Invisible Formal Methods for Embedded Control Systems

A. Tiwari, N. Shankar, and J. Rushby

***Abstract*— Embedded control systems typically comprise continuous control laws combined with discrete mode logic. These systems are modeled using a hybrid automaton formalism, which is obtained by combining the discrete transition system formalism with continuous dynamical systems. This paper develops automated analysis techniques for asserting correctness of hybrid system designs. Our approach is based on symbolic representation of the state space of the system using mathematical formulas in an appropriate logic. Such formulas are manipulated using symbolic theorem proving techniques.**

It is important that formal analysis should be unobtrusive and acceptable to engineering practice. We motivate a methodology called "invisible formal methods" that provides a graded sequence of formal analysis technologies ranging from extended typechecking, through approximation and abstraction, to model checking and theorem proving. As an instance of invisible formal methods, we describe techniques to check inductive invariants, or extended types, for hybrid systems and compute discrete finite state abstractions automatically to perform reachability set computation. The abstract system is sound with respect to the formal semantics of hybrid automata. We also discuss techniques for performing analysis on non-standard semantics of hybrid automata. We also briefly discuss the problem of translating models in Simulink/Stateflow language, which is widely used in practice, into the modeling formalisms, like hybrid automata, for which analysis tools are being developed.

***Index Terms*—Hybrid dynamical systems, Inductive invariants, Abstraction.**

I. Introduction

HYBRID systems involve a combination of discrete and continuous dynamics and are used for modeling embedded control systems. Many of the embedded control systems are safety critical and require formal guarantees of safe operation. Formal design and analysis of hybrid system models has received much attention in the research community recently, from both the computer science and control theory worlds.

The systems that have been traditionally studied in the computer science community have been discrete. Such systems evolve in discrete time steps over a countable state space and are formalized using *discrete transition systems*. Typical examples include hardware circuit designs, network communication protocols, and software components. Good advances have been made in the techniques and tools for analyzing discrete transition systems [9, 24]. Some of the most effective techniques include *model checking* and *abstraction*. Abstraction is typically used to reduce the possibly infinite state space system into a finite state space abstract system, and model checking is subsequently used to exhaustively search through all behaviors of the finite abstraction.

Hybrid systems differ from purely discrete systems in that they also contain a continuous component. Such systems evolve in continuous time with discrete jumps at particular time instances. The techniques developed for discrete systems are thus not directly applicable. First, the state space now is uncountably infinite. Second, a continuous evolution results in uncountably many successor states from a given state of the hybrid system. Furthermore, the problem of checking if a hybrid system ever reaches a certain bad state is known to be intractable (undecidable) for even simple classes of hybrid systems (for example, systems whose continuous dynamics involves variables that proceed at two constant slopes [18]).

One approach to overcoming undecidability involves restricting the continuous dynamics of the hybrid system so that suitable abstractions can be successfully applied to yield conservative discrete transition systems. Timed automata [2], multirate automata [1], and rectangular automata [18] are some such examples. Another approach is to restrict the discrete transitions and the continuous flows so that finite abstractions can again be constructed. The idea of o-minimal hybrid systems [25] is motivated by this.

There is a huge gap between the interesting large and complex systems that are typically used in practice, and the restricted and simple systems that are tractable by known automated analysis techniques. Moreover, the available techniques are still far removed from the tools engineers most often use in practice to design embedded control systems. As a result, huge amounts of effort is spent on validating the models through extensive simulation and testing on particular scenarios developed by the engineer. In this paper, we describe a range of formal analysis technology to incrementally establish properties that hold for all behaviors of the model, thus replacing the tedious task of testing.

Our approach is based on using a symbolic representation for the state space of the system. In contrast to simulation, where a state is represented by the numerical values for the state variables, we represent states symbolically using a mathematical formula over some language. Whereas states are updated using numerical techniques in simulation, the symbolic representation of states is manipulated using symbolic inferential analysis in our approach.

Typechecking is a simple kind of formal analysis technology that has been successfully used for early error detection

Research supported by DARPA under the MoBIES and SEC programs administered by AFRL under contracts F33615-00-C-1700 and F33615-00-C-3043, and the NASA Langley Research Center contract NAS1-00108 to Rannoch Corporation.

All authors are with the Computer Science Laboratory, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, U.S.A. E-mail: {tiwari,shankar,rushby}@csl.sri.com

in software development. The simple notion of types, where each state variable is specified to be either an integer, real, or boolean, can be extended to a more general notion, say that of non-zero integers, positive reals, and even to more complex subsets specified using formulas. Typechecking these more complex type specifications is equivalent to inductive invariant checking. For discrete transition systems, this can be accomplished by verifying validity of certain formulas, see Section V-C. In Section V-D we show how to generalize this to hybrid systems.

Typechecking is a simple, though powerful, technique for early error detection in designs. However, it might not be sufficient to establish all the correctness properties of interest in the design. A hybrid dynamical system evolves in time via different trajectories through its state space. The complete behavior of the system is given by the set of all its trajectories under all possible input vectors. Simulation techniques test single trajectories from this set. An exhaustive search through all the system behaviors over the state space is required to verify any given property of the system. However, for hybrid systems, owing to the presence of continuous dynamics, it is not clear how this process, called model-checking, can be done. In Section V-E we describe the technique of creating sound finite state abstractions for the hybrid system, which preserve the behaviors of the system relevant to the property of interest. In the final step, the finite state system is model-checked against the property.

The paper is organized as follows. Discrete transition systems, continuous dynamical systems, and hybrid automata are formalized in Sections II, III, and IV respectively. While these are mathematical models, in practice tools like Simulink and Stateflow from MathWorks Inc. are used to create models of hybrid systems. The issues related to bridging the gap between designs in Simulink/Stateflow and the mathematical formalisms are discussed in Section IV-A. Section V is devoted to a description of a broad range of analysis techniques and Section VI presents related work and conclusion.

A. Notation

We denote the set of natural numbers by $\mathbb{N}$, rationals by $\mathbb{Q}$, and reals by $\mathbb{R}$. We use capital letters, like X, Y, with possible subscripts, to denote sets of real variables and Q to denote a set of boolean variables. A valuation of a set X of real variables is an assignment of real values to the variables in X and similarly, a valuation of a set Q of boolean variables is an assignment of boolean values (true, false) to the variables in Q. We denote the set of all valuations of X and Q by $\mathbf{X}$ and $\mathbf{Q}$, respectively. Clearly, $\mathbf{X}$ is isomorphic to $\mathbb{R}^{|X|}$ and $\mathbf{Q}$ is isomorphic to $2^{|Q|}$. If X and Q represent the set of all real and boolean variables in a system, then the set $\mathbf{Q} \times \mathbf{X}$ represents the *state space* of this system.

Our formal analysis approach is based on representing subsets of the state space symbolically by a first-order formula over variables X and Q. Such formulas are constructed using symbols from a set of function symbols $\{+, -, \cdot\}$, constants $\mathbb{Q}$, and predicate symbols $\{=, >, \geq, <, \leq\}$. The set of terms over a set X of variables corresponds to the set of polynomials $\mathbb{Q}[X]$ over the variables X and coefficients from $\mathbb{Q}$. The set $ATM(X)$, defined as $\{p \sim 0 : p \in \mathbb{Q}[X] \text{ and } \sim \in \{=, <, \leq, >, \geq\}\}$, is the set of all *atomic* formulas. The set $WFF(X)$ of first-order formulas (over X) is defined as the smallest set containing $ATM(X)$ and closed under the boolean operations (conjunction $\wedge$, disjunction $\vee$, implication $\Rightarrow$, and negation $\neg$) and quantification (existential $\exists$ and universal $\forall$). The *first-order theory* of *reals*, denoted by $\Re$, is defined as the set of all first-order formulas over the above signature (and a countable set of variables) that are true over the real numbers. We use the notation $\Re \models \phi$ to denote the fact that the (first-order) formula ϕ is true in the theory of reals. The first-order theory of the real closed fields is a complete theory, that is every sentence in $WFF(X)$ is either true or its negation is true in this theory, and is known to be decidable [34, 13].

