controller for the European Train Control System (ETCS) Level 3. The ETCS case study is introduced in Subsection 2.1. Subsection 2.2 deals with the input language of HySAT and presents the idea of encoding the ETCS benchmark into that language. The interpretation of the analysis results are given in Subsection 2.3. Section 3 concludes the paper and lists some directions for future and ongoing work.

2. Reachability analysis with HySAT

We demonstrate the reachability analysis with HySAT by introducing a concrete application benchmark from the transportation domain. The model was generated using the Matlab/Simulink tool. A structure-driven and compositional, currently manually applied translation — of which we show the most interesting aspects in the second subsection — then allows for fully automatic bounded model checking. The retrieved error trace is subsequently shown in a side-by-side comparison with a simulation run of the Simulink model.

2.1. ETCS benchmark

The benchmark deals with analyzing the safety of a railway system when operated under a "moving block" principle of operation. In contrast to conventional interlocking schemes in the railway domain, where static track segments are locked in full, the moving block principle applies headway control as required by the braking distance, reserving a "moving block" ahead of the train depending on speed and braking capabilities. There are two variants of this principle, namely train separation in relative braking distance, where the spacing of two following trains depends on the current speeds and braking capabilities of both trains, and train separation in absolute braking distance, where the distance of two following trains equals the braking distance of the second train plus an additional safety distance (Figure 1). Within this case study we apply the second variant which will also be used in the forthcoming European Train Control System (ETCS) Level 3.

We consider an abstract model of ETCS Level 3. Within this simplified version, all trains operate in obedience of the following procedures and regulations:

1. All trains on the track travel in the same direction, i.e. no train may ever change its direction.

2. The train sequence is fixed (no overtaking).

3. Each train broadcasts the position of its end to the following train every 8 seconds via radio.

4. Whenever a train receives an update of the position of the train running ahead, it computes its *movement authority* m, i.e. the stopping point it must not cross, and the deceleration a which is required to meet that stopping point. These are computed according to the formulae

$$m = xr - (xh + S) \quad \text{and} \quad a = \frac{v^2}{2m}$$

where xr is the position of the rear end of the first train, xh is the position of the head of the second train, and v is its velocity.

Braking is automatically applied whenever the value of a exceeds a certain threshold b_{on}. Automatic braking ends if a falls below b_{off}.

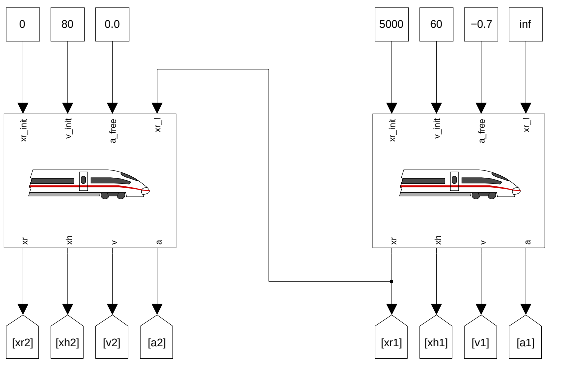

Figure 3. Top-level view of the Matlab/Simulink model.

5. When a train is not in automatic braking mode, acceleration and deceleration are freely controlled by the train operator within the physical bounds of the train.

We chose the parameters of the model to roughly match the characteristics of an ICE 3 half-train:

Parameter		Value
length of the train	[m]	200
maximum speed	[m/s]	83.4
maximum acceleration	[m/s^2]	0.7
maximum deceleration	[m/s^2]	-1.4
b_{on}	[m/s^2]	-0.7
b_{off}	[m/s^2]	-0.3
safety distance S	[m]	400

Figure 3 shows the top-level view of the Matlab/Simulink implementation of our model in a version with two trains. Inputs of a train block are the initial position of the train, its initial speed, the acceleration applied in free-running mode and the position of the rear end of the train which is running ahead. Outputs are the positions of the rear and of the head of the train, its velocity and current acceleration. The implementation of a train block is shown in Fig. 2.

A sample trace of the model, showing position, speed, acceleration and distance of the two trains (see Figure 4 (a)), seems to suggest that the controller works correctly: The trains are started with an initial distance of 5000 m, the second train being 20 m/s faster than the first train, which is braking with a deceleration of -0.7 m/s^2. The second train automatically starts braking, adjusting its deceleration in intervals of 8 seconds, and comes to stop exactly 400 m behind the first train.

Instead of performing a potentially unlimited number of simulations to cover all possible traces of the system, we encode the model for HySAT. This allows us to check all traces (up to a certain unwinding depth) for collisions of the trains without having to guess scenarios for the open inputs that may lead to these unsafe states.

