Formal Verification of Hybrid Systems

Rajeev Alur
University of Pennsylvania
alur@cis.upenn.edu

ABSTRACT

In *formal verification*, a designer first constructs a model, with mathematically precise semantics, of the system under design, and performs extensive analysis with respect to correctness requirements. The appropriate mathematical model for embedded control systems is *hybrid systems* that combines the traditional state-machine based models for discrete control with classical differential-equations based models for continuously evolving physical activities. In this article, we briefly review selected existing approaches to formal verification of hybrid systems, along with directions for future research.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; J.7 [**Computers in other systems**]: Real time, Process control

General Terms

Verification.

Keywords

Hybrid systems, Control systems, Model checking, Formal methods

1. REVIEW OF CURRENT APPROACHES

Model-based design offers a promising approach for detecting and correcting errors in early stages of system design [33, 37, 49]. In this methodology, a designer first constructs a model, with mathematically precise semantics, of the system under design, and performs extensive analysis with respect to correctness requirements before generating the implementation from the model. Embedded systems, such as controllers in automotive, medical, and avionic systems, consist of a collection of interacting software modules reacting to a continuously evolving environment. The appropriate mathematical model for design of embedded control systems is *hybrid systems* that combines the traditional models for discrete behavior with classical differential- and algebraic-equations based models for dynamical systems. Such models can capture both the controller — the system under design, and the plant — the environment with continuously evolving physical activities in which the system operates. Given that (1) automated verification tools have recently been successful in finding bugs in "real-world" hardware protocols and device drivers [13, 15], (2) tools such as Stateflow/Simulink are commonly used in automotive and avionics industry for modeling, and (3) high assurance is a necessity in safety-critical applications that deploy embedded software, formal verification of hybrid systems has been a vibrant research area for the past 20 years. This article is an overview of current research directions, aimed at providing an introductory "roadmap" rather than a comprehensive survey.

Modeling

In early 1990s, formal models for discrete *reactive* systems were integrated with models for dynamical systems [2, 41]. The model of *hybrid automata* [1, 2, 29] has emerged to be a popular choice. A hybrid automaton is an extended finite-state machine whose state consists of a *mode* that ranges over finitely many discrete values and a finite set of real-valued variables. Each mode is annotated with constraints that specify the continuous evolution of the system, and edges between modes are annotated with guards and updates that specify discrete transitions. For example, the behavior of a self-regulating thermostat can be described by a hybrid automaton with two modes, *on* and *off*, and a single real-valued variable T modeling the temperature. To specify how the temperature changes in the mode *on*, we can annotate the mode with a differential equation, say, $\dot{T} = k(70 - T)$, for a constant parameter k. Alternatively, we can use a differential inequality, say, $k_1 \leq \dot{T} \leq k_2$, for two constants k_1 and k_2, to specify the dynamics approximately using only bounds on the derivative. Associating the mode *on* with an invariant constraint $T \leq 70$ specifies that the system must exit this mode before the temperature exceeds 70. An edge from the mode *on* to the mode *off* with the guard $T \geq 68$ specifies that, whenever the temperature exceeds 68, the system can discretely change its state by updating the mode to *off*. A hybrid automaton can naturally be interpreted as an infinite-state transition system, and this forms the basis for formalizing classical notions such

EMSOFT'11, October 9–14, 2011, Taipei, Taiwan.

as safety verification, property-preserving abstractions, and simulation relations, for hybrid systems.

Hybrid automata are analogs of state machines, with little support for structured descriptions, and consequently, a number of formalisms have been proposed to facilitate modular descriptions of complex systems. These include modeling environments such as SHIFT [17] and PTOLEMY [19] for hierarchical specifications of hybrid behavior; models such as hybrid I/O automata [40], hybrid modules [6], and CHARON [3], for compositional treatment of concurrent hybrid behavior; and differential dynamic logic for logic-based specification and compositional analysis of sequential hybrid behavior [43].

The commercial modeling tools such as Stateflow/Simulink (see `www.mathworks.com`) and Modelica (see `www.modelica.org`) are routinely used in a wide range of industries. Conceptually, it should be possible to compile models expressed in such tools into formal notations such as hybrid automata. This has turned out to be difficult in practice due to the richness of features in commercial tools and the lack of a standardized formal semantics of such features (see [35] for efforts aimed at semantics-preserving transformations across modeling notations).