Since hybrid systems will be described using both real valued variables X and boolean variables Q, we extend the set of atomic formulas to also include the set $\{true, false\} \cup Q$. The resulting first-order theory is then an extension of the theory of real closed fields with boolean variables and constants. This resulting theory, denoted by $\Re^e$, is still decidable, since the boolean variables can only take values in a 2-valued domain $\{true, false\}$. We denote formulas in $WFF(X, Q)$ by ϕ, ψ, possibly with subscripts and use p to denote polynomials in the set $\mathbb{Q}[X]$. We say a polynomial p occurs in a formula ϕ if there is an atomic formula $p \sim 0$ in ϕ. The rest of the notation follows the standard practice in hybrid systems literature.

II. Discrete Transition Systems

A discrete state transition system DS is a tuple $(Q, Init, t)$ where Q is a finite set of boolean variables, $\mathbf{Q}$ denotes the (countable) set of all valuations of the variables Q, $Init \subseteq \mathbf{Q}$ is a set of initial states, and $t \subseteq \mathbf{Q} \times \mathbf{Q}$ is a set of transitions. The semantics, $[\![DS]\!]$, of a discrete state transition system $DS = (Q, Init, t)$ is the collection of all mappings $\theta : \mathbb{N} \mapsto \mathbf{Q}$ satisfying

(a) initial condition: $\theta(0) \in Init$, and
(b) discrete evolution: for all $i \in \mathbb{N}$, $(\theta(i), \theta(i+1)) \in t$.

The transition relation t can be specified in many different ways. Typically, it is given using a guarded command syntax:

$$\phi(Q) \quad \longrightarrow \quad \bigwedge_i q_i' = e_i(Q)$$

with the meaning that there is a transition from state $\mathbf{q} \in \mathbf{Q}$ to $\mathbf{q}' \in \mathbf{Q}$ whenever the valuation $\mathbf{q}$ makes the guard expression $\phi(Q)$ evaluate to true and $\mathbf{q}'$ is obtained from $\mathbf{q}$ using the assignments, on the right-hand side of the arrow above. There is one assignment $q_i' = e_i(Q)$ for each $q_i \in Q$. Here $e_i(Q)$ is a boolean formula over the variables Q. Note that q_i' denotes the value of state variable q_i after the transition is completed.

The guard $\phi(Q)$ can be any boolean formula over the boolean variables in Q. Later, when we consider hybrid systems over boolean variables Q and real variables X, the guard would be any formula in $WFF(Q, X)$. Here we have assumed Q to only contain boolean variables. In general, Q could contain integer variables, and variables interpreted over finite domains as well.

III. Continuous Dynamical Systems

A continuous dynamical system CS is a tuple $(X, Init, f)$ where X is a finite set of variables interpreted over the reals

$\mathbb{R}$, $\mathbf{X} = \mathbb{R}^X$ is the set of all valuations of the variables X, $Init \subseteq \mathbf{X}$ is the set of initial states, and $f : \mathbf{X} \mapsto T\mathbf{X}$ is a vector field that specifies the continuous dynamics. Here $T\mathbf{X}$ denotes the tangent space of $\mathbf{X}$. We assume that f satisfies the standard assumptions for existence and uniqueness of solutions to ordinary differential equations. Note that the continuous dynamical systems we consider here are autonomous, that is, they have no inputs.

The semantics, $[\![CS]\!]$, of a continuous dynamical system $CS = (X, Init, f)$ over an interval $I = [\tau_a, \tau_z] \subseteq \mathbb{R}$ is a collection of mappings $\sigma : I \mapsto \mathbf{X}$ satisfying

(a) initial condition: $\sigma(\tau_a) \in Init$, and
(b) continuous evolution: for all $\tau \in (\tau_a, \tau_z)$, $\dot{\sigma}(\tau) = f(\sigma(\tau))$.

In case the interval I is left unspecified, it is assumed to be the interval $[0, \infty)$.

We assume that the flow derivative, f, is specified using polynomial expressions over the state variables X, that is $f \in (\mathbb{Q}[X])^{|X|}$, where $\mathbb{Q}[X]$ denotes the set of polynomials over the indeterminates X and coefficients in $\mathbb{Q}$, and $|X|$ denotes the cardinality of X. These polynomials can be nonlinear in general.

Example 1: As an example of a purely continuous system, we consider simplified leader control from the design of automated highway systems [33]. Suppose vehicle A is following vehicle B in a lane. Let gap denote the distance between the two vehicles, v_0 be the velocity of vehicle A, a_0 be the acceleration of vehicle A, and v_1 be the velocity of vehicle B. In the leader control mode, vehicle A follows vehicle B by suitably adjusting its acceleration a_0 based on the sensor readings for gap, v_0, v_1, and a_0. Let us assume that the dynamics of the system are given by the following equations:

$$\begin{aligned} \dot{v_0} &= a_0 \\ \dot{a_0} &= -4v_0 + 3v_1 - 3a_0 + gap \\ \dot{gap} &= v_1 - v_0 \end{aligned}$$

We consider the velocity v_1 of vehicle B as a parameter, that is, an unspecified symbolic constant. We have changed variables in the above to make $[0, 0, 0, 0]$ an equilibrium point so that the two cars collide when $gap = -10$. Let us say that the initialization condition, or equivalently the condition under which this control mode is triggered, is given by $a_0 = 0 \wedge 10 \leq v_0, v_1 \leq 20 \wedge gap \geq 20$. The problem is to show that the rear car does not crash into the car in front, that is, $gap \geq -10$ at all times. In Example 7 we show how to *prove* this automatically.

Example 2: In the same setting as Example 1, consider now that the following simplified dynamics of the system:

$$\begin{aligned} \dot{v_1} &= a_1 \\ \dot{v_0} &= a_1 + v_1 - v_0 \\ \dot{gap} &= v_1 - v_0 \end{aligned}$$

Let us say that the initialization condition, or equivalently the condition under which this control mode is triggered, is given by $gap \geq 2$ and $v1 \geq v0$. The problem is to show that the rear car does not crash into the car in front, that is, $gap \geq 0$ at all times. In this case, we assume that the velocity of the front car is changing according to an uncontrolled *input* a_1.

We remark here that although we have excluded parameters and input variables in the formal definition of continuous dynamical systems (to keep the definition simple), they can be easily incorporated into the analysis methods we shall outline later in Section V. A simulation of the above systems given in Examples 1 and 2 for a specific initial valuation for the state variables can be done using available tools, like Simulink. However, multiple simulation runs can not provably demonstrate the safety of the control laws. See Examples 4 and 7 for an analysis using symbolic techniques that reason about the complete state space.

A. Simulink Simulation Semantics

For purposes of simulation, several tools use either a fixed-step or variable-step numerical solver and approximate the differential equation by a difference equation. Assuming the use of a fixed-step solver with step size δ, the fixed-step (non-standard) semantics of the continuous dynamical system $(X, Init, f)$ over an interval $I = [\tau_a, \tau_z]$ is the collection of mappings $\sigma : I \mapsto \mathbf{X}$ satisfying condition (a) from before and the new condition

(b) continuous evolution: for all natural numbers n such that $0 \leq n < (\tau_z - \tau_a)/\delta$, $\sigma(\tau_a + (n+1)\delta) = \sigma(\tau_a + n\delta) + \delta f(\sigma(\tau_a + n\delta))$, and $\sigma(\tau_a + n\delta + \delta') = \sigma(\tau_a + n\delta)$ for all $0 < \delta' < \delta$.

In case of a variable-step solver, the variable-step (non-standard) semantics of the continuous dynamical system $(X, Init, f)$ over an interval $I = [\tau_a, \tau_z]$ is the collection of mappings $\sigma : I \mapsto \mathbf{X}$ satisfying condition (a) from before and the new condition

(b) continuous evolution: for all $\tau \in (\tau_a, \tau_z]$, there exists a $\delta > 0$ such that $\sigma(\tau) = \sigma(\tau - \delta) + \delta f(\sigma(\tau - \delta))$.