2.2. Encoding into HySAT

In order to encode the model described in Subsec. 2.1, we first introduce the input language of the HySAT tool. The input file format consists of four parts:

- `DECL`: This part contains declarations of all variables. Types supported by HySAT are `float`, `int` and `boole`. When declaring a float or an integer variable you have to specify a bounded range of this variable, e.g. `float [0, 1000] x; boole jump;`. Boolean variables are identified with integer variables of range $[0, 1]$. Furthermore, you can define symbolic constants here, e.g. `define f = 2.0;`.
- `INIT`: This part is a formula describing the initial state(s) of the system to be investigated. Such a formula is an arbitrary Boolean combination of arithmetic constraints[2], e.g. `x = 0.6; !jump;`. Integer and float variables can be mixed within the same arithmetic constraint. The semicolon which terminates each constraint can be read as an AND-operator. Hence, `INIT` is a conjunction of both constraints.
- `TRANS`: This formula describes the transition relation of the system. Variables may occur in primed or unprimed form. A primed variable represents the value of that variable in the successor step, i.e. after the transition has taken place. E.g., `jump' <-> !jump; jump -> f * x' = x; !jump -> x' = x + 2;`.
- `TARGET`: This formula characterizes the state(s) whose reachability is to be checked, e.g. `x > 3.5;`.

When calling HySAT with the input described above, it successively unwinds the transition relation $k = 0, 1, 2, \ldots$ times, conjoins the resulting formula with the formulae describing the initial state and the target states, and thereafter solves the formula thus obtained. For $k = 0, 1, 2, 3, 4$, the formulae are all unsatisfiable, for $k = 5$ however, a solution is found. The output of HySAT for $k = 4$ and for $k = 5$ is as follows:

```
SOLVING:
    k = 4

RESULT:
    unsatisfiable

SOLVING:
```

[2]For a detailed list of all supported Boolean and arithmetic operators consult the user manual on the HySAT website.

```
    k = 5

RESULT:
    candidate solution box found

SOLUTION:
    jump (boole):
        @0: [0, 0]
        @1: [1, 1]
        @2: [0, 0]
        @3: [1, 1]
        @4: [0, 0]
        @5: [1, 1]

    x (float):
        @0: [0.6, 0.6]
        @1: [2.6, 2.6]
        @2: [1.3, 1.3]
        @3: [3.3, 3.3]
        @4: [1.65, 1.65]
        @5: [3.65, 3.65]
```

HySAT reports the values of `jump` and `x` for each step k of the system. After the last transition, as required, `x > 3.5` holds.

If HySAT terminates with the result '`unsatisfiable`', then the formula is actually unsatisfiable. If the solver stops with the result '`candidate solution box found`', then the solver could not detect any conflicts within the reported intervals which, however, does *not* mean that the intervals are guaranteed to actually contain a solution (cf. [9]). Nevertheless, the sizes of the returned intervals do not exceed a user-specified parameter ε, which can be set by the command-line option `--msw`. From a practical point of view, this means that the solver returns a solution with precision ε.

For encoding the Matlab/Simulink model of the ETCS case study (cf. Fig. 2 and 3) we first introduce a variable (of a corresponding type and domain) for each connecting line of the Simulink model and declare them in the `DECL` part. (Please note that by substituting the input-output functions of some blocks for occurrences of their outputs, we can reduce the total number of variables.) In the `INIT` part we require that the trains are stopped and their distance is 1000 meters. Contrary to the initial state of the simulation, the initial values of the accelerations are not fixed but may be chosen freely from their domain $[-1.4, 0.7]$. For the translation of the Simulink blocks into the `TRANS` part of HySAT we illustrate the encodings of the most interesting blocks of Fig. 2, i.e. the relay, switch, and integrator blocks. Simpler blocks, e.g. the sum block, can be encoded straightforwardly, e.g. by `o = i1 + i2` where `o` is the output and `i1,i2` are the inputs of the sum block. The predicative encodings of all blocks are conjoined by logical conjunction, represented by a semicolon in concrete HySAT syntax.