Symbolic reachability analysis

In the safety verification problem for hybrid systems, we are given a hybrid systems model M, a set I of initial states of M, and a set S of "safe" states of M, and we want to check whether every execution of M starting in an initial state always stays within the set of safe states, and if not, report a violating execution as a counter-example. For instance, given the hybrid systems model of a collision avoidance protocol, we want to check whether the distance between two vehicles stays greater than the safety threshold for given initial conditions. Let us first note that the safety verification problem of hybrid systems cannot be solved algorithmically: with the exception of classes that severely restrict the allowed dynamics of real-valued variables such as timed automata [5] and initialized rectangular automata [32], the safety verification problem is undecidable [1, 32]. Symbolic reachability algorithms for safety verification try to compute the set R of reachable states of a hybrid system M in an iterative manner starting from the set I of initial states. The algorithm checks, at every step of the iteration, if the current set R of reachable states is a subset of the set S of safe states, and if not, it terminates with a counter-example. In general, there is no termination guarantee, as the algorithm may keep adding more and more states to R without being able to deduce that the system is safe. The key challenge to efficient implementation is to identify a suitable representation for the set of states that supports the operations used by the iterative reachability computation.

The tool HYTECH was the first model checker to implement symbolic reachability analysis for hybrid systems [7,31]. For a hybrid automaton with n real-valued variables, each reachable set is represented by associating a finite union of n-dimensional polyhedra with each mode, where a polyhedron is represented as a conjunction of linear inequalities over variables. Such a polyhedra-based representation is appealing due to the use of polyhedra in many computing applications and the availability of open-source libraries for manipulating them (c.f. [12]). The models are restricted to the class of *linear hybrid automata* (LHA): guards, updates, and invariants involve only linear expressions, and the dynamics is specified using differential inequalities that are linear constraints over first-order derivatives. For example, the LHA-admissible dynamics $\dot{x} = \dot{y} \wedge 1 \leq \dot{x} \leq 2$ describes two-dimensional motion along the diagonal line with bounds on speed. For LHA, the polyhedral representation is *closed* under both discrete transitions corresponding to mode-switches and continuous evolution according to differential constraints in a mode: given a polyhedron R describing the set of current states, the set R' of states that the system can reach after a discrete mode-switch to a mode m, is a polyhedron that can be computed effectively from R, and the set R'' of states that the system can reach as a result letting it evolve continuously according to the dynamics associated with the mode m, is also a polyhedron that can be computed effectively from R'.

The most commonly used dynamics in mathematical design of control systems involves *linear differential equations*: if $\mathbf{x}$ represents the vector of state variables, $\mathbf{u}$ represents the vector of input variables, a linear system is described by the equation $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$, where A and B are matrices of appropriate dimensions. Such dynamics is *not* allowed in LHA. *Linear hybrid systems* (LHS) refers to the class of hybrid automata where guards, updates, and invariants involve only linear expressions, and the dynamics is specified using linear differential equations. First note that the polyhedral representation is not closed under continuous evolution specified by linear differential equations: if $x = 1$ is the initial state and the dynamics is given by the linear differential equation $\dot{x} = x$, the set of reachable states is the exponential curve given by e^t, for real numbers $t \geq 0$. Since manipulating transcendental functions is computationally difficult, a popular strategy, first advocated by the tool CHECKMATE [14], and later refined by the tool **d/dt** [11], is to compute *over-approximations* of reachable sets using polyhedral representations. Given a polyhedron R representing the set of current states, to compute the set R' of states that are reachable within a fixed time horizon Δ according to a given linear differential equation, the algorithm implements the following strategy. For every corner vertex v of R, it first computes the state v' of the system at time Δ starting in state v. Then it computes the convex hull R_1 of all the corner vertices v of R and their respective images v'. We are guaranteed that all the system trajectories start in the polyhedron R_1 and end in R_1 at time Δ, but are not necessarily inside R_1 at intermediate times. The final step involves computing the polyhedron R_2 obtained by "face-lifting" R_1: the number and normal vectors for facets of R_2 coincide with R_1, but the facets are shifted outwards so as to include all reachable states upto time Δ. All these steps can be efficiently implemented for linear systems, and the resulting polyhedron R_2 is guaranteed to be a superset of the desired set R'. The process can be repeated for successive time intervals, and the resulting approximation of the reachable set is called the *flowpipe* approximation.