There is usually an upper-bound on δ in this case.

In Section V, we describe different techniques that handle the continuous dynamics in different ways, based on the what semantics we wish to use for analysis.

IV. Hybrid Systems

Hybrid systems involve the interaction of discrete and continuous dynamics. They are formalized using hybrid automata, which combine a discrete transition system with a continuous dynamical system. An autonomous hybrid automaton HS is a tuple $(Q, X, Init, t, f)$, where Q is a finite set of discrete variables, X is a set of continuous variables, $Init \subseteq \mathbf{Q} \times \mathbf{X}$ is a set of initial states, $t \subset \mathbf{Q} \times \mathbf{X} \times \mathbf{Q} \times \mathbf{X}$ is a set of (guarded) discrete transitions, $f : \mathbf{Q} \mapsto (\mathbf{X} \mapsto T\mathbf{X})$ is a mapping from the discrete states to vector fields that specify the continuous flow in that discrete state. We refer to $(\mathbf{q}, \mathbf{x}) \in \mathbf{Q} \times \mathbf{X}$ as the *state* of the hybrid automaton HS.

The semantics of hybrid automata is defined in terms of discrete and continuous evolutions. Formally, the semantics of an hybrid automaton $HS = (Q, X, Init, t, f)$ over an interval $I = [\tau_a, \tau_z]$ is a collection $[\![HS]\!]$ of *runs* $\sigma : \mathbf{I} \mapsto \mathbf{Q} \times \mathbf{X}$, where $\mathbf{I}$ is a dense interval defined as the multiset union $\cup_{i=0}^{l+1}[\tau'_{i-1}, \tau_i]$ of an ordered collection, called a *hybrid time trajectory*, $\{[\tau_a, \tau_0], [\tau'_0, \tau_1], \ldots, [\tau'_{l-1}, \tau_l], [\tau'_l, \tau_z]\}$ with $\tau_a =$

$\tau'_{-1} \leq \tau_0 = \tau'_0 \leq \tau_1 = \tau'_1 \leq \cdots \leq \tau_l = \tau'_l \leq \tau_z = \tau_{l+1}$, satisfying

(a) initial condition: $\sigma(\tau_a) \in \mathit{Init}$,

(b) continuous evolution: for all i such that $\tau'_i \neq \tau_{i+1}$, the projection σ_X of σ over X is continuous over $[\tau'_i, \tau_{i+1}]$ and for all $\tau \in (\tau'_i, \tau_{i+1})$ such that $\tau'_i \neq \tau_{i+1}$, $\dot{\sigma_X}(\tau) = f(\sigma_Q(\tau'_i))(\sigma_X(\tau))$ with $\sigma_Q(\tau)$ remaining unchanged,

(b') for all i such that $\tau'_i = \tau_{i+1}$, $\sigma(\tau'_i) = \sigma(\tau_{i+1})$, and

(c) discrete evolution: for all i such that $0 \leq i \leq l$, it is the case that $(\sigma_Q(\tau_i), \sigma_X(\tau_i), \sigma_Q(\tau'_i), \sigma_X(\tau'_i)) \in t$.

Here we have used $\sigma_Q : \mathbf{I} \mapsto \mathbf{Q}$ and $\sigma_X : \mathbf{I} \mapsto \mathbf{X}$ to denote the projections of $\sigma : \mathbf{I} \mapsto \mathbf{Q} \times \mathbf{X}$ so that $\sigma(\tau) = (\sigma_Q(\tau), \sigma_X(\tau))$. Thus, a hybrid automaton can either make discrete jumps at time points possibly changing its discrete and continuous state components (condition (c)), or evolve according to the flow equations in time while keeping the discrete state component unchanged (condition (b)). Condition (b') allows for the possibility of multiple discrete jumps at the same time instance.

These are the standard semantics of hybrid automata. However, one can consider other options where the continuous dynamics are interpreted using the simulation semantics given earlier. Again, for simplicity, we have assumed that there are no inputs, parameters, and invariant sets, which can all be handled by the analysis methods described later. A parameter behaves just like other variable except that its value remains unchanged through a run of the hybrid system. An invariant set for a discrete mode specifies a region of state space that the system can not leave while it is in that mode.

Example 3: As a simple example of a hybrid system, consider a thermostat that controls the temperature x of a room. The thermostat senses the temperature and turns a heater on and off if the threshold values x_{min} and x_{max} are reached, where $0 < x_{min} < x_{max}$ and $x_{min}, x_{max} \in \mathbb{R}^+$. When the heater is off, the temperature of the room decreases and when the heater is turned on, the temperature increases according to the following dynamics:

$$\mathit{off} \quad : \quad \dot{x} = -Kx \qquad \mathit{on} \quad : \quad \dot{x} = -K(x - h)$$

Here, the parameter $K \in \mathbb{R}^+$ is the room constant and the parameter $h > x_{min}, x_{max}$ is a real-valued constant that depends on the power of the heater.

The discrete logic to switch between these two modes is given by the following two guarded transitions:

$$\begin{array}{l} \mathit{state} = \mathit{off} \ \wedge \ x \leq x_{min} \ \longrightarrow \ \mathit{state}' = \mathit{on} \\ \mathit{state} = \mathit{on} \ \wedge \ x \geq x_{max} \ \longrightarrow \ \mathit{state}' = \mathit{off} \end{array}$$

Let us say that the initial condition is given by $x \geq x_{min} \ \wedge \ x \leq x_{max}$ and the heater is off. This thermostat example can be formally modeled as the hybrid automaton $HS = (Q, X, \mathit{Init}, t, f)$, where $Q = \{q_1\}$ (here the boolean variable q_1 is true when the heater is on, and false when it is off), $X = \{x_1\}$ (here x_1 is the temperature of the room), $\mathit{Init} = \{(q_1 = \mathit{false}, x_1 = x) : x_{min} < x < x_{max}\}$, $t = \{(q_1 = \mathit{true}, x_1 = x, q_1 = \mathit{false}, x_1 = x) : x \geq x_{max}\} \cup \{(q_1 = \mathit{false}, x_1 = x, q_1 = \mathit{true}, x_1 = x) : x \leq x_{min}\}$, and $f(q_1 = \mathit{true}) = -K(x - h)$ and $f(q_1 = \mathit{false}) = -Kx$. Note that this is a parametric hybrid automaton with parameters K, h, x_{min}, and x_{max}. There are also invariant sets in the two modes which make sure that the discrete transitions are taken when they are enabled.

A. Expressive Power of the Modeling Formalism

One of the most extensively used tools for modeling, simulation, and rapid prototyping of control designs for embedded applications is the Simulink/Stateflow development suite provided by MathWorks Inc. Hence, it is of considerable interest to study the problem of translating models in Simulink/Stateflow to the kinds of formalisms for which we are building analysis capabilities. This would offer benefits in early error detection and more complete assurance of the designs in Simulink/Stateflow. But this dream is hampered by the lack of formal and rigorous semantics for the modeling language of this tool. It is potentially valuable, therefore, to provide formal semantics and to develop formal analysis techniques for important features of the modeling language provided by the MathWorks tool.

MathWorks' Simulink/Stateflow development suite consists of two modeling languages: Simulink is used to model the continuous dynamics and Stateflow is used to specify the discrete control logic and the modal behavior of the system. We have developed a formal semantics for the Stateflow modeling language by translating a Stateflow model into a set of *communicating pushdown automata*. We skip the details of this translation in this paper and refer the reader to [35] for details. The one non-trivial feature of this translation was the need of an infinite stack, which we describe briefly.