Relay block. When the relay is 'on' (indicated by the Boolean variable `is_on`), it remains 'on' until the input drops below the value of the switch-off-point parameter `param_off`. When the relay is 'off' (i.e. not `is_on` or `!is_on` holds), it remains 'off' until the input exceeds the value of the switch-on-point parameter `param_on`. The switch-on/off-point parameters are defined as symbolic constants in the `DECL` part, i.e. `define param_on = 0.7;` and `define param_off = 0.3;`.

```
( is_on and h >  param_off) -> ( is_on' and  brake);
( is_on and h <= param_off) -> (!is_on' and !brake);
(!is_on and h <  param_on ) -> (!is_on' and !brake);
(!is_on and h >= param_on ) -> ( is_on' and  brake);
```

The *switch block* passes through the first input `a_brake` or the third input `a_free` based on the value of the second input `brake`.

```
 brake -> a = a_brake;
!brake -> a = a_free;
```

Integrator block with saturation. The potentially new value `v'` of the velocity is determined by an Euler approximation with sampling time `dt` $= 8, 2$, and 1 seconds for the encodings A, B, and C, resp., and stored temporarily in the auxiliary variable `aux`. According to the saturation parameters, `v'` is set to its value as shown below. The lower and upper saturation limits are 0.0 and `v_max` $= 83.4$, respectively.

```
aux = v + dt * a;
aux <= 0.0                      -> v' = 0.0;
aux >= v_max                    -> v' = v_max;
(aux > 0.0 and aux < v_max) -> v' = aux;
```

Note that other (exact or safe) approximation methods are applicable here. For the sake of clarity, we opt for the simple, in general inexact, Euler method. We refer the interested reader to [15, 16] — just to name two different approaches to safely approximating the continuous behaviour.

Finally, completing the HySAT input we specify a target state, i.e. an undesired property of the system to be checked. In our case study, we want to know whether the controller is incorrect in the sense that collisions of the trains are possible. Hence, we add the formula `xr1 - (xr2 + length) <= 0.0;` to the `TARGET` section, meaning that the distance of the rear position of the first train `xr1` and the head position of the second train, i.e. rear position `xr2` plus length of the train, is less than or equal zero.

Recently, an automatic translation of a subset of Matlab/Simulink models to HySAT has been implemented in [17]. This translation follows the scheme sketched above. While not currently being able to translate the full model from Fig. 2 due to some of its Simulink blocks not being supported, it will in the near future cover all these blocks as well as a representative subset of Stateflow statecharts, as embedded into Simulink (cf. [11]).

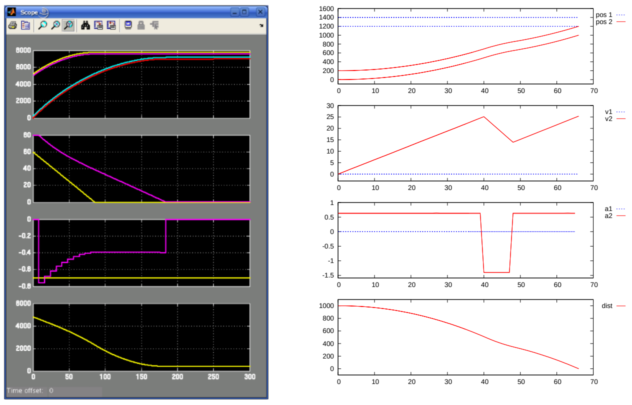

Figure 4. (a) left: simulation run of the system with fixed parameters, from top to bottom the charts show the positions, speeds, accelerations and distances of the two trains over the simulated time; (b) right: error trace found by HySAT

2.3. Results

Running HySAT on the encoded models yields error traces of lengths 8 for encoding A, 33 for encoding B, and 66 for encoding C. Bounded model checking thus revealed a simple bug of the controller that was yet subtle enough not to be noticed when designing the model: If the moving authority m becomes zero or even negative (which may happen since the controller re-computes the deceleration setting only every 8 seconds), then instead of applying the maximum braking force, the controller switches back to free-running mode, allowing the operator of the train to accelerate and crash into the rear of the train ahead. The error trace that was yielded for encoding C is shown in Fig. 4 (b) side-by-side with a simulation run of the system.

The experiments were performed on a 2.5 GHz Opteron machine with 4 GByte physical memory, running Linux. The total runtimes for solving all BMC instances up to the error trace were about 10 seconds for encoding A (with sampling time $\texttt{dt} = 8$ seconds), 1.8 minutes for encoding B ($\texttt{dt} = 2$ seconds) and 21.5 minutes for encoding C ($\texttt{dt} = 1$ second). The runtime largely depends on the solver settings, e.g. the splitting heuristics chosen, with the runtimes reported above being the best we could obtain for the respective encoding.

3. Conclusion and ongoing work

Based on a representative case study, we have in this paper described the use of the bounded model checker HySAT for bounded reachability analysis of safety-critical embedded systems. The workflow was exemplified on a controller for train seperation in the European Train Control System Level 3, with the model covering the joint dynamics of both the embedded controller and its physical environment. We hope that this paper will help and encourage other researchers to apply the HySAT tool to their respective problem domains.

