Supervisory Control of Discrete-Event Systems with Output: Application to Hybrid Systems

Pedram Mahdavinezhad

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

April 2007

© Pedram Mahdavinezhad, 2007

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-40889-6
Our file Notre référence
ISBN: 978-0-494-40889-6

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

ABSTRACT

Supervisory Control of Discrete-Event Systems with Output: Application to Hybrid Systems

Pedram Mahdavinezhad

In this thesis, the problem of supervisory control of Discrete-Event Systems (DES) with output is presented and discussed at length. In such systems, causal output functions are employed to assign each sequence of inputs with a corresponding sequence of outputs. When the specification of the desired behavior is given by a formal language over the output alphabet, necessary and sufficient conditions are derived for the existence of nonblocking input as well as nonblocking output supervisory controls. An algorithm is presented to extend the results of nonblocking input/output supervisory control from language-based framework into finite automata framework, making the proposed results applicable to large scale discrete-event systems. The idea of siblings is introduced to solve the problem of nondeterminism in discrete-event abstractions of hybrid systems, giving rise to the development of a theory for nonblocking supervisory control of hybrid systems. Our results enable one to apply classical supervisory control theory to design supervisors for DES approximations of hybrid systems, and to import many interesting concepts from classical theory such as modular and hierarchical control.

I dedicate this thesis to my parents for their support, encourage, and patience during all years of my academic education.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Peyman Gohari, and my Co-supervisor, Dr. Amir G Aghdam, for their strong support on this work by their knowledge, encourage, and patience. Being exposed to their experience and professional insight, caused me to grow up professionally during my master degree which can help me in different aspect of my life.

I would also appreciate the help from many friends who guided me thorough this work and provided me with an enjoyable environment.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

A discrete-event system (DES) can be considered as a dynamical system whose dynamics depend on any occurrence of physical events. These physical events are considered as *transitions* in the DES framework. The DES paradigm is a research area which has a wide range of applications in different industries such as information systems, manufacturing systems, traffic management, communication protocols, and logistic systems. The study of such systems has captured a lot of attention over the past few decades because of fast-increasing advances in computer-controlled systems. Such systems which are governed by computer typically own some sort of "discrete" dynamics. For example, the process of counting the number of active telephone calls, or the number of parts in the buffer, which are simple examples of real application of DES, includes some sort of discrete dynamics which should be controlled via appropriate control configurations. Broadly stated, different examples of DES applications exist whenever digital computers are employed or any kind of execution of a computer program is involved in the dynamics of the system.

A supervisor is the controlling agent of a DES *plant* which can observe all the transitions or simply any occurrence of events of the DES plant. Whenever an event is observed, the supervisor will enable or disable a set of different DES events in order to keep the plant away from a set of 'forbidden' states, and also to prevent the occurrence of any undesirable event sequences. Generally, the supervisor's duty is to restrict the system behavior such that it satisfies some desired specifications such as *safety* specifications. Examples of desired specifications could be avoidance of a set of forbidden states, or observation of different service priorities in a manufacturing process.

Broadly speaking, DES are modeled by finite state machines or generators or finite automata, where formal languages are employed to describe the dynamics of such systems. In addition, a supervisor, which is also modeled by a finite automaton, as an external agent enables or disables certain transitions such that the objective of the system will be satisfied. As a common objective of DES, a supervisor should be designed such that the performance of the controlled system satisfies the specified 'legal' behavior.

In this paradigm, a control theoretic framework has been proposed by Peter J. Ramadge and W. Murray Wonham [1]. In their modeling, DES are defined by finite state machines, in which different sequences of transitions (or strings) can happen. The set of all such strings forms a language that contains every possible event sequence that can occur in the DES. When a supervisor is to disable a set of different events in the plant, the resulting behavior of the

DES will change, resulting in the generation of a new closed-loop language. This closed-loop language can describe the dynamics of the controlled system. To ensure that the controlled performance of the plant satisfies a specified 'legal' behavior, one can specify the legal behavior as a legal language that contains all desirable event sequences. The objective of the supervisor is to ensure that every possible sequence of events in the controlled system is a member of the legal language. A brief review of the Ramadge-Wonham supervisory control framework is presented in the next chapter.