The Stateflow modeling language is based on hierarchical state machines with discrete transitions between states. A Stateflow chart consists of states, transitions between states, events that enable or disable transitions, and a hierarchy on the states. A state could itself consist of several substates with transition between them and can be classified as AND- or OR-state. The semantics of Stateflow models is specified informally through examples in the Matlab documents. Broadly speaking, an input event e causes execution of the root state. A state *executes* by firing any of its transitions that can be fired. If none of the transitions can be fired, the state causes execution of its (either one or all, depending on if it is an OR-state or AND-state) descendants. When the descendents have finished executing, the control returns to the parent state. A transition can fire if the condition on it is true (and the triggering event present). A transition fires by inactivating the source state (recursively if it is an non-leaf AND- or OR-state), performing condition actions, activating the destination state, and performing transition actions. Actions could change the value of a variable or broadcast an event. Broadcasting an event is like a function call invocation in imperative programming languages. Hence, there is a need of a global stack to keep track of the control passing information during the execution of a Statechart model. Each state is mapped onto a pushdown automaton (a pushdown automata is a discrete transition system, but with access to an infinite stack) and the full Stateflow chart gets translated into a composition of these pushdown automata, which is itself a single pushdown automaton.

The informal semantics of Stateflow is clearly different from the semantics of Statecharts. Stateflow works only on one

event at a time and there is no notion of "maximal and non-conflicting" transitions. Event broadcasting is recursive. Moreover, after an event is processed, control needs to return to the state that generated that event. We need a stack to store this additional information. The communication between different pushdown automata allows for passing of control between any two automata and not only between automata adjacent in the *hierarchy* specified in the original model. This is required for the translation of important features like *supertransitions* and *directed event broadcasting* in the Stateflow language. In addition, the automata share a global pushdown stack that is used to keep track of events that have been broadcast.

The Stateflow chart represented as a pushdown system can be statically analyzed. By using the algorithm for reachability in pushdown systems [10], it can be determined if a pushdown system requires a bounded or an unbounded stack depth. An unbounded stack depth corresponds to infinite recursive event broadcasting in the Stateflow charts. The Simulink/Stateflow tool detects loops in event broadcasting at simulation time. To perform the bounded stack depth analysis, all non-boolean data variables are abstracted and the analysis is performed on a finite state abstraction. The pushdown system can also be analyzed to detect any nondeterminacy in the Stateflow chart and other such properties. All of this analysis can be performed in an completely automated and non-intrusive way. For some theoretical results on analysis of recursive state machines, the reader is referred to [3].

The Simulink component of the MathWorks tool suite is used to model the continuous dynamical component of hybrid systems. However, again the semantics of the Simulink environment are not formally specified in the MathWorks documentation. But, a large class of Simulink designs can be captured using the continuous dynamical system and hybrid automaton formalisms. Note here that we have a choice on the semantics for the continuous dynamics—using either the standard one, or the simulation semantics where the differential equations are discretized into difference equations (the discretization parameter is either fixed or left symbolic), the latter is used for Simulink models in MathWorks tool suite.

V. Analysis Techniques for Hybrid Systems

We now describe formal analyses techniques for hybrid systems. Analysis tools for Simulink/Stateflow or hybrid models include symbolic simulation, invariance checking, typechecking, abstraction, and model checking. This tool set provides a graded sequence of formal analysis technologies. On one end are completely automated techniques that determine bounds on recursive event calls and perform extended typechecking. Although such analysis helps in early error detection, it does not provide full verification. Abstraction and invariant generation is used to make the model amenable for exhaustive search. Thus, complete assurance can be provided using theorem proving and model checking.

In Section V-A, we generalize the notion of simulation to symbolic simulation and show how it can be useful. Symbolic simulation refers to computing the set of next states reached from the set of current states using either the continuous or discrete transition. We show how this can be done for the simulation semantics of hybrid systems. Symbolic simulation can be used in different ways to do forward and backward propagation, reachability computation, invariance checking, and typechecking.

For the standard semantics of hybrid systems, we describe techniques to perform extended typechecking (inductive invariant checking) and creating sound abstractions of hybrid models. For hybrid systems described using only polynomial expressions, both of these technique can be completely automated. In case of the abstractor, the resulting abstraction is always a finite state discrete transition system, which we model-check using a generic explicit state model-checker.

Prototype implementations of the techniques described in this paper have been built over the Symbolic Analysis Laboratory (SAL) framework. The extended SAL framework provides a specification language for hybrid automata and models can be explicitly discretized using a fixed or symbolic step for further analysis, or they can be analyzed as is using the qualitative abstractor and model-checking technologies.

A. Symbolic Simulation

The Matlab Simulink tool provides extensive simulation facility. Simulation refers to traversing one trajectory of the system behavior from the possible infinite. To run a simulation, the designer must (a) specify initial conditions by giving values of all state variables, (b) choose a particular input function (in case the system has inputs coming from the environment), (c) give some default values to all parameters used in the modeling of the system, and optionally (d) choose a solver and/or a sample time for certain blocks. The simulation tool then computes the system behavior under these specific choices.

Even after doing several simulations with different choices for (a)–(d) above, the designer cannot be sure that the system works correctly in *all* possible scenarios. For instance, in the leader control system of Example 2, simulation would show the behavior of the system under a particular profile of the acceleration of the car in front. But, safety requires that there be no crash under *every* possible acceleration maneuver of the leading car. Similarly, running simulation on the thermostat model of Example 3 will show that the thermostat works as desired for the particular values of parameters h, K, x_{min}, and x_{max} that were chosen for the simulation.

Symbolic simulation refers to performing simulation on sets of states represented symbolically. Thus, symbolic simulation differs from regular simulation in two respects. First, it simultaneously traverses a bunch of trajectories instead of a single trajectory through the state space. Second, a set of states is represented symbolically rather than explicitly. This allows representation of a potentially infinite number of states and simulation of a potentially infinite number of trajectories in one symbolic simulation.

We use the language of first-order logic to symbolically represent sets of states. We recall that a state is a valuation of all the state and output variables. A set of states can be specified using a first-order formula over the state variables. A crucial step in performing symbolic simulation is the computation of the set of all states that are reachable from the current set of

states (represented as a first-order formula). If $\phi(Q,X)$ is a first-order formula that represents the current set of states, and

$$\psi(Q,X,U) \quad \longrightarrow \quad \begin{array}{l} \bigwedge_i (x_i' = e_i(Q,X,U)) \wedge \\ \bigwedge_i (q_i' = b_i(Q,X,U)) \end{array}$$

is a guarded transition with guard $\psi(Q,X,U)$ and assignments $x_i = e_i(Q,X,U)$, $q_i = b_i(Q,X,U)$, where Q,X are state variables and U is the input set of variables, e_i and b_i are expressions over these state variables that evaluate to a real number and boolean constant respectively, then the set of states reached after taking this transition is given by

$$\begin{array}{ll} \exists(\bar{Q},\bar{X},\bar{U}): & [\phi(\bar{Q},\bar{X}) \ \wedge \ \psi(\bar{Q},\bar{X},\bar{U}) \wedge \\ & \bigwedge_i (x_i = e_i(\bar{Q},\bar{X},\bar{U})) \wedge \\ & \bigwedge_i (q_i = b_i(\bar{Q},\bar{X},\bar{U}))]. \end{array}$$

For a system represented by a set of guarded transitions, the set of states reached from the set $\phi(Q,X)$ by one step application of a guarded transition can be computed by combining all such formulas, one for each guarded transition, by disjunction. The resulting formula, denoted by $\mathbf{post}(\phi(Q,X))$, represents the set of states reached from the set $\phi(Q,X)$ by following one step of the system. For suitable discretizations of the continuous dynamics of the hybrid systems, as described in previous sections, this can be done.

Note also that the input variables U are existentially quantified, which means no assumption is being made on them. However, if the input is known to satisfy certain constraints, then these can be incorporated as a conjunct in the above formula as well. The existential quantifier in the expression for **post** must be eliminated to ensure that the formulas do not get arbitrarily large very soon, and we discuss this in Section V-B.

Other approaches to performing simulation that are not based on the use of a quantifier elimination procedure have been discussed in the literature as well. A particular case of symbolic simulation is the idea of using intervals to represent sets of states. The polygonal state space can then be simulated using particular numerical methods. There is a need to do an over-approximation whenever the state set is not representable by a polygon [12]. Similarly, level set methods represent the set of states as level sets [31] and appropriately compute the **post** operator in that representation.

