Application of constraint solving and ODE-enclosure methods to the analysis of hybrid systems

Andreas Eggers, Martin Fränzle and Christian Herde

Carl von Ossietzky Universität Oldenburg, Dept. of Computing Science, 26111 Oldenburg, Germany
{eggers,fraenzle,herde}@informatik.uni-oldenburg.de

Abstract. In this short paper, we summarize our approach to analyzing hybrid discrete-continuous systems by reduction to constraint solving paired with enclosure methods for sets of initial value problems of ordinary differential equations.

Keywords: hybrid discrete-continuous systems, verification, ordinary differential equations, constraint solving
PACS: 02.70.-c, 02.60.Lj, 89.75.-k

INTRODUCTION

Many systems consist of discrete and continuous aspects that influence each other. This is especially true in the domain of so-called embedded systems, where often a digital controller manipulates the behavior of a physical plant via actuators and bases its decisions on measurements of the system obtained by sensors. Our focus is on embedded systems which play a safety-critical role, e.g. because they have to decide whether an airbag ignition sequence should be started or a landing gear may be retracted or not. The goal of our work is to devise methods that can automatically prove the correctness of such complex hybrid discrete-continuous systems despite the infinite number of possible states they can reach. In this paper we present an approach to automatically find errors in models that characterize the discrete and continuous behavior of hybrid systems.

HYBRID SYSTEMS

Hybrid systems can be regarded as having a finite number of discrete modes along with a finite number of continuous variables whose evolution in each mode can be characterized by a mode-dependent ordinary differential equation. Changes from one mode to another may be restricted by certain guard conditions and may cause the valuation of the continuous variables to change as well. A formalism that captures this perspective on hybrid systems are *hybrid automata* (see e.g. [1]). Formally, a hybrid automaton $A = (V, M, I, F, J)$ is given by a set of continuous variables $V = \{x_1, \ldots, x_n\}$, discrete modes M, and sets I, F, and J characterizing the initial states, the continuous evolutions (*flows*), and the discrete mode changes (*jumps*), respectively. These sets are defined as follows: $I = \{(m, i_m) | m \in M\}$, where i_m is a predicate over the variables from V, $F = \{(m, ODE_m, Invar_m) | m \in M\}$, where ODE_m is an ordinary differential equation over V and $Invar_m$ a mode invariant, i.e. again a predicate over V, and $J = \{(m, Guard_{m,m'}, Action_{m,m'}, m') | m \in M\}$, where $Guard_{m,m'}$, the guard, is a predicate over V and $Action_{m,m'}$, the action of the jump, is a predicate over the variables from V and decorated versions (e.g. x_1') thereof — the latter denoting the successor value of the variable after the jump.

Semantics. We call $\sigma : V \to \mathbb{R}$ a *valuation* which assigns each variable from V a value from the real numbers. Together with a discrete mode $m \in M$, the pair $\langle m, \sigma \rangle$ is called a *state* of the system. Using this definition, an alternating

A. Eggers, M. Fränzle, and C. Herde, *Application of Constraint Solving and ODE-Enclosure Methods to the Analysis of Hybrid Systems,* in *NUMERICAL ANALYSIS AND APPLIED MATHEMATICS: International Conference on Numerical Analysis and Applied Mathematics 2009*, edited by T. E. Simos, G. Psihoyios, and C. Tsitouras, American Institue of Physics, Melville, New York, 2009, vol. 2 of *AIP Conference Proceedings*, pp. 1326–1330, http://link.aip.org/link/?APC/1168/1326/1.

sequence of states

$$\rho : \langle m_0, \sigma_0 \rangle \rightsquigarrow_{\Delta t_0} \langle m_0, \hat{\sigma}_0 \rangle \rightarrow \langle m_1, \sigma_1 \rangle \rightsquigarrow_{\Delta t_1} \langle m_1, \hat{\sigma}_1 \rangle \rightarrow \cdots,$$

where $\rightsquigarrow_{\Delta t_i}$ denotes a continuous flow of duration Δt_i mediated by an ODE and $\rightarrow$ denotes an instantaneous jump, is called a *run* of the hybrid automaton A iff the following properties are satisfied:

- *Initiation:* the first element $\langle m_0, \sigma_0 \rangle$ of the run is a valid initial state, i.e. $\exists (m, i_m) \in I : m_0 = m \wedge \sigma_0 \models i_m$.
- *Jump step:* each pair of states $\langle m_i, \hat{\sigma}_i \rangle \rightarrow \langle m_{i+1}, \sigma_{i+1} \rangle$ connected by a jump ($\rightarrow$) is valid with respect to J, i.e. $\exists (m, Guard_{m,m'}, Action_{m,m'}, m') \in J : m_i = m \wedge m_{i+1} = m' \wedge \hat{\sigma}_i \models Guard_{m,m'} \wedge (\hat{\sigma}_i, \sigma_{i+1}) \models Action_{m,m'}$.
- *Flow step:* each pair of states $\langle m_i, \sigma_i \rangle \rightsquigarrow_{\Delta t_i} \langle m_i, \hat{\sigma}_i \rangle$ connected by a flow ($\rightsquigarrow_{\Delta t_i}$) is valid with respect to F, i.e. $\exists (m, ODE_m, Invar_m) \in F$ (with $ODE_m \equiv \frac{d\vec{x}}{dt} = \vec{f}(\vec{x})$ an ordinary differential equation): $m_i = m \wedge \exists \vec{x} : [0, \Delta t_i] \rightarrow \mathbb{R}^n : \sigma_i(\vec{x}) = \vec{x}(0) \wedge \hat{\sigma}_i(\vec{x}) = \vec{x}(\Delta t_i) \wedge \forall \tau \in [0, \Delta t_i] : \vec{x}(\tau) \models Invar_m \wedge \frac{d\vec{x}}{dt}(\tau) = \vec{f}(\vec{x}(\tau))$. This property can be intuitively described as follows: the valuation σ_i of the variables $\vec{x} = (x_1, \ldots, x_n)$, with the x_j from V, is the starting point of a solution trajectory of the differential equation ODE_m that after Δt_i time units reaches the valuation $\hat{\sigma}_i$ of $\vec{x}$ while never leaving the space described by the mode invariant $Invar_m$.

The states in ρ thus characterize a run over the real time from 0 to $\sum_i \Delta t_i$, where alternatingly either time is spent in a mode and the continuous variables change according to the differential equation of that mode or a mode change occurs and variables are updated according to the action enforced by the particular jump taken. Please note that this model allows several sources of non-determinism, among them a potentially infinite number of initial states, multiple jump guards being true at the same time, thereby enabling multiple jumps or the choice between a jump and staying within the current mode if its invariant allows, and non-deterministic assignments in the action predicate of a jump, e.g. $x' \geq 4.2 \wedge x' \leq 7.3$. Even for a fixed initial state, there can thus be infinitely many valid runs starting from there.

Reachability. Using this formalism, the initial question of how to detect errors in hybrid systems can now be refined. Given a hybrid automaton A and a *target predicate* describing e.g. unsafe states of the system, the analysis goal is to find out, whether there exists a run ρ of A that leads to a state satisfying the target predicate. The existence of such a run would then be witness to an incorrect behavior of the system, while a proof of its non-existence would prove the safety of the system with respect to the given target predicate. Though this problem is undecidable in the general case [1], its importance justifies to investigate algorithms that try to tackle it at least for practically interesting types of systems.

BOUNDED MODEL CHECKING USING CONSTRAINT SOLVING

A version of reachability is *bounded reachability*, where only runs of a given length are considered. From an application perspective, this provides a powerful debugging tool but is as useful in mechanizing inductive arguments based on transition induction [2].

Given above semantics, the *bounded model checking* (BMC) [3, 4] problem can be formulated as follows: does there exist a run of at most k steps length whose last state satisfies a given target predicate? In order to answer this question, the semantic definitions are used to compose a formula from the automaton which is satisfiable if such a run exists. Given predicative encodings $init(Y)$ of the possible initial states, $trans(Y, Y')$ of all possible transitions from one state to another, i.e. all jumps and flows, and $target(Y)$ of the target states whose reachability is to be checked, the BMC problem corresponds to the satisfiability of the formula

$$\Phi := init(Y_0) \wedge trans(Y_0, Y_1) \wedge trans(Y_1, Y_2) \wedge \ldots \wedge trans(Y_{k-1}, Y_k) \wedge target(Y_k),$$

where the $Y_0, \ldots, Y_k$ are sets of variables representing the state of the automaton, i.e. mode and valuation of the continuous variables, in the 0-th, ..., k-th step, respectively. Thus, by encoding the automaton A by such a *constraint system* Φ, i.e. a Boolean combination of arithmetic constraints and differential equations, deciding the BMC problem becomes equivalent to checking satisfiability of a formula [4].

SOLVING CONSTRAINT SYSTEMS INVOLVING DIFFERENTIAL EQUATIONS

In order to solve the kind of formula introduced in the previous section, our approach extends the iSAT algorithm [5] with safe numerical integration of ODEs. We therefore first summarize the iSAT algorithm itself and then present our approach to embed handling of ODEs into it.

