Parametric Analysis of Hybrid Systems Using HyMITATOR

Étienne André[1] and Ulrich Kühne[2]

[1]LIPN, CNRS UMR 7030, Université Paris 13, France
[2]Group for Computer Architecture, University of Bremen, Germany

Abstract. Hybrid automata are a powerful formalism for modeling and verifying hybrid systems, that combine continuous and discrete behavior. A common problem for hybrid systems is the good parameters problem, which consists in identifying a set of parameter valuations guaranteeing a certain good behavior of a system. We introduce here HyMITATOR, a tool for efficient parameter synthesis for hybrid automata, providing hybrid systems with a quantitative measure of their robustness.

Keywords: Hybrid automata, Verification, Parameter synthesis, Robustness

1 Motivation and History

Hybrid systems combine discrete and continuous behavior. This corresponds for instance to the discrete control logic and continuous physical variables in an embedded system. Hybrid automata are a popular and powerful model for such systems, where the continuous variables evolve according to ordinary differential equations in each control mode.

In [4], we proposed the *inverse method* for timed automata, a subclass of hybrid systems whose variables (named clocks) all have constant rates equal to 1. Different from CEGAR-based methods, this original semi-algorithm for parameter synthesis is based on a "good" parameter valuation (also named *point*) π_0 instead of a set of "bad" states. This method synthesizes a constraint K_0 on the parameters such that, for each parameter valuation π satisfying K_0, the trace set (i.e., the discrete behavior) of $\mathcal{A}$ under π is the same as for $\mathcal{A}$ under π_0. This preserves in particular linear time properties. This also provides the system with a criterion of robustness, in the sense that the resulting constraint gives a quantitative measure of the allowed "drift" such that the discrete behavior of the system is not impacted. By iterating the inverse method on all integer points within a bounded reference parameter domain, we get a set of constraints (or *tiles*) such that, for each parameter valuation in each such tile, the time-abstract behavior is the same: this gives a *behavioral cartography* of the system [5].

A basic implementation named IMITATOR (for *Inverse Method for Inferring Time AbstracT behaviOR*) has first been proposed, under the form of a Python script calling HyTech. The tool has then been entirely rewritten in IMITATOR 2.0 [3], under the form of a standalone OCaml program making use of the

Parma Polyhedra Library (PPL) [8]. A number of case studies containing up to 60 timing parameters could be efficiently verified in the purely timed framework. The latest version (2.5) includes stopwatches and arbitrary updates, and has been applied to several classes of scheduling problems [6].

The inverse method and the behavioral cartography have been extended to hybrid systems in [11]. Due to the strong syntactic and algorithmic differences between timed automata and hybrid automata, the work of [11] had to be implemented in an experimental "fork" of IMITATOR 2.0, and not in the main version. We present in this paper HyMITATOR, a now mature extension of that prototype, performing parameter synthesis on hybrid systems.

2 Implementation and Features

HyMITATOR takes as input a network of hybrid automata synchronized on shared actions. The input syntax, inspired by HyTech, allows the use of analog variables (such as time, velocity or temperature), rational-valued discrete variables, and parameters (i.e., unknown constants). The dynamics of the analog variables is described by ordinary differential equations. The tool directly supports linear dynamics, while affine dynamics can be approximated with arbitrary precision.

The core of the program is written in the object-oriented language OCaml, and interacts with PPL. Exact arithmetics with unbounded precision is used. A constraint is output in text format; furthermore, the set of traces computed by the analysis can be output under a graphical form (using Graphviz).

HyMITATOR implements the following algorithms for hybrid systems:

Full reachability analysis Given a model, it computes the set of symbolic reachable states.

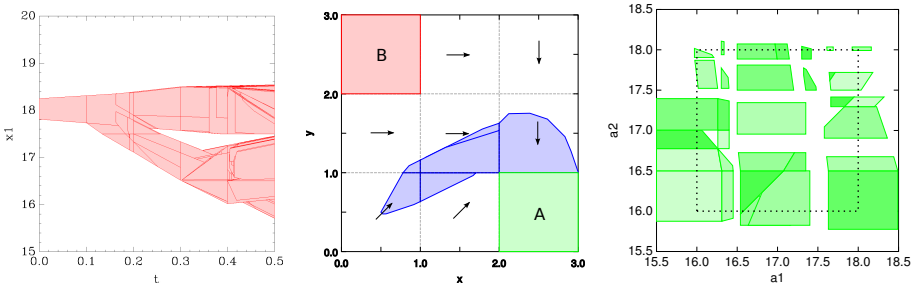
Predicate Abstraction Safety verification can alternatively be performed using a counterexample-guided abstraction refinement loop. The abstract state space is constructed w.r.t. a set of linear predicates [1].

Inverse method Given a model and a reference parameter valuation π_0, it computes a constraint on the parameter guaranteeing the same time-abstract behavior as under π_0 [4,11].

Behavioral cartography Given a model and a bounded parameter domain for each parameter valuation, it computes a set of constraints [5,11].