B. Quantifier Elimination

The cylindrical algebraic decomposition (CAD) algorithm [13, 21] decides the full first-order theory (equality and the greater-than relation included) of ordered real closed fields. Given a set of polynomials over n variables, the CAD procedure decomposes the real n-dimensional space into a finite set of regions where each polynomial's evaluation is sign-invariant. The quantifier elimination procedure for real closed fields is obtained as a side effect of the CAD decomposition. Over the last 25 years, the CAD algorithm has been improved and made more efficient [28,27,19]. One such efficient implementation is available via the tool QEPCAD [20], which is built over a symbolic algebra library called SACLIB [11].

The tool QEPCAD can be used to perform quantifier elimination over the first-order theory of real closed fields and, consequently, it can be used as a decision procedure for the same theory. As seen above, quantifier elimination is a crucial step in symbolic simulation and reachability algorithms. Note that the QEPCAD tool cannot handle variables that are not of type real, and hence it can be used only on formulas in which all the non-real variables can be eliminated by suitable preprocessing. In our applications, the boolean variables are the only non-real variables, and they can be easily eliminated by expansion.

Example 4: Following up on Examples 2, we now show a symbolic simulation step for a variable step discretization of the system in Example 2:

$$\begin{array}{lll} \phi_0 & : & gap \geq 2 \ \wedge \ v_1 \geq v_0 \\ \phi_1 & : & \exists(\bar{gap},\bar{v_1},\bar{v_0},\bar{a_1},\delta) : \bar{gap} \geq 2 \ \wedge \ \bar{v_1} \geq \bar{v_0} \ \wedge \\ & & \quad v_1 = \bar{v_1} + \delta\bar{a_1} \ \wedge \\ & & \quad v_0 = \bar{v_0} + \delta(\bar{a_1} + \bar{v_1} - \bar{v_0}) \ \wedge \\ & & \quad gap = \bar{gap} + \delta(\bar{v_1} - \bar{v_0}) \ \wedge \\ & & \quad 0 < \delta \leq 1 \\ \phi_1' & : & (gap > 2 \ \wedge \ v_1 - v_0 - gap + 2 \geq 0 \ \vee \\ & & \quad (v_0 \leq v_1 \ \wedge \ gap + v_0 - v_1 - 2 \geq 0) \end{array}$$

Note that δ is used as a symbolic discretization time step. Here ϕ_1' is obtained from quantifier elimination on ϕ_1. We have shown only one simulation step in the example above because we can *prove* that $gap \geq 0$ always using the results from this one symbolic propagation step. See Example 5.

The quantifier elimination problem has a high time and space complexity. Consequently, techniques for simplification are required before the quantifier elimination tool can be used. In particular, we perform the following two simplifications:

- *Solving for quantified variable*: Certain quantified variables can be easily eliminated by solving for them. For example, given the equality $x + y = z + 5$, one can solve for x to obtain $x = z + 5 - y$. Thus, a quantified formula $\exists x : x + y = z + 5 \ \wedge \ \phi(x)$ is equivalent to the formula $\phi(x/z+5-y)$, where $x/z+5-y$ denotes that we replace all occurrences of x in ϕ by the expression $z + 5 - y$.
- *Logical simplification*: We can use logical equivalences to reduce the size of the formula that is given to the quantifier elimination tool. One of the tautologies that is very useful is $(\exists x : \phi(x) \ \wedge \ \psi) \leftrightarrow (\exists x : \phi(x)) \ \wedge \ \psi$, if x does not occur in ψ. This allows us to move parts of the formula that do not contain the quantified variable outside the scope of the quantifier, thus reducing the size of the quantified formula in the process.

Finally, the quantifier elimination procedure is quite sensitive to the ordering of quantified variables. Logically equivalent quantified formulas $\exists x \exists y : \phi(x,y)$ and $\exists y \exists x : \phi(x,y)$ may take drastically different time and space resources for computation.

C. Invariant Generation and Checking

Symbolic simulation can be used to compute the reachability region as well. In the i-th simulation step, the symbolic simulation procedure yields the set of states that are reached in exactly i transitions. Thus, in order to compute the reachable state set, one must collect the set of all states that are reachable in i-steps

for $i = 0, 1, 2, \ldots$. Each successive iteration would then yield successive approximations of the reachable state set. The exact reachable state space is obtained only in the condition that this process terminates. In case of termination, the set of reachable states is obtained as a formula, which by definition is also the strongest invariant for the given transition system.

Example 5: In Example 4 we showed a symbolic simulation step for the the leader control system. Assuming the same notation and same formulas ϕ_i's from before, successive approximations ψ_i's of the reachability set would be

$$\begin{array}{rcl} \psi_0 & = & \phi_0 \;=\; gap \geq 2 \;\wedge\; v1 \geq v0 \\ \psi_1 & = & \psi_0 \;\vee\; \phi_1' \end{array}$$

The formula ψ_1 is *logically equivalent* to the formula $gap \geq 2 \wedge v1 \geq v0$. This logical equivalence can also be shown using the quantifier elimination decision procedure that is used in the symbolic simulation steps. This establishes that the formula ψ_0 is an invariant of the system. The invariant ψ_0 implies that $gap > 0$, and this establishes that the rear car never crashes onto the car in front under the given leader control law.

The method outlined above for generating an invariant assertion by computing the exact reachable region using forward symbolic propagation is, in general, not sufficient in many cases. In some of these other cases, a combination of approaches based on forward and backward propagation with suitable narrowing and widening might be required. See [37] for the details. For an example of some of these ideas, see also Example 6.

However, the technology outlined above is *sufficient* for invariant *checking*. A formula ϕ is an inductive invariant if (i) the formula describing the initial states implies the formula ϕ and (ii) the result of symbolic propagation starting from the formula ϕ (logically) implies the formula ϕ, that is, $\mathbf{post}(\phi) \Rightarrow \phi$ is valid in the theory $\Re^e$. Both of these tests can be done using a quantifier elimination procedure. In fact, Examples 4 and 5 can also be seen as checking that the formula given as the initial condition is an inductive invariant. Note that inductive invariants are over-approximations for the set of reachable states.

Simple invariants on the values of variables can also be specified using *types*. Richer type system allows specification of more complex relations between the values of different variables. Invariant checking can be used to perform *typechecking* on such rich type systems. The designer can easily annotate his Simulink/Stateflow model by such type information using additional Simulink blocks.

To illustrate that the symbolic propagation method can in fact *generate* invariants, we consider the thermostat example.

Example 6: The thermostat hybrid system discussed in Example 3 can be symbolically discretized using the variable δ which is constrained to be between 0 and $1/K$.

$$\begin{array}{l} q_1 = false \;\wedge\; x_1 \leq x_{min} \longrightarrow \quad q_1' = true \\ q_1 = true \;\wedge\; x_1 \geq x_{max} \longrightarrow \quad q_1' = false \\ q_1 = true \;\wedge\; x_1 < x_{max} \;\wedge\; \delta > 0 \;\wedge\; K\delta \leq 1 \longrightarrow \\ \qquad\qquad\qquad\qquad\qquad\qquad x_1' = x_1 + \delta(-K)(x_1 - h) \\ q_1 = false \;\wedge\; x_1 > x_{min} \;\wedge\; \delta > 0 \;\wedge\; K\delta \leq 1 \longrightarrow \\ \qquad\qquad\qquad\qquad\qquad\qquad x_1' = x_1 + \delta(-Kx) \end{array}$$

The parameters x_{min}, x_{max}, and h satisfy the condition $0 < x_{min} < x_{max} < h$ (this is part of the specification of the problem). We do not explicitly mention this conjunct in the expressions below, but it is implicitly assumed in the computation.