From a general point of view, constructing abstract models of any complex dynamical system is essential in many areas of engineering for analysis and verification purposes. In control systems engineering paradigm, a good abstract model is one that is complex enough to contain all the important system characteristics, but simple enough to allow application of existing analysis and design methods. Broadly speaking, traditional control has just considered continuous dynamical systems, where both plant and controller are modeled by differential equations. Despite the great effect which this approach has had on control design and analysis, it has several limitations mainly because of the fact that not all real control systems can be well modeled by differential equations. Therefore, traditional control is unable to provide a suitable framework for analysis of complex dynamical systems. Networked and embedded control systems are examples of emerging areas that need a new modeling paradigm. In modern control framework, most practical systems include both analogue and discrete components. Complex systems typically possess a hierarchical structure, characterized by continuous variable dynamics at the lowest level and logical decision-making at the highest.

The interaction between continuous and discrete dynamics in such systems results in the complexity of their analysis. Therefore, a large amount of research is dedicated to the modeling of combinations of discrete and continuous (*hybrid*) systems in order to facilitate their verification and design methods. The main contribution of most of these modeling frameworks is to somehow separate the continuous and discrete dynamics of a hybrid system while the hybrid nature of the system which includes the mixture of both dynamics is preserved. The major role of discrete-event systems theory in the development of hybrid technology is to model the logical decision-making process of the hybrid structure, and to supervise the continuous dynamics of the system from the highest level of abstraction. The idea of this thesis is to redevelop the main contribution of Koutsoukos et al [2] in which they tried to develop a theory for supervisory control of hybrid systems based on supervisory control of discrete-event systems. While we believe that Koutsoukos's contribution [2] had impressive impact on the development of supervisory control of hybrid systems framework, there are major issues in their proposed method of designing DES supervisors which motivated us to develop a modified framework to fix all the shortcomings of Koutsoukos et al [2]. The result is the developed framework of supervisory control of DES with outputs and its application to hybrid systems. A summary of this work is presented in [3]

1.1 Thesis Motivations and Contributions

In this thesis, we follow [2] and approximate the behavior of hybrid systems with DES plants. Further, the desired behavior is specified by a formal language over the alphabet of plant symbols, and the objective is to design a discrete-event supervisor such that the controlled system satisfies the specification.

Although [2] proposes to apply supervisory control of DES to hybrid systems, the theory of supervisory control of hybrid systems is not clearly connected to that of supervisory control of DES. The most notable problem in [2] is that a specification language therein that forces the supervisor to disable all controllable events at its disposal is called *uncontrollable*. We believe the unacceptability of a supervisor implementing such a specification has little to do with its controllability—after all, since each event is a command generated by the supervisor, all events can be considered to be controllable. In classical supervisory control of DES, a system in which all events out of a non-marker state are disabled is said to be *blocking*. Thus, a unified theory for supervisory control of DES and hybrid systems is achievable only if the notion of blocking is properly introduced in supervisory control of hybrid systems which is not the case in [2].

Motivated by the above observations, in this thesis we address the problem of nonblocking supervisory control of hybrid systems, where hybrid systems are approximated by nondeterministic finite automata, proposed in [2]. We first extend classical supervisory control of DES to develop a theory for nonblocking supervisory control of DES with output. Then, a theory for output supervisory control is introduced and necessary and sufficient conditions for the existence of nonblocking output supervisors as well as nonblocking input supervisors are presented when the desired behavior is specified by a language over output events.

The other contribution is an algorithm which is proposed to solve the problem of nonblocking supervisory control of DES with outputs when the desired specification is specified by finite automata. The proposed procedure enables one to simply examine the output consistency property of a language by inspecting its generating automaton, and facilitates the design of nonblocking output supervisory control for large scale discrete-event systems.

Eventually, we adjust the theory to make it applicable to DES models of hybrid systems by requiring that a supervisor treat all transitions carrying the same label out of a state *consistently*; that is, either to enable or to disable all. In order to do this, event siblings are introduced to convert the nondeterministic DES plant of a hybrid system to a deterministic model such that our proposed theory of output nonblocking supervisory control can be applied.