The complexity of operations on polyhedra is exponential in the number of dimensions (that is, the number of real-valued variables of the hybrid system), and since such operations are invoked repeatedly in symbolic analysis based on polyhedral representation, a significant body of work has been

aimed at battling this complexity and/or replacing polyhedra with alternative representations [11, 21, 36, 42]. Representation using zonotopes and support functions [22,26] has so far resulted in the most scalable approach for the analysis of linear hybrid systems, leading to the tool SpaceEx that is able to analyze, for instance, a complex 28-dimensional helicopter controller [20].

Deductive verification

In deductive verification, a designer interacts with a mechanized theorem prover to generate proofs of correctness of systems. For safety verification of discrete systems, a classical proof principle relies on the notion of *inductive invariants*: to show that all executions of a system M starting in an initial set I stay within a safe set S, we identify a state property φ such that (1) all initial states satisfy φ; (2) φ is a subset of the desired safety property S; and (3) the property φ is preserved locally across system transitions (that is, no transition of the system changes the value of φ from 1 to 0). In interactive verification, the user proposes a property φ, and the analysis tool checks if φ is an inductive invariant.

The concept of inductive invariants for discrete systems has been generalized and adopted to continuous-time dynamical (and hybrid) systems. We will informally explain the first such notion, called *barrier certificates* [47,48]. To show that a dynamical system M with dynamics $\dot{\mathbf{x}} = f(\mathbf{x})$ with initial set I satisfies a safety property S, we identify a function ψ from the states to reals such that (1) in all initial states, the value of ψ is nonnegative; (2) in all unsafe states (that is, states not in the safe set S), the value of ψ is negative; and (3) the Lie derivate of ψ with respect to the vector field f is positive on the boundary set (called the *barrier*) characterized by $\psi(\mathbf{x}) = 0$. The first two conditions ensure that the barrier separates the initial states from the unsafe states, and the third condition ensures that system trajectories cannot escape the barrier from inside as at the barrier the flow field points inwards. Together, these conditions imply that $\psi(\mathbf{x}) \geq 0$ is an inductive invariant of the system. Typically, the desired function ψ is a polynomial function of the system variables. It is also worth noting that the barrier certificates are closely related to the notion of *Lyapunov certificates* for stability in classical control theory. The notion of *differential invariants* generalizes barrier certificates [44]: it relaxes the third condition, and also allows more general forms of logical assertions as potential invariants.

Verification of safety properties based on identifying inductive invariants avoids the iterative calculation of reachable state sets and is not limited to linear systems. The tool KeYmaera offers support to prove correctness of hybrid systems using deductive verification [43,44]. To check whether a given polynomial certificate satisfies all the conditions necessary for it to be a barrier certificate, the tool needs to perform symbolic differentiation, and calculations such as simplification and quantifier elimination, with formulas in the theory of reals with arithmetic operators. To fully automate deductive verification, such a tool needs to automatically generate candidates for inductive invariants, and this remains an active area of research (see [28,50] for automatic generation of invariants by instantiating templates and [45] for generating invariants by fixpoint computation).

Abstraction

An abstraction A of a model M is a "simplified" model obtained from M such that proving safety and temporal properties of A is a sufficient condition for proving the corresponding properties of M. Abstraction is an effective strategy for scalability of verification tools, provided there is a way to compute A from M in a tool-supported manner, and a way to refine the current abstraction A if it is not adequate to prove the desired properties. In the case of hybrid systems, the simplicity of the abstract model A can be of various forms: A can be discrete while M is continuous; A can have linear dynamics while M has non-linear dynamics; and A can be a linear model of dimensionality lower than that of M. There is an extensive literature on automatic abstraction of hybrid systems. We note three representative examples.

We have already seen that the dynamics admissible in the model of linear hybrid automata is simple enough to permit exact computation of reachable states using polyhedral representation. In *phase portrait* approximation [30], the dynamics $\dot{\mathbf{x}} = f(\mathbf{x})$ in a mode m of a hybrid system is replaced by $\mathbf{l} \leq \dot{\mathbf{x}} \leq \mathbf{u}$, where the vectors $\mathbf{l}$ and $\mathbf{u}$ represent the lower and upper bounds on the function f over the range of values specified by the invariant constraint associated with the mode m. This clearly yields an over-approximation of the allowed system trajectories in each mode. The error introduced by the approximation can be reduced if we split the mode m into submodes, each corresponding to a different region of the state-space.