Starting with an initial state in which we assume nothing on the value of x and $state$ variables, symbolic simulation gives the following:

$$\begin{array}{rcl} \phi_0 & : & true \\ \phi_1 & : & (\exists(\bar{q_1}) : \bar{q_1} = false \;\wedge\; x_1 \leq x_{min} \;\wedge\; q_1 = true \\ & \vee & (\exists(\bar{q_1}) : \bar{q_1} = true \;\wedge\; x_1 \geq x_{max} \;\wedge\; q_1 = false) \\ & \vee & (\exists(\bar{x_1}, \delta) : q_1 = true \;\wedge\; \bar{x_1} < x_{max} \;\wedge \\ & & \quad 0 < \delta \leq 1/K \;\wedge\; x_1 = \bar{x_1} - K\delta(\bar{x_1} - h)) \\ & \vee & (\exists(\bar{x_1}, \delta) : q_1 = false \;\wedge\; \bar{x_1} > x_{min} \;\wedge \\ & & \quad 0 < \delta \leq 1/K \;\wedge\; x_1 = \bar{x_1} - K\delta\bar{x_1}) \\ \phi_1 & : & (q_1 = true \;\wedge\; (x_1 \leq x_{min} \;\vee\; x_1 < h)) \;\vee \\ & & (q_1 = false \;\wedge\; (x_1 \geq x_{max} \;\vee\; x_1 > 0)) \;\vee \\ \phi_1 & : & (q_1 = true \;\wedge\; x_1 < h) \;\vee\; (q_1 = false \;\wedge\; x_1 > 0) \end{array}$$

We do not show the rest of the computation here, but it can be checked that we get the same formula after the second symbolic simulation step as well. Thus, the set of states represented by ϕ_1 is an invariant of the system.

Note that we can get a stronger invariant if we make a stronger assumption on the parameter δ. As δ is constrained to be in a smaller neighborhood of 0^+, the upper and lower bound on x in the invariant gets closer to x_{max} and x_{min}, so that in the limit, the invariant is $(state = on \;\wedge\; x_1 \leq x_{max}) \;\vee\; (state = off \;\wedge\; x_1 \geq x_{min})$.

We emphasize here that in this computation, no assumption was made on (i) the values for the parameters h, x_{min}, and x_{max}, or (ii) the initial state of the system.

D. Hybrid Systems with Standard Semantics

The techniques described in Sections V-A and V-C are generic techniques that have been developed for discrete transitions systems. Hence, they can be used on hybrid systems only when we consider the simulation semantics of the continuous components, which allows for a translation of the hybrid system model into a discrete transition system model. We now describe analysis techniques to deal with the standard semantics of hybrid systems.

The procedure to test for inductive invariants for transition systems was considered in Section V-C. Since the same technique works for discrete transitions of a hybrid system, we now restrict our attention to proving inductive invariants for continuous dynamical systems. Let $\phi(X)$ be a formula representing a set of states that we need to test for being inductive over some continuous dynamical system. Assume that the formula $\phi(X)$ is of the form $\bigvee_i \psi_i(X)$, where each $\psi_i(X)$ is of the form

$$\bigwedge_j p_j > 0 \;\wedge\; \bigwedge_j q_j = 0 \;\wedge\; \bigwedge_j r_j \geq 0,$$

where p_j, q_j, and r_j are polynomials over the variables X. A *sufficient* condition to establish that $\phi(X)$ is inductive over the

continuous dynamics is that for each ψ_i, the following implication be valid in $\Re$:

$$\psi_i \Rightarrow \bigwedge_j \dot{p_j} \geq 0 \ \wedge \ \bigwedge_j \dot{q_j} = 0 \ \wedge \ \bigwedge_j (r_j = 0 \Rightarrow \dot{r_j} \geq 0),$$

where $\dot{p_j}, \dot{q_j}$, and $\dot{r_j}$ denote the expressions obtained by symbolically differentiating the polynomials p_j, q_j, and r_j with respective to the continuous dynamics. To prove that this condition is sufficient, note that if p_j is positive in a state and the above implication is true, then $\dot{p_j}$ would be non-negative, and hence p_j would continue to remain positive. The case of q_j is similar. If $r_j \geq 0$ in a state and $r_j = 0 \Rightarrow \dot{r_j} \geq 0$, then r_j can not ever be negative. For example, consider a simple exponential decay system $\dot{x} = -x$. For this system, if $x > 0$ in the initial state, then it is easily proved that $x \geq 0$ is an inductive invariant using the above sufficient condition since $x = 0 \Rightarrow \dot{x} \geq 0$ is valid in $\Re$. Validity in the theory $\Re$, or $\Re^e$ in general (when we consider hybrid systems), can be checked by using either a suitable extension of the quantifier elimination procedure described in Section V-B or some other decision procedure for these theories.

E. Abstracting the Continuous Component

An abstraction of a system is any system that exhibits all the behaviors (trajectories) of the original system, possibly more. Abstract systems are usually smaller and are obtained by suitable generalization or pruning of information from the original system. Abstraction is essential for analyzing systems containing a large number of state variables. Since fairly efficient model checking tools are available for searching through a large, but finite, (discrete) state space, one of the challenges in building analysis tools for hybrid systems is to come up with suitable abstractions for the continuous components that are refined enough to suffice for proving the properties of interest.

We construct sound finite state abstractions for continuous dynamical systems by mapping the uncountable state space into a finite state space by an abstraction function. More specifically, if $CS = (X, Init, f)$ is a continuous dynamical system such that $|X| = n$, then the n-dimensional real space $\mathbb{R}^n$ is partitioned into zones which are sign-invariant for all polynomials in some finite set P, say $\{p_1, p_2, \ldots, p_m\}$. The number of abstract states is bounded by 3^m as each polynomial can be either positive, negative, or zero in each zone. Formally, the set of state variables Q in the corresponding abstract discrete system $DS = (Q, Init, t)$ contains exactly one new variable for each polynomial $p \in P$. Thus, $Q = \{q_p : p \in P\}$. These new variables are interpreted over the domain $\{pos, neg, zero\}$ and consequently the set $\mathbf{Q}$ of all discrete states is the set $\{pos, neg, zero\}^Q$ of all valuations of the variables Q over this domain. We shall represent any such valuation by the corresponding conjunction of atomic formulas. For example, the valuation $\langle q_{p_1} \mapsto pos, q_{p_2} \mapsto neg, q_{p_3} \mapsto zero \rangle$ will be thought of as the formula $p_1 > 0 \ \wedge \ p_2 < 0 \ \wedge \ p_3 = 0$. We shall use such conjunctions and valuations interchangeably. The set of all conjunctions representing such valuations will also be denoted by $\mathbf{Q}$. Note that these conjunctions are in the set $WFF(X)$ of formulas over free variables X.

The continuous dynamics are mapped onto the abstract system using qualitative reasoning. We add an abstract transition $(\psi_1, \psi_2) \in t$ if all of the following conditions hold (for all polynomials $p \in P$):
(a) if $p < 0$ is a conjunct in ψ_1, then (a1) if $\Re \models \psi_1 \Rightarrow \dot{p} \leq 0$, then $p < 0$ is a conjunct in ψ_2; (a2) otherwise, either $p < 0$ or $p = 0$ is a conjunct in ψ_2;
(b) if $p = 0$ is a conjunct in ψ_1, then (b1) if $\Re \models \psi_1 \Rightarrow \dot{p} < 0$, then $p < 0$ is a conjunct in ψ_2; (b2) if $\Re \models \psi_1 \Rightarrow \dot{p} = 0$, then $p = 0$ is a conjunct in ψ_2; (b3) if $\Re \models \psi_1 \Rightarrow \dot{p} > 0$, then $p > 0$ is a conjunct in ψ_2; and (b4) if the valuation of $\dot{p}$ cannot be deduced from ψ_1, then either $p > 0$, $p = 0$, or $p < 0$ is a conjunct in ψ_2;
(c) if $p > 0$ is a conjunct in ψ_1, then (c1) if $\Re \models \psi_1 \Rightarrow \dot{p} \geq 0$, then $p > 0$ is a conjunct in ψ_2; (c2) otherwise, either $p > 0$ or $p = 0$ is a conjunct in ψ_2.