1.2 Thesis Outline

The thesis is organized as follows: Chapter 2 presents the fundamental concepts of supervisory control of DES. In this chapter, the definitions and results which are used throughout the thesis are covered. This chapter also briefly discusses some commands of the software package TTCT which can be used in designing DES supervisors. An overview of hybrid systems, different methodologies to hybrid systems and their applications are presented in Chapter 3. Chapter 4 presents supervisory control of hybrid systems introduced in [2], and reviews basic concepts of DES with output. After defining input and output supervisory control, in Chapter 4 we develop a theory for nonblocking supervisory control of DES with output, and find necessary and sufficient conditions for the existence of a supervisor. In Chapter 5 we first present an algorithm to determine whether the language generated by an automaton is *output-consistent*, and then minor adjustments are made to the theory of supervisory control of DES with output to make it applicable to nondeterministic DES models of hybrid systems, and an example is worked out to illustrate the approach. Finally, we conclude the thesis in Chapter 6 and point out directions for future research.

Chapter 2

Overview of Discrete-Event System (DES)

2.1 Languages

In order to model a discrete-event system in a formal language framework, first a finite set of distinct symbols Σ, called the alphabet of events, is defined such as $\Sigma : \alpha, \beta, \lambda, \delta$. In this paradigm, ε denotes an empty sequence, (i.e. a sequence with no symbols), where ε is not a member of Σ. If Σ represents an alphabet, then $L \subseteq \Sigma^*$ can be considered as a language where Σ^* is the set of all finite sequences of symbols in Σ, including the empty sequence. The usual set operations such as union, intersection, difference, and complement are applicable to languages.

Suppose that $s = tu$, where $s, t, u \in \Sigma^*$. Then the two strings t and u are called *prefix* and *suffix* of the string s, respectively. The *prefix-closure* of a language L is denoted by $\overline{L}$. The prefix-closure of L is the set of prefixes of all strings of L; in other words,

$$\overline{L} = \{s \in \Sigma^* | \exists t \in \Sigma^*, st \in L\}$$

One can easily verify that $L \subseteq \overline{L}$ always holds. The language L is called *prefix-closed* or simply *closed* if $L = \overline{L}$. Similarly, for two languages L, $M \subseteq \Sigma^*$, L is called *M-closed* if $L = \overline{L} \cap M$ [4].

Two languages L_1 and L_2 are *nonconflicting* [4] if

$$\overline{L_1 \cap L_2} = \overline{L_1} \cap \overline{L_2}$$

The following example is presented to illustrate the above definitions.

Example 1 *Consider $\Sigma = \{a,b,c,d,e\}$ as an alphabet. The sets of finite sequences L_1, L_2 and $L_3 \subseteq \Sigma^*$ defined as*

$$L_1 = \{adece\}$$
$$L_2 = \{\varepsilon, a, ad, ade, adec, adece\}$$
$$L_3 = \{adece, e\}$$

are languages over alphabet Σ. L_2 is the prefix-closure of L_1, i.e. $L_2 = \overline{L_1}$. In addition, L_2 is closed, since $L_2 = \overline{L_2}$. Also, 'a' and 'e', for instance, are a prefix and a suffix of 'adece', respectively. L_1 is not a closed language since $\overline{L_1} \neq L_1$. However, L_1 is L_3-closed since $L_1 = \overline{L_1} \cap L_3$. L_1 and L_2 are nonconflicting, since $L_1 \cap L_2 = L_1$ and

$$\overline{L_1} \cap \overline{L_2} = \{\varepsilon, a, ad, ade, adec, adece\} = \overline{L_1 \cap L_2}.$$

2.2 Generators

In Ramadge-Wonham framework [4] a deterministic generator is the fundamental tool to model a discrete-event system. A generator (finite state machine) can be represented by a five-tuple:

$$G = (Q, \Sigma, \delta, q_0, Q_m)$$

where Q represents the finite set of states, Σ represents the finite set of events, δ represents the partial transition function $\delta : Q \times \Sigma \rightarrow Q$, $Q_m \subseteq Q$ is the set of marked states, and q_0 is the initial state.