Predicate abstraction is a powerful technique for extracting finite-state models from complex, potentially infinite-state, systems, and has been extended and adopted for hybrid systems [4, 16]. In this approach, the input to the verification tool consists of a linear hybrid system, the safety property to be verified, and a finite set of Boolean predicates over system variables to be used for abstraction. An abstract state is a valid combination of truth values to the Boolean predicates, and thus, corresponds to a polyhedral set of the concrete state-space. The verifier performs an on-the-fly search of the abstract system by symbolic manipulation of polyhedra, where the computation of continuous-time successors of abstract states can be performed using flow-pipe approximations. The key computational benefit is that the continuous reachability computation is applied only to an abstract state, instead of intermediate sets of arbitrary complexity generated during iterative computation. If the initial choice of predicates is too coarse, the search finds abstract counter-examples that are infeasible in the original hybrid system, and such counter-examples can be analyzed to discover new predicates that will rule out related spurious counter-examples.

A classical notion of equivalence of nondeterministic systems is *simulation*: a relation between states of two systems is a simulation relation, if (1) two related states have identical observations, and (2) whenever two states are related, for every transition from the first state, there exists a matching transition from the second state such that the targets of the transitions remain related. For simpler classes of hybrid sys-

tems such as timed automata and O-minimal systems, one can algorithmically compute the maximal simulation relation over the states of a given system, and use the discrete quotient with respect to this relation as the abstract system which can replace the original system for verification purposes [8]. In the context of hybrid systems, since states contain real-valued vectors, there is a natural metric over states, and this can be used to also define a coarser notion of simulation called *approximating simulations*: an approximating simulation relation with parameter ε requires observations of two related states to be ε-close of one another, and transitions can be matched step-by-step by staying ε-close [23]. The resulting theory of approximating relations leads to algorithms for constructing lower-dimensional abstractions of linear systems [52].

2. EMERGING RESEARCH DIRECTIONS

Automated verification is a computationally intractable problem. Consequently, even though there are many demonstrations of interesting analyses using tools for verification of hybrid systems, scalability remains a challenge, and a significant fraction of the current research is aimed at addressing this challenge. A complementary challenge is to integrate verification tools and techniques in the design flow so as to improve the overall system reliability. We conclude this article by discussing some promising research directions towards this goal.

Symbolic simulation

In simulation, a possible execution of the model upto a finite time horizon is obtained using numerical methods, and this is a well-accepted industrial practice. A single simulation corresponds to a specific choice of inputs. A promising idea is to analyze a simulation trace corresponding to a particular choice of inputs using symbolic analysis techniques to compute the space of inputs that are close enough to the chosen one so that no inputs from this space need to be considered for subsequent simulations (see [9, 18, 34] for recent efforts). This integration of simulation and symbolic analysis can lead to improved coverage, and is similar to *concolic testing* which has proved to be effective in debugging of large-scale software systems [24]. An added benefit is that such an approach can be implemented directly within the native simulation engine of an industrial-strength modeling environment such as Simulink/Stateflow.

Synthesis

Historically, synthesis refers to the process of computing an implementation (the "how") from a specification of the desired behavior and performance (the "what") and the assumptions on the environment (the "where"). In the more recent view, the synthesis tool facilitates the design by consistently integrating different views: a designer expresses her insights about the design using synthesis artifacts of different kinds such as models that may contain ambiguities and declarative specifications of high-level requirements, and the synthesis tool composes these different views about the structure and functionality of the system into a unified concrete implementation using a combination of algorithmic techniques. Illustrative examples of this new view of synthesis include programming by examples for spreadsheet transformations in Microsoft Excel [27], and *sketching* of bit-streaming programs using program skeletons [51]. We believe that these ideas emerging in the programming languages community should be explored in the context of design of hybrid systems to integrate synthesis in a pragmatic way in the design cycle (see [53] for recent work on synthesizing switching conditions in hybrid automata).

From models to code

Generating embedded software directly from high-level models, such as hybrid systems, is appealing, but challenging due to the wide gap between the two. In current practice, this gap is bridged with significant manual effort by exploiting the run-time support offered by operating systems for managing tasks and interrupts. A key challenge to systematic software generation from hybrid models is to ensure that one can infer properties of the software from the properties of the model, and this problem is receiving increasing attention from researchers. Sample research directions include integration of control and scheduling [10] and static analysis of errors introduced by finite precision computations [25].