The initial set of states in CS can be mapped onto an initial set of states in DS. For complete details on the abstraction technique and its proof of correctness, see [36]. The effectiveness of the constructed abstract system crucially depends on the choice P of polynomials. In general, the set P is constructed by starting with a small set P_0 of polynomials of interest and adding to this set the time derivatives of polynomials in P_0 and their derivative and so on. The initial set P_0 could contain, for example, the polynomials that appear in the statement of the property of interest that we want to establish for the given continuous system, or the polynomials that occur in the guards of mode change transitions for exiting a mode in a hybrid system, etc. Adding more polynomials to P results in finer abstractions.

Although the procedure for constructing an abstraction is described using polynomials, the method is general and works for other function choices as well. There are two issues to consider in that case: first, polynomials guarantee that the partition of the real space will be finite. For other functions, for example, trigonometric functions, there could be infinitely many sign-invariant regions. Second, as long as everything is polynomial, we can use a decision procedure for the reals to construct the abstraction. For more general functions, we might need additional inference engines to deal with them.

For linear systems, effective abstractions can be constructed using eigenvectors of the transpose of the matrix specifying the continuous dynamics corresponding to real-valued eigenvalues. The details of this method will be described in a future paper. However, it is worth pointing out here that this observation gives a way to go beyond the known decidability results for reachability set computation for certain classes of linear systems. In particular, it is known that the reachability sets can be computed for linear systems specified by (a) nilpotent matrices, (b) diagonalizable matrices with rational eigenvalues, (c) diagonalizable matrices with purely imaginary eigenvalues with rational imaginary part. In case where even a single eigenvalue is (a) zero, or (b) real, or (c) purely imaginary, we can create non-trivial abstractions using corresponding eigenvectors. The more eigenvalues are real or purely imaginary, the more refined abstraction can be created.

The abstraction technique is completely automatic for hybrid systems specified using polynomials (possibly nonlinear) for flows, transition guards, and resets. We are currently inves-

tigating novel approaches to enrich the set P of polynomials so that finer abstractions are created for non-linear systems as well.

Example 7: For the adaptive cruise control law given in Example 1, let $[v_0, v_1, a_0, gap]^T$ be the state vector. It is easily established that the the characteristic polynomial of the 4×4 matrix

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -4 & 3 & -3 & 1 \\ -1 & 1 & 0 & 0 \end{bmatrix}$$

corresponding to the dynamical system of Example 1 is $l^3 + 3l^2 + 4l + 1 = 0$, which has a negative real root. The eigenvector of the transpose A^T of A, corresponding to this negative real eigenvalue, is $[r_3, -r_3 - 1, r_4, 1]^T$, where $r_3 = l^2 + 3l$ and $r_4 = l$. Hence, the corresponding polynomial $p = gap - r_3 v_1 + r_3 v_0 - v_1 + r_4 a_0$ is used to construct an abstraction of the continuous dynamical system. The reason this is useful is the fact that for this polynomial, the derivative $\dot{p}$ is l times p, and for negative real value l, the value of p would exponentially decrease on each trajectory of the dynamical system. The resulting abstraction allows us to prove collision avoidance for sets of initial states where $p > 0$ (assuming a bound on the maximum deceleration). The initial condition specified in Example 1 satisfies this formula. We have assumed here that the velocity of the leading car v_1 is an unspecified symbolic constant.

Example 8: Consider the thermostat hybrid model from Example 3. Starting with the set $P_0 = \{x_1\}$ of seed polynomials, the saturation process adds new polynomials from the specification of the thermostat and the property to be proved. The final set of polynomials generated is

$$P = \{-Kx_1 + Kh,\ x_1,\ x_{min} - x_1,\ x_1 - x_{max}\}$$

Note here that $-Kx_1 + Kh$ is the derivative of x_1 in the heater "on" mode. The second derivative of $-Kx_1 + Kh$ in this mode would be $-K^2(h - x_1)$, which is a constant multiple of $-Kx_1 + Kh$ and hence it is not added to the set P.

Using this set of polynomials and the known relationships on the parameters x_{min}, x_{max}, and h, the abstract system is created. Model-checking the abstract system against the safety property that the temperature is between x_{min} and x_{max} shows that this property is valid always.

VI. Conclusion

Due to large scale deployment of embedded processors in several physical systems, many of which are safety critical, automated tools for analysis of embedded control systems is becoming an important and challenging task. Much work has been done in the design and analysis of hybrid systems [14, 7, 4, 8, 17]. This paper describes tools and techniques for performing formal analysis on hybrid models using a symbolic approach to represent states and manipulate these representations using formal techniques such as theorem provers and decision procedures. The attractive features of this methodology is that it avoids the theoretical intractability results by providing a graded sequence of tools to perform incremental analysis and provide incremental assurance. Since the techniques are based on symbolic representations and manipulations, they promise to scale more easily to larger systems.

We have presented a wide range of formal technologies for hybrid control systems starting from completely automated and invisible techniques like static analysis of simple program properties and symbolic simulation, through extended typechecking, to abstraction and invariant generation. These analysis tools can be embedded into design languages like Stateflow/Simulink to provide greater assurance and quick error detection.

Stateflow design language is based on the concept of hierarchical automata from Statecharts [15], but the semantics of Stateflow diagrams is different from the semantics of Statecharts in several ways. There have been efforts at providing semantics to Statecharts [16]. Hierarchical automata were used for this purpose in [29], and a set of several different semantics for Statecharts was given in [38].

Quantifier elimination tools have been used in the hybrid system world in a variety of contexts. Formulas and expressions over the first-order theory of real closed fields arise naturally when linear and non-linear control systems are described. Many problems in control theory can be reduced to finding solutions of systems of polynomial equations, disequations, and inequalities [22]. Quantifier elimination is also used in obtaining decidability results for reachability in safety-critical embedded systems and hybrid systems [26]. Many applications, especially in mechanical engineering and in numerical analysis, lead to formulas with trigonometric functions involved [32]. In fact, CAD-based quantifier elimination procedures have been used to solve problems regarding stationarity, stability, and reachability of control system designs [23]. Requiem [30] is a tool for performing exact reachability state set computation for linear systems specified using nilpotent matrices. It uses the quantifier elimination procedure implemented inside Mathematica. The computation of the reach set for parametric inhomogenous linear differential systems is done using implicitization and quantifier elimination in [6].

A *finite* decomposition of the real space $\mathbb{R}^n$ into open sets and points such that each partition element preserves a first-order formula over reals is crucial not only for getting a decision procedure for the first-order theory, but also for obtaining finite abstractions of certain hybrid systems [5]. In fact, a model-theoretic structure over the reals in which every (first-order) definable subset of $\mathbb{R}^n$ is a *finite* union of points and open intervals is called a *o-minimal* structure. It is shown in [5] that hybrid systems that are definable over some o-minimal structure admit finite abstractions. The class of o-minimal structures over the reals includes structures with richer signatures as well.

The techniques for verification of hybrid systems described in this paper are being extended in several different ways. First, compositional techniques for creating the abstract transition system are being developed. This will make the tools scalable to handle larger system designs. Second, new techniques for identifying interesting polynomials to partition the state space are required to create fine abstractions for non-linear systems. The basic premise is that even though the general reachability problem might be intractable, there should be ways to extract sufficient, though incomplete, information from the model descriptions to prove the properties of interest for a given sys-

tem. Finally, the abstraction technique relies strongly on the ability to check validity of formulas in a given theory. If good decision procedures for this problem can be developed for richer theories, then the abstraction method can be used on non-polynomial systems as well. A unique feature in our way of using the decision procedures is that our algorithm is tolerant to failures of the decision procedure. More specifically, even if the procedure fails to prove a theorem which was valid, the abstraction algorithm continues to remain sound, though it might result in weaker abstractions.