The iSAT algorithm

Given a Boolean combination of arithmetic constraints along with finite bounds for the domains of the variables involved in the given constraint system, the frontend of the solver converts the formula into conjunctive normal form by introducing auxiliary variables. Internally, the formula to be solved thus becomes a conjunction of clauses, which themselves are disjunctions of arithmetic constraints, e.g. $(a \geq 1 \vee x \leq 4.2 \cdot y) \wedge (\sin(x) < 0.4) \wedge (x^2 \geq 2.0 \vee y = 0)$. The goal of solving this formula is to find out whether there exists a valuation under which in each clause at least one constraint is satisfied and that thereby satisfies the formula in its entirety.

Taking the given bounds for each variable occuring in the formula as their initially assigned *interval valuation* ξ, the iSAT algorithm prunes off definitive non-solutions from these intervals using *deduction* of the following two types. The requirement that at least one constraint per clause must be satisfied, gives rise to the first deduction mechanism, *unit propagation*, which is directly inherited from (DPLL-based) Boolean satisfiability (SAT) solving [6, 7]: if in some clause all constraints but one evaluate to false under the current valuation, the remaining constraint must be satisfied in order to preserve satisfiability of the entire formula. In the example given above, from a valuation that assigns $\xi(a) = [-10, -3.2]$, unit propagation could deduce that in the first clause the second constraint $x \leq 4.2 \cdot y$ determines satisfaction such that all satisfying valuations must obey this constraint. This remaining constraint is called *unit* and may now imply new deductions itself which necessitates the second type of deductions: taking the current bounds for the variables involved in constraints that have been found to be unit, *interval constraint propagation* is used to deduce further bounds. In above example, current interval assignments for x and y could be used to deduce new bounds for the respective other variable, e.g. an upper bound for $y \leq 2$ would be sufficient to deduce that $x \leq 8.4$ (in this case even independently of y's lower bound). Such bounds may themselves give rise to further deductions of either type.

There are three different ways, in which a deduction phase may terminate: (1) no further bounds can be deduced, (2) newly deduced bounds yield only very small progress over previously known bounds, or (3) a deduced bound is contradictory to an existing bound. In the first and second cases, deduction alone does not help in further narrowing the search space. It thus calls for a *decision* by the solver, i.e. the solver (heuristically) selects a variable, splits its interval, and then performs the deductions now possible due to the newly assigned bound. In the third case, a *conflict* has occurred. This means that the part of the search space currently under consideration by the solver does not contain any solutions. In general the conflict does not only apply to the very box under consideration but may be the result of a relatively small subset of the bounds defining this box. Again akin to propositional SAT solving, a small set of reasons for the conflict is identified and the bounds that constitute this set are used to formulate a *conflict clause* (see e.g. [8]). Whenever all but one of the conditions that lead to the conflict are met again, the negation of the remaining reason becomes unit and thereby prevents the solver from running again into the same conflict. Having added this conflict clause, the solver performs a *backjump* [8], thereby undoing the decisions that lead to the conflict.

If a conflict is detected to be independent from any decisions, i.e. backtracking cannot extricate the solver from the conflict, it has successfully found out that the formula does not have any solutions in the entire given domain. This unsatisfiability result is safe since deductions never prune off solutions from the search space. If, on the other hand, a box (of a small user-defined size) is found for which deduction did not detect a conflict, the solver returns the interval assignment as a candidate solution box, yet without guaranteeing the existence of a point-valued solution inside it.

Using ODE enclosure methods as propagators within iSAT

In [9], we have presented a seamless integration of safe Taylor-series based enclosures of ODEs [10, 11] into the iSAT algorithm. Without going into the technical details, this integration essentially works as follows. Like the other parts of the transition system, each ODE constraint occurs in all steps of the bounded model checking formula Φ. As an example take a simple system of ODE constraints $(\frac{dx}{dt} = v) \wedge (\frac{dv}{dt} = 2.3)$. If this formula occurs in the transition system it will be instantiated at all BMC steps. Depending on previous deductions and decisions, unit propagation may detect a

definitionally closed system of ODE constraints (in the sense of each variable being defined by an ODE constraint) e.g. among those originating from BMC step i. In the example above, this would be an ODE defining the evolution of the variables x and v between BMC step i and its successor $i+1$. For the respective instances x_i, v_i and x_{i+1}, v_{i+1} as well as for a freshly introduced duration variable Δt with its instantiation Δt_i, current interval assignments $\xi(x_i), \xi(v_i), \ldots$ are known. Exactly mirroring the semantics of hybrid systems, this system of ODE constraints is satisfied if there exists a point in the box $\xi(x_i) \times \xi(v_i)$ from which a solution trajectory of the ODE leads to a point in the box $\xi(x_{i+1}) \times \xi(v_{i+1})$ with a length that lies in $\xi(\Delta t_i)$, the length of the flow. More generally, the valuations of the variables representing the state at BMC step i define a *prebox*, and those representing the state at BMC step $i+1$ define a *postbox* between which connecting trajectories of lengths $\xi(\Delta t_i)$ have to be searched.