Industrial applications

The value of formal modeling and verification on industrially relevant problems has been demonstrated on a number of case studies. Examples of these include design and analysis of vehicle platooning protocols [17], identification of optimal tolerances for audio control protocol [31], safety verification of collision avoidance protocol for aircrafts [46, 54], and verification of adaptive cruise control [39]. Yet, the level of commitment from embedded software industry remains limited to exploratory projects in collaboration with academic researchers. This is in contrast to, say, Intel's investment in formal hardware verification and Microsoft's investment in static analysis of software, which can be attributed to the identification of specific classes of errors that can be largely eliminated using verification techniques (for example, deadlocks in cache coherence protocols and misuse of API rules by third-party device driver code). Thus, a key challenge for research in formal verification of hybrid systems is to identify a compelling class of errors that designers routinely make and can be eliminated using verification techniques. An alternative path to industrial adoption is to integrate verification tools in the certification process, and this seems plausible in safety-critical domains such as software for medical devices [38].

Acknowledgments: This research was partially supported by NSF awards CNS 0931239, CNS 1035715, and CCF 0915777. We thank Oded Maler and André Platzer for their feedback on this article.

3. REFERENCES

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[2] R. Alur, C. Courcoubetis, T. Henzinger, and P. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume LNCS 736, pages 209–229. Springer-Verlag, 1993.

[3] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. *Proceedings of the IEEE*, 91(1), 2003.

[4] R. Alur, T. Dang, and F. Ivancic. Predicate abstraction for reachability analysis of hybrid systems. *ACM Transactions on Embedded Computing Systems*, 5(1):152–199, 2006.

[5] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[6] R. Alur and T. Henzinger. Modularity for timed and hybrid systems. In *CONCUR '97: Eighth International Conference on Concurrency Theory*, LNCS 1243, pages 74–88. Springer-Verlag, 1997.

[7] R. Alur, T. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22(3):181–201, 1996.

[8] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, 2000.

[9] R. Alur, A. Kanade, S. Ramesh, and K. Shashidhar. Symbolic analysis for improving simulation coverage of Simulink/Stateflow models. In *Proceedings of the 8th Annual ACM Conference on Embedded Software (EMSOFT)*, pages 89–98, 2008.

[10] R. Alur and G. Weiss. RTComposer: a framework for real-time components with scheduling interfaces. In *Proceedings of the 8th ACM & IEEE International Conference on Embedded Software*, pages 159–168, 2008.

[11] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control, Third International Workshop*, LNCS 1790, pages 21–31. Springer, 2000.

[12] 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. *Sci. Comput. Program.*, 72(1-2):3–21, 2008.

[13] T. Ball, V. Levin, and S. K. Rajamani. A decade of software model checking with SLAM. *Commun. ACM*, 54(7):68–76, 2011.

[14] A. Chutinan and B. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *Hybrid Systems: Computation and Control, Second International Workshop*, LNCS 1569, pages 76–90. Springer, 1999.

[15] E. M. Clarke, E. A. Emerson, and J. Sifakis. Model checking: algorithmic verification and debugging. *Commun. ACM*, 52(11):74–84, 2009.

[16] E. M. Clarke, A. Fehnker, Z. Han, B. H. Krogh, O. Stursberg, and M. Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *Tools and Algorithms for the Construction and Analysis of Systems, 9th International Conference*, LNCS2619, pages 192–207, 2003.

[17] A. Deshpande, A. Göllu, and P. Varaiya. SHIFT: a formalism and a programming language for dynamic networks of hybrid automata. In *Hybrid Systems III*, LNCS 1567. Springer, 1996.

[18] A. Donzé and O. Maler. Systematic simulation using sensitivity analysis. In *Hybrid Systems: Computation and Control, 10th International Conference*, LNCS 4416, pages 174–189. Springer, 2007.

[19] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Luvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity–the Ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.

[20] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. SpaceEx: Scalable verification of hybrid systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV)*, LNCS 6806, pages 379–395. Springer, 2011.

[21] A. Girard. Reachability of uncertain linear systems using Zonotopes. In *Hybrid Systems: Computation and Control, 8th International Workshop*, LNCS 3414, pages 291–305. Springer, 2005.

[22] A. Girard, C. L. Guernic, and O. Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In *Hybrid Systems: Computation and Control, 9th International Workshop*, LNCS 3927, pages 257–271, 2006.

[23] A. Girard and G. Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, 2007.

[24] P. Godefroid, N. Klarlund, and K. Sen. DART: directed automated random testing. In *Proceedings of the ACM Conference on Programming Language Design and Implementation*, pages 213–223, 2005.

[25] E. Goubault and S. Putot. Static analysis of finite precision computations. In *Verification, Model Checking, and Abstract Interpretation - 12th International Conference*, LNCS 6538, pages 232–247, 2011.

