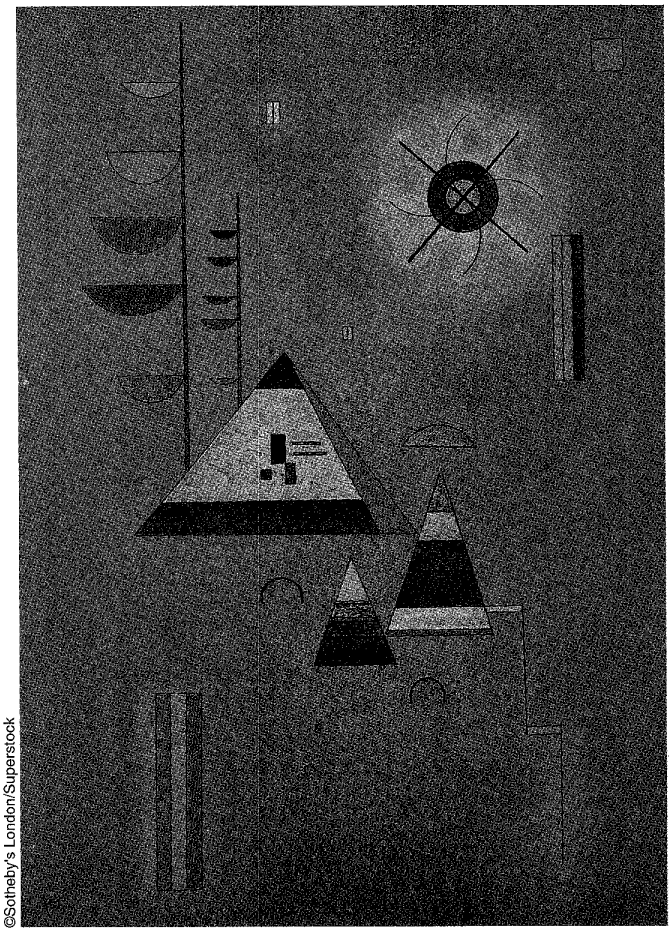


©Sotheby's London/Superstock

Supervisory Hybrid Systems

Michael D. Lemmon,
Kevin X. He, and Ivan Markovsky

Supervisory hybrid systems are systems generating a mixture of continuous-valued and discrete-valued signals. This systems paradigm is particularly useful in modeling applications where high-level decision making is used to supervise process behavior. This occurs, for instance, whenever a network of computers is used to control the physical plant in a decentralized manner, as is found in chemical process control or flexible manufacturing facilities. Hybrid system methodologies are also applicable to switched systems where the system switches between various setpoints or operational modes to extend its effective operating range. Such applications are found in aerospace and power systems. Hybrid systems, therefore, embrace a diverse set of applications [1]-[6] ranging from embedded real-time systems to large-scale manufacturing facilities and from aerospace control to traffic control. Over the past five years there has been considerable activity in the area of hybrid systems theory, and this article provides an introduction to some of the basic concepts and trends in this emergent field.

The term *hybrid* refers to a mixing of two fundamentally different types of objects or methods. Hybrid neural networks arise when we combine artificial neural networks with fuzzy logic or with statistical methods. A system modeled by the interconnection of lumped and distributed parameter systems may be referred to as a hybrid system. Sampled data control systems are hybrid in that they combine discrete-time and continuous-time systems. This paper deals with *supervisory hybrid systems*. Supervisory hybrid systems are systems that combine discrete-event and continuous-valued dynamics.

The motivation for introducing the hybrid system framework is the need to consider continuous and discrete dynamics simul-

Lemmon is an associate professor of electrical engineering at the University of Notre Dame, Notre Dame, IN 46556 (lemmon@maddog.ee.nd.edu). Markovsky and He are Ph.D students at the same institution. *The authors gratefully acknowledge the partial financial support of the Army Research Office (DAAH04-96-10285 and DAAG5-98-1-0199).*

 0272-1708/99/$10.00©1999IEEE

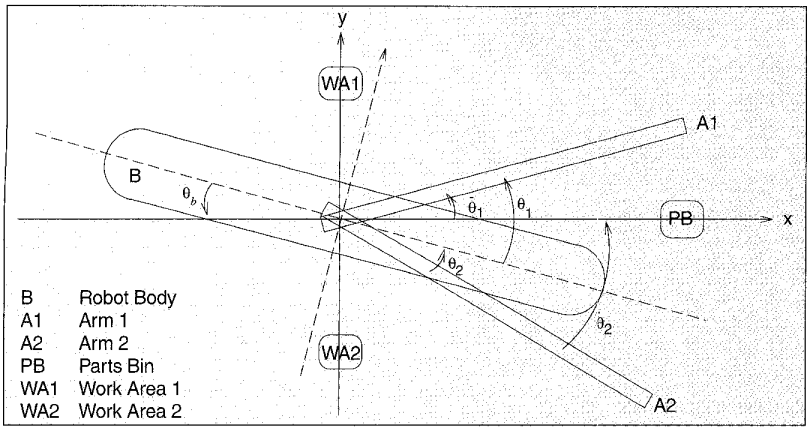


Fig. 1. Free floating robotic system.

taneously. Classical control theory provides tools for analysis and synthesis of continuous systems. There is also a theory for the analysis and synthesis of supervisory discrete-event systems. We are currently lacking a formal approach to analyze systems containing discrete and continuous dynamics; solutions are approached in an ad hoc manner with extensive computer simulation testing. Supervisory hybrid systems theory is intended to provide automated and guaranteed solutions to such problems.

Systems science provides a formal mathematical approach to the study of dynamical systems. The systems scientist treats the system as an abstract mathematical mapping between various sets of *signals*. The signals are functions between a set of time indices, I, and a set of measurements M. Signals, therefore, are functions of the form $x{:}I \rightarrow M$, which map a time $t \in I$ onto a measurement $x(t)$ in the set M. Hybrid systems arise when they generate signals whose index or measurement sets can be treated as the Cartesian product of a discrete and continuous set. One example of a hybrid system is a sampled data system. In sampled data systems the index set I is the Cartesian product of the integers (discrete-time) and real numbers (continuous-time). *Supervisory hybrid systems* arise when the measurement set M is the Cartesian product of a discrete set (usually taken to be a finite set of symbols or integers) and a continuous set (usually taken to be some subset of Euclidean n-space). The discrete-valued signals are sometimes referred to as *discrete-event* signals. The hybrid nature of supervisory hybrid systems, therefore, is a consequence of the fact that these systems generate a mixture of continuous-valued and discrete-valued signals.

A common example of supervisory hybrid systems is found whenever a computer is used to *supervise* the behavior of a continuous-valued process. The continuous process may be a closed-loop control system whose mathematical representation takes the form of an ordinary differential equation. The computer program may be seen as supervising this control loop by selecting various reference inputs or setpoints for the plant. This program's current state evolves over a discrete set, and the dynamics of the associated *discrete-event* process are formally modeled using language theoretic or graph theoretic constructions. Thus this computer-supervised system is a hybrid system since it mixes continuous-valued (the control loop's state) and discrete-event (the program state) variables.

Over the past five years there has been considerable interest in supervisory hybrid systems [7]-[13]. A great deal of this interest is driven by the need to test the logic and timing of very large scale integrated digital circuits. Another reason for this interest is that rapid advances in computer and networking technology have greatly accelerated the deployment of large-scale supervisory systems. Examples of such large-scale systems include the air traffic control grid, communication networks, and the power distribution grid. Traffic control is concerned with the supervision (a discrete process) of vehicles (continuous processes). Congestion control in communication networks is similar in that we are interested in supervising the flow of data packets so that some continuous-valued measure of service quality is optimized. Power distribution involves discrete switching to ensure the stable and continuous delivery of electrical power across the grid. The safe operation of such systems is of paramount interest to all nations, as these systems constitute major components of national infrastructures. Current methods for the design and analysis of such systems rely heavily on simulation testing, a costly and time-consuming method of analysis providing no provable guarantees of safe system operation. The hope is that hybrid systems theory will provide a systematic framework for system engineers that will greatly reduce the cost of large-scale system development with concurrent increases in system reliability.

This article is organized as follows. We first provide a concrete example of a hybrid system, to be used throughout as a pedagogical tool illustrating various concepts in hybrid systems theory. We then discuss modeling frameworks for hybrid systems, paying specific attention to the *hybrid automaton*. Not only may the system have a hybrid character, but the specifications on desired system behaviors may also be hybrid. This article also discusses specification logics that express system requirements on both the discrete and continuous states of the system. The article continues with a survey of current methods and concepts used to verify or validate desired system behaviors, and concludes with a survey of current methods for hybrid control system synthesis.

Hybrid System Example

Fig. 1 shows a concrete example of a supervisory hybrid system. The figure shows a free-floating robotic vehicle with two articulated arms. The system is required to obtain components from a *parts bin* and move these components to a *work area* where an assembly operation is to be performed. The tasks of fetching the workpiece, transporting it to the work area, and then returning to the parts bin to fetch another workpiece are performed repeatedly. As illustrated in Fig. 1, however, it is assumed that the parts bin is shared by both robotic arms. The introduction of a shared resource (i.e., the parts bin) generates a *mutual exclusion* requirement on the system. Not only must the robotic arms complete their repetitively performed tasks, they must also be sure to execute the tasks in a way that ensures both arms do not enter the parts bin at the same time. In other words,

the robotic system needs to treat the parts bin as a *critical section* that both arms access in a mutually exclusive manner.

A candidate solution to the mutual exclusion problem can be readily developed [14]. Let's assume that each arm is controlled by a computer process (an instantiation of the arm control program). Both of these processes execute on the same computer. These concurrent processes need to coordinate their actions if they are to ensure the physical system (i.e., the robotic arms) enters the parts bin in a mutually exclusive manner. Assuming that a multitasking operating system (O/S) controls the execution of both computer processes, we can then use O/S control structures such as *semaphores* or *mutexes* [15] to ensure that both processes execute, in a mutually exclusive manner, that section of their code requesting access to the parts bin. In other words, by requiring that the virtual (i.e., computer) processes respect the mutual exclusion requirement, we expect the robotic arms (i.e., the physical system) to respect that requirement as well.