From this description, the role of ODE enclosure methods within the overall algorithm becomes apparent: by safely enclosing all trajectories emerging from the prebox over the interval range determined by $\xi(\Delta t_i)$, new safe bounds for the trajectories can be deduced and thereafter be used to tighten the postbox. Using the same mechanism, switching the role of pre- and postbox and multiplying the righthand side of the (time-invariant) ODE with -1, similar narrowings for the prebox can be obtained.

Our current work adresses different aspects that we consider essential to the performance of the method. While our own prototypical implementation of an ODE enclosure method proved feasibility of the approach and allowed us to generate some first benchmark results [9], we expect that using more optimized enclosure code can reduce some of the penalty incurred upon deduction for ODE constraints compared to the relatively cheap deductions through unit propagation or interval constraint propagation. We are currently integrating Nedialkov's VNODE-LP [12] as an enclosure method replacing our own.

A second optimization is to avoid re-enclosing trajectories by memoizing the results of deductions persistently. This idea is similar to learning conflict clauses: by storing deduced knowledge in clauses and adding these to the constraint system, the solver will not have to revisit parts of the search space for which previous deductions imply that there exist no solutions inside. Due to the expressiveness of the constraint language, schemes that not merely store bounds on variables but also e.g. Taylor series with error bounds can be interesting research directions.

A third interesting direction will be to combine different enclosure methods like Taylor models [13] with their stronghold in the case of nonlinear ODEs with computationally cheap methods, e.g. based on monotonicity [14], along with heuristics selecting between the different available propagation mechanisms. This allows the solver to use coarse but quick enclosures as long as they are effective, proceeding to computationally more expensive, tight enclosures thereafter in order to prune off spurious solutions of the constraint systems.

ACKNOWLEDGMENTS

This work has been partially funded 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. T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's Decidable about Hybrid Automata," in *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, ACM, 1995, pp. 373–382.
2. Z. Manna, and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, vol. 1, Springer-Verlag, Berlin, 1992.
3. 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*, National Institute of Standards and Technology, 1995, pp. 57–68.
4. A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," in *Tools and Algorithms for the Construction and Analysis of Systems*, edited by R. Cleaveland, Springer, 1999, vol. 1579 of *Lecture Notes in Computer Science*, pp. 193–207.
5. M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, *JSAT Special Issue on SAT/CP Integration* **1**, 209–236 (2007).
6. M. Davis, and H. Putnam, *Journal of the ACM* **7**, 201–215 (1960).
7. M. Davis, G. Logemann, and D. Loveland, *Communications of the ACM* **5**, 394–397 (1962).
8. L. Zhang, and S. Malik, "The Quest for Efficient Boolean Satisfiability Solvers," in *Automated Deduction – CADE-18*, edited by A. Voronkov, Springer, Berlin, 2002, vol. 2392 of *Lecture Notes in Computer Science*, pp. 313–331.
9. A. Eggers, M. Fränzle, and C. Herde, "SAT Modulo ODE: A Direct SAT Approach to Hybrid Systems," in *Proceedings of the 6th International Symposium on Automated Technology for Verification and Analysis (ATVA'08)*, edited by S. S. Cha, J.-Y. Choi, M. Kim, I. Lee, and M. Viswanathan, Springer, 2008, vol. 5311 of *Lecture Notes in Computer Science*, pp. 171–185.

10. R. E. Moore, "Automatic local coordinate transformation to reduce the growth of error bounds in interval computation of solutions of ordinary differential equations," in *Error in Digital Computation*, edited by L. B. Ball, Wiley, New York, 1965, vol. II, pp. 103–140.
11. R. J. Lohner, "Enclosing the Solutions of Ordinary Initial and Boundary Value Problems," in *Computerarithmetic: Scientific Computation and Programming Languages*, Teubner, Stuttgart, 1987, pp. 255–286.
12. N. S. Nedialkov, VNODE-LP – A Validated Solver for Initial Value Problems in Ordinary Differential Equations, Tech. Rep. CAS-06-06-NN, Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada (2006), `www.cas.mcmaster.ca/~nedialk/vnodelp/doc/vnode.pdf`.
13. M. Berz, and K. Makino, *Reliable Computing* **4**, 361–369 (1998).
14. N. Ramdani, N. Meslem, and Y. Candau, "Reachability of Uncertain Nonlinear Systems Using a Nonlinear Hybridization," in *HSCC*, edited by M. Egerstedt, and B. Mishra, Springer, 2008, vol. 4981 of *Lecture Notes in Computer Science*, pp. 415–428.