HySAT is an ongoing project. On the one hand, we are continuously improving its algorithmic core iSAT, e.g. through more efficient internal data-structures, through low-level code optimizations for improving cache behavior, through acceleration by parallelization, and through various heuristics motivated by the problem structure. On the other hand, we are currently extending the scope of HySAT to support reachability analysis of broader classes of hybrid systems. More precisely, these extensions cover (a) the integration of safe numerical solving of *ordinary differential equations* (ODEs) in order to directly handle ODEs in the solver without an a priori approximation [7], (b) a generalization of the iSAT algorithm wrt. supporting *stochastic quantification* of discrete variables for the fully symbolic bounded reachability analysis of probabilistic hybrid systems [19, 10], and (c) the *generation of Craig interpolants* for *unbounded* model checking of hybrid systems.

Acknowledgements. This work has been partially supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS, `www.avacs.org`).

References

[1] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying industrial hybrid systems with MathSAT. In *Bounded Model Checking (BMC’04)*, volume 119 of *ENTCS*, pages 17–32, 2004.

[2] A. Bauer, M. Pister, and M. Tautschnig. Tool-support for the analysis of hybrid systems and models. In *Conference on Design, Automation and Test in Europe (DATE’07)*. IEEE Computer Society, 2007.

[3] F. Benhamou and L. Granvilliers. Continuous and interval constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, chapter 16, pages 571–603. Elsevier, Amsterdam, 2006.

[4] A. Biere, A. Cimatti, and Y. Zhu. Symbolic model checking without BDDs. In *TACAS’99*, volume 1579, 1999.

[5] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *Commun. ACM*, 5:394–397, 1962.

[6] M. Davis and H. Putnam. A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3):201–215, 1960.

[7] A. Eggers. Einbettung sicherer numerischer Integration von Differentialgleichungen in DPLL-basiertes arithmetisches Constraint-Solving für hybride Systeme. Master's thesis, Carl von Ossietzky Universität, Dpt. Informatik, Oldenburg, Germany, 2006. in German.

[8] M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30:179–198, 2007.

[9] M. Fränzle, C. Herde, S. Ratschan, T. Schubert, and T. Teige. Efficient Solving of Large Non-linear Arithmetic Constraint Systems with Complex Boolean Structure. *JSAT Special Issue on SAT/CP Integration*, 1:209–236, 2007.

[10] M. Fränzle, H. Hermanns, and T. Teige. Stochastic Satisfiability Modulo Theory: A Novel Technique for the Analysis of Probabilistic Hybrid Systems. In *Proceedings of the 11th International Conference on Hybrid Systems: Computation and Control (HSCC'08)*, 2008. to appear.

[11] M. Fränzle, H. Hungar, C. Schmitt, and B. Wirtz. HLang: Compositional Representation of Hybrid Systems via Predicates. Reports of SFB/TR 14 AVACS 20, SFB/TR 14 AVACS, July 2007. ISSN: 1860-9821, `http://www.avacs.org`.

[12] G. Frehse. PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. In *HSCC*, volume 3414 of *LNCS*, pages 258–273. Springer, 2005.

[13] J. F. Groote, J. W. C. Koorn, and S. F. M. van Vlijmen. The safety guaranteeing system at station hoorn-kersenboogerd. In *Conference on Computer Assurance*, pages 57–68. National Institute of Standards and Technology, 1995.

[14] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: The next generation. In *16th Annual IEEE Real-time Systems Symposium (RTSS 1995)*, pages 56–65. IEEE Computer Society Press, 1995.

[15] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HYTECH: A model checker for hybrid systems. *International Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

[16] R. J. Lohner. Enclosing the solutions of ordinary initial and boundary value problems. In *Computerarithmetic: Scientific Computation and Programming Languages*, pages 255–286. Teubner, Stuttgart, 1987.

[17] S. Ollhoff. Automatische Übersetzung von Simulink Modellen in HySAT-Formeln. BSc thesis, Carl von Ossietzky Universität, Dpt. Informatik, Oldenburg, Germany, 2007. in German.

[18] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Trans. Embedded Comput. Syst.*, 6(1), 2007.

[19] T. Teige. SAT-Modulo-Theory based Analysis of Probabilistic Hybrid Systems. In *Proceedings of the Dagstuhl Graduate School Meeting 2007 "Dagstuhl Zehn plus Eins"*, pages 86–87, Aachen, 2007. Verlag Mainz. ISBN: 3-86130-882-7.