2.3 Languages Represented by Generators

The *closed behavior of* G, denoted by $L(G)$, represents all possible event sequences taking G from the initial state to some reachable states:

$$L(G) := \{s | s \in \Sigma^*, \delta(q_0, s)!\}$$

Here $\delta(q_0, s)!$ means that $\delta(q_0, s)$ is defined. Also, the *marked behavior of* G defined as

$$L_m(G) := \{s | s \in L(G), \delta(q_0, s) \in Q_m\}$$

represents the set of all possible event sequences taking G from the initial state to some marked states. One can easily verify that $L(G)$ is always a closed language ($L(G) = \overline{L(G)}$) while $L_m(G)$ may not be closed. In general, the following relation always holds: $L_m(G) \subseteq \overline{L_m(G)} \subseteq L(G) = \overline{L(G)}$.

2.4 Non-deterministic Generators

A generator G is called *deterministic* if for a given current state and event symbol, the transition function δ uniquely determines the next state, when defined, while it is called *non-deterministic* otherwise, or when there are two or more initial states. Usually, the transition function in a non-deterministic generator can be defined as: $\delta : Q \times \Sigma \rightarrow Pwr(Q)$ which illustrates the possibility of different states as destinations of a transition from a state $q \in Q$ in response to the occurrence of an event $\sigma \in \Sigma$. Although one can follow the appropriate procedures(e.g., the ones proposed in [4]) to convert the non-deterministic generator into a deterministic one, where

$$L_{Det}(G) = L_{Ndet}(G)$$

and

$$L_{Det,m(G)} = L_{Ndet,m(G)}$$

. A non-deterministic model can be preferred to a deterministic one in many cases due to its general form and capabilities of modeling complicated systems.

2.5 Operations on Generators

This section introduces three different operations on generators which will be used directly or indirectly in this thesis.

Reachable Generator Consider a generator $G = (Q, \Sigma, \delta, q_0, Q_m)$. A state $q \in Q$ is reachable if there exists an $s \in \Sigma^*$ such that $\delta(q_0, s) = q$. The set of reachable states is denoted by Q_r. The reachable generator G_r is defined as $G_r = (Q_r, \Sigma, \delta_r, q_0, Q_{r,m})$, where

$$Q_r = \{q \in Q | \exists s \in \Sigma^*.\ \delta(q_0, s) = q\}$$

$$Q_{r,m} = Q_m \bigcap Q_r$$

$$\delta_r = \delta|_{Q_r \times \Sigma \to Q_r}$$

The function $\delta|_{Q_r \times \Sigma \to Q_r}$ denotes the restriction of δ to the smaller domain of accessible states Q_r. One can notice that the reachability has no effect on $L(G)$ and $L_m(G)$. Thus, the following equalities always hold

$$L(G) = L(G_r)$$

$$L_m(G) = L_m(G_r)$$

Meet and Synchronous Product In order to be able to construct large scale models of DES using individual small components, one can exploit the DES operation of *synchronous product*.

The meet of two DES models G_1 and G_2, represented by $meet(G_1, G_2)$, includes the synchronized occurrence of the common events in the two models. Let

$$G_1 = (Q_1, \Sigma_1, \delta_1, q_{0_1}, Q_{m_1})$$

$$G_2 = (Q_2, \Sigma_2, \delta_2, q_{0_2}, Q_{m_2})$$

Then $meet(G_1, G_2)$ is the reachable subgenerator of $(Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, (q_{0_1}, q_{0_2}), Q_{m_1} \times Q_{m_2})$. The transition function of $\delta\ :\ (Q_1 \times Q_2) \times \Sigma \to Q_1 \times Q_2$ is defined as: for $(x_1, x_2) \in Q_1 \times Q_2$

and $\sigma \in \Sigma$,

$$\delta((x_1,x_2),\sigma) = \begin{cases} (\delta_1(x_1,\sigma),\delta_2(x_2,\sigma)) & \text{if } \delta_1(x_1,\sigma)! \text{ and } \delta_2(x_2,\sigma)! \\ \text{not defined} & \text{otherwise.} \end{cases}$$

Therefore, it can be easily verified that

$$L(meet(G_1,G_2)) = L(G_1) \cap L(G_2)$$

$$L_m(meet(G_1,G_2)) = L_m(G_1) \cap L_m(G_2)$$

On the other hand, the synchronous product of two DES models G_1 and G_2, denoted by $sync(G_1,G_2)$, is the reachable subgenerator of

$$(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, (q_{0_1}, q_{0_2}), Q_{m_1} \times Q_{m_2})$$

where

$$\delta \ : \ (Q_1 \times Q_2) \times \Sigma \rightarrow Q_1 \times Q_2$$

is defined as: for $(x_1,x_2) \in Q_1 \times Q_2$ and $\sigma \in \Sigma$,

$$\delta((x_1,x_2),\sigma) = \begin{cases} (\delta_1(x_1,\sigma),\delta_2(x_2,\sigma)) & \text{if } \delta_1(x_1,\sigma)! \text{ and } \delta_2(x_2,\sigma)! \\ (\delta_1(x_1,\sigma),x_2) & \text{if } \delta_1(x_1,\sigma)! \text{ and } \sigma \in \Sigma_1 - \Sigma_2 \\ (x_1,\delta_2(x_2,\sigma)) & \text{if } \delta_2(x_2,\sigma)! \text{ and } \sigma \in \Sigma_2 - \Sigma_1 \\ \text{not defined} & \text{otherwise.} \end{cases}$$

In general, synchronous product is used to model the joint operation of two generators; in other words, it combines the models of individual components to form a large scale model for the entire system. The following example illustrates the outlined definitions.

Example 2 *Consider two generators G_1 and G_2 shown in Figures 2.1 and 2.2.*

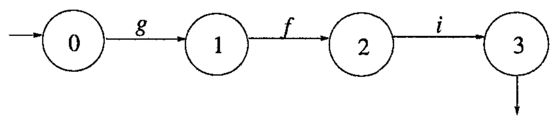


Figure 2.1: G_1.

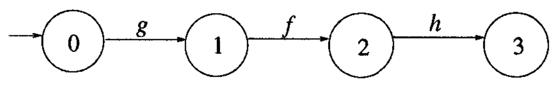


Figure 2.2: G_2.

The generator meet(G_1, G_2) represents the parallel product of G_1 and G_2 and is shown in Figure 2.3. One can notice that only eligible events in both machines can occur in meet(G_1, G_2).

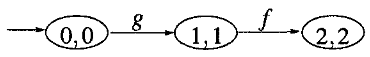


Figure 2.3: *Meet* of G_1 and G_2.

The generator sync(G_1, G_2) represents the synchronous product of G_1 and G_2. It should be noted that the occurrence of 'h' is determined by the transition function of G_2 only since G_1 does not include 'h' in its alphabet. This case happens for 'i' too, since its occurrence is just determined by G_1.

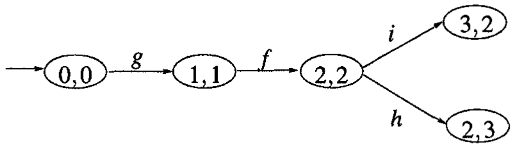


Figure 2.4: *sync* of G_1 and G_2.

2.6 Non-blocking and Safety Properties

A system is called *blocking* if there is a deadlock or livelock in the system, defined below. Deadlock happens whenever the system can reach an unmarked state from which there is no transition going out. Livelock happens whenever the system can reach a set of strongly connected unmarked states with no transition going out. Therefore, a deterministic DES is called *non-blocking* if there is not any deadlock or livelock, i.e. mathematically speaking: $\overline{L}_m(G) = L(G)$. In other words, G is nonblocking if from every state reachable from the initial state of G there is a path to a marked state.

Example 3 *Consider the generator* $G = (Q, \Sigma, \delta, q_0, Q_m)$ *which is shown in Figure 2.5. The marked state* $\{6\}$ *can be identified by the outgoing arrow. The initial state is numbered* 0. *The corresponding alphabet for this generator is* $\Sigma = \{a, b, c, d, e, f\}$.

G blocks since there is a deadlock in state $\{8\}$. *The marked behavior of the generator can be represented by* $L_m(G) = \{afdece\}$ *whose only string leads the generator to the marked state* $\{6\}$.

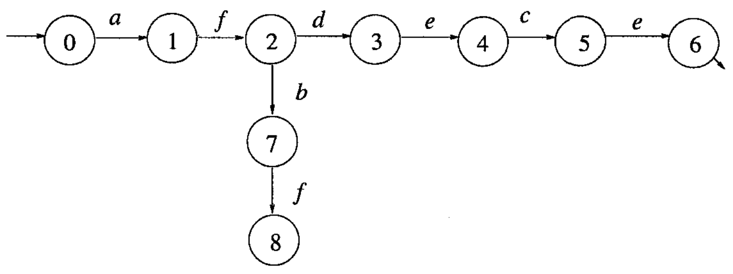


Figure 2.5: G.

An important property of discrete-event systems is called safety property. A system is safe if all strings in the closed behavior of the system belong to a legal or admissible language. In other words, whatever happens in the system should be a part of some desirable behavior if the system is to satisfy a safety property.

In the DES framework, if L_{Spec} represents the specification of some desired behavior, one can investigate the safety property of the system just by checking the subset inclusion $L(G) \subseteq L_{Spec}$, or equivalently $L(G) \cap L_{Spec}^{Comp} = \emptyset$.

2.7 Supervisory Control of Discrete-Event Systems

Usually in DES framework it is desired that the system behavior satisfies a given specification defined either in the form of a formal language or a finite state machine. Performance specifications can be viewed as requiring that certain undesirable sequences of events are not permitted to occur, while at the same time certain other sequences are allowed to happen. Therefore, if the system behavior does not satisfy the desired specification, in order to modify the system's behavior, some kind of control seems to be essential. This can be carried out by restricting the system behavior to a subset of $L(G)$ or $L_m(G)$ that satisfies the desired specification. Assume that the plant is modeled by a generator $G = (Q, \Sigma, \delta, q_0, Q_m)$. Two types of input events are identified. First the set of *controllable* events, denoted by Σ_c, which consists of events that can be prevented from happening by a supervisor or an external agent. Second, the set of *uncontrollable* events, denoted by Σ_{uc}, consists of events which cannot be disabled by any means of control. Examples of uncontrollable events are changes in sensor reading or clock ticks. The set of controllable and uncontrollable events are disjoint, i.e., the relation $\Sigma_c \cap \Sigma_{uc} = \emptyset$ always holds.

Suppose the desired behavior of the DES is represented in the form of a finite state machine $Spec$. Then $LB = L_m(G) \cap L_m(Spec)$ is called the *legal behavior* and is a subset of the DES language that satisfies the specification, that is $LB \subseteq L_m(Spec)$.

Supervisor is an agent which can enable or disable the controllable events such that it prevents G from generating undesirable sequences of events. The supervisor can be represented by a map

$$V : L(G) \longrightarrow \Gamma := \{\gamma \in 2^{\Sigma} : \Sigma_{uc} \subseteq \gamma\}$$

where at string $s \in L(G)$ the event $\sigma \in \Sigma$ is enabled iff $\sigma \in V(s)$.

It can be inferred from the above definition that the supervisor is not allowed to disable an uncontrollable event; therefore, the set of enabled events should always include the set of uncontrollable events. For our purpose in this thesis, we consider that a supervisor can be modeled as a finite state machine

$$S = (X, \Sigma, \eta, x_0, X_m)$$

where X, x_0, $X_m \subseteq X$, and $\eta : X \times \Sigma \rightarrow X$ are the set of states, initial state, set of marked states, and transition function, respectively. An event $\sigma \in \Sigma_c$ is enabled at state $x \in X$ if $\eta(x, \sigma)$ is defined.

System Under Supervision V/G In the Ramadge-Wonham framework [4], there is an interaction between the supervisor and the DES generator, which indicates that they are coupled, and form a closed-loop system. In this paradigm $V(s)$ represents the set of events permitted by the supervisor to happen after $s \in L(G)$, and $\Upsilon : Q \rightarrow 2^{\Sigma}$ is the active event function defined as $\sigma \in \Upsilon(q) \Leftrightarrow \delta(q, \sigma)!$. Thus, for each $s \in L(G)$ generated by G, $V(s) \bigcap \Upsilon(\delta(q_0, s))$ is the set of enabled events that G can execute at its current state $q = \delta(q_0, s)$. The supervisor cannot disable uncontrollable events, particularly those that are active at $\delta(q_0, s)$; therefore, we must have

$$(\Sigma_{uc} \bigcap \Upsilon(\delta(q_0, s)) \subseteq V(s),$$

in this case, the supervisor is said to be "admissible".

When the supervisor is represented by a finite state machine S, the system under supervision is denoted by S/G. The closed behavior of S/G, denoted by $L(S/G) = L(G) \cap L(S)$, contains the sequences of uncontrolled events that survive under the supervision of S. In addition, $L_m(S/G) := L_m(G) \cap L(S/G)$ represents the marked behavior of S/G and includes the sequences of events marked by the uncontrolled process that can be generated by the system under supervision. It should be noted that if all states of the supervisor are marked, i.e. $X = X_m$, then

$$L_m(S/G) = L_m(G) \cap L(S/G) = L_m(G) \cap L(G) \cap L(S) = L_m(G) \cap L_m(S)$$

thus S/G can be represented as $meet(S,G)$.

Thus, the supervisor's duty is to observe the events generated by the plant, and to disable or enable the controllable events in G such that the system under supervision is nonblocking, i.e. $\overline{L}_m(S/G) = L(S/G)$, and also satisfies the desired specification $L_m(S/G) \subseteq L_m(Spec)$.

The supervisor S is said to be **nonblocking** if the closed-loop system S/G is nonblocking, or equivalently,

$$\overline{L}_m(S/G) = \overline{L_m(G) \cap L(S/G)} = L(S/G).$$

Definition 1 *([4]) A language $K \subseteq \Sigma^*$ is controllable with respect to G if $\overline{K}\Sigma_{uc} \cap L(G) \subseteq \overline{K}$.*

A supervisor S is admissible if and only if $L(S/G)$ is controllable. When a language K is not controllable, it is not possible to design a nonblocking supervisor such that the system under supervision satisfies the specification; therefore, one seeks the largest element of the class of controllable sublanguages of K, denoted by

$$\mathscr{C}_{in}(K) = \{M \subseteq K \mid \overline{M}\Sigma_{uc} \cap L(G) \subseteq \overline{M}\}.$$

The largest element of $\mathscr{C}_{in}(K)$ is denoted by $\sup \mathscr{C}_{in}(K)$. Then, designing a nonblocking supervisor for $\sup \mathscr{C}_{in}(K)$ such that the modified specification is satisfied would be feasible.

Theorem 1 *([5]) Given a generator G, $K \neq \emptyset$ and $K \subseteq L_m(G)$, there exists a non-blocking supervisor S for G such that $L_m(S/G) = K$ if and only if*

- *K is controllable with respect to G and Σ_{uc},*
- *K is $L_m(G)$-closed.*

Example 4 *Consider the DES plant G and specification Spec defined over $\Sigma = \{a,b,c,e,f\}$. The set of controllable events is $\Sigma_c = \{b,c\}$.*

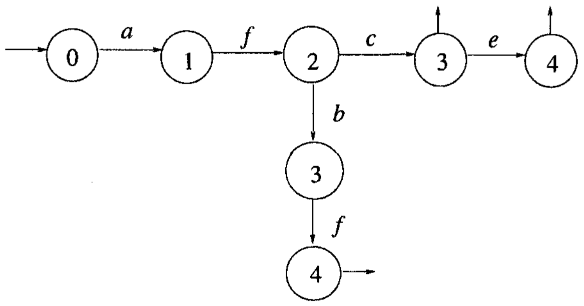


Figure 2.6: Plant G.

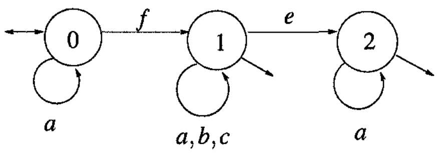


Figure 2.7: *Spec* in the form of a finite automaton.

In order to find the appropriate supervisor one should use the outlined Theorem 1. The optimal supervisor which makes the system under supervision satisfy the desired specification is displayed in Figure 2.8.