HyMITATOR uses several algorithmic optimizations, some of which have initially been developed for IMITATOR. In particular, the efficient merging technique presented in [7] has been successfully extended to the hybrid case: we merge any two states sharing the same discrete part (location and value of the discrete variables) and such that the union of their constraint on the analog variables and parameters is convex. This optimization preserves the correctness of all our algorithms; better, the constraint output by the inverse method in that case may be weaker, i.e., covers a larger set of parameter valuations.

For affine hybrid systems, further optimizations are needed. Due to the linear over-approximation by partitioning the state space, a lot of additional branching is introduced, which renders the inverse method ineffective. To solve this



(a) Reachable states (b) Trajectories (c) Behavioral cartography

Fig. 1. Examples of graphics output by HyMITATOR

problem, the algorithm has been extended as described in [11]. Basically, the partitioning is performed locally, and partitions belonging to the same discrete state are merged by taking their convex hull.

The post image computation can be costly for hybrid automata. To overcome this problem, an abstraction technique for the verification of simple safety properties (non-reachability of bad states) has been presented in [1]. Based on a set of linear predicates, reachability is performed on the abstract state space induced by these predicates. Refinement can be performed by discovering separation planes. While the original method is based on flow-pipe construction, we adapted the algorithm to the linear approximation by state space partitioning.

The behavioral cartography has been adapted to the framework of hybrid systems. Different from timed automata, hybrid automata do not restrict coefficients appearing in clock constraints to be integers, and allow variables to be compared with any rational value. For this reason, instead of considering only integer points as starting points for the inverse method, an arbitrary rational step size can be used for each parameter dimension in HyMITATOR. This gives more accurate results, by reducing the size of the possible "holes" not covered by any tile of the cartography.

HyMITATOR (with sources, binaries and case studies) is available on its Web page: http://www.lsv.ens-cachan.fr/Software/hymitator/.

3 Applications

HyMITATOR can be used for the *parametric verification* of hybrid systems. An application to sampled data hybrid systems has been presented in [11]. As a special case, such systems can be parametrized over the initial states. Then, a single run satisfying a desirable reachability property can be generalized to a larger set of initial states. As an example, Figure 1(a) shows the enlarged reachable states of a single run for the *room heating benchmark* from [9]. This also proves the *robustness* of the system w.r.t. the tested property. Figure 1(b) shows an over-approximation of the reachable states for the *navigation benchmark* [9], proving that all trajectories will eventually enter the green target zone.

Another problem that can be addressed using HyMITATOR is *test coverage*. In order to ensure the quality of an implementation of a hybrid system, a set of tests is generated which is then applied to the system. However, since the state space of hybrid systems is infinite in general, it is hard to decide when enough tests have been performed. Using the inverse method, a tile (dense set of points) around each test point is generated which entails the same discrete behavior. This means that any point in this tile can be considered covered. Figure 1(c) shows the coverage of a parameter rectangle for the room heating benchmark.

4 Related Work

One of the first powerful model checkers for analyzing hybrid automata is HyTech [12]. Unfortunately, it can hardly verify even medium sized examples due to exact arithmetics with limited precision and static composition of automata, quickly leading to memory overflows. The tool PHAVer [10] improves on the computation of the reachable states by using efficient over-approximations. Techniques similar to those in PHAVer have also been implemented in HyMITATOR, with additional algorithmic improvements. The work in [2] presents an analysis on Simulink models which shares similar goals with our approach.

References

1. R. Alur, T. Dang, and F. Ivančić. Predicate abstraction for reachability analysis of hybrid systems. *ACM Trans. Embedded Computing Systems*, 5:152–199, 2006.
2. R. Alur, A. Kanade, S. Ramesh, and K. C. Shashidhar. Symbolic analysis for improving simulation coverage of simulink/stateflow models. In *EMSOFT'08*, pages 89–98. ACM, 2008.
3. É. André. IMITATOR II: A tool for solving the good parameters problem in timed automata. In *INFINITY'10*, volume 39 of *EPTCS*, pages 91–99, 2010.
4. É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, 2009.
5. É. André and L. Fribourg. Behavioral cartography of timed automata. In *RP'10*, volume 6227 of *LNCS*, pages 76–90. Springer, 2010.
6. É. André, L. Fribourg, U. Kühne, and R. Soulat. IMITATOR 2.5. Available at www.lsv.ens-cachan.fr/Software/imitator/.
7. É. André, L. Fribourg, and R. Soulat. Enhancing the inverse method with state merging. In *NFM'12*, volume 7226 of *LNCS*, pages 100–105. Springer, 2012.
8. R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.
9. A. Fehnker and F. Ivancic. Benchmarks for hybrid systems verification. In *HSCC'04*, volume 2993 of *LNCS*, pages 326–341. Springer, 2004.
10. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. *Software Tools for Technology Transfer*, 10(3):263–279, 2008.
11. L. Fribourg and U. Kühne. Parametric verification and test coverage for hybrid automata using the inverse method. In *RP'11*, volume 6945 of *LNCS*, pages 191–204. Springer, 2011.
12. T. A. Henzinger, P. H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:460–463, 1997.