The pseudocode for one of the computer processes is shown below:

```
ENTRY:  if(mutex==1) goto ENTRY;
        mutex=1;
CRIT1:  if(arm_not_in_partsbin) goto_partsbin();
EXIT:   mutex=0;
ERR:    if(arm_locked) STOP;
REM:    if(arm_not_in_workarea) goto_workarea();
        goto ENTRY;
```

This code has four distinct segments. There is an entry section (ENTRY) which tests the lock variable mutex to see if the other arm is moving towards the parts bin. In practice, the lock variable could be implemented as an O/S semaphore. If the lock variable mutex is 0, then the program sets the lock variable to alert the other process that it is heading to the parts bin. This process then enters its critical section (CRIT1), which represents that code which must be executed mutually exclusively. In other words, both computer processes cannot be executing their critical sections at the same time. While in the critical section, the program checks to see if the arm is in the parts bin (the function call arm_not_in_partsbin) and outputs the command signal to the arm's motor (the function call goto_partsbin()). Upon leaving the parts bin, the process releases the lock variable and then enters its remainder (REM) section, from which it commands the arm to move back to the work area. This remainder section checks to see if the arm is in the work area (the function call arm_not_in_workarea) and outputs the command signal to the arm (the function call goto_workarea()), which moves the arm toward the work area. We have also included an *error* state (ERR) that aborts the program's execution if the arm hits its mechanical limits (arm_locked evaluates to true).

From this pseudocode, we see that the state of the program can be characterized by three different state variables: the lock variable, the program counter for the first process, and the program counter for the second process. Since these variables take values in a discrete set, the supervisory logic embodied in this program is a discrete-event system.

Whether or not ensuring mutually exclusive execution of the process's critical sections is sufficient to guarantee the safe operation of the physical system is not immediately apparent. From our earlier discussion, we saw that the computer process controlling each arm occupies several distinct states. Are these discrete states sufficient to represent the behavior of the physical process? For this particular system, the answer is no, because there is a subtle coupling between the arm and body dynamics. The equations of motion for the arms can be expressed by the following ordinary differential equations

$$\ddot{\theta}_1 = -\dot{\theta}_1 + k(\theta_1 + \theta_b - r_1)$$
$$\ddot{\theta}_2 = -\dot{\theta}_2 + k(\theta_2 + \theta_b - r_2),$$

where θ_1 and θ_2 are the angular positions of arm 1 and arm 2 with respect to the robot's body axis (see Fig. 1). For this example the control law is a proportional feedback law with gain k and with reference inputs r_1 and r_2. These reference inputs represent commands that direct the arm to move to the parts bin or work area. Due to the Hamiltonian nature of the system, the movement of the arms will induce a body rotation so that the system's total angular momentum is conserved. Let θ_b be the body angle with respect to an inertial frame. Let $\bar{\theta}_1 = \theta_1 + \theta_b$ and $\bar{\theta}_2 = \theta_2 + \theta_b$ denote the inertial angles of robot arms 1 and 2, respectively. We therefore know that the body angle θ_b with respect to an inertial frame must satisfy

$$J_b\dot{\theta}_b + J_a\dot{\bar{\theta}}_1 + J_a\dot{\bar{\theta}}_2 = 0, \tag{1}$$

where J_b and J_a are the moments of inertia for the body and arms, respectively.

Note that the state of the continuous-valued subsystem is not entirely reflected in the discrete-event state of the computer process. Transition between discrete states is triggered on entry into or exit from the parts bin, and this knowledge is determined by explicitly examining the arm's position with respect to the position of the parts bin (e.g., by measuring $\theta_1 + \theta_b$). We assume that the computer process for arm 1 can only measure $\bar{\theta}_1 = \theta_1 + \theta_b$, the angle between the arm and the parts bin (or work area). Under our assumptions, the body angle cannot be obtained independently from $\bar{\theta}_1$. The fact that the arm angle, θ_1, (relative to the body) and body angle, θ_b, are not directly observable suggests that this system could fail in ways that cannot be predicted by an examination of the controlling computer processes.

From (1), we see that arm motions will induce a body rotation, since the system conserves total system angular momentum. It may, therefore, be possible for the system's body to work itself into a position from which one of the arms cannot reach the parts bin. If this were to occur, the system would *deadlock* (i.e., the program would be stuck in one of its discrete states). In other cases, system dynamics imply a subtle coupling between both arms that might make it possible for an arm to enter the parts bin when the controlling computer process is not in its critical section. As a result, it is possible to violate the mutual exclusion constraint without the computer process actually detecting this violation.

The conclusion to be drawn from the preceding discussion is that even relatively simple systems such as that shown in Fig. 1 may need to be studied using a formal framework in which both discrete and continuous system dynamics are examined simultaneously. The goal of hybrid systems science is to provide such a formal framework. The following sections discuss what progress has been made to date in this direction.

Hybrid System Modeling

Hybrid systems have been studied extensively by computer scientists and systems scientists. Computer scientists have been interested in the behavior of real-time and multiprocessor programs. The system models developed for such systems are usually based on extensions of traditional finite state machine or Petri net formalisms. System scientists, on the other hand, have tended to employ *equational* models in which system trajectories are represented as functions solving some set of equations. Each approach has its strengths and weaknesses. What has become apparent in recent years is that a successful modeling paradigm for hybrid systems must integrate ideas and methodologies from both disciplines in a complementary way. Early system theoretic models for hybrid systems tended to focus on *switched systems* [16]. These are systems that can be implicitly modeled by the following set of equations:

$$\dot{x} = f(x(t), i(t)) \tag{2}$$

$$i(t) = q(x(t), i(t^-)), \tag{3}$$

where $x:\Re \rightarrow \Re^n$ is the continuous-valued state trajectory and $i:\Re \rightarrow \Omega$ is a discrete-valued state trajectory taking values in the discrete set Ω. $x(t)$ and $i(t)$ denote the values that the continuous and discrete trajectories take at time t, respectively. The function $f:\Re^n \times \Omega \rightarrow \Re^n$ represents a set of continuous dynamical systems (vector fields). These individual systems are referred to as *modes*. The dynamical system used at time t is represented by the discrete state $i(t)$ at time t. The dynamics of the discrete state are embodied in (3). In this equation,

$$i(t^-) = \lim_{\tau \uparrow t} i(\tau)$$

represents the right-hand limit of the function $i(t)$ at t. Equation (3) means that the discrete state transitions at time t from state $i(t^-)$ to $i(t)$, and that this transition is conditioned on the current value of the continuous state, $x(t)$. The set of discrete states we might transition to is characterized by the discrete transition function, $q:\Re^n \times \Omega \rightarrow \Omega$.

It is convenient in switched systems to define a *switching set* between the ith and jth modes by the following equation:

$$\Omega_{ij} = \{x \in \Re^n : \ j = q(x,i)\}.$$

This is the set of all continuous states that enable a transition from discrete state i to discrete state j. We usually assume Ω_{ij} is a "nice" set in the sense that its boundary is an $(n-1)$-dimensional manifold (a hypersurface). Thus the switching action may be initiated whenever the system's continuous state evolves across that boundary.

A natural question about such equational representations is whether or not they are well-posed. In other words, are there any continuous or discrete trajectories that satisfy these equations? Conditions for the existence of absolutely continuous trajectories generally require that a set-valued mapping associated with these system equations be upper semicontinuous and convex [17]. These are very general conditions and are satisfied by most of the systems of interest.

The existence conditions [17], however, do not preclude the existence of hybrid system continuous state trajectories that are not absolutely continuous. Switched systems of the form shown in (2) and (3) are well known to exhibit *chattering* solutions in which the system switches infinitely fast between two different types of vector fields. In the limit these chattering solutions become *sliding mode* solutions, which are frequently found in variable structure systems [18]. In hybrid dynamical systems, sliding mode behaviors are often considered undesirable, and a sufficient condition guaranteeing that such solutions do not exist is to require that state trajectories cross the switching surfaces in a transverse manner. It is interesting to note that computer scientists also have a term for this chattering behavior. Systems capable of exhibiting such chattering solutions are sometimes referred to as *Zeno* systems. The name refers to the classical Zeno's paradox in which the concept of a limit is first informally introduced. In supervisory hybrid systems, we usually want all of our systems to be non-Zeno.

Although we can usually ensure the existence of solutions to such equations, there is no guarantee that these solutions will be unique. The switched systems represented in (2) and (3) can also be treated as differential inclusions for which it is well known that nondeterministic solutions may exist. In other words, if we know the system state at time t, the future behavior of the system may take any one of a number of different paths. Hybrid systems commonly exhibit some form of nondeterministic behavior. Such nondeterminism may arise out of the discrete switching between various vector fields, or it may arise due to an inherent nondeterminism in supervisory decisions.

The switched system introduced in (2) and (3) provides a convenient model for many physical systems, but it does not capture the full range of possible hybrid behaviors. The preceding model assumed solutions in which the continuous state trajectories were continuous across the switching boundary. There are, however, many systems in which the continuous state makes discontinuous jumps on the switching boundary [19]. One example of such a system is the bouncing ball where, due to an elastic collision, the ball's velocity vector makes an instantaneous sign change upon hitting the floor. A variety of hybrid system models have been developed to allow the representation of such discontinuous or autonomous jumping. A good reference to some of these models is found in [20].

The preceding references to the hybrid system modeling literature refer exclusively to the efforts of traditional system scientists. These scientists were trying to develop an equational framework capturing a sufficiently rich array of possible hybrid behaviors (chattering, switching, and autonomous jumping). A key challenge faced by any hybrid system paradigm, however, involves developing a framework that not only treats continuous-state jumping, but also captures the switching nature of the discrete-event process. Equational representations familiar to most system scientists, unfortunately, do not provide a convenient way of capturing discrete-event behaviors. What is really needed for hybrid systems theory to advance is a modeling paradigm providing greater insight into the discrete-event dynamics of the hybrid system.

An early hybrid system model dealing explicitly with discrete and continuous dynamics is found in [21]. In this case, the hybrid system was viewed as a logical discrete-event supervisor connected to a continuous subsystem. The discrete and continuous systems were interconnected through an interface that transformed continuous-valued measurements into discrete event sig-

nals and vice versa. This work suggested a logical discrete-event system (DES) approach to hybrid controller synthesis that was reminiscent of traditional approaches to sampled data control. The approach advocated the extraction of an equivalent discrete-event model of the continuous subsystem, which could then be supervised using extensions of the Ramadge-Wonham supervisory control theory [22].

Although it provided a very general framework for hybrid systems, the model in [21] and [23] was of limited utility due to the restrictive nature of the control. A framework with significant potential for practical usage was developed by the computer science community [24], [25]. Computer scientists have long used formal graph theoretic models for concurrent computer processes. Finite state machines and Petri nets represent two well-known examples of such models. Although powerful computational tools were developed for the manipulation of such formal models, it was apparent that in dealing with multiprocessors and real-time applications, the continuous nature of time would require some extension of these traditional computer science methodologies. This realization led to an attempt to extend traditional and highly successful model checking [26] for finite state machines to real-time systems. The result was a *timed* [25] and *hybrid automaton* [24]. These automata are generalizations of traditional finite state machines in which event transitions are conditioned on the truth value of logical propositions defined over a set of continuous-valued dynamical processes. The work was very influential in that it led to the development of automatic verification tools [27]-[29] for real-time and hybrid systems, and has served as the starting point for much of the recent research in hybrid systems.

The hybrid automaton is closely related to the differential automaton, which was introduced in [30]. Another related version of the hybrid automaton can be found in [31]. Extensions of the approach using Petri nets (rather than finite state machines) will be found in [32]-[34]. The hybrid automaton has been very influential in the study of hybrid systems; the following section presents the hybrid automaton in more detail.

Hybrid Automaton

The hybrid automaton is an extension of the traditional finite state machine [35]. It can be defined by a three-tuple $(\mathcal{N}, \Delta, \mathcal{L})$ where $\mathcal{N}$ is a marked directed graph called a *network*. The network $\mathcal{N}$ is also called a finite state machine or finite automaton. Δ is a set of Lipschitz continuous vector fields over $\Re^n$ representing the continuous dynamics of the system, and $\mathcal{L}$ is a mapping from the network's vertices and arcs onto formulae in a propositional logic. The network models the discrete-event subsystem and the set Δ represents the continuous dynamics of the hybrid system. The relationship between these two subsystems is captured by the labeling function $\mathcal{L}$. Each of these three components is discussed in greater detail below.

The network $\mathcal{N}$ is a directed graph characterized by the ordered pair (V, A). V is a set of vertices and $A \subset V \times V$ is a set of directed arcs between vertices. The vertex set is finite with its cardinality denoted as $|V|$. Networks are often represented graphically. An open circle is used to represent each vertex of the network. An arrow starting at vertex v_i and terminating with an arrowhead at vertex v_j represents the arc (v_i, v_j). The network shown in Fig. 2 illustrates a network with five vertices, $V = \{v_1, v_2, v_3, v_4, v_5\}$, and the arcs

$$A = \{(v_1,v_2),(v_2,v_3),(v_3,v_4),(v_4,v_1),(v_2,v_5),(v_4,v_5)\}.$$

Network $\mathcal{N} = (V, A)$ enumerates all possible states that a discrete-event system might occupy. The current state of the system is denoted by *marking* the network. A marked network is the triple (V, A, μ) where V and A are the network's vertices and arcs, respectively. The final element of the triple is a function $\mu: V \rightarrow \{0,1\}$ that associates either zero or one with each vertex of the network. If $\mu(v) = 1$, then vertex v is marked. Otherwise the vertex is unmarked. Graphically, a marked vertex is denoted by placing a small solid circle (also called a *token*) in the marked vertex. In Fig. 2, the vertex v_2 is marked. It is common to think of μ as a vector $\bar{\mu}$ in which the value of the ith element of this vector is the marking of the ith vertex. This *marking vector* constitutes the discrete state of the hybrid system. The value that this vector takes at time τ is denoted as $\bar{\mu}(\tau)$. In Fig. 2, we've labeled the vertices of our network with the names of the discrete states that a robot arm can occupy, GoToBin, WorkArea, Stop, PartsBin, and LeaveBin. The network therefore characterizes the various discrete states that our robotic arm can transition between. The marking indicates that this robotic arm is currently heading toward the parts bin.

The set Δ is a finite set of vector fields over $\Re^n$ characterized as

$$\Delta = \{f_1, f_2, \cdots, f_n\}.$$

Each $f_i: \Re^n \rightarrow \Re^n$ (for $i = 1,\ldots,n$) maps the continuous state space $\Re^n$ back into itself. The elements of Δ represent continuous dynamical systems (i.e., modes) which generate state trajectories $x: \Re \rightarrow \Re^n$ through the differential equation $\dot{x}(t) = f_i(x(t))$ for any $f_i \in \Delta$. The *continuous state* of the hybrid system is characterized by the following four-tuple, $z = (\dot{x}, x, \tau_0, x_0)$ where $\dot{x}$ is one of the mappings (say f_i) in Δ, $x: \Re \rightarrow \Re^n$ is a continuous function of $\Re$ taking values in $\Re^n$, $\tau_0 \in \Re$, and $x_0 \in \Re^n$. The objects in $z = (\dot{x}, x, \tau_0, x_0)$ are referred to as the continuous state's rate, value, initial time, and initial value, respectively. Together these objects form an initial value problem

$$\dot{x}(t) = f_i(x(t)) \tag{4}$$

$$x(\tau_0) = x_0 \tag{5}$$

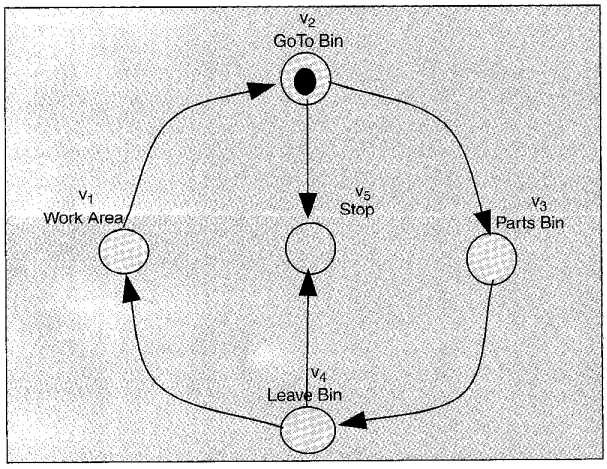


Fig. 2. Network for a discrete-event system.

that the function $x:[\tau_0, \infty) \rightarrow \Re^n$ satisfies for all $\tau > \tau_0$.

Combining the four-tuple, $z = (\dot{x}, x, \tau_0, x_0)$ with the marking vector $\bar{\mu}$ produces the system's *hybrid state*. The hybrid automaton's state, therefore, is represented by an ordered pair $\sigma = (z, \bar{\mu})$ consisting of the discrete state $\bar{\mu}$ and the continuous state z. The set of all ordered pairs $(z, \bar{\mu})$ is denoted as H and will be called the *hybrid state space*. We will be interested in functions $\sigma:\Re \rightarrow \mathcal{H}$ of time that take values in the hybrid state space. Such a function will be called a *hybrid trajectory*.

The event labels $\mathcal{L}$ form the third component of the hybrid automaton. $\mathcal{L}$ is the interface between the discrete subsystem, $\mathcal{N}$, and the continuous subsystems in Δ. The event labeling is represented as a mapping L: $V \cup A \rightarrow \mathcal{P}$ from the arcs and vertices of the network onto formulae in a propositional logic $\mathcal{P}$ whose atomic formulae are defined with respect to the continuous state z. The logical propositions labeling the network nodes and arcs can be defined in a variety of ways. In this article we choose the following. We first introduce the following set of *atomic equations*.

- *Invariant equations* are equations of three forms. The first form of the invariant looks like $[\dot{x} = f_i]$, where $\dot{x}$ is the rate of the continuous state and $f_i \in \Delta$. The second type of invariant equation has the form $[h(x_0) = 0]$, where $h:\Re^n \rightarrow \Re$ are functions of the initial conditions x_0 in the continuous part of the hybrid state. The third type of invariant has the form $[\tau_0 = \tau]$ and acts to reset the initial time in the continuous state z to the current time, τ.
- *Guard equations* have the form $[g(x) > 0]$ where $g:\Re^n \rightarrow \Re$ is a function defined over the continuous-state space, $\Re^n$.

An atomic formula p is said to be satisfied by hybrid state σ if and only if the equation is true when the hybrid state σ is substituted into the equation. We denote the satisfaction of p by σ as $\sigma \vDash p$. Other legal formulae in $\mathcal{P}$ are generated recursively by the conjunction and negation of other formulae in $\mathcal{P}$. For instance, if both p and q are in $\mathcal{P}$, then the conjunction $p \wedge q$ is in $\mathcal{P}$. We say that the hybrid state σ satisfies $p \wedge q$ (denoted as $\sigma \vDash p \wedge q$) if and only if $\sigma \vDash p$ and $\sigma \vDash q$. In a similar way, if $p \in \mathcal{P}$, then the negation $\neg p$ is also in $\mathcal{P}$. Moreover, we say that the hybrid state σ satisfies $\neg p$ (i.e., $\sigma \vDash \neg p$) if and only if σ does not satisfy p. All formulae in $\mathcal{P}$ can be generated by the recursive application of conjunction ($\wedge$) and negation ($\neg$). We define the *disjunction* as $p \vee q = \neg(\neg p \wedge \neg q)$.

The labeling function $\mathcal{L}$ associates each vertex and arc of the network with a proposition in $\mathcal{P}$. The bindings implied by $\mathcal{L}$ determine how the continuous and discrete parts of our hybrid system interact. For the hybrid automata considered in this article, we assume that network arcs are labeled with guard equations and network vertices are labeled with invariant equations. Note that the definition is sufficiently general so that other interpretations can be applied to arc and vertex labels. It is customary in many applications, for example, to label arcs (rather than vertices) with invariant formulae resetting the initial state x_0.

We now define **the dynamics of the hybrid automaton** by characterizing all hybrid trajectories generated by the hybrid system. A time $\tau \in \Re$ is said to be a *switching time* if and only if $\bar{\mu}(\tau^-) \neq \bar{\mu}(\tau^+)$. In other words, a switching time is the instant when the marking of the network $\mathcal{N}$ changes. Given an arbitrary hybrid trajectory $\sigma:\Re \rightarrow \mathcal{H}$, we say that this trajectory is generated by the hybrid automaton $(\mathcal{N}, \Delta, \mathcal{L})$ if and only if the trajectory σ satisfies the following conditions:

- For all $\tau \in (\tau_1, \tau_2)$ that are not switching times and where τ_1 is a switching time, there exists a marked vertex v such that the hybrid state $\sigma(\tau) \vDash L(v)$.

  This condition requires, essentially, that between switching times the continuous part of the hybrid system state satisfies the differential equations implied by the vertex predicate $L(v)$. Recall that the vertex predicates are invariant equations of the form $[\dot{x} = f_i]$, $[h(x_0) = 0]$, and $[\tau_0 = \tau]$. For these predicates to be satisfied, the continuous state z must be reset so that the rate $\dot{x}$, the initial time τ_0, and the initial state x_0, satisfy the equations in $L(v)$. These objects characterize the initial value problem in (4) and (5), generating the hybrid system's continuous state x. Therefore, the switch at time τ_1 causes the hybrid system to switch the underlying continuous-time dynamics of the hybrid system.
- If τ is a switching time, then there is an arc (w,v) such that $\sigma(\tau^-) \vDash L((w,v))$, $\mu(w(\tau^+)) = \mu(w(\tau^-)) - 1 = 0$ and $\mu(v(\tau^+)) = \mu(v(\tau^-)) + 1 = 1$.

  This condition states that at the switching time τ, there is an arc (w,v) in the network that *fires*. The arc fires when the continuous state trajectory $x(\tau)$ satisfies the guard equation $L((w,v))$ on the arc and when the discrete enabling conditions of the marking vector are satisfied. Recall that the label $L((w,v))$ is a guard equation representing an inequality constraint on the continuous state's value x. This label, therefore, represents a necessary condition that the continuous state x has to satisfy before the arc (w,v) can fire. In addition to this continuous enabling condition, there are discrete enabling conditions. The final two conditions state that the vertex w must be marked just prior to the switch. These conditions also tell us what must happen to the marking vector after the switch. The *firing* of arc (w,v) will modify the network's marking vector by removing a token from w and placing a token in vertex v.

Two important classes of hybrid automata are obtained by restricting the nature of the objects in the triple, $(\mathcal{N}, \Delta, \mathcal{L})$. If the elements of Δ are all unity (i.e., $\dot{x} = 1$), then the hybrid automaton is called a *timed automaton*. The class of *rectangular* hybrid automata occurs when elements of Δ are set valued mappings in $\Re^n$ characterizing rectangular regions and when the guard and invariant equations form rectangles in the continuous-state space $\Re^n$.

The system whose continuous dynamics are illustrated in Fig. 1 and whose discrete dynamics are illustrated in Fig. 2 can be easily modeled using a hybrid automaton. The resulting hybrid automaton is shown in Fig. 3. The network $\mathcal{N}$ is simply the network given earlier and the set Δ consists of the following vector fields. The labels for the network are shown in Fig. 3.

$$\Delta = \left\{ \begin{array}{ll} f_{11} = -\dot{\theta}_1 + k(\theta_1 + \theta_b), & f_{21} = -\dot{\theta}_2 + k(\theta_2 + \theta_b), \\ f_{12} = -\dot{\theta}_1 + k(\theta_1 + \theta_b - \pi/2), & f_{22} = -\dot{\theta}_2 + k(\theta_2 + \theta_b - \pi/2), \\ f_{13} = -\dot{\theta}_1, & f_{23} = -\dot{\theta}_2 \end{array} \right\}.$$

The automaton consists of two concurrent parts, one for robot arm 1 and another for robot arm 2. Let's examine the automaton for arm 1 in more detail. In this automaton, we see that the vertex label WorkArea has no predicate associated with it. The arc, however, between WorkArea and GoToBin is labeled with the conditional formula $\neg$[mutex > 0]. In other words, when the lock

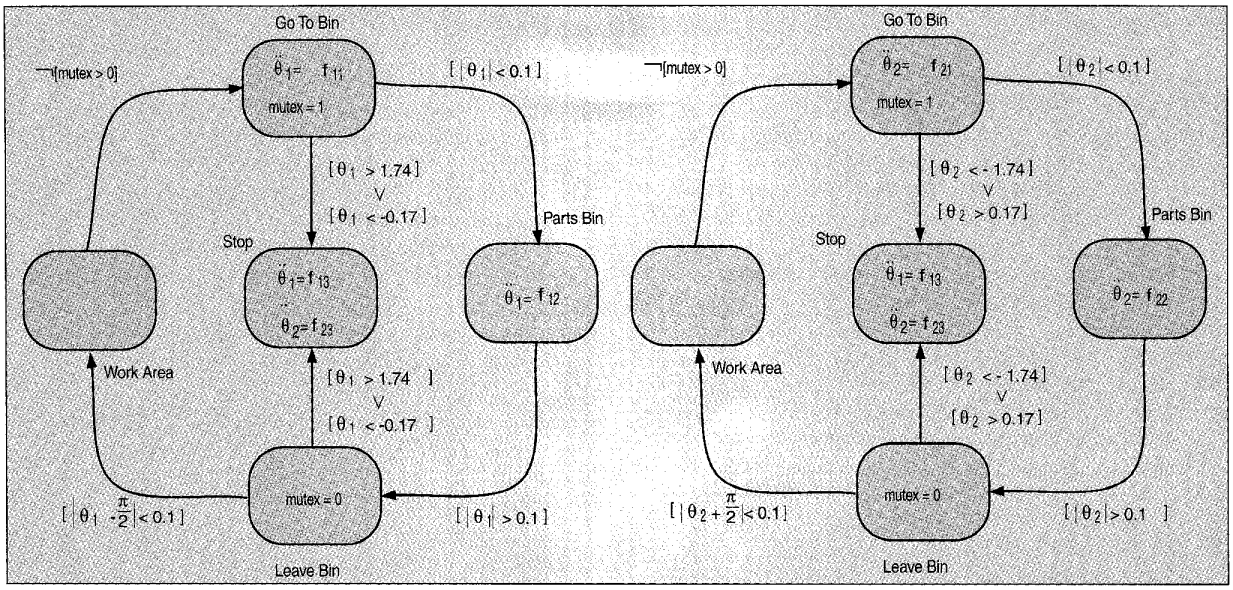


Fig. 3. Hybrid automaton for robotic system.

variable mutex is no longer nonzero, this arc may fire and the system will switch to the logical state GoToBin.

We assume that all vertices in this network also have the invariant equations, $[x_0 = x(\tau)]$ and $[\tau_0 = \tau]$ where τ is a switching time. These invariants ensure that the continuous state trajectories are continuous across switching surfaces. Since these invariants are common to all vertices, they are not explicitly noted on the graph in Fig. 3. The discrete state GoToBin is labeled with the predicate $[\text{mutex} = 1] \wedge [\ddot{\theta}_1 = f_{11}]$. This predicate sets the lock variable to 1, thereby indicating to arm 2 that the parts bin will be occupied. While in this state, the system also sets its timer rate $\ddot{\theta}_1$ to the vector field that begins moving the arm toward the parts bin. The arc connecting GoToBin to the discrete state PartsBin is labeled with the conditional predicate $[|\dot{\theta}_1| < 0.1]$. This guard condition represents a strip in the continuous state space characterizing the extent of the parts bin for robot arm 1.

Also note that the transition out of discrete state GoToBin has a nondeterministic next state in the sense that we can either transition to PartsBin or Stop. The condition for transitioning to the Stop state is $[\theta_1 > 1.74] \vee [\theta_1 < -0.17]$. This is a safety condition that is triggered if the arm moves too far (i.e., hits its physical stops). The Stop state is a deadlocked state from which all forward progress in the system ceases. It is labeled with the predicate $[\ddot{\theta}_1 = f_{13}] \wedge [\ddot{\theta}_2 = f_{23}]$, which turns off the system, thereby causing both arms to eventually stop their motion.

Once the arm is in the parts bin, the system begins moving it out of the bin; therefore, the state PartsBin is labeled with the predicate $[\ddot{\theta}_1 = f_{12}]$. Once the arm is out of the bin, we allow the system's discrete state to transition to the state LeaveBin. The predicate guarding this transition is $[|\dot{\theta}_1| > 0.1]$. Upon leaving the bin, the system resets the lock variable so the other arm can access the parts bin, hence the predicate on LeaveBin is [mutex = 0]. Finally, the system either returns to the WorkArea state if the appropriate conditions on the angle are satisfied, or exits to the Stop state if the limit conditions on the arm's angular position are violated.

Note that the preceding discussion stepped through the various discrete states of the automaton controlling the first arm of the vehicle. A similar automaton shown in Fig. 3 is also used to control the second arm of the vehicle. The coupling between these two discrete structures is through the lock variable, mutex, and the body angle, θ_b.

System Specifications

Control theoretic measures of system performance are frequently taken to be the *size* of some important signals within the control system's feedback loop. Signal size is measured using a function that maps each signal (a function of time) onto a positive real number. Common measures of signal size include signal energy, power, and amplitude. In a more abstract setting, these measures are referred to as signal *norms,* and norm-based measures of system performance represent the starting point for most optimal controller design methods.

In practice, however, a single norm-based measure of performance is rarely adequate to completely characterize what the designer wants the system to do. It may, for example, be necessary to condition system performance on the system's reference signal. A gain scheduled system may need to satisfy one norm-bounded specification at one of its setpoints, and yet this specification may be relaxed at another setpoint without hurting the system's ability to satisfactorily meet specified performance goals. Finally, it should be noted that norm-based performance measures are clearly inappropriate for supervised systems such as the system in Fig. 1. In this case, the mutual exclusion requirement is a high-level behavioral constraint that is not easily expressed in terms of a signal with bounded norm. The conclusion that must be drawn from the preceding observations is that while traditional control theoretic performance measures are valuable, they do not provide sufficient flexibility to characterize the wide range of desired behaviors our systems need to satisfy. To meet these more complex and realistic system specifications, more expressive methods must be adopted for capturing the designer's requirements.

Formal logics can be used to express complex system specifications. We have already used a propositional logic to character-

ize the labels for a hybrid automaton. We now turn to the use of formal logics, in particular temporal logic, to express requirements on desired system behavior. These logics consist of propositions whose truth values are evaluated with respect to norm bounds on the continuous states of the hybrid system.

A logic may be characterized by three things: its atomic formulae, its syntax, and its semantics. The atomic formulae are a set of elementary formulae or equations. The syntax of the logic is the set of rules defining how atomic formulae may be combined to form legal formulae or predicates in the logic. The semantics characterize the meaning of the logical predicates with respect to a specified *frame*. The frame is a set of states through which a system might evolve (i.e., our hybrid automaton). The meaning of logical formulae is then determined by defining the truth values of all logical equations with respect to the frame states. In particular, if a logical formula, p, is true with respect to the frame state, s, then we say that s satisfies p and denote this as $s \models_F p$, where F is the frame on which s is defined. In cases where the frame is clear, we drop the subscript F.

In temporal logics, the frame states can be ordered (i.e., with respect to order of occurrence), which allows us to introduce and reason about several notions of time. A linear temporal logic assumes all states are strictly ordered and therefore allows us to reason about purely deterministic strings of events. A branching temporal logic assumes all frame states are partially ordered and allows us to reason about systems with nondeterministic dynamics. In our case, we will look at system specifications that can be expressed as formulae in a branching temporal logic since hybrid systems are usually nondeterministic. This logic, referred to as CTL1, is a subset of the computation tree logic (CTL) [36].

Let $\sigma(t)$ be a hybrid system trajectory, then the *atomic* formulae for our specification logic take the form $[g(x(t)) > 0]$ or $[\mu(t) = \mu_0]$. The first atomic formula is the conditional formula used earlier as a guard condition in the hybrid automaton. The current hybrid state σ is said to satisfy this atomic formula if the inequality is true for the given state at time t. The second atomic formula is a specific marking of the network. In this case, the hybrid state at time t satisfies the predicate if and only if the network's marking at time t equals μ.

The frame is a hybrid automaton and so explicit mention of it is dropped. The *syntax* and *semantics* of CTL1 are defined below:

$s \models p \quad \Leftrightarrow$ p is satisfied by state s

$s \models \neg p \quad \Leftrightarrow$ p is not satisfied by state s

$s \models p \wedge q \quad \Leftrightarrow$ p and q are satisfied by state s

$s \models p \exists \mathcal{U} q \quad \Leftrightarrow$ there exists a hybrid trajectory $\sigma(t)$ such that $\sigma(0) = s$ and a time t_1 such that $\sigma(t) \models p \vee q$ for $0 < t < t_1$ and $\sigma(t_1) \models q$.

$s \models p \forall \mathcal{U} q \quad \Leftrightarrow$ for all hybrid trajectories $\sigma(t)$ such that $\sigma(0) = s$, there exists a time t_1 such that $\sigma(t) \models p \vee q$ *for* $0 < t < t_1$ *and* $\sigma(t_1) \models q$.

Thus the formula $p \wedge q$ represents our usual notion of logical conjunction, $p \vee q$ represents logical disjunction, and $\neg p$ represent the logical not operation. The other two formulae, $p \forall \mathcal{U} q$ and $p \exists \mathcal{U} q$, have a special meaning that is specific to temporal logics. These operators provide a way of describing temporal relationships between predicates. The formula $p \forall \mathcal{U} q$ can be seen as saying that *for all* hybrid trajectories, predicate p is true *until* predicate q is true. The formula $p \exists \mathcal{U} q$ is the existential formula, meaning that *there exists* a trajectory in which p is true *until* q is true. The formulae $\forall \mathcal{U} p$ and $\exists \mathcal{U} p$ are equivalent to $[\text{true}] \forall \mathcal{U} p$ and $[\text{true}] \exists \mathcal{U} p$, respectively.

CTL1 allows us to express complex specifications relating the discrete and continuous states of the hybrid system. We have made no attempt in this article to construct a complete logic. More powerful temporal logics using the hybrid automaton as a frame will be found in [37] and [38]. CTL1, as introduced in this article, is only intended as a pedagogical tool illustrating some of the basic concepts encountered in using temporal logics to express specifications for hybrid dynamical systems. In the remainder of this section we present some specific examples illustrating the use of CTL1 in specifying acceptable behaviors for the robotic system illustrated in Fig. 1.

In referring to the example in Fig. 1, the first requirement is that the system must satisfy a mutual exclusion requirement. A temporal logic specification capturing this desired constraint is

$$\forall \mathcal{U} \neg [[|\bar{\theta}_1| < .1] \wedge [|\bar{\theta}_2| < .1]].$$

This particular specification equation says that for all possible discrete states (markings), the computer programs controlling both arms will not enter their critical sections at the same time. The critical sections are defined by inequality constraints on the absolute value of the arm angles $\bar{\theta}_1$ and $\bar{\theta}_2$ in the inertial frame.

Not all solutions to the mutual exclusion problem are equally desirable. An easy way to guarantee mutual exclusion is to require that one arm be deadlocked. In other words, if one of the arms stops moving, then we can always ensure the other arm accesses the parts bin in a mutually exclusive manner. For this reason, it is also essential to require that the system be *deadlock-free*. A system attempting to enforce a mutual exclusion constraint is deadlock-free if each process in its entry section is guaranteed of eventually transitioning into its critical section after a finite waiting time. A weak version of deadlock-freedom may be expressed by the CTL1 formula,

$$[\text{WorkArea}] \forall \mathcal{U} [\text{PartsBin}].$$

This requirement is weak because no finite constraints have been imposed on the amount of time before deadlock is broken. A time limit on the duration of deadlock might be imposed by introducing a clock into the system that measures how long the arm has been deadlocked. Let x_1 denote the state of such a clock and let's assume the clock is reset and restarted when the system first marks the vertex WorkArea. In this case, the following equation provides a useful characterization of the deadlock-freedom requirement:

$$[\text{WorkArea}] \forall \mathcal{U} [[\text{PartsBin}] \wedge [x_1 < c]].$$

The specification is requiring that all trajectories starting in WorkArea enter PartsBin in less than c time units.

The particular logic used here, however, is extremely simple, and considerable work is still being done to investigate specification logics for hybrid systems. Recent work in [37] and [38] has augmented CTL to reason about time intervals, and the *duration calculus* [29], [39] also appears to form a very attractive specification language for hybrid systems. This article has only presented some of the basic principles and ideas behind using logics to formally specify hybrid system behavior.

Verification, Validation, and Synthesis

Given a system model and a design specification, there are two classes of problems to consider: the *analysis* and *synthesis* problems. The analysis problem asks whether or not the model satisfies the specification. Solving this problem involves identifying *sufficient* or *necessary and sufficient* tests for satisfiability of the specification with respect to the assumed model (a hybrid automaton). Necessary and sufficient tests are often referred to as *verification* tests, whereas merely sufficient conditions are often referred to as *validation* tests. Verification methods have been studied extensively by computer scientists interested in extending symbolic model checking to real-time systems. Validation methods are frequently used in the control systems community where it is often impractical from a computational standpoint to verify system properties such as stability and robust performance. In both cases, we are concerned with determining whether there exists a set of initial conditions from which there emanate trajectories satisfying the formal specification.

The second problem of interest concerns the synthesis of controllers that enforce a specification on the plant behavior. Synthesis is closely related to analysis in that parameterizing the verification or validation problems may make it possible to effectively search for system parameters that ensure the satisfiability of the specification. Control theorists are very familiar with this approach to system synthesis. Modern robust control synthesis involves searching over a parameterization of the feedback control loop to find stabilizing systems satisfying norm bounds on a specified set of objective signals. A similar approach has been proposed [40] for using existing verification tools to help synthesize hybrid system controllers.

This section provides an overview of concepts and methods used in hybrid system analysis and synthesis. The discussion begins with a look at current verification methods based on extensions of symbolic model checking. We then consider Lyapunov theory approaches for validating hybrid system performance. The two approaches are compared and their relevance to hybrid system synthesis is discussed.

Verification

Much of the early work in hybrid systems analysis can be found in the computer science literature. This body of work assumes that the specification is posed in a temporal logic such as the timed computation tree logic (TCTL), the time μ-calculus [37], or the duration calculus [39]. The frame for these logics is often taken to be the hybrid automaton, although recent efforts have looked at alternative frames such as hybrid Petri nets. By far the best known work has been done for hybrid automata with specification logics based on Emerson's computation tree logic [41]. This work [24], [27], [28], [37], [38] attempts to extend symbolic model checking (SMC) to real-time systems.

SMC [26] is a verification method in which the frame is a finite state machine and the specification language is the branching temporal logic, CTL. The use of binary decision diagrams (BDD) has made it possible to answer queries posed in CTL in a computationally efficient manner [26]. As a result, symbolic model checking has become a standard way of checking digital VLSI circuits [42].

Early verification work for hybrid systems attempted to duplicate the success of SMC on real-time systems. This work developed an extension of CTL so that specifications could be formulated on continuous state variables as well as discrete network states. A hybrid system verification method based on earlier SMC approaches was reported in [24], and software implementing the approach was also developed [27].

How does model checking for hybrid systems work? It is easiest to begin by considering classical SMC for finite state machines and then examining the extensions required for checking hybrid systems. Traditional model checking [26] assumes that the specification is framed in CTL. The interesting thing about CTL formulae is that they are fixed points of special recursive operators, which means that the satisfiability of such formulae can be readily computed by the repeated application of these operators [36]. A full discussion of symbolic model checking is beyond the scope of this article, but a simple example will serve to illustrate the basic principle.

Let us consider the finite state machine shown in Fig. 3 and the CTL predicate,

$$p = \exists\mathcal{U}[\mathsf{PartsBin}].$$

This CTL specification asks us to identify each discrete state from which there exists a state trajectory eventually ending up in the parts bin, PartsBin.

Now consider a sequence of sets, Ω_i, for $i = 0,1,2,\ldots$. The first set, Ω_0, consists of all those discrete states for which the predicate p in the CTL formula $\exists\mathcal{U}p$ is true. In this case, we see that $\Omega_0 = \{\mathsf{PartsBin}\}$. The next set, Ω_1, is generated by the relation

$$\Omega_1 = \Omega_0 \cup \Delta,$$

where the set Δ consists of the presets of all vertices in Ω_0. These presets represent those discrete states from each of which there exists at least one trajectory reaching Ω_0. In this case, we see that

$$\Omega_1 = \{\mathsf{GoToBin}, \mathsf{PartsBin}\}.$$

We repeat the above iteration until, in the ith iteration, we compute all of those discrete states that can reach Ω_0. The first observation that can be made about this iteration is that it is monotonic, since $\Omega_i \subseteq \Omega_{i+1}$. The second observation is that because the state machine has a finite number of vertices, we are guaranteed that there exists some j such that $\Omega_j = \Omega_k$ for all $k \geq j$. In other words, the iteration has a *fixed point*, which we denote as Ω. This fixed point represents all the discrete states of the system satisfying the CTL formula $\exists\mathcal{U}p$. Moreover, this fixed point can be identified after a finite number of iterations, so the fixed point is computable. In this example, the fixed point is the set

$$\Omega = \{\mathsf{PartsBin}, \mathsf{WorkArea}, \mathsf{GoToBin}, \mathsf{LeaveBin}\}.$$

Fig. 4 illustrates the basic steps in this iteration leading to the final determination of the fixed point. This figure shows each set of states in the sequence Ω_i. This set consists of all discrete states that can reach a discrete state satisfying the predicate p in CTL formula $\exists\mathcal{U}p$. Thus the iterative procedures used in symbolic model checking are essentially solving reachability problems over the discrete-event system's state space. The specification $\exists\mathcal{U}p$ is then verified by comparing this fixed point against the initial starting states for our system. If the starting states are contained within this fixed point, the specification can be considered to have been verified.

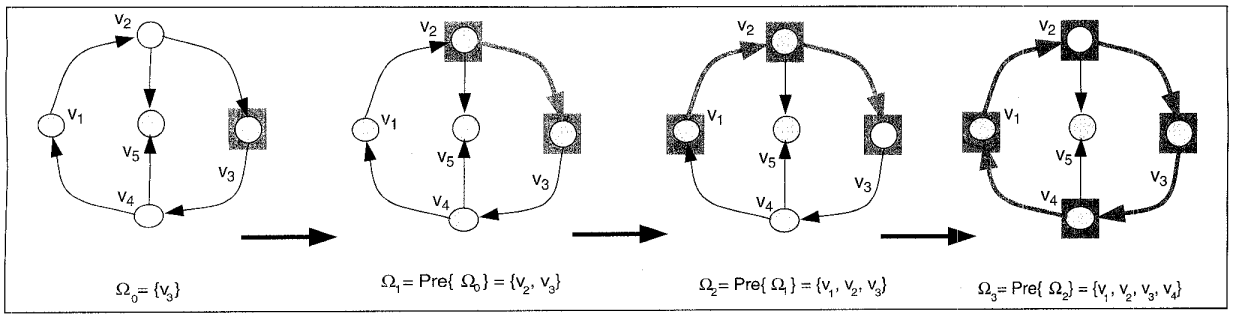


Fig. 4. Model checking iterations.

Extending SMC methods to hybrid systems involves solving the reachability problem for both continuous and discrete system states. As before, let us consider the verification of the CTL formula $\exists \mathcal{U} p$ where p = [PartsBin]. The SMC method described earlier identifies those discrete states that can reach the parts bin solely on the basis of the connectivity between logical states in the network. The enabling and firing of arcs in hybrid automata, however, are also conditioned on satisfaction of the guard equation labeling the arc in question. This implies that although connectivity between discrete states is certainly necessary for reachability, it is by no means sufficient. To fire the arc between the discrete states GoToBin and PartsBin, we must also ensure that the continuous state θ_1 satisfies the guard condition, $|\theta_1| < 0.1$. Extensions of SMC methods to hybrid systems must therefore determine methods for computing subsets, Ξ, of continuous states that allow the firing of the arc.

These subsets can be computed using a recursive procedure similar to that used in traditional SMC methods [25]. This recursive procedure computes a sequences of discrete sets Ω_i and continuous sets Ξ_i. Unlike traditional model checking, however, there is no guarantee that the sequence of continuous state subsets Ξ_i will ever converge to a fixed point after a finite number of steps. This last point concerning the nonfinite nature of the computation highlights one of the weaknesses of model-checking methods as applied to hybrid systems. Since the computation may not terminate in a finite number of steps, the computation of these reachable sets is not decidable [35].

The decidability of the verification problem for hybrid systems has been an important issue driving a great deal of current work. In general, verification problems for hybrid automata are undecidable. Restricted classes of rectangular hybrid automata, however, have been shown to be decidable [43]. However, for minor perturbations of these restricted classes decidability can be lost [43]. The primary obstacle in establishing decidability of hybrid systems rests with the fact that the precursor operation for determining Ξ may not converge. In particular, it was implied in [43] that the decidability boundary for hybrid systems may well rest with rectangular hybrid automata, and thus it was important to see how useful that class of systems would prove to be. In [44], it was suggested that the flow-box theorem could be used to straighten out hybrid systems represented by nonlinear differential inclusions into rectangular hybrid automata. The necessary conditions for this transformation, however, were so restrictive that it was apparent rectangular hybrid automata were of limited utility in modeling many hybrid systems arising in practice. Recently, a larger class of decidable hybrid systems has been identified. These systems are referred to as *o-minimal* systems [45]. The significance of this larger class of decidable hybrid systems is still being investigated.

The preceding discussion has focused on algorithmic verification methods. These methods search through the state space of the system in order to verify a given specification. In concurrent systems, however, the number of states grows exponentially with the number of concurrent processes. As a result, algorithmic model-checking methods are frequently impractical for highly concurrent systems. One way around this limitation is to use *deductive verification* tools [46]-[48]. Deductive verification uses automated theorem-proving techniques to deduce the satisfiability of the specification. The symbolic model-checking methods are often referred to as automatic verification methods since they require little user intervention. Deductive verification, however, often requires some set of heuristics or user input to help guide the proof process.

Validation

In view of the undecidability of verification problems for many hybrid systems, it is natural to ask whether or not we should relax our demands and settle for *validation* tests. Recall that validation only requires finding sufficient conditions for a specification's satisfiability. The hope, of course, is that the sufficient condition is easier to compute yet is sufficiently tight to be useful. The use of sufficient conditions in control theory has a long history. A number of fundamental control problems can be shown to be undecidable, but this fact has not prevented the development of sufficient methods that are still very useful.

An example of a very useful sufficient test can be found in Lyapunov's second method. This method provides a sufficient test for system stability and serves as the basis for a number of analysis and synthesis methods in control theory. Given a dynamical system $\dot{x} = f(x)$ with state trajectories $x(t)$, we say that x_0 is an *equilibrium* point if and only if $f(x_0) = 0$. The equilibrium point is stable in the sense of Lyapunov if for all $\epsilon > 0$ there is a $\delta > 0$ such that $|x(0)| < \delta$ implies $|x(t)| < \epsilon$ for all $t \geq 0$. Lyapunov's method states that if there exists a positive definite functional $V: \Re^n \rightarrow \Re$ such that $V(x_0) = 0$ and $\dot{V}(x(t)) < 0$, then the equilibrium point is Lyapunov stable. Lyapunov methods are well known to only provide sufficient tests for system stability (though converse results exist for linear systems). Despite this shortcoming, however, Lyapunov methods are still an extremely useful tool in the study of nonlinear dynamical systems.

Given the importance of Lyapunov methods, it is not surprising to find a variety of results on the Lyapunov stability of hybrid systems. In [49] a single Lyapunov function was used to determine sufficient tests for switched system (Eqs. (2) and (3)) sta-

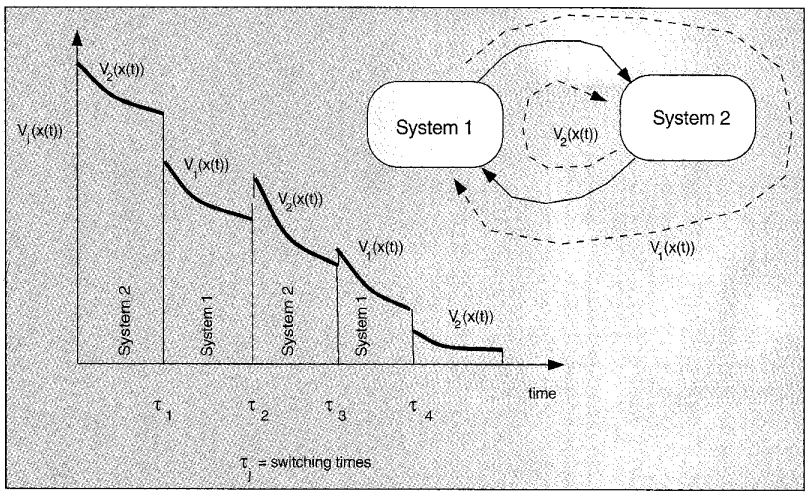


Fig. 5. Switched Lyapunov system.

bility. Multiple Lyapunov function approaches [50], [51] have greatly extended the applicability of Lyapunov analyses for hybrid systems. In [50], a sufficient condition for switched system stability using multiple Lyapunov-like functionals was established. Recall that a switched system consists of a collection of continuous systems $\dot{x} = f_i(x)$ that are switched on the basis of some supervisory control logic. Assuming that system switching is non-Zeno, for the jth mode, we can identify a collection of closed bounded time intervals over which that system is active. Fig. 5 illustrates one such hybrid system trajectory and identifies the set of disjoint time intervals over which the first mode is active. Assuming there are $\mathcal{N}$ systems to switch among, we associate a function V_j ($j = 1,\ldots, N$) with the jth subsystem. We say that this family of functionals is *Lyapunov-like* if $V_j(x(t))$ is decreasing over the intervals in which the jth mode is active. Fig. 5 illustrates a set of Lyapunov-like functionals for this particular system. The result in [50] states that if there exists such a family of Lyapunov-like functions, then the switched system is stable in the sense of Lyapunov. Note that in this result, there can be discontinuous jumps in the value of $V_j(t)$ between different modes.

The fundamental concept in results such as [50] and [51] is that we only have to consider the behavior of the Lyapunov function over *cycles* in the switching behavior. Examining Fig. 5, we see that a Lyapunov-like function is associated with each mode. However each mode shown in the attached automaton is also associated with a vertex. The decreasing nature of a specific Lyapunov-like function, say V_1, is only evaluated when the hybrid system is in mode 1 or rather vertex v_1. Therefore, the contours of the Lyapunov-like function represent subsets of continuous states to which the system returns whenever the discrete part of the system executes a cycle returning to vertex v_1. This close relationship between the cycles of the discrete-event part of the hybrid system and the Lyapunov-like functions used in the stability analysis represents a fundamental way in which the continuous and discrete dynamics of the hybrid system are coupled together.

The results in [50] provide an important extension of Lyapunov analysis methods for the validation of switched system stability. Related work in [51] has relaxed some of the assumptions in [50] to provide tighter sufficient conditions on hybrid system stability. Neither of these results is constructive. As is usually the case with Lyapunov methods, the determination of such functions can only be done systematically for special classes of systems. One such class occurs when the switched modes are linear time invariant and the switching sets are polytopic. In [52] and [53], it was shown that piecewise quadratic Lyapunov-like functions could be determined by checking the feasibility of a certain linear matrix inequality (LMI) [54] based on a modified version of Lyapunov's equation. It is also possible to use convex programming to compute other types of Lyapunov functions [55], [56].

Although these prior results have provided great insight into the Lyapunov stability of switched systems, they do not address the role of the switching law on overall system stability. It was assumed that all traces generated by the switching law were available to be tested. It is more practical to use the discrete-event system's switching logic to directly assess system stability. A hint on how this objective might be accomplished is contained in the results of [50] and [51]. Researchers in both cases observed that it is important to check for monotone decreasing behavior of Lyapunov-like functions over *cycles* within the discrete trace. The *cycles* are cycles of activated systems that begin and end with the same modes. For switching logics generated by finite automata or Petri nets, the pumping lemma [35] assures us that all traces can be finitely generated by a finite set of discrete-event cycles. Thus by investigating the cycles within the discrete part of the hybrid system, it should be possible to formally establish the role that discrete dynamics play in determining overall hybrid system stability. This idea was developed more fully in [33] and [57], which showed that the use of fundamental cycles extracted from the discrete-event subsystem could be used in conjunction with the results of [52] and [53] to provide sufficient conditions for hybrid system stability.

It is interesting to note that the use of fundamental cycles and Lyapunov functionals in [57] is related to earlier studies of cyclic behaviors in hybrid automata. Recall that the SMC iteration discussed earlier may converge to a fixed point consisting of a set of discrete states Ω and a subset of the continuous-state space Ξ. Because these points are fixed points of the iteration, they also constitute sets of states that can be revisited repeatedly by the system. Such sets are sometimes called *viability kernels* [58]. The fixed points Ξ represent viable sets of states associated with cycles in the discrete-event dynamics of the hybrid system. This is precisely what the Lyapunov analysis discussed above approximates for the fundamental cycles of the switching logic. The computational methods discussed in [57] identify fundamental cycles of the switching logic and then use LMI techniques to find Lyapunov-like functions. The level sets of these functions are invariant under the cycle and therefore represent a viability kernel for the specified cycle. These sets, of course, represent approximations to the viability sets Ξ, which the fixed point computation in the SMC iteration attempts to determine.

Hybrid System Synthesis

Synthesis methods for hybrid control systems are still at an early stage of development. This subsection identifies some of the major trends in this area. Early work in hybrid control synthesis attempted to follow the route of traditional sampled data control design. One method proposed extracting a discrete-event model of the continuous part of the hybrid system and then using discrete-event supervision schemes to synthesize a supervisory controller [21]. The strength of this approach rested with its attention to the interface between the continuous and discrete subsystems. Specifically, it was argued that an important aspect of hybrid control synthesis was to design interfaces in which the extracted DES plant *accurately* modeled the behavior of the continuous system [59]. Precisely what constitutes accurate modeling has been discussed in a variety of papers [60]-[62]. In most of these works it is essential that some property of the original continuous system (such as controllability or observability) be preserved under the discretization. Precisely how such ideas can be used to help in the design and analysis of hybrid control systems remains an open question.

An alternative approach to *discretization* is to *continualize* the entire system. Research in the dynamics of switched systems and variable structure systems might be taken as early examples of this continualization approach. A very sophisticated approach [63] to continualization was based on the use of formal power series (Lie-Fliess series) [64] to represent the continuous state trajectories generated by the hybrid systems. The synthesis problem, then, involves determining a controller such that the series representation of the continuous state trajectory satisfies specified safety conditions. The resulting series representations of the controlled systems, by way of the Schutzenberger representation theorems [65], could then be used to extract finite automata generating the desired switching logic for the hybrid system. This continualized approach was first discussed in [63], and that work combines concepts from logic, automata theory, and differential geometry in an attempt to produce a unified framework for the synthesis of controlled hybrid systems. An elementary example illustrating how such switching policies might be synthesized is found in [66].

Another approach for hybrid control system synthesis is based on *gain-scheduling* ideas [66]. Gain scheduling [67] assumes a set of linear controllers that are switched among as the system moves through its operating range. Gain scheduling has been studied extensively in the control systems community. Early work in this area [68], which was of direct interest to the study of hybrid systems, established conditions under which switched behavior would result in stable behavior. The method's primary strength is that it draws heavily upon mature results in modern linear robust control and thus provides a mechanism by which to actually design controllers that achieve tight norm-bounded performance measures [69]. The weakness of this approach, unfortunately, is that it pays relatively little attention to the switching logic and how that logic might be used to enhance system stability and performance.

The preceding approaches to controller synthesis reduced the hybrid synthesis problem to a well-understood set of continuous or discrete system synthesis problems. In contrast to these efforts, some work has attempted to define an integrated framework that directly addresses synthesis issues for both continuous and discrete parts of the system. In this case, the optimal controller is viewed as a saddle point in a noncooperative game played between the supervisor and the controller [70]. The nondeterministic nature of the supervisor means that continuous-state controllers must be optimized over the worst-case decisions adopted by the supervisor, and at the same time we need to ensure that the supervisor is as permissive as possible. These objectives are usually conflicting, and this conflict leads to the noncooperative nature of the game. This viewpoint of hybrid controller synthesis is very elegant from a theoretical standpoint, but computing the saddlepoint is nontrivial. Applications [2] where this method have been applied have relied on computationally intensive approaches for determining the saddle point. For this approach to succeed, a computationally efficient method [71] of determining saddle points will need to be found.

Summary

This article has provided an introduction to many of the concepts and trends in hybrid systems science. Hybrid systems science is an interdisciplinary field requiring a familiarity with methods and concepts from computer science and traditional systems science. Due to the introductory nature of this article, it was impossible to itemize all of the important work being performed. Much of the work outlined here will be found in a series of workshop proceedings published by Springer-Verlag [7]-[13] as well as numerous other workshops and various special issues of technical journals (*Theoretical Computer Science*, *IEEE Transactions on Automatic Control*, *Discrete Event Dynamic Systems*, *International Journal of Control*, *System and Control Letters*, *Automatica*). The field is still in an early stage of development, but some important results and discoveries have already been made. Principal milestones in hybrid system science include early work on verification using hybrid automata, the extension of Lyapunov methods to hybrid systems, and the growing body of recent work dealing with hybrid system synthesis. There have recently been significant applications of these methods in traffic control, automotive systems, and chemical process control [1]-[6]. All of these accomplishments point to a science that shows excellent potential for having a profound impact on the way we design and develop the engineering applications of the future.

Hybrid systems science, however, is far from a mature field. This article summarizes a recent synthesis of computer science and systems science methods, and a number of open questions remain. One of the dominant issues concerns computational complexity. As noted above, algorithmic model checking can be computationally intensive. Recursive computation of the viability kernel is not guaranteed to converge, and the finite state machine formalism suffers from state explosion problems when dealing with highly concurrent systems. Many approaches are being considered to address this problem. Deductive model checking [46] and hybrid Petri net formalisms [33] represent two different directions in addressing the state explosion problem. Decidability issues also need to be treated. Current directions include attempts at extending the decidability boundary [45] or relaxing our goals to obtain tight sufficient conditions (i.e., model validation) for system safety [50]. Validation methods are currently based on Lyapunov-based arguments, but the automatic computation of such piecewise continuous Lyapunov functions is just beginning to be studied in detail. Other open issues in hybrid systems concern synthesis and identification issues. Although a variety of frameworks has been proposed for hybrid control system synthesis, no universal agreement has formed

concerning the best approach to follow. Future work is needed in the development of automatic synthesis methods that integrate discrete-event and continuous controllers satisfying complex temporal logic specifications. Additional work is needed in determining the role that Zeno-type or sliding mode control might play in hybrid system supervision. Finally, it should be noted that very little attention has been paid to the issue of hybrid system identification and event detection.

References

[1] S. Kowalewski, M. Fritz, H. Graf, J. Preussig, S. Simon, O. Stursberg, and T. Treseler, "A case study in tool-aided analysis of discretely controlled continuous systems: The two tanks problem," in *Hybrid Systems V*, P.J. Antsaklis, W. Kohn, M.D. Lemmon, A.N. Nerode, and S. Sastry, eds., Lecture Notes in Computer Science, vol. 1567, Springer-Verlag, 1999.

[2] C. Tomlin, G. Pappas, and S. Sastry, "Conflict resolution for air traffic management: A study in multiagent hybrid systems," *IEEE Trans. Automat. Contr.*, vol. 43, no.4, 1998.

[3] B. Lennartson, M. Tittus, B. Egardt, and S. Petterson, "Hybrid systems in process control," *Control Systems Magazine*, vol.16, no. 5, pp. 45-56, 1996.

[4] J. Raisch and E. Klein, "Approximating automata and discrete control for continuous systems: Two examples from chemical process control," in *Hybrid Systems V*, P.J. Antsaklis, W. Kohn, M.D. Lemmon, A.N. Nerode, and S. Sastry, eds., Lecture Notes in Computer Science, vol. 1567, Springer-Verlag, 1999.

[5] C.W. Seibel and J.-M. Farines, "Towards using hybrid automata for the mission planning of unmanned aerial vehicles," in *Hybrid Systems V*, P.J. Antsaklis, W. Kohn, M.D. Lemmon, A.N. Nerode, and S. Sastry, eds., Lecture Notes in Computer Science, vol. 1567, Springer-Verlag, 1999.

[6] R. Balluchi, M. De Benedetto, C. Pinello, C. Rossi, and A. Sangiovanni-Vincentelli, "Hybrid control for automotive engine management: The cut-off case," in *Hybrid Systems: Computation and Control*, T.A. Henzinger and S. Sastry, eds., Lecture Notes in Computer Science, vol. 1386, Springer-Verlag, 1998.

[7] *Hybrid Systems*, R.L. Grossman, A.N. Nerode, A.P. Ravn, and H. Rischel, eds., Lecture Notes in Computer Science, vol. 736, Springer-Verlag, 1993.

[8] *Hybrid Systems II*, P.J. Antsakis, W. Kohn, A.N. Nerode, and S. Sastry, eds., Lecture Notes in Computer Science, vol. 999, Springer Verlag, 1995.

[9] *Hybrid Systems III; verification and control*, R. Alur, T.A. Henzinger, and E.D. Sontag, eds., Lecture Notes in Computer Science, vol. 1066, Springer Verlag, 1996.

[10] *Hybrid Systems IV*, P.J. Antsaklis, W. Kohn, A.N. Nerode, and S. Sastry, eds., Lecture Notes in Computer Science, vol. 1273, Springer-Verlag, 1997.

[11] *Hybrid Systems V*, P.J. Antsaklis, W. Kohn, M.D. Lemmon, A.N. Nerode, S. Sastry, eds., Lecture Notes in Computer Science, vol. 1567, Springer-Verlag, 1999.

[12] *Hybrid Systems: Control and Computation*, T.A. Henzinger and S. Sastry, eds., Lecture Notes in Computer Science, vol. 1386, Springer-Verlag, 1998.

[13] *Hybrid and Real-Time Systems: Hart'97*, O. Maller, ed., Lecture Notes in Computer Science, vol. 1201, Springer Verlag, 1997.

[14] N. Lynch and N. Shavit, "Timing-based mutual exclusion," *Proc. 13th IEEE Real-Time Systems Symp.*, IEEE Computer Society Press, pp. 3-11, 1993.

[15] R. Gallmeister, *POSIX.4, Programming for the Real World*, O'Reilly and Associates, 1995.

[16] H.S. Witsenhausen, "A class of hybrid-state continuous time dynamic systems," *IEEE Trans. Automat. Contr.*, vol. 11, no. 2, pp. 161-167, 1966.

[17] J.P. Aubin and A. Cellina, *Differential Inclusions*, Springer-Verlag, Berlin, 1984.

[18] R.A. DeCarlo, S.H. Zak, and G.P. Matthews, "Variable structure control of nonlinear multivariable systems: A tutorial," *Proc. IEEE*, vol. 76, no. 3., March 1988.

[19] R.W. Brockett, "Hybrid models for motion control systems," in *Essays in Control: Perspectives in the Theory and Its Applications*, pp. 29-53, Boston: Birkhauser, 1993.

[20] M.S. Branicky, "Studies in Hybrid Systems: Modeling, Analysis, and Control," LIDS-TH-2304, Ph.D. Dissertation, Massachusets Institute of Technology, LIDS, 1995.

[21] J.A. Stiver, P.J. Antsaklis, and M.D. Lemmon, "A logical DES approach to the design of hybrid control systems," in *Mathematical Computer Modeling*, vol. 23, no. 11/12, pp. 55-76, 1996.

[22] P.J. Ramadge and W.M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control and Optimization*, vol. 25, no. 1, pp. 206-230, 1987.

[23] J. Raisch and S.D. O'Young, "Discrete approximations and supervisory control of continuous systems," *IEEE Trans. Automat. Contr.*, vol. 43, no. 4, pp. 569-573, 1998.

[24] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Po, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," in *Hybrid Systems*, R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, eds., Lecture Notes in Computer Science, vol. 736, Springer-Verlag, 1993.

[25] R. Alur and D. Dill, "The theory of timed automata," *Theoretical Computer Science*, vol. 126, pp. 193-235, 1994.

[26] K. McMillan, *Symbolic Model Checking*, Kluwer Academic, 1993.

[27] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi, "A user's guide to HyTech," in *First Workshop on Tools and Algorithms for the Construction and Analysis of Systems: TACAS94*, Lecture Notes in Computer Science, vol. 1019, Springer-Verlag, 1995.

[28] R. Alur, C. Courcoubetis, Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, Olivero, J. Sifjakis, and S. Yovine, "The algorithmic analysis of hybrid systems," in *Theoretical Computer Science,* vol. 138, pp. 3-34, 1995.

[29] K.G. Larsen, P. Pettersson, and W. Yi, "Model-Checking for Real-Time Systems," *Proc. 10th Int. Conf. Fundamentals of Computation Theory*, Dresden, Germany, H. Reichel, ed., Lecture Notes in Computer Science, vol. 965, Springer-Verlag, 1995.

[30] L. Tavernini, "Differential automata and their discrete simulators," in *Nonlinear Analysis, Theory, Methods and Applications*, vol. 11, no. 6, pp. 665-683, 1987.

[31] M. Heymann, F. Lin, and G. Meyer, "Synthesis and viability of minimally interventive legal controllers for hybrid systems," in *Discrete Event Dynamic Systems: Theory and Applications*, vol. 8, no. 2, pp. 105-136, 1998.

[32] J. LeBail, H. Alla, and R. David, "Hybrid petri nets," *Proc. 1st Euopean Control Conf.*, Grenoble, France, 1991.

[33] M.D. Lemmon and K.X. He, "Modeling hybrid control systems using programmable Petri nets," in *JESA—European J. Automation*, vol. 32, no. 9-10, pp. 1187-1208, 1999.

[34] I. Demongodin and N.T. Koussoulas, "Differential Petri nets: Representing continuous system in a discrete-event world," *IEEE Trans. Automat. Contr.*, vol. 44, no. 3, pp. 573-578, 1998.

[35] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

[36] E.M. Clarke and E.A. Emerson, "Characterizing properties of algorithms as fixed points," in *7th Int. Colloq. Automata Languages and Programming*, Lecture Notes in Computer Science, vol. 85, Springer-Verlag, 1981.

[37] R. Alur, C. Courcoubetis, and D. Dill, "Model checking in dense real time," *Information and Computation*, vol. 104, pp. 2-34, 1993.

[38] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, "Symbolic model checking for real-time systems," in *Information and Computation*, vol. 111, pp. 193-244, 1994.

[39] C. Zhou, "Duration calculii: An overview," *Proc. Formal Methods in Programming and Their Application*, D. Bjorner, M. Broy, and I. Pottosin, eds., Lecture Notes in Computer Science, vol. 735, Springer-Verlag, 1993.

[40] H. Wong-Toi, "Synthesis of contollers for linear hybrid automata," *Proc. 36th IEEE Conf. Decision and Control*, San Diego, California, 1997.

[41] E.M. Clarke and E.A. Emerson, "Synthesis of synchronization skeletons for branching time temporal logic," in *Logic of Programs: Workshop*, Dexter and Kozen, eds., Lecture Notes in Computer Science, vol. 131, Springer-Verlag, 1981.

[42] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and T. Hwang, "Symbolic model checking: 10^{20} states and beyond," *Proc. 4th Ann. Symp. Logic in Computer Science,* June 1990.

[43] T.A. Henzinger, P. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata," *Proc. 27th Ann. ACM Symp. Theory of Computing*, 1995.

[44] G.J. Pappas and S. Sastry, "Straightening out rectangular differential inclusions," *System and Control Letters*, vol. 32, no. 2, pp.79-85, Sep., 1998.

[45] G. Lafferriere, G. J. Pappas, and S. Sastry, "O-minimal hybrid systems," Technical report UCB/ERL M98/29, Univ. California, Berkeley, May 1998.

[46] Z. Manna and A. Pnueli, *Temporal Verification of Reactive Systems: Safety*, Springer-Verlag, 1995.

[47] H. Sipma, T. Uribe, and Z. Manna, "Deductive model checking," *Proc. 8th Int. Conf. Computer Aided Verification*, Lecture Notes in Computer Science, vol. 1102, Springer-Verlag, 1996.

[48] Z. Manna and H. Sipma, "Deductive verification of hybrid systems using STeP," in *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science, vol. 1386, Springer-Verlag, 1998.

[49] P. Peleties and R.A. DeCarlo, "Asymptotic stability of *m*-switched systems using Lyapunov like functions," *Proc. American Control Conf.*, pp. 1679-1684, 1991.

[50] M.S. Branicky, "Multiple Lyapunov functions and other analysis tools for switched and hybrid systems," *IEEE Trans. Automat. Contr.*, vol. 43, no. 4, April 1998.

[51] L. Hou, A.N. Michel, and H. Ye, "Stability analysis of switched systems," *Proc. 35th Conf. Decision and Control*, Kobe, Japan, 1996.

[52] S. Petterson and B. Lennartson, "Stability and robustness of hybrid systems," *Proc. 35th Conf. Decision and Control*, Kobe, Japan, 1996.

[53] M. Johansson and A. Rantzer, "Computation of piecewise quadratic Lyapunov functions for hybrid systems," *IEEE Trans. Automat. Contr.*, vol. 43, no. 4, 1998.

[54] S. Boyd, L.E. Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*, SIAM, Studies in Applied Mathematics, vol. 15, 1994.

[55] C.A. Yfoulis, A. Muir, N.B.O.L. Pettit, and P.E. Wellstead, "Stabilization of orthogonal piecewise linear Lyapunov-like functions," *Proc. Conf. Decision and Control*, Dec. 1998.

[56] A. Hassibi and S. Boyd, "A class of Lyapunov functionals for analyzing hybrid dynamical systems," *Proc. American Contr. Conf.*, Feb. 1999.

[57] K.X. He and M.D. Lemmon, "Lyapunov stability of continuous valued systems under the supervision of discrete event transition systems," *Hybrid Systems: Control and Computation*, Lecture Notes in Computer Science, vol. 1386, Springer Verlag, 1998.

[58] A. Deshpande and P. Varaiya, "Viable control of hybrid systems," in *Hybrid Systems II*, P.J. Antsakis, W. Kohn, A.N. Nerode, and S. Sastry, eds., Lecture Notes in Computer Science, vol. 999, Springer Verlag, 1995.

[59] J.S. Stiver, P.J. Antsakis, and M.D. Lemmon, "An invariant based approach to the design of hybrid control systems," *Proc. IFAC 13th Triennial World Congress*, vol. J, pp. 457-472, San Francisco, CA, 1996.

[60] M.D. Lemmon and P.J. Antaklis, "Inductively inferring valid logical models of continuous state dynamical systems," in *Theoretical Computer Science*, vol. 138, pp. 201-210, 1995.

[61] P.E. Caines and Y.J. Wei, "The hierarchical lattices of finite state machines," in *System and Control Letters*, vol. 25, pp. 257-263, 1994.

[62] G.J. Pappas, G. Lafferriere, and S. Sastry, "Hierarchically consistent control systems," *Proc. 37th IEEE Conf. Decison and Control*, Tampa, FL, Dec. 1998.

[63] A. Nerode and W. Kohn, "Multiple agent hybrid control architecture," in *Hybrid Systems*, R.L. Grossman, A.N. Nerode, A.P. Ravn, and H. Rischel, eds., Lecture Notes in Computer Science, vol. 736, Springer-Verlag, 1993.

[64] A. Isidori, *Nonlinear Control Systems*, Springer-Verlag, Berlin, 2nd ed., 1989.

[65] A. Salomaa and M. Soittola, *Automata Theoretic Aspects of Formal Power Series*, Springer-Verlag, 1973.

[66] M.D. Lemmon and C.J. Bett, "Safe implementations of supervisory commands," *Int. J. Control*, Vol. 70, no. 2, pp. 271-288, 1998.

[67] J.S. Shamma and M. Athans, "Guaranteed properties of gain scheduled control for linear parameter-varying plants," *Automatica*, vol. 27, pp. 559-564, 1990.

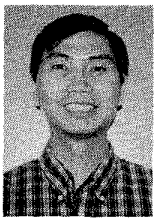
[68] A.S. Morse, *Control Using Logic Based Switching*, Lecture Notes in Control and Information Sciences, vol. 222, Springer-Verlag, 1997.

[69] C.J. Bett and M.D. Lemmon, "Bounded amplitude performance of switched LPV systems with applications to hybrid systems," to appear in *Automatica*, 1999.

[70] J. Lygeros, D.N. Godbole, and S. Sastry, "Multiagent hybrid system design using game theory and optimal control," *Proc. Conf. Decision and Control*, Kobe Japan, pp. 1190-1195, 1996.

[71] J. Lygeros, C.J. Tomlin, and S. Sastry, "On controller synthesis for nonlinear hybrid systems," *Proc. 37th IEEE Conf. Decision and Control*, Tampa, Florida, Dec. 1998.

Michael D. Lemmon received the B.S. degree in electrical engineering from Stanford University in 1979. After 7 years as a guidance and control systems engineer in the aerospace industry, Dr. Lemmon returned to graduate school and received M.S. and Ph.D. degrees in electrical and computer engineering from Carnegie Mellon University in 1987 and 1990, respectively. Since 1990, Dr. Lemmon has been affiliated with the University of Notre Dame where he currently holds the position of associate professor of electrical engineering. Dr. Lemmon's research interests focus on the development of high-autonomy control systems. Dr. Lemmon is a former associate editor of the *IEEE Transactions on Neural Networks* and was program chair for the 5th International Workshop on Hybrid Systems (1997). He currently chairs the IEEE Working Group on Hybrid Dynamical Systems, is an associate editor of the *IEEE Transactions on Control Systems Technology*, and is Program Chair for the 1999 International Symposium on Intelligent Control.



Kevin X. He was born in Tianjin, P.R. China, in 1972. He obtained the B.S.E.E. degree in 1995 from Tsinghua University, Beijing, P.R. China, and the M.S.E.E degree in 1998 from the University of Notre Dame. Presently he is pursuing his Ph.D. degree in Electrical Engineering at the University of Notre Dame. His research interests lie in the modeling, verification and supervision of hybrid systerms.

Ivan Valentinov Markovsky was born in Sofia, Bulgaria, in 1974. He received the B.S. (1997) and MS. (1998) degrees in control engineering from Technical University of Sofia, Bulgaria. The master's thesis was completed in Delft University of Technology in 1998. Currently he is working toward his Ph.D. at the University of Notre Dame. His research interests are optimization methods in control, switched system stability and nonlinear control.