References

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

[2] R. Alur and D. Dill.
A theory of timed automata.
Theoretical Computer Science, 126:183–235, 1994.
1

[3] R. Alur, K. Etessami, and M. Yannakakis.
Analysis of recursive state machines.
In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *Lecture Notes in Computer Science*. Springer, 2001.
5

[4] R. Alur, T. Henzinger, and E. D. Sontag (eds.).
Hybrid Systems III.
Springer-Verlag, Berlin, 1996.
volume 1066 of *Lecture Notes in Computer Science*.
9

[5] Rajeev Alur, Tom Henzinger, Gerardo Lafferriere, and George J. Pappas.
Discrete abstractions of hybrid systems.
Proceedings of the IEEE, 88(2):971–984, July 2000.
9

[6] H. Anai and V. Weispfenning.
Reach set computations using real quantifier elimination.
In M. D. Di Benedetto and A. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control HSCC 2001*, volume 2034 of *LNCS*, pages 63–76. Springer-Verlag, 2001.
9

[7] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry (eds.).
Hybrid Systems II.
Springer-Verlag, Berlin, 1995.
volume 999 of *Lecture Notes in Computer Science*.
9

[8] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry (eds.).
Hybrid Systems IV.
Springer-Verlag, Berlin, 1997.
volume 1273 of *Lecture Notes in Computer Science*.
9

[9] Gérard Berry, Hubert Comon, and Alain Finkel, editors.
Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings, volume 2102 of *Lecture Notes in Computer Science*. Springer, 2001.
1

[10] A. Bouajjani, J. Esparza, and O. Maler.
Reachability analysis of pushdown automata: Application to model-checking.
In A. W. Mazurkiewicz and J. Winkowski, editors, *CONCUR 97: 8th International Conference on Concurrency Theory*, volume 1243 of *LNCS*, pages 135–150. Springer-Verlag, 1997.
5

[11] B. Buchberger, G. E. Collins, M. J. Encarnacion, H. Hong, J. R. Johnson, W. Krandick, R. Loos, A. M. Mandache, A. Neubacher, and H. Vielhaber.
SACLIB 1.1 user's guide.
In *RISC-Linz Report Series, Tech Report No 93-19*. Kurt Gödel Institute, 1993.
`www.eecis.udel.edu/~saclib/`.
6

[12] Alongkrit Chutinam and Bruce H. Krogh.
Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations.
In Frits W. Vaandrager and Jan H. van Schuppen, editors, *Hybrid Systems: Computation and Control*, volume 1569 of *LNCS*, pages 76–90. Springer-Verlag, 1999.
6

[13] G. E. Collins.
Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition.
In *Proc. Second GI Conf. Automata Theory and Formal Languages*, pages 134–183, 1975.
Vol. 33 of *Lecture Notes in Comp. Sci., Springer, Berlin*.
2, 6

[14] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel (eds.).
Hybrid Systems.
Springer-Verlag, Berlin, 1993.
volume 736 of *Lecture Notes in Computer Science*.
9

[15] D. Harel.
Statecharts: A visual formalism for complex systems.
Sci. Comput. Program., 8:231–274, 1987.
9

[16] D. Harel and A. Naamad.
The statemate semantics of statecharts.
ACM Transactions on Software Engineering and Methodology, 5(4):293–333, October 1996.
9

[17] T. Henzinger and S. Sastry (eds.).
Hybrid Systems: Computation and Control.
Springer-Verlag, Berlin, 1998.
volume 1386 of *Lecture Notes in Computer Science*.
9

[18] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya.
What's decidable about hybrid automata?

Journal of Computer and System Sciences, 57:94–124, 1998.
1

[19] H. Hong.
An improvement of the projection operator in cylindrical algebraic decomposition.
In *Proc. ISAAC 90*, pages 261–264, 1990.
6

[20] H. Hong.
Quantifier elimination in elementary algebra and geometry by partial cylindrical algebraic decomposition version 13.
In *The world wide web*, 1995.
`http://www.gwdg.de/~cais/systeme/-saclib, www.eecis.udel.edu/~saclib/`.
6

[21] Mats Jirstrand.
Cylindrical algebraic decomposition - an introduction.
Technical Report LiTH-ISY-R-1807, Dept of EE. Linköping University, S-581 83 Linköping, Sweden, Dec 1995.
Available by anonumous ftp at ftp.control.ee.liu.se.
6

[22] Mats Jirstrand.
Algebraic methods for modeling and design in control.
Licentiate thesis LIU-TEK-LIC-1996:05 Linkping Studies in Science and Technology. Thesis No 540, Department of Electrical Engineering, Li, 1996.
9

[23] Mats Jirstrand.
Nonlinear control system design by quantifier elimination.
Journal of Symbolic Computation, 24(2):137–152, Aug 1997.
9

[24] Warren A. Hunt Jr. and Steven D. Johnson, editors.
Formal Methods in Computer-Aided Design, Third International Conference, FMCAD 2000, Austin, Texas, USA, November 1-3, 2000, Proceedings, volume 1954 of *Lecture Notes in Computer Science*. Springer, 2000.
1

[25] G. Lafferriere, G. J. Pappas, and S. Sastry.
O-minimal hybrid systems.
Mathematics of Control, Signals, and Systems, 13(1):1–21, 2000.
1

[26] Gerardo Lafferriere, George J. Pappas, and Sergio Yovine.
Symbolic reachability computations for families of linear vector fields.
J. Symbolic Computation, 2001.
To appear.
9

[27] D. Lazard.
An improved projection for cylindrical algebraic decomposition.
Technical Report, Informatique, Universite Paris IV, F-75252 Paris Cedex 05, France, 1990.
6

[28] S. McCallum.
An improved projection operator for cylindrical algebraic decomposition of three dimensional space.
J. Symbolic Computation, 5:141–161, 1988.
6

[29] E. Mikk, Y. Lakhnech, and M. Siegel.
Hierarchical automata as model for statecharts.
In *Asian Computing Science Conference (ASIAN'97)*, volume 1345 of *LNCS*. Springer-Verlag, 1997.
9

[30] P. Mishra and G. J. Pappas.
Reachability using quantifier elimination.
In *University of Pennsylvania hybrid systems group webpage*, 2001.
`www.seas.upenn.edu/hybrid/requiem.html`.
9

[31] I. Mitchell and C. Tomlin.
Level set methods for computation in hybrid systems.
In *Hybrid Systems: Computation and Control HSCC 2000*, 2000.
LNCS 1790.
6

[32] Petru Pau and Josef Schicho.
Quantifier elimination for trignometric polynomials by cylindrical algebraic trignometric decomposition.
www.risc.uni-linz.ac.at/people/ppau, Research Institute for Symbolic Computation, Johannes Kepler University, A-4040 Linz, Austria, 1999.
9

[33] A. Puri and P. Varaiya.
Driving safely in smart cars.
In *Proceedings of the 1995 American Control Conference*, 1995.
3

[34] A. Tarski.
A Decision Method for Elementary Algebra and Geometry.
University of California Press, 1948.
Second edition.
2

[35] A. Tiwari.
Formal semantics and analysis methods for Simulink Stateflow models.
Technical report, SRI International, 2001.
`http://www.csl.sri.com/~tiwari/-stateflow.html`.
4

[36] A. Tiwari and G. Khanna.
Series of abstractions for hybrid automata.
In C. J. Tomlin and M. R. Greenstreet, editors, *Hybrid Systems: Computation and Control HSCC*, volume 2289 of *LNCS*, pages 465–478. Springer, March 2002.
8

[37] A. Tiwari, H. Rueß, H. Saïdi, and N. Shankar.
A technique for invariant generation.
In Tiziana Margaria and Wang Yi, editors, *TACAS 2001 - Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *LNCS*, pages 113–127, Genova,

Italy, April 2001. Springer-Verlag.
7

[38] M. von der Beek.
A comparison of stateflow variants.
In L. de Roever and J. Vytopil, editors, *Formal techniques in real-time and fault tolerant systems*, volume 863 of *LNCS*, pages 128–148. Springer-Verlag, 1994.
9