[26] C. L. Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *Computer Aided Verification, 21st International Conference*, LNCS 5643, pages 540–554, 2009.

[27] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of 38th ACM Symposium on Principles of Programming Languages*, pages 317–330, 2011.

[28] S. Gulwani and A. Tiwari. Constraint-based approach for analysis of hybrid systems. In *Computer Aided Verification, 20th International Conference*, LNCS 5123, pages 190–203, 2008.

[29] T. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th IEEE Symposium on Logic in Computer Science*, pages 278–293, 1996.

[30] T. Henzinger and P. Ho. Algorithmic analysis of nonlinear hybrid systems. In *Proceedings of the Seventh Conference on Computer-Aided Verification*, LNCS 939, pages 225–238. Springer-Verlag, 1995.

[31] T. Henzinger, P. Ho, and H. Wong-Toi. HyTech: a model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1-2):110–122, 1997.

[32] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata. In

Proceedings of the 27th ACM Symposium on Theory of Computing, pages 373–382, 1995.

[33] T. Henzinger and J. Sifakis. The embedded systems design challenge. In *FM 2006: 14th International Symposium on Formal Methods*, LNCS 4085, pages 1–15, 2006.

[34] A. Julius, G. Fainekos, M. Anand, I. Lee, and G. Pappas. Robust test generation and coverage for hybrid systems. In *Hybrid Systems: Computation and Control, 10th International Conference*, LNCS 4416, pages 329–342. Springer, 2007.

[35] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145–164, 2003.

[36] A. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *Hybrid Systems: Computation and Control, Third International Workshop*, LNCS 1790, pages 202–214. Springer, 2000.

[37] E. Lee. What's ahead for embedded software. *IEEE Computer*, pages 18–26, September 2000.

[38] I. Lee and O. Sokolsky. Medical cyber physical systems. In *Proc. 47th Design Automation Conference*, pages 743–748, 2010.

[39] S. M. Loos, A. Platzer, and L. Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In *FM 2011: Formal Methods, 17th International Symposium on Formal Methods*, LNCS 6664, pages 42–56. Springer, 2011.

[40] N. Lynch, R. Segala, F. Vaandrager, and H. Weinberg. Hybrid I/O automata. In *Hybrid Systems III: Verification and Control*, LNCS 1066, pages 496–510, 1996.

[41] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In *Real-Time: Theory in Practice, REX Workshop*, LNCS 600, pages 447–484. Springer-Verlag, 1991.

[42] I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In *Hybrid Systems: Computation and Control, Third International Workshop*, LNCS 1790, pages 310–323. Springer, 2000.

[43] A. Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reasoning*, 41(2):143–189, 2008.

[44] A. Platzer. *Logical Analysis of Hybrid Systems - Proving Theorems for Complex Dynamics*. Springer, 2010.

[45] A. Platzer and E. M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In *Computer Aided Verification, 20th International Conference*, LNCS 5123, pages 176–189, 2008.

[46] A. Platzer and E. M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In *FM 2009: Formal Methods*, LNCS 5850, pages 547–562, 2009.

[47] S. Prajna and A. Jadbabaie. Safety verification of hybrid systems using barrier certificates. In *Hybrid Systems: Computation and Control, 7th International Workshop*, LNCS 2993, pages 477–492, 2004.

[48] S. Prajna, A. Jadbabaie, and G. J. Pappas. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE Transactions on Automatic Control*, 52(8):1415–1429, 2007.

[49] A. Sangiovanni-Vincentelli. Quo Vadis SLD: Reasoning about trends and challenges of system-level design. *Proceedings of the IEEE*, 95(3):467–506, 2007.

[50] S. Sankaranarayanan, H. B. Sipma, and Z. Manna. Constructing invariants for hybrid systems. *Formal Methods in System Design*, 32(1):25–55, 2008.

[51] A. Solar-Lezama, R. Rabbah, R. Bodík, and K. Ebcioglu. Programming by sketching for bit-streaming programs. In *Proc. 2005 ACM Conference on Programming Language Design and Implementation*, pages 281–294, 2005.

[52] P. Tabuada. *Verification and control of hybrid systems*. Springer, 2009.

[53] A. Taly and A. Tiwari. Switching logic synthesis for reachability. In *Proceedings of the 10th International Conference on Embedded software*, pages 19–28, 2010.

[54] C. Tomlin, G. J. Pappas, and S. Sastry. Conflict resolution for air traffic management: A study in muti-agent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, 1998.