A Tutorial Introduction to Supervisory Hybrid Systems

Technical Report of the ISIS Group
at the University of Notre Dame
ISIS-98-004
October, 1998

M.D. Lemmon, K.X. He, and I. Markovsky
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556

Interdisciplinary Studies of Intelligent Systems

A Tutorial Introduction to Supervisory Hybrid Systems

Michael D. Lemmon [*], Kevin X. He, and Ivan Markovsky
Department of Electrical Engineering
University of Notre Dame

October, 1998

Abstract

Supervisory hybrid systems are systems generating a mixture of continuous-valued and discrete-valued signals. These systems provide convenient models for a wide range of complex engineering applications ranging from small real-time embedded systems to large-scale traffic control and manufacturing facilities. In recent years there has been considerable interest in using hybrid systems theory to develop a systematic framework for the analysis and design of complex engineering systems. This paper provides an introduction to some of the concepts and trends in hybrid dynamical systems theory.

1 Introduction

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 in order to extend its effective operating range. Such applications are found in aerospace and power systems. Hybrid systems, therefore, embrace a wide range of applications [32] [71] [75] [76] [77] [78] [79] ranging from embedded real-time systems to large-scale manufacturing facilties, from aerospace control to traffic control. Over the past 5 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*. Owing to the diverse useage of the term hybrid, we need to be precise in delineating the hybrid nature of these supervisory systems.

[*]We gratefully acknowledge the partial financial support of the Army Research Office (DAAH04-96-10285 and DAAG5-98-1-0199) and the National Science Foundation (NSF-ECS95-31485)

System science provides a formal mathematical approach to the study of dynamical systems. The system scientist treats the system as an abstract mathematical mapping between various sets of *signals*. The signals are, themselves, functions between a set of time indices, I, and a set of measurments M. Signals, therefore, are functions of the form $x : I \to M$ 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. The obvious example, in this case, is the class of sampled data systems in which the index set I is the Cartesian product of the integers (discrete-time) and real numbers (continuous-time). *Supervisory hybrid systems*, on the other hand, 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.

This paper is concerned with the study of *supervisory hybrid systems.* A common example of such 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. 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. We therefore see that 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 [63] [64] [65] [66] [67] [68] [69]. One of the primary motivations for this interest rests with the fact 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're 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 the nation as these systems constitute major components of the national infrastructure. Current methods for the design and analysis of such systems rely heavily on simulation testing which is 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.

The remainder of this paper is organized as follows. Section 2 provides a concrete example of a hybrid system which will be used throughout the paper as a pedagogical tool illustrating various concepts in hybrid systems theory. Section 3 discusses modeling frameworks for hybrid systems with specific attention being paid to the *hybrid automaton* in section 4. Not only may the system have a hybrid character, but the specifications on desired system behaviors may also be hybrid. Section 5 discusses specification logics for hybrid systems that express system requirements on both the discrete and continuous states of the system. Section 6 surveys current methods and concepts used to verify or validate desired system behaviors and then concludes with a survey of current methods for hybrid control system synthesis. Final remarks will be found in section 7.

2 Hybrid System Example

A concrete example of a supervisory hybrid system will be found in figure 1. This 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 figure 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 ensure both arms don't enter the parts bin at the same time. In other words, the robotic system needs to treat the parts bin as a *critical section* which both arms access in a mutually exclusive manner.

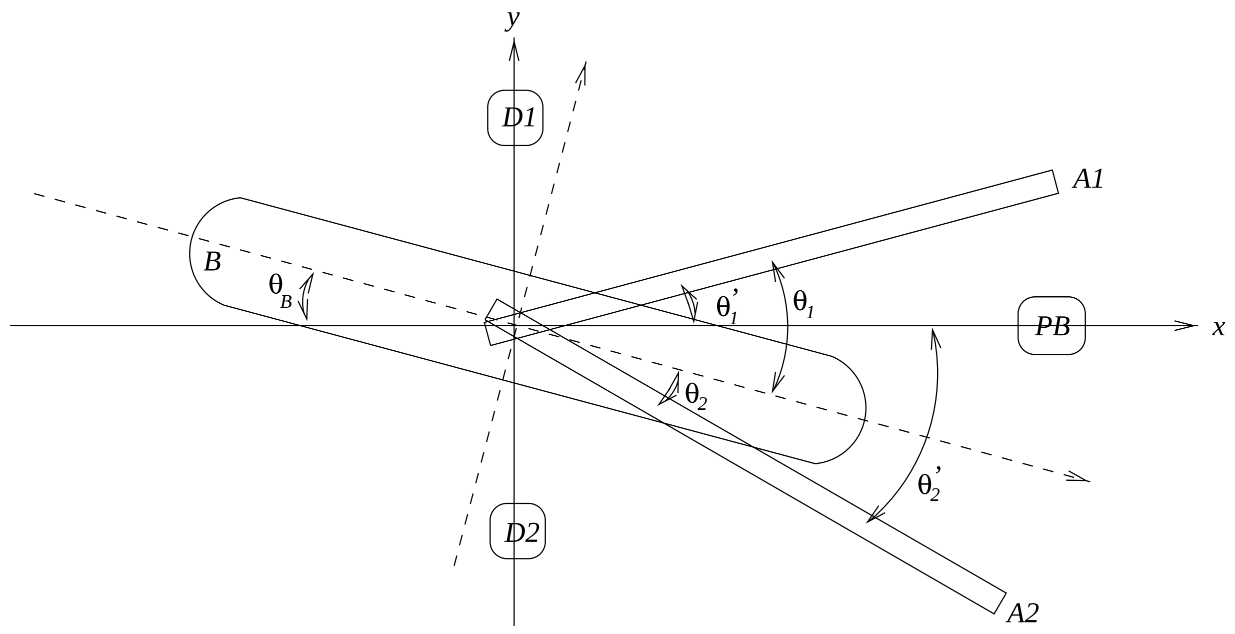

Figure 1: Free Floating Robotic System

A candidate solution to the mutual exclusion problem can be readily developed [58] [59] [62]. Let's assume that each arm is controlled by a computer process (an instantiation of the arm control program). We therefore have two concurrently running computer processes that 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 multi-tasking operating system (O/S) controls the execution of both computer processes, we can then use O/S control structures such as *semaphores* or *mutexes* [60] 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 hopefully expect the robotic arms (i.e. the physical system) to respect that requirement as well.

The pseudo-code for one of the computer processes is shown below

```
ENTRY: if(x==1) goto ENTRY
       x=1;
CRIT1: if(arm_not_in_partsbin) command_arm(1);
EXIT:  x=0;
ERR:   if(arm_locked) STOP;
REM:   if(arm_not_in_workarea) command_arm(-1);
       goto ENTRY;
```

This code segment has four distinct segments. There is an entry section (`ENTRY`) which tests the lock variable `x` to see if the other arm is moving towards the parts bin. In practice, the lock variable could be implemented as a O/S semaphore. If the lock variable `x` is 1, 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 `command_arm(1)`). 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 ouptuts the command signal to the arm (the function call `command_arm(-1)`) which moves the arm towards the work area. We've also included an *error* state (`ERR`) in the program that aborts the program's execution if the arm hits its mechanical limits (i.e. `arm_locked` evaluates to true). From this pseudo-code, we see that the state of the program can be characterized by 3 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' 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 a number of distinct states. Are these discrete states sufficient to represent the behavior of the physical process? For this particular system, the answer is negative 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

$$\begin{aligned} \ddot{\theta}_1 &= -\dot{\theta}_1 + k(\theta_1 + \theta_b - r_1) \\ \ddot{\theta}_2 &= -\dot{\theta}_2 + k(\theta_2 + \theta_b - r_2) \end{aligned} \tag{1}$$

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 figure 1). For this example, we see that 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. We therefore know that the body angle θ_b with respect to an inertial frame must satisfy

$$J_b\dot{\theta}_b + J_a\dot{\theta}_1 + J_a\dot{\theta}_2 = 0 \tag{2}$$

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 the 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. by measuring $\theta_1 + \theta_b$. We assume that the computer process 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, it is assumed that 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 it may be possible for this system to fail in ways that cannot be predicted by an examination of the controlling computer processes. From equation 2, 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, then the system would *deadlock* (i.e. the program would be get stuck in one of its discrete states). In other cases, system dynamics imply a subtle coupling between both arms which 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 figure 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 system science is to provide such a formal framework and the following sections discuss what progress has been made to date in this direction.

3 Hybrid System Modeling

Hybrid systems have been studied extensively by computer scientists and system scientists. Computer scientists have been interested in the behavior of real-time and multi-processor 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* [74]. These are systems that can be implicitly modeled by the following set of equations,

$$\begin{aligned} \dot{x} &= f(x(t), i(t)) \qquad (3) \\ i(t) &= q(x(t), i(t^-)) \qquad (4) \end{aligned}$$

where $x : \Re \to \Re^n$ is the continuous valued state trajectory and $i : \Re \to \Omega$ 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 $f : \Re^n \times \Omega \to \Re^n$ represents a set of continuous dynamical systems (vector fields). 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 equation 4. In this equation, $i(t^-) = \lim_{\tau \uparrow t} i(\tau)$ represents the righthand limit of the function $i(t)$ at t. Equation 4 means that the discrete state transtions 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 sets we might transition to is characterized by the discrete transition function, $q : \Re^n \times \Omega \to \Omega$.

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

$$\Omega_{ij} = \{x \in \Re^n \ : \ j = q(x,i)\} \tag{5}$$

This set is the set of all continuous states which 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). We therefore see that the switching action may be initiated whenever the system's continuous state evolves across that boundary.

A natural question to be asked 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 [7]. These are very general conditions and are satisfied by most of the systems of interest.

The existence conditions [7], however, do not preclude the existence of hybrid system continuous state trajectories which are not absolutely continuous. Switching systems of the form shown in equations 3 and 4 are well known to exhibit *chattering* solutions in which the system switches infintely fast between two different types of vector fields. The existence and exploitation of this relaxed behavior is, in fact, a basic principle behind another important class of hybrid systems known as variable structure systems [8]. It is interesting to note that computer scientists also have an interesting 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 want all of our systems to be non-Zeno.

While we can usually ensure the existence of solutions to such equations, there is no guarantee that these solutions will be unique. The switching systems represented in equations 3 and 4 can also be treated as differential inclusions for which it is well known that nondeterministic solutions 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. This nondeterminism is, in fact, a fundamental property of hybrid systems and it represents one important way in which hybrid systems theory differs from traditional linear systems theory.

The switching system introduced in equation 3 and 4 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 [6] [5]. 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 will be found in [4].

The preceding references to the hybrid system's 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 to be faced by any hybrid systems paradigm, however, involves developing a framework which 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 will be found in

[9]. In this case, the hybrid system was viewed as logical discrete event supervisor connected to a continuous subsystem. The discrete and continous systems were interconnected through an interface that transformed continuous-valued measurements into discrete event signals and vice versa. This work suggested a logical discrete-event system (DES) approach to hybrid controller synthesis which 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 [61].

While providing a very general framework for hybrid systems, the model in [9] [11] [10] was of limited utility due to the restrictive nature of the control. A framework with significant potential for practical useage was developed by the computer science community [1] [2]. 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 and while powerful computational tools were developed for the manipulation of such formal models, it was apparent that in dealing with multiprocessors and real-time applications, that 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 [28] for finite state machines to real-time systems. The result was a *timed* [2] and *hybrid automaton* [1] . These automata were generalizations of traditional finite state machines in which event transitions were conditioned on the truth value of logical propositions defined over a set of continuous-valued dyanmical processes. The work was very influential in that it led to the development of verification tool [3] [22] [23] [27] 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 [13] [70]. Another related version of the hybrid automaton will be found in [14]. Extensions of the approach using Petri nets (rather than finite state machines) will be found in [15] [46] [16] [17] [18] [19]. While the hybrid automaton has been very influential in the study of hybrid systems, there are some significant limitations and much recent research has attempted to define the boundary of these limitations. Nonetheless, the hybrid automata in spite of its limitations represents an necessary starting point for the study of hybrid systems theory. For this reason the following section will present the hybrid automaton in more detail.

4 Hybrid Automata

The hybrid automaton is an extension of the traditional finite state machine [88]. It can be defined as a 3-tuple $(\mathcal{N}, \mathcal{X}, \mathcal{L})$ where $\mathcal{N}$ is a labeled marked directed graph called a *network*, $\mathcal{X}$ is a set of continuous-valued dynamical processes called *timers* 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 timers $\mathcal{X}$ represent the continuous dynamics of the hybrid system. The relationship between these two subsystems is captured by the labeling function $\mathcal{L}$.

A network or directed graph is the ordered pair (V, A) where 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 node v_j is used to represent the arc (v_i, v_j). As a specific example of a network, let's consider the set of vertices

$$V = \{v_1, v_2, v_3, v_4, v_5\} \tag{6}$$

and the set of arcs

$$A = \{(v_1,v_2),(v_2,v_3),(v_3,v_4),(v_4,v_1),(v_2,v_5),(v_4,v_5)\} \tag{7}$$

Figure 2 shows the graphical representation of this network.

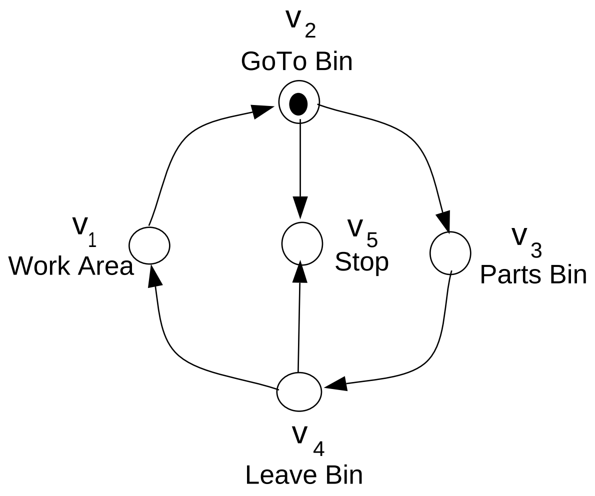

Figure 2: Network for a Discrete Event System's State Space

The network (V,A) denotes all possible states that a discrete-event system might occupy. Which state a specific system is currently occupying is shown by *marking* the network. A marked network is the 3-tuple (V,A,μ) where V and A are the network vertices and arcs, respectively. The final element of the triple is a function $\mu : V \to \{0,1\}$ which associates either zero or one with each vertex of the network (V,A). If $\mu(v) = 1$, then we say vertex v is marked. Otherwise the vertex is unmarked. Graphically, we mark a network by placing a small solid circle (also called a *token*) in the marked vertex. As shown in figure 2, the vertex v_2 is marked.

By itself, the marked network (V,A,μ) is an abstract mathematical object. We now *bind* this object to a specific interpretation so it becomes a model of something. Such a binding is accomplished by labeling the vertices of the network with strings or names that have a concrete meaning. Formally, we denote a labeled marked network by the 4-tuple, (V,A,μ,ℓ) where (V,A,μ) is a marked network and $\ell : V \cup A \to \Omega$ maps the vertices and arcs of the network onto a discrete set of labels. Consider for example, the network shown in figure 2 and let's introduce the following labeling function on the vertices

$$\begin{aligned}
\ell(v_1) &= \texttt{WorkArea} && (8)\\
\ell(v_2) &= \texttt{GoToBin} && (9)\\
\ell(v_3) &= \texttt{PartsBin} && (10)\\
\ell(v_4) &= \texttt{LeaveBin} && (11)\\
\ell(v_5) &= \texttt{Stopped} && (12)
\end{aligned}$$

The labeled vertices are shown in figure 2. With these labels, the network shown in figure 2 provides a graphical model for the computer program we used to control the arms of the vehicle in figure 1. There are, in this figure, 4 discrete states associated with each of the program segments given in the pseudo-code described above. In addition to these 4 states, we've also included a fifth state (`Stopped`) that represents a failure condition under which the system does an emergency stop.

As presented so far, the labeled marked network $N = (V, A, \mu, \ell)$ represents the discrete state of the program controlling the arms in the paper's robot example. We can also, however, introduce a very simple dynamical rule which allows us to view N as a discrete-event dynamical system. Denote the *preset* and *postset* of a vertex v as $\bullet v$ and $v\bullet$, respectively. Define both of these objects as

$$\begin{aligned} \bullet v &= \{w \in V \,:\, (w,v) \in A\} &(13)\\ v\bullet &= \{w \in V \,:\, (v,w) \in A\} &(14) \end{aligned}$$

The preset (postset) of v therefore consists of all vertices which are connected to v by an input arc , (w,v) (output arc, (v,w)). An arc (w,v) will be said to be *enabled* if and only if $\mu(w) = 1$. Any enabled arc may *fire*. Let μ be the network's marking function before enabled arc (v_1, v_2) fires and let μ' denote the marking function after the arc fires. The relationship between μ and μ' is

$$\mu'(w) = \begin{cases} 1 & \text{if } w = v_2 \\ 0 & \text{if } w = v_1 \\ \mu(w) & \text{otherwise} \end{cases} \tag{15}$$

In other words, the firing of arc (v_1, v_2) unmarks vertex v_1, marks vertex v_2, and leaves all other vertices in the network unchanged.

The labeled marked network described above is sometimes referred to as a *finite state machine*. Finite state machines are often referred to as finite automata. This paper does not distinguish between the two structures. It should be noted that finite automata are usually defined from a language theoretic formulation. To keep the presentation more compact, we treat finite state machines and finite automata in the same way using a graph theoretic formalism. While these models are very useful, it is common practice to augment the structure by labeling the arcs and vertices with statements conditioning the firing of arcs. One common example of such an augmented network is found in logical DES control where finite state machines are augmented with conditional labels that disable the firing of specific arcs of the network as a function of the network's current marking. When these conditional statements are also functionally related to the states of a continuous-valued dynamical system, then we obtain the Alur-Dill hybrid automaton [1].

The Alur-Dill hybrid automaton was introduced in response to a need to accurately model the behavior of real-time programs. Real-time systems, of course, contain an implicit dynamical system; a clock with associated differential equation $\dot{x} = 1$. The clock can be used to condition program execution so that program code segments at executed at the correct real-time, not just in the correct order. Any control systems engineer with experience in the development of embedded control systems will be aware of the use of interval timers in controlling program execution in real-time. The hybrid automata model was introduced to capture this aspect of real-time programming.

To formally define the hybrid automaton, we need to introduce the timers and labels mentioned in the opening paragraph of this section. We define the *i*th *timer* by the ordered triple $x_i = (f_i, x_{i0}, t_{i0})$ where $f_i : \Re^n \to \Re^n$ is a Lipschitz continuous vector field defined over the continuous state space $\Re^n$, x_{i0} is an initial condition in $\Re^n$, and t_{i0} is an initial time in $\Re$. The timer triple, therefore, can be viewed as an initial value problem and the *time* of our timer is denoted by the state trajectory, $x_i(t)$ for $t \geq t_{i0}$ that satisfies the following initial value problem

$$\begin{aligned} \dot{x}_i &= f_i(x) &(16)\\ x_i(t_{i0}) &= x_{i0} &(17) \end{aligned}$$

We let X denote a set of timers of the form given above. The set X characterizes the continuous dynamics of our hybrid system. If the vector field f is unity, then we call the timer a *clock*. The *state* of the *i*th timer at time t will be denoted as $z_i(t) = (\dot{x}_i(t), x_i(t))$, i.e. it is defined with respect to the *i*th timer's rate and value.

In a hybrid automaton, $(\mathcal{N}, \mathcal{X}, \mathcal{L})$, the labels $\mathcal{L}$ tie the discrete and continuous parts of the system together. These labels are mappings from the vertices and arcs of the network $\mathcal{N}$ onto formulae in a propositional logic, $\mathcal{P}$, whose truth values are evaluated with respect to the current timer states, z. The logical propositions labeling the network nodes and arcs can be defined in a variety of ways. In this paper we choose the following. We first introduce a set of *atomic equations* defined over the variables $\dot{x}_i$, x_i, x_{i0}, and f_i. Let a and b be real vectors and let c be a real constant, then the basic atomic equations are:

- *switching equations* of the form $[\dot{x}_i = f_j]$. This formula states that the ith timer's rate is equal to vector field f_j,
- *guard equations* of the form $[a'x_iRb'x_j]$ or $[a'x_iRc]$. These inequalties mean that the inner product of a and x_i, $a'x_i$, stands in relation R (either $<$ or $>$) to $b'x_j$ or real constant c.
- *reset equations* of the form $[a'x_{i0} = c]$ which means that $a'x_i$ is equal to real constant c.

Legal formulae in $\mathcal{P}$ are defined inductively by the following rules:

- Any atomic equation is in $\mathcal{P}$,
- If p and q are in $\mathcal{P}$ then $[p \wedge q]$ is in $\mathcal{P}$
- if p is in $\mathcal{P}$ then$\tilde{}p$ is in $\mathcal{P}$.

The preceding paragraph defined the syntax for formulae in $\mathcal{P}$. The *meaning* or *interpretation* of these formulae is made with respect to the timer states $z = (\dot{x}, x)$. In particular, we say that an atomic formula is satisfied by timer state $z(t) = (\dot{x}(t), x(t))$ if and only if the equation is true when evaluated with respect to those states at time t. The formula $p \wedge q$ is true if both p and q are true under the given timer states. The formula$\tilde{}p$ is true if p is not true under the current timer states. The current timer state is said to satisfy a formulae $p \in \mathcal{P}$ if and only if it has the truth value of true.

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 hybrid automata, this interaction is defined according to the following rules:

- The network arcs are labeled with equations in $\mathcal{P}$ formed from guard atomic equations, $[a'x_iRc]$ or $[a'x_iRb'x_j]$. These conditions on the arcs represent additional enabling conditions for an arc's firing. Recall that an arc $((w,v))$ of a network may only fire if vertex w is marked. In the hybrid automaton, this same arc may fire at time t if and only if vertex w is marked and $\mathcal{L}((w,v))$ is true at time t.
- The network vertices are labeled with formulae whose atomic equations are switching or reset equations. These formulae are interpreted as follows. If the formulae do not evaluate to true when the vertex is first marked , then the timer states will be reset to make these predicates true. In the case of the reset equations, this means that the clock time, x is reset to the specified value. For switching equations, the timer's rate, $\dot{x}$ is set to the specified vector field. We therefore see that these reset/switching conditions allow the hybrid automaton to model autonomous jumping and switching behaviors.

There are several important classes of hybrid automata. When the timers are chosen to be clocks, then we obtain the class of linear hybrid automata. Timed automata are linear hybrid automata whose guard conditions define rectangles in the continuous state space. The class of rectangular hybrid automata occur when the timers are represented as rectangular differential inclusions of the form $\dot{x} \in [a,b]$ and the guard conditions also define rectangular sets.

The system whose continuous dynamics are illustrated in figure 1 and whose discrete dynamics are illustrated in figure 2 can be easily modeled using a hybrid automaton. The resulting hybrid automaton is shown in figure 3. 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 $\tilde{}[x > 0]$. In other words when the lock variable x is no longer nonzero this arc may fire and the system will switch to the logical state `GoToBin`.

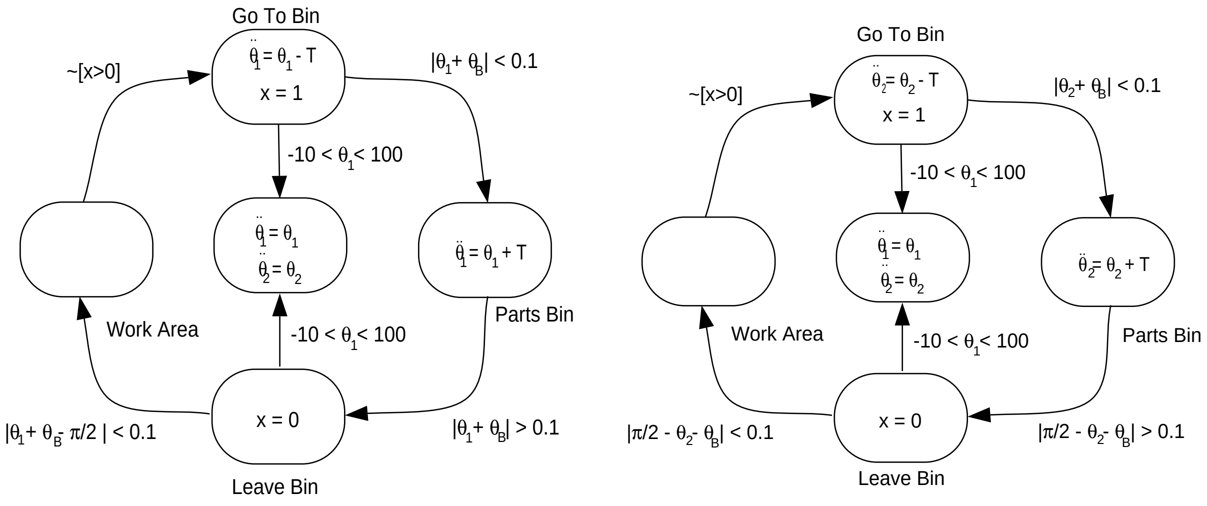
Go To Bin
$\ddot{\theta}_1 = \theta_1$ - T
x = 1
~[x>0]
$|\theta_1 + \theta_B| < 0.1$
-10 < θ_1< 100
$\ddot{\theta}_1 = \theta_1$
$\ddot{\theta}_2 = \theta_2$
$\ddot{\theta}_1 = \theta_1$ + T
Work Area
Parts Bin
-10 < θ_1< 100
x = 0
$|\theta_1 + \theta_B - \pi/2| < 0.1$
$|\theta_1 + \theta_B| > 0.1$
Leave Bin
Go To Bin
$\ddot{\theta}_2 = \theta_2$ - T
x = 1
~[x>0]
$|\theta_2 + \theta_B| < 0.1$
-10 < θ_1< 100
$\ddot{\theta}_1 = \theta_1$
$\ddot{\theta}_2 = \theta_2$
$\ddot{\theta}_2 = \theta_2$ + T
Work Area
Parts Bin
-10 < θ_1< 100
x = 0
$|\pi/2 - \theta_2 - \theta_B| < 0.1$
$|\pi/2 - \theta_2 - \theta_B| > 0.1$
Leave Bin

Figure 3: Hybrid Automaton for Robotic System

The discrete state `GoToBin` is labeled with the predicate $[x = 1] \wedge [\ddot{\theta}_1 = -\dot{\theta}_1 + k(\theta_1 + \theta_b)]$ This predicate sets the lock variable to 1 thereby indicating to arm 2 that it is heading towards the parts bin. While in this state, the system also sets its timer rate $\ddot{\theta}_1$ to the vector field which begins moving the arm towards the parts bin. The arc connecting `GoToBin` to the discrete state `PartsBin` is labeled with the conditional predicate $[\overline{\theta}_1^2 - 100\overline{\theta}_2^2] < 0]$. This conditional predicate is a indefinite quadratic form representing a conic sector enclosing the parts bin.

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 `Stopped`. The condition for transitioning to the `Stopped` state is $[\theta_1 > 100] \vee [\theta_1 < -10]$ This is a safety condition which is triggered if the arm moves too far (i.e. hits its physical stops). In this case, we transition to the `Stopped` state. The `Stopped` state is a deadlocked state from which all forward progress in the system ceases. It is labeled by a predicate which turns off the system, so it is labeled with the predicate $[\ddot{\theta}_1 = -\theta_1] \wedge [\ddot{\theta}_2 = -\theta_2]$ thereby causing both arms to eventually stop their motion.

Once in the parts bin, the system begins moving the arm out of the bin. Therefore the state `PartsBin` is labeled with the predicate $[\ddot{\theta}_1 = -\dot{\theta}_1 + k(\theta_1 + \theta_b - \pi/2)]$ 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 $[\overline{\theta}_1^2 - 100\overline{\theta}_2] > 0$. 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 $[x = 0]$. Finally, the system returns to the WorkArea state if the appropriate conditions on the angle are satisfied or exits to the Stopped state if the limit conditions on the arm's angular position are violated.

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

5 System Specifications

Control theoretic measures of system performance are frequently taken to be the *size* of some important signal within the control system's feedback loop. Signal size is measured using a functional that maps each signal (function) 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 bound 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 figure 1. In this case, the mutual exclusion requirement is a high level behavioral constraint which 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 it is imperative that a more expressive method be adopted for capturing the designer's requirements.

Formal logics can be used to express more complex system specifications. We've already used a propositon logic to characterize the labels for a hybrid automaton. We now turn to the use of formal logics, and in particular temporal logic, to express requirements on desired system behavior.

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 we 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 will drop the subscript F.

In temporal logics, the frame states can be ordered (i.e. with respect to order of occurrence) and this allows us to introduce and reason about several notions of time. A linear temporal logic assumes all states are strictly ordered and hence 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. We refer to this logic as CTL1. It is a subset of the well-known computation tree logic (CTL).

Before defining the atomic propositions, syntax and semantics of our specification logic, we need to introduce some preliminary notation. Consider the hybrid automaton, $\mathcal{H} = (\mathcal{N}, \mathcal{X}, \mathcal{L})$. The hybrid state of the system at time t is denoted as $\sigma(t) = (x(t), \mu(t))$ where $x(t)$ is the continuous-valued state trajectory and $\mu(t)$ is the network marking history (as a function of time). We define the *event projection* $\pi_e(\sigma(t))$ by the equation

$$\pi_e(\sigma(t)) = \mu(t_1), \mu(t_2), \cdots, \mu(t_n), \cdots \tag{18}$$

In other words, the event projection of $\sigma(t)$ is a sequence of discrete network states in which no two adjacent states are the same. The event projection $\pi_e(\sigma)$ is sometimes called the *trace* of the hybrid trajectory.

Let $\sigma(t)$ be a hybrid system trajectory, then the *atomic* formulae for our specification logic take the form, $[a'x(t) > b]$ 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 trace at time t equals μ.

The *syntax* of a logic is a set of fundamental legal formulas. These fundamental formulas provide a way of inductively defining all legal formulas in CTL1. In our case, the syntax is as follows.

- p is in CTL1 if p is atomic
- if p is in CTL1 then $\tilde{}p$ is in CTL1
- if p and q are in CTL1 then $p \wedge q$ is in CTL1.
- if p and q are in CTL1 then $p \exists \mathcal{U} q$ is in CTL1,
- if p and q are in CTL1 then $p \forall \mathcal{U} q$ is in CTL1

The formula $\forall \mathcal{U} p$ and $\exists \mathcal{U} p$ are equivalent to [true]$\forall \mathcal{U} p$ and [true]$\exists \mathcal{U} p$, respectively. The notation above may seem confusing, but essentially, we are assuming that all legal formulas in CTL1 can be expressed as either a predicate, p, a predicate with a unary operator, (i.e. $\tilde{}\, p$, $\forall \mathcal{U} p$, $\exists \mathcal{U} p$) or a binary operation on legal predicates, (i.e. $p \wedge q$, $p \forall \mathcal{U} q$, $p \exists \mathcal{U} q$). Note that $\forall \mathcal{U}$ and $\exists \mathcal{U}$ represent binary or unary operators on predicates. These formulas provide a way of building up more complex formulas. Therefore the formula $p \vee [q \forall \mathcal{U} r]$ is legal because p and $[q \forall \mathcal{U} r]$ are both legal predicates according to the syntactical rules given above.

The preceding syntactical formulas represent a set of abstract formulas but provide no interpretation or meaning to these formulas. The interpretation of the formulas is determined by the logic's semantics. The *semantics* of CTL1 are defined with respect to a hybrid state, s. Let $\sigma(t) = (x(t), \mu(t))$ be a hybrid trajectory generated by the hybrid automaton, $(\mathcal{N}, \mathcal{X}, \mathcal{L})$. As the frame is given, we drop explicit mention of it in the formulae. The meaning of the generating CTL1 formulae is as follows:

$$s \models p \quad \Leftrightarrow \quad p \text{ is satisfied by state } s \tag{19}$$

$$s \models \tilde{}p \quad \Leftrightarrow \quad p \text{ is not satisfied by state } s \tag{20}$$

$$s \models p \wedge q \quad \Leftrightarrow \quad p \text{ and } q \text{ are satisfied at state } s \tag{21}$$

$$\begin{aligned} s \models p\exists U q \quad &\Leftrightarrow \quad \text{there exists a hybrid trajectory } \sigma(t) \text{ such that } \sigma(0) = s \text{ and a time } t_1 \text{ such that} && (22)\\ &\qquad \sigma(t) \models p \vee q \text{ for } t < t_1 \text{ and } \sigma(t_1) \models q. && (23)\\ s \models p\forall U q \quad &\Leftrightarrow \quad \text{for all hybrid trajectories } \sigma(t) \text{ such that } \sigma(0) = s\text{, there exists a time } t_1 \text{ such that} && (24)\\ &\qquad \sigma(t) \models p \vee q \text{ for } t < t_1 \text{ and } \sigma(t_1) \models q. && (25) \end{aligned}$$

We therefore see that the formula $p \vee q$ represents our usual notion of logical conjunction where as$\tilde{}\, p$ represent the logical not operation. The other two formulae $p\forall Uq$ and $p\exists Uq$ have a special meaning which is specific to temporal logics. These operators provide a way of describing temporal relationships between predicates. The formula $p\forall Uq$ can be seen as saying that *for all* hybrid trajectories, predicate p is true *u*ntil predicate q is true. The formula $p\exists Uq$ is the other existential formula meaning that *there exists* a trajectory in which p is true *u*ntil q is true.

CTL1 allows us to express complex specifications relating the discrete and continuous states of the hybrid system. It should be noted that we've made no attempt in this paper to construct a complete logic. More powerful temporal logics using the hybrid automaton as a frame will be found in [20] and [21]. CTL1, as introduced in this paper, 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 figure 1.

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

$$\forall U\tilde{}\,\tilde{}\,[\mathtt{PartsBin}_1 \wedge \mathtt{PartsBin}_2] \tag{26}$$

This particular specification equation says that for all possible traces, the computer programs controlling both arms will not enter their critical sections at the same time. This mutual exclusion requirement, however, is only on the discrete part of the system and does not necessarily capture the true constraint we're interested in. A more realistic constraint on the system would be expressed as follows:

$$\forall U\tilde{}\,[[|\theta_1 + \theta_b| < .1] \wedge [|\theta_2 + \theta_b| < .1]] \tag{27}$$

This constraint addresses the mutual exclusion constraint on the physical system by specifying that all hybrid trajectories respect the physical constraints defining entry into the parts bin.

By itself, of course, equation 27 still does not precisely capture our desired behavior. A stronger constraint would require that the conditions defining mutual exclusion for the discrete and continuous systems coincide. This later requirement can be captured by the following specifcation on arm 1's angular position,

$$\forall U[[|\theta_1 + \theta_b| < .1] \wedge [\mathtt{PartsBin}_1]] \tag{28}$$

and by a similar specification on arm 2. This specification requires that the arm is in the parts bin if and only if the desired angle conditions are satisfied. In this case, the state conditions ensuring the first condition is satisfied become an invariant characterization of the discrete state `PartsBin` and if we can ensure this along with the preceding discrete constraint in equation 26, then mutual exclusion is guaranteed in the system in a very strong sense.

The specification in equation 28 illustrates one important approach to hybrid system analysis and design. The approach involves abstracting a discrete-event model for the hybrid system in such a way that checking the behaviour of the discrete event system is sufficient to ensure the safety of the complete system. We sometimes refer to this abstracted model as a *bisimulation* of the original hybrid system. In our example, the high level logical model for the system will be a bisimulation of the physical plant if we can guarantee that the physical constraints defining the parts bin are entered if and only if the discrete system state is `PartsBin`. If this is the case, then controlling the discrete-event structure will clearly be sufficient to ensure the safe operation of both the discrete and continuous parts of the hybrid system.

Not all solutions to the mutual exclusion problem are equally desirable. An easy way to guarantee mutual exclusion is to require that the system deadlocks in a safe state. 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 weakly deadlock-free if each process in its entry section is guaranteed of eventually transitioning into its critical section. The weakly deadlock free specification may be expressed by the CTL1 formula,

$$[\texttt{WorkArea}]\forall U[\texttt{PartsBin}] \tag{29}$$

This says that for all hybrid trajectories which start in the `WorkArea`, there is some time when the system ends up in the discrete state `PartsBin`.

As noted before, the specification in equation 29 is a requirement for weak deadlock freedom. The 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,

$$[\texttt{WorkArea}]\forall U[[\texttt{PartsBin}] \wedge [x_1 < c]] \tag{30}$$

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

The preceding examples illustrate how various hybrid specifications might be expressed as formulae of a computational tree logic. This logical framework is clearly more expressive than the traditional norm bounded approach to specifying control system performance. The particular logic used here, however, is extremely simple and it should be noted that there is still considerable work being done to investigate specification logics for hybrid systems. Recent work in [20] and [21] have augmented CTL to reason about time intervals and there is a *duration calculus* [24] [25] [26] [27] which also appears to form a very attractive specification language for hybrid systems. This paper has only presented some of the basic principles and ideas behind using logics to formally specify hybrid system behaviour.

6 Verification, Validation, and Synthesis

Given a system model and a specification on that model there are two classes of problems to consider; the *analysis* and *synthesis* problems. The analysis problems 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 only 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 frequently impractical from a computational standpoint to verify system properties such as stability and robust performance. In both cases, we're 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 by parametrizing the verification or validation problems, it may be possible to effectively search for system parameters ensuring 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. In [29] a similar approach has been proposed for using existing verification tools to help synthesize hybrid system controllers.

The objective of this section is to provide 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 then discussed.

6.1 Verification

Much of the early work in hybrid systems analysis will be found in the computer science literature. This body of work assumes that the specification is posed in a temporal logic such the timed computation tree logic (TCTL), the time μ-calculus [20] or the duration calculus [25]. The frame for these logics is often taken to be the hybrid automaton, though there have been recent effort looking at alternative frames such as Petri nets. By far the best known work has been done for hybrid automata with specification logics based on Emerson's computation tree logic [34]. This work [23][22] [21][20] [1] [3] attempts to extend symbolic model checking to real-time systems.

Symbolic model checking (SMC) [28] is a verification method in which the frame is a finite state machine and the specification language is the branching temporal logic, CTL. Through the use of binary decision diagrams (BDD) it has been possible to answer queries posed in CTL in a computationally efficient manner [28]. As a result symbolic model checking has become a standard way of checking digital VLSI circuits [35].

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 [1] and software implementing the approach was also developed [3].

How does model checking for hybrid systems work? It is easiest to begin by considering classical SMC for finite state machines and then examine the extensions required for checking hybrid systems. Traditional model checking [28] assumes that the specifcation is framed in CTL. The interesting thing about CTL formulae is that they are fixed points of special recursive operators and this means, therefore, that the satisfiability of such formulae can be readily computed by the repeated application of these operators [33]. A full discussion of symbolic model checking is beyond the scope of this paper, but a simple example will serve to illustrate the basic principle.

Let's consider the finite state machine shown in figure 3 and the CTL predicate,

$$p = \exists U[\texttt{PartsBin}] \tag{31}$$

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

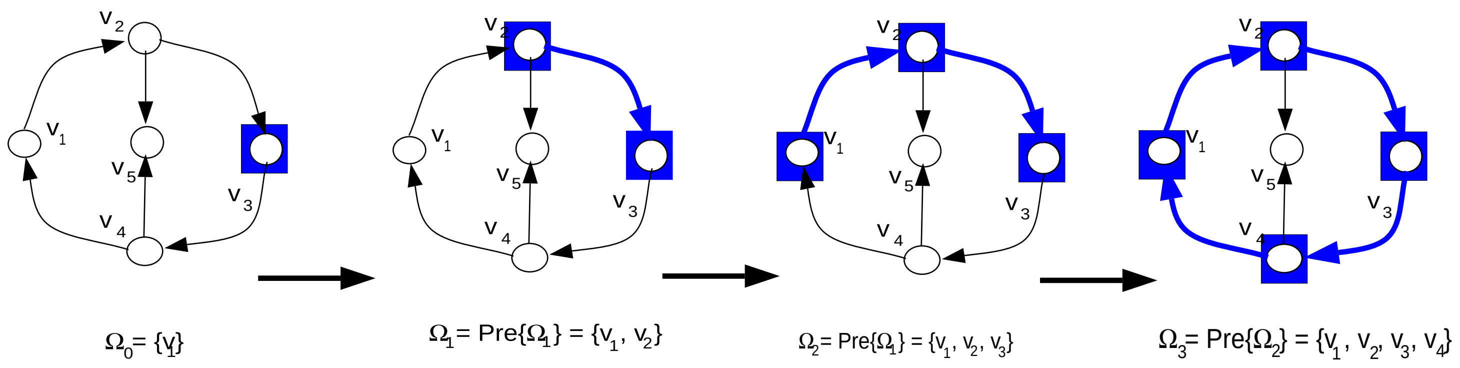
v_2
v_1
v_5
v_3
v_4
$\Omega_0 = \{v_1\}$
$\Omega_1 = \mathrm{Pre}\{\Omega_1\} = \{v_1, v_2\}$
$\Omega_2 = \mathrm{Pre}\{\Omega_1\} = \{v_1, v_2, v_3\}$
$\Omega_3 = \mathrm{Pre}\{\Omega_2\} = \{v_1, v_2, v_3, v_4\}$

Figure 4: Model Checking Iteration

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 $\exists U p$ is true. In this case, we see that $\Omega_0 = \{\texttt{PartsBin}\}$. The next set Ω_1 is generated by the relation

$$\Omega_1 = \Omega_0 \cup \Delta \tag{32}$$

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

$$\Omega_1 = \{\texttt{GoToBin}, \texttt{PartsBin}\} \tag{33}$$

We repeat the above iteration repeatedly, computing in the ith iteration, the set of discrete states that may reach Ω_{i-1} in a single step. 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 satisfyng the CTL formula $\exists U p$. 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

$$\Omega = \{\texttt{PartsBin}, \texttt{WorkArea}, \texttt{GoToBin}, \texttt{LeaveBin}\} \tag{34}$$

Figure 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 represents the set of discrete states which can reach a discrete state satisfying the predicate p in CTL formula $\exists U p$. We therefore see that the iterative procedures used in symbolic model checking are essentially solving reachability problems over the discrete event system's state space. The specification $\exists U p$ 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, then the specification can be considered to be verified.

Extending SMC methods to hybrid systems involves solving the reachability problem for both continuous and discrete system states. As before, let's consider the verification of the CTL formula $\exists U p$ where $p = [\texttt{PartsBin}]$. 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 the satisfaction of the guard equation labeling the arc in question. This implies that while 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 states θ_1 and θ_b satisfy the guard condition, $\theta_1^2 - 100\theta_2^2 < 0$. Extensions of SMC methods to hybrid systems must therefore determine methods for computing subsets 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. Let's consider a sequence of pairs of sets, $(\Omega_0, \Xi_0), (\Omega_1, \Xi_1), \cdots, (\Omega_n, \Xi_n), \cdots$. The subsequence $\{\Omega_i\}$ consists of sets of discrete states and the subsequence $\{\Xi_i\}$ consists of subsets of continuous states. We now introduce a recursive procedure for computing a sequence of sets that can verify the existential formula, $\exists Up$. Let Ω_0 consist (as before) of all states that satisfy the given predicate p. Let Ξ_0 consist be a subset of the continuous state space that satisfies the guard condition on the arcs leading into the states in Ω_0. We determine the next discrete set Ω_1, as before, by adding vertices from the preset of Ω_0. The next set Ξ_1, is computed as follows. Let's consider all arcs between elements of Ω_0 and Ω_1. The dynamics of the continuous part of the system are determined by the *switching equations* labeling the vertices of Ω_1. We therefore propagate back from Ξ_0 using these continuous-dynamics until the guard conditions on the arcs leading into the vertices of Ω_1 are satisfied. The resulting set Ξ_1, then represents a subset of the continuous state space from which the hybrid automaton can reach Ξ_0 under the appropriately selected dynamics.

Figure 5 illustrates how this computation might appear for the specific example considered here. In this case, Ξ_0, is the set outlined at time $t = t_f$. By propgating back until the states enter the work area, we obtain a polytopic region representing the possible states (and times) which can reach Ξ_0. This set becomes Ξ_1. This operation is sometimes called the *precursor* operation and the set Ξ_1 is often written as $\Xi_1 = \text{Pre}(\Xi_0)$. Recursive application of this precusor operator constructs a sequence of discrete states Ω_i and continuous subsets Ξ_i, which represent the states (both discrete and continuous) that can reach those states satisfying the CTL formula's predicate, p. If the iteration converges to a fixed point (Ω, Ξ), then this fixed point represents all of the states that can satisfy the specification, $\exists Up$. As in the case of SMC verification, we then compare the allowed initial states against this fixed point (if it exists). If the starting states are a proper subset of the fixed point, then we know that this system will satisfy the specification. In other words, we've *verified* the specification for this particular system and set of starting states.

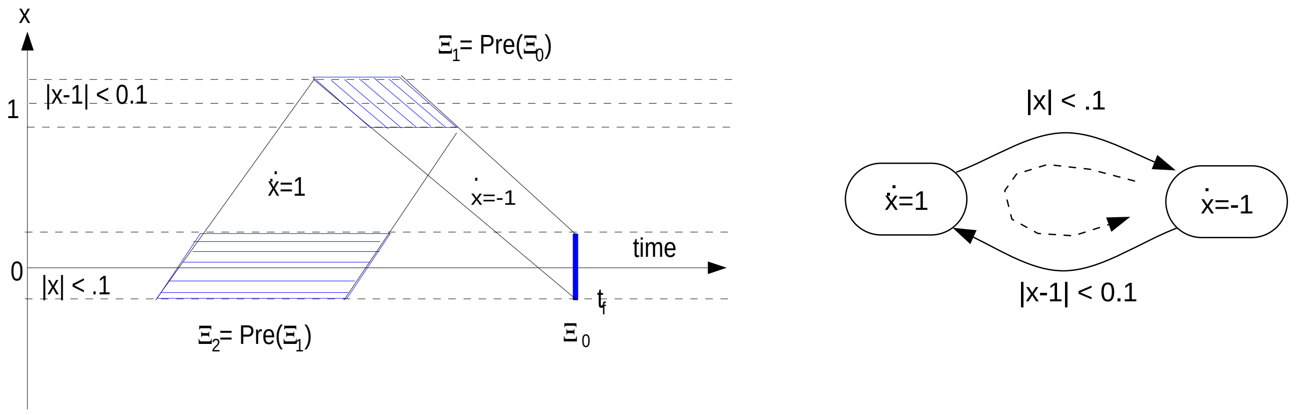

Figure 5: Preset of a Transition

The preceding discussion provides a simplified example of the basic concepts behind symbolic model checking for hybrid systems. As can be seen, this is a direct extension of traditional model checking methods. 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 non-finite nature of the computation highlights one of the great 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 [88].

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. It has been shown that even for the very restricted class of linear hybrid automata that verification is undecidable [30]. Suitable restrictions of linear hybrid automata, however, have yielded decidable verification problems. Timed automata and rectangular hybrid automata represent two such classes of decidable hybrid systems [31]. 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 [31] that the decidability boundary for hybrid systems may well rest with rectangular hybrid automata and therefore it was important to see how useful that class of systems would prove to be. In [36], 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 that rectangular hybrid automata were of limited utility in modeling many hybrid systems arising in practice. Very recently a larger class of decidable systems hybrid systems has been identified. These systems are referred to as *o-minimal* systems [81]. The significance of this larger class of decidable hybrid systems is still being investigated.

6.2 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 people from developing sufficient methods which are still of great utility.

An example of a very useful sufficient test will be found in Lyapunov's second method. Lyapunov's second method provides a sufficient test for system stability and it 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 $f(x_0) = 0$. We say that the equilibrium point is stable in the sense of Lyapunov if for all $\varepsilon > 0$ there is a $\delta > 0$ such that $\|x(0)\| < \delta$ implies $\|x(t)\| < \varepsilon$ 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). In spite of this shortcoming, however, Lyapunov methods still provide an extremely useful tool in the study of nonlinear dynamical systems.

Given the importance of Lypuanov methods, it is not surprising to find a variety of results on the Lyapunov stability of hybrid systems. In [37] a single Lyapunov function was used to determine sufficient tests for switched system (equations 3-4) stability. Multiple Lyapunov function approaches in [38] [40] [42] and

[41] greatly extended the applicability of Lyapunov analyses for hybrid systems. We now review some of this recent work on multiple Lyapunov functions. In [38] , 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 $\dot{x} = f_i(x)$ which are switched between on the basis of some supervisory control logic. Assuming that system switching is non-Zeno in character, then for the jth subsystem, we can identify a collection of closed bounded intervals, over which that system is active. Figure 6 illustrates one such hybrid system trajectory and identifies the set of disjoint time intervals over which the first subsystem is active. Assuming there are N systems to switch between, we associate a function V_j $(j = 1, \ldots, N)$ with the jth subystem. We say that this family of functionals is *Lyapunov-like* if $V_j(x(t))$ is decreasing over the intervals in which the jth subsystem is active. Figure 6 illustrates a set of Lyapunov-like functionals for this particular system. The result in [38] 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, it is possible for there to be discontinuous jumps in the value of $V_j(t)$ between different subsystems.

The fundamental concept in results such as [41] and [38] is that we only have to consider the behavior of the Lyapunov function over *cycles* in the switching behaviour. Examining figure 6, we see that a Lypaunov-like function is associated with each subsystem. Each subsystem, however, as 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 system 1 or rather vertex v_1. Therefore the contours of the Lyapunov-like function represent subsets of continuous states which the system returns to 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 represent a fundamental way in which the continuous and discrete dynamics of the hybrid system are coupled together.

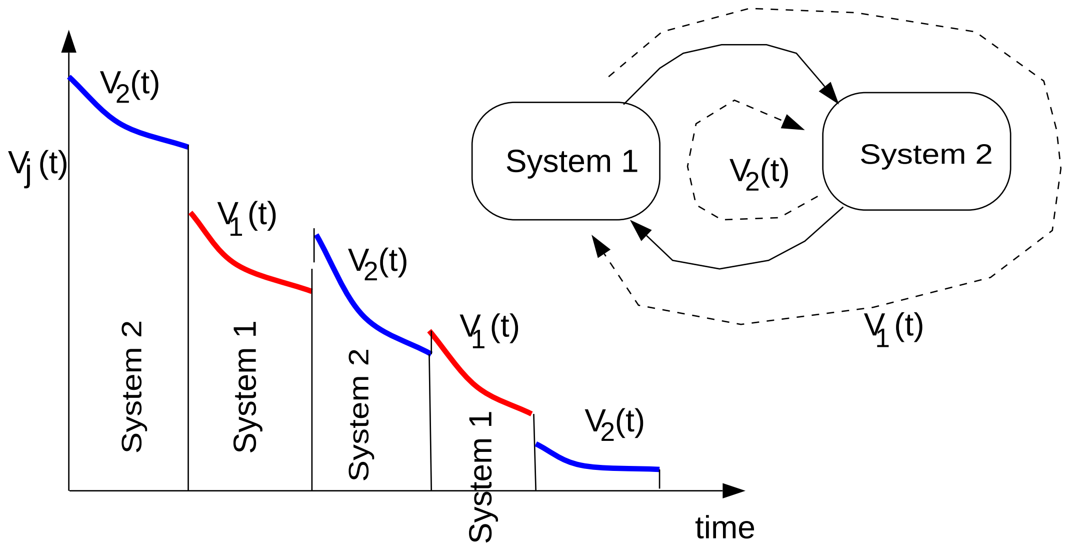

Figure 6: Switched Lyapunov System

The results in [38] provide an important extension of Lyapunov analysis methods for the validation of switched system stability. Related work in [41] [42] has relaxed some of the assumptions in [38] thereby providing tighter sufficient conditions on hybrid system stability. Neither of these results is constructive. As is usually the case in using Lyapunov methods, the determination of such functions can only be done systematically for special classes of systems. One such class occurs when the switched subsystems are linear time invariant and the switching sets are conic sectors. In [44] and [43], it was shown that Lyapunov like functions could be determined by checking the feasibility of a certain linear matrix inequality (LMI) [39] based on a modified version of Lyapunov's equation.

While these prior results have provided great insight into the Lyapunov stability of switched systems, these results 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 buried in the result of [38] and [41]. In both results, it was observed that it is important to check for monotone decreasing behaviour of Lyapunov-like functions over *cycles* within the discrete trace. The *cycles* are cycles of switched systems that begin and end with the same subsystem. For switching logics generated by finite automata or Petri nets, the pumping lemma [88] assures us that all traces can be finitely generated by a finite set of discrete event cycles. It therefore seems plausible that 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 ideas was developed more fully in [45] and [46], where it was shown that the use of fundamental cycles extracted from the discrete event subsystem could be used in conjunction with the results of [44] and [43] to provide sufficient conditions for hybrid system stability.

It is interesting to note that the use of fundamental cycles and Lyapunov functionals in [45] is related to earlier work studying 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 which can be revisited repeatedly by the system. Such sets are sometimes called *viability kernels* [47]. 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 [45] identify fundamental cycles of the switching logic and then uses linear matrix inequality (LMI) techniques to find Lyapunov like functions. The level sets of these functions are invariant under the cycle and hence represent a viability kernel for the entire system. These sets, of course, represent approximations to the viability sets, Ξ which the fixed point computation in the SMC iteration attempts to determine.

This section has surveyed recent work providing sufficient conditions for the stability of switched systems. The multiple Lyapunov function methods discussed here were related to earlier work in model checking for hybrid systems and it was noted that these Lyapunov methods can be seen as computing approximations to the fixed points determined by model checking methods. In the following subsection a concrete example illustrating the use of Lyapunov type methods in hybrid system validation is presented for the robotic example of figure 1.

6.3 Validating the Example

This subsection illustrates some of the principles discussed in subsection 6.2. In particular, this subsection will validate the mutual exclusion and deadlock-freedom requirements on the robotic system of figure 1. The methodology will be a variation on the Lyapunov stability approaches discussed in [46].

The performance specifications for our example will be posed as temporal logic formulae. The mutual exclusion constraint was that both arms should not be in the parts bin at the same time. This requirement is expressed by the formula,

$$\tilde{}\,\exists U[[|\bar{\theta}_1| < 10°] \wedge [|\bar{\theta}_2| < 10°]] \tag{35}$$

where $\bar{\theta}_i = \theta_i + \theta_b$ $(i = 1, 2)$. This formula states that there should not exist any trajectory which eventually reaches a state where the angular position of both arms is less than the prespecified limit of 0.1. These limits, of course, define the extent of the parts bin, so the formula is requiring that the physical system should never violate the mutual exclusion requirement.

The deadlock freedom requirement was that neither arm should reach a state from which it is impossible to proceed (actually we're only insisting that the system be weakly deadlock free). Deadlock effectively occurs if either arm reaches its physical stops at any time. This specification may therefore be represented by the following formula,

$$\forall U[[-10 < \theta_1 < 100] \wedge [-100 < \theta_2 < 10]] \tag{36}$$

This formula states that for all trajectories starting in the initial states that both of the conditions $[-10 < \theta_1 < 100]$ and $[-100 < \theta_2 < 10]$ must be satisfied by all state trajectories.

The preceding requirements are associated with regions in the continuous state space of the system. The mutual exclusion requirement defines a *forbidden set* which the system trajectories must never enter. This forbidden set is the smaller box centered at the origin in figure 7. The deadlock freedom requirement is associated with the larger box in figure 7. This box represents the set of *safe* arm positions and therefore represents an *safe set* which all continuous-state trajectories must remain in for all time.

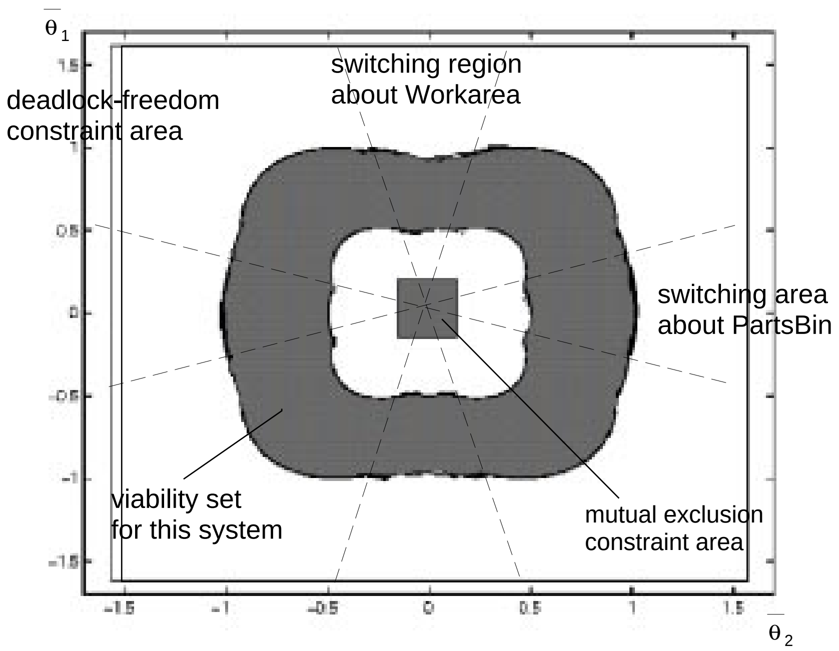

Figure 7: Forbidden and Safe Sets Associated with System Specifications

An ellipsoidal approximation to the validating set of initial conditions may be computed using the Lyapunov techniques discussed in section 6.2. We use a variation [45] on the linear matrix inequality (LMI) method first proposed in [44] and [43]. The methodology in [44] [43] formulated a linear matrix inequality whose feasible solutions were a set of positive definite matrices P_j characterizing a family of Lyapunov like functions for this switched system. The method used in [45] employed a linear matrix inequality to form a sufficient test for the uniform ultimate boundedness (UUB) [89] of the switched system. This method extends the earlier results of [44] and [43] so that systems with bounded exogenous disturbances could be treated using the same type of analysis. The analysis method works as follows. Consider all fundamental cycles of discrete events generated by the discrete part of the hybrid system. A linear matrix inequality (LMI) may be derived from this set of cycles whose feasible solution (if it exists) consists of a set of positive definite matrices and constants characterizing ellipsoidal sets in the continuous-state space whose intersection is invariant under all possible concatenations of fundamental cycles. In other words, the sets identified by this method represent sets of continuous states which the system is guaranteed to reenter as it traverses its discrete events. Technical details on the formulation of the LMI will be found in [56].

The invariant sets identified using the preceding method can be used to validate the mutual exclusion and deadlock freedom constraints posed above. Let's first consider the deadlock freedom constraint. Associated with this requirement was a region in the system's state space which was required to be invariant under all possible system trajectories. The validation problem involves finding a set of initial conditions within this box whose future trajectories remain in the set. The invariant set identified using the preceding method is an obvious candidate for this set. Therefore, if we can find a set of feasible LMI's whose solution identify an invariant set which is properly contained in the *safe set* of the system, then the system is guaranteed to be safe as long as our initial condition is taken from within this invariant set of states. The proper inclusion of the invariant in the safe set therefore validates the existential specification we posed earlier.

Validating the mutual exclusion requirement may also be accomplished using this method. In this case, however, we reverse the sense of the inequalities, so that their feasible solution identifies a set which is guaranteed to be *repelling* under all concatenations of fundamental cycles. The union of the ellipsoidal sets identified by the feasible solutions of this LMI, this represents a repelling set of initial conditions, in the sense that if the system starts outside this set, then it is guaranteed to remain outside of this set for all time. We use this *repelling set* manner to validate the mutual exclusion requirement. If the *forbidden set* associated with the specification is properly contained within the repelling set, then the specification is validated in the sense that any initial conditions starting outside of this region will be guaranteed to not violate the mutual exclusion constraint.

Ensuring that both mutual exclusion and deadlock freedom constraints are met is simply a matter of ensuring that the intersection of the repelling and invariant sets is non-empty. Figure 7 provides an illustration of the invariant and repelling sets identified using this LMI method for the case where the feedback gain is $k = 1.6$ and the ratio of the moments of inertia $J_a/J_b = .1$. In this case, it is easy to see that the non-empty intersection of the repelling and invariant sets indeed lies within the accepting region for this problem. For this particular system, therefore, we can validate the specified behavioral constraints.

6.4 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 hybrid control system synthesis.

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. [9] The strength in 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 [12]. Precisely what constitutes accurate modeling has been discussed in a variety of papers [83] [84] [85] [82]. In most of these works it is essential that some property of the original continuous system (such as controllability or observability) is preserved under the discretization. Discrete event systems that satisfy this condition are sometimes referred to as *bisimulations* of the continuous plant. Precisely how such bisimulations might be used to help in the design and analysis of hybrid control systems, however, remains an open question for the research community.

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 [48] to continualization was based on the use of formal power series (Lie-Fliess series) [49] 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 [50] representation theorems, could then be used to extract finite automata generating the desired switching logic for the hybrid system. This continualized approach was first discussed in [48] [80] and that work combines concepts from logic, automata theory, and differential geometry to attempt to produce a unified framework for the synthesis of controlled hybrid systems. The method is very similar the motion planning approaches found in robotics [54] and an elementary example illustrating how such switching policies might be synthesized will be found in [55].

Another approach for hybrid control system synthesis is based on *gain-scheduling* ideas [55]. Gain scheduling [51] [52] [53] assumes a set of linear controllers that are switched between as the system moves through its operating range. Gain scheduling has been extensively studied in the control systems community. Early work in this area [57], 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 therefore provides a mechanism by which to actually design controllers that achieve tight norm bounded performance measures [56]. 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, there has been some work attempting to define an integrated framework which 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 non-cooperative game played between the supervisor and the controller [86] [87]. The nondeterministic nature of the supervisor means that continuous-state controllers must be optimized over the worst case decisions adopted by the supervisor and we need, at the same time, to ensure that the supervisor is as permissive as possible. Both of these objectives are usually conflicting and this conflict leads to the non-cooperative nature of the game. This viewpoint of hybrid controller synthesis is very elegant from a theoretical standpoint, but computing the saddle-point is non-trivial. Applications [71] 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 [72] of determining saddle points will need to be found.

7 Future Directions

The paper provided an introduction to many of the concepts and trends in hybrid system science. As can be seen from the paper, hybrid system science it an interdisciplinary field requiring a familiarity with methods and concepts from computer science and traditional system science. Due to the introductory nature of this paper, 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, [63] [64] [65] [66] [67] [68] [69] as well as numerous other workshops not mentioned here 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 already some important results and discoveries have been made. Principle milestones in hybrid system science include the early work on verification using hybrid automata, 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 [71] [75] [76] [77] [32] [78] [79]. All of these accomplishments point to a science which shows excellent potential for having a profound impact on the way in which we design and develop the engineering applications of the future.

References

[1] 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 Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems, Lecture Notes in Computer Science*, vol. 736, Springer-Verlag, pp. 209-229, 1993.

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

[3] 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, pp. 41-71, 1995.

[4] M.S. Branicky, *Studies in Hybrid Systems: modeling, analysis, and control*, LIDS-TH-2304, Ph.D. Dissertation, Massachusets Institute of Technology, LIDS, 1995.

[5] A.J. Van der Schaft and J.M. Schumacher, Complementary Modeling of Hybrid Systems, *IEEE Trans. on Automatic Control*, 43(4):483-490, 1998.

[6] R.W. Brockett, Hybrid models for motion control systems, in *Essays in control: perspectives in the theory and its applications*, pp. 29-53, Birkhauser, Boston, 1993.

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

[8] R.A. DeCarlo, S.H. Zak, and G.P. Matthews, Variable Structure Control of Nonlinear Multivariable Systems: a tutorial, *Proceedings of the IEEE*, Vol. 76, No. 3., March 1988.

[9] J.A. Stiver, P.J. Antsaklis, and M.D. Lemmon, A logical DES approach to the design of hybrid control systems, *Mathematical Computer Modeling*, Vol. 23(11/12),pp. 55-76, 1996.

[10] X. Yang, M.D. Lemmon, and P.J. Antsaklis, On the supremal controllable sublanguage in the discrete event model of nondeterministic hybrid control systems, *IEEE Trans. on Automatic Control*, Vol. 40(12) , pp. 2098-2102, 1996.

[11] J. Raisch, and S.D. O'Young, Discrete approximations and supervisory control of continuous systems. *IEEE Trans. on Automatic Control*, 43(4): 569-573, 1998.

[12] J.S. Stiver, P.J. Antsakis, and M.D. Lemmon, An invariant based approach to the design of hybrid control systems, in *Proceedings of the IFAC 13th Triennial World Congress*, Vol. J, pp. 457-472, San Franciscio, CA, 1996.

[13] L. Tavernini, Differential automata and their discrete simulators, *Nonlinear analysis, theory, methods and applications*, vol. 11(6), 665-683, 1987.

[14] M. Heymann, F. Lin, and G. Meyer, Synthesis and viability of minimally interventive legal controllers for hybrid systems, *Discrete Event Dynamic Systems: theory and applications*, Volume 8(2):105-136, 1998.

[15] J. LeBail, H. Alla, and R. David, Hybrid Petri nets, in *Proceedings 1st Euopean Control Conference*, Grenoble, France, 1991.

[16] I. Demongodin, and N.T. Koussoulas, Differential Petri nets: representing continuous system in a discrete-event world, *IEEE Transactions on Automatic Control*, vol 44:3, pp. 573-578, 1998.

[17] J.-M. Flaus, and H. Alla, Structural analysis of hybrid systems modeled by hybrid flow nets, in *Proceedings of the European Control Conference 97*, Brussels Belgium, 1997.

[18] A. Guia, and E. Usai, High-level hybrid Petri nets; a definition, in *Proceedings of the IEEE 35th Conference on Decision and Control*, Kobe Japan, 1996.

[19] X. Koutsoukos, K.X. He, M.D. Lemmon, and P.J. Antsaklis, Timed Petri nets in hybrid systems: stability and supervisory control, *Journal of discrete event and dynamical systems*, Vol 8(2), pp. 137-174, 1998.

[20] R. Alur, C. Courcoubetis, and D. Dill, Model Checking in Dense Real Time, *Information and Computation*, Vol. 104, pp. 2-34, 1993.

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

[22] R. Alur, T.A. Henzinger and P.-H. Ho, Automatic symbolic verification of embedded systems. *IEEE Trans. on Software Engineering*, 22: 181-201, 1996.

[23] 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. *Theoretical Computer Science* Vol. 138:, pp. 3-34, 1995.

[24] M.R. Hansen, M.R. and C. Zhou, Semantics and completeness of the duration calculus, in editors, *Real-time: theory in practice*, De Bakker, Huizing, and de Roever (editors), 1991, Lecture Notes in Computer Science Vol. 600, pp. 209-225, Springer-Verlag, Berline, 1992.

[25] C. Zhou, Duration calculii: an overview, in *Proc. Formal Methods in Programming and their applciation*, Bjorner, Broy and Pottosin (editors), Lecture Notes in Computer Science Vol. 735, pages 256-266, 1993.

[26] C. Zhou, M.R. Hansen, M.R., and P. Sestoft, Decidability Results for duraiton calculus, *Proc. STACS 93*, Enjalbert, Finkel, and Wagner (editors), Lecture Notes in Computer Science 665, pp. 58-68, Springer-Verlag, 1993.

[27] K.G. Larsen, P. Pettersson, and W. Yi, Model-Checking for Real-Time Systems, in *Proceedings of the 10th International Conference on Fundamentals of Computation Theory*, Dresden, Germany, Lecture Notes in Computer Science Vol. 965, pages 62-88, Horst Reichel (editor.), Springer-Verlag, 1995.

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

[29] H. Wong-Toi, Synthesis of Contollers for Linear Hybrid Automata, *Proceedings of the 36th IEEE Conference on Decision and Control*, San Diego, California, 1997.

[30] A. Puri, and P. Varaiya, Decidability of hybrid systems with rectangular differential inclusions., in *Computer Aided Verification: CAV'94*, Dill (editor), Lecture Notes in Computer Science Voo. 818., Springer-Verlag, pp. 81-84, 1994.

[31] T.A. Henzinger P. Kopke, A. Puri, and P. Varaiya , What's decidable about hybrid automata, *Proc. of the 27th Annual ACM symposium on the theory of computing* , 1995.

[32] S. Kowalewski, M. Fritz, H. Graf, J. Preussig, S. Simon, O. Stursberg,and Treseler., A case study in tool-aided analysis of discretely controlled continuous systems: the two tanks problem, Hybrid Systems V, (in print), Springer Verlag, 1999.

[33] E.M. Clarke and E.A. Emerson, Characterizing properties of algorithms as fixed points, in *7th international colloquium on automata languages and programming*, Lecture Notes in Computer Scinece Vol. 85, Springer-Verlag, 1981.

[34] E.M. Clarke and E.A. Emerson, Synthesis of synchronization skeletons for branching time temporal logic. in *Logic of Programs: Workshop* Lecture Notes in Computer Science Vol 131, Dexter and Kozen (editors), Yorktown Heights, New York, Springer-Verlag, 1981.

[35] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and Hwang, Symbolic model checking: 10^{20} states and beyond, *Proceedings of the 4th annual sympsoium on logic in computer science*, June 1990.

[36] G.J. Pappas and S. Sastry, Straightening out rectangular differential inclusions, to appear in *System and Control Letters*, 1998.

[37] P. Peleties and R.A. DeCarlo, Asymptotic stability of *m*-switched systems using lypaunov like functions, in *Proceedings of the American Control Conference*, pp. 1679-1684, 1991.

[38] M. Branicky , Stability of switched and hybrid systems. in *Proceedings of the 33th Conference on Decision and Control*, Lake Buena Vista, Florida, pp. 3498-3503, 1994.

[39] S. BOyd, L.E. Ghaoui, E. Feron, and V. Balakrishnan, *Linear matrix inequalities in system and control theory*, SIAM, studies in applied mathematics 15, 1994.

[40] M.S. Branicky, Multiple Lypaunov functions and other analysis tools for switched and hybrid systems, *IEEE Trans. on Automatic Control*, 43(4), April 1998.

[41] L. Hou, A.N. Michel,and H. Ye, Stasbility analysis of switched systems, in *Proceedings of the 35th Conference on Decision and Control*, Kobe Japan, 1996.

[42] H. Ye, A.N. Michel, and L. Hou, Stability theory for hybrid dynamical systems, *IEEE Trans. on Automatic Control*, 43(4):461-474.

[43] M. Johansson and A. Rantzer, Computation of peicewise quadratic Lyapunov functions for hybrid systems, *IEEE Transactions on Automatic Control*, 1998.

[44] S. Petterson and B. Lennartson, Stability and robustness of hybrid systems. in *Proceedings of 35th Conference on Decision and Control*, Kobe Japan, 1996.

[45] K.X. He and M.D. Lemmon, Lyapunov stability of continuous valued systems under the supervision of discrete event transition systems, in *Proceedings of Hybrid Systems: Control and Computation*, lecture notes in computer science Vol. 1386, Springer Verlag, 1998.

[46] M.D. Lemmon, and K.X. He , Modeling hybrid control systems using programmable Petri nets, *JESA - European Journal of Automation*, to appear in 1999.

[47] A. Deshpande and P. Varaiya, Viable control of hybrid systems, in *Hybrid systems II* [64], pp. 128-147.

[48] A. Nerode and W. Kohn, Multiple Agent hybrid control architecture, hybrid systems, in *Hybrid Systems* [63], pp297-316

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

[50] A. Salomaa, M. Soittola, *Automata Theoretic Aspects of formal power series*, Springer-Verlag, Berlin, 1973.

[51] J.S. Shamma and M. Athans, Analysis of gain shceduled control for nonlinear plants, *IEEE Trans. on Automatic Control*, Vol 35, 878-907, 1991.

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

[53] A. Packard, Gain scheduling via linear fractional transformation, *System and Control Letters*, Vol 22, 79-92, 1994.

[54] J.W. Goodwine and J. Burdick, Trajecotry generation for legged robotic systems, *IEEE International Conference on Robotics and Automation*, 1997.

[55] M.D. Lemmon, and C.J. Bett, Safe implementations of supervisory commands, *International Journal of Control*, Vol 70(2), pp. 271-288, 1998.

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

[57] A.S. Morse, *Control using logic based switching*, lecture ntoes in control and information sciences vol. 222, Springer-Verlag, 1997.

[58] L. Lamport, A fast mutual exclusion algorithm, *ACM Trans. on Computer Systems*, 5(1):1-11, 1987.

[59] Raynal, M., *Algorithms for mutual exclusion*, MIT Press, 1986.

[60] R. Gallmeister, *POSIX.4, programming for the real world*, O'Reilley and Associates, 1995.

[61] P.J. Ramadge and W.M. Wonham, Supervisory control of a class of discrete event processes, *SIAM Journal of Control and Optimization*, 25(1): 206-230, 1987.

[62] N. Lynch and N. Shavit, Timing-based mutual exclusion, *Proceedings 13th IEEE Real-Time Systems Symposium,* IEEE Computer Society Press, Los Alamtibes, CA, pp. 3-11, 1993.

[63] R.L. Grossman, A.N. Nerode , A.P. Ravn , and H. Rischel (editors), *Hybrid Systems*, lecture notes in computer science volume 736, Springer-Verlag, 1993.

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

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

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

[67] P.J. Antsaklis, W. Kohn, M.D. Lemmon, A.N. Nerode , S. Sastry (editors) *Hybrid Systems V*, Lecture Notes in Computer Science (in press), Springer-Verlag, 1999.

[68] T.A. Henzinger and S. Sastry (editors), *Hybrid systems: control and computation* , Lecture Notes in Computer Science 1386, Springer-Verlag, 1998.

[69] O. Maller (editor), Hybrid and Real-Time Systems: Hart'97, Lecture Notes in Computer Science Vol. 1201, Springer Verlag, 1997.

[70] A.S. Matveev and A.V. Savkin, Reduction and decompositionof differential automata: theory and applications, in *Hybrid Systems: computation and control* [68], 1998.

[71] C. Tomlin, G. Pappas and S. Sastry, Conflict resolution for air traffic managmeent: a study in multiagent hybrid sytems, *IEEE Trans. of Automatic Control*, Vol. 43(4), 1998.

[72] J. Lygeros, C.J. Tomlin, and S. Sastry, On controller synthesis for nonlinear hybrid systems, *Proceedings of 37th IEEE Conference on Decision and Control*, Tampa, Florida, Dec. 1998.

[73] J. Lygeros, D. Godbole, and S. Sastry, Verified hybrid cotnrollers for automated vehicles, *IEEE Trans of Automatic Control*, 43(4), 1998.

[74] H.S. Witsenhausen, A class of hybrid-state continuous time dynamic systems, *IEEE Transactions on Automatic Control*, 11(2):161-167, 1966.

[75] B. Lennartson, M. Tittus, B. Egardt, and S. Petterson, Hybrid systems in process control, *Control Systems Magazine*, 16(5):45-56, 1996.

[76] M. Tittus, Control Synthesis for Batch Processes, Ph.D. thesis, Control engineering lab, Chalmers University of Technology, Goteburg, Sweden, 1994.

[77] J. Raisch and E. Klein, Approximating automata and discrete control for continuous systems: two examples from chemical process control, *Hybrid Systems V.* [67], 1999.

[78] C.W. Seibel and J.-M. Farines, Towards using hybrid automata for the mission planning of unmanned aerial vehicles. in *Hybrid Systems V.* [67], 1999.

[79] R. Balluchi,M. De Benedetto, C. Pinello, C. Rossi, A. Sangiovanni-Vincentelli, Hybrid control for automotive engine management: the cut-off case. *Hybrid Systems: computation and control* [68], 1998.

[80] A.N. Nerode and W. Kohn, Models for hybird systems: automata topologies, controlability observability, in *Hybrid Systems* [63], 1993.

[81] G. Lafferriere, G. J. Pappas, and S. Sastry, o-minimal hybrid systems, *Technical report UCB/ERL M98/29*, Universityof Califory Berkeley, May 1998.

[82] G.J. Pappas, G. Lafferriere and S. Sastry, Hierarchically consisting control systems in *Proceedings of 37th IEEE Conference in Decison and Control*, Tampa, FL, Dec. 1998.

[83] M.D. Lemmon and P.J. Antaklis, Inductively inferring valid logical models of continuous state dynamical systems *Theoretical computer science*, volume 138, 201-210, 1995.

[84] P.E. Caines and Y.J. Wei, The hierarchical lattices of finite state machines, *System and Control Letters*, 25:257-263, 1994.

[85] P.E. Caines and Y.J. Wei, Hierarchical hybrid control systems: a lattice theoretic formulation. *IEEE Trans. on Automatic Control*, 43(4):501-508, April 1998.

[86] J. Lygeros, C. Tomlin, and S. Sastry, Multiobjective hybrid controller synthesis, in *Proc. Hart'97* [69], 1997.

[87] J. Lygeros, D.N. Godbole, and S. Sastry, Multiagent hybrid system design using game theory and optimal control, *Proc. of Conference on Decision and Control*, Kobe Japan, 1190-1195, 1996.

[88] J.E. Hopcroft, J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley Publishing Co., Reading Massachusetts, 1979.

[89] H.K. Khalil, *Nonlinear Systems*, 2nd edition, Prentice-Hall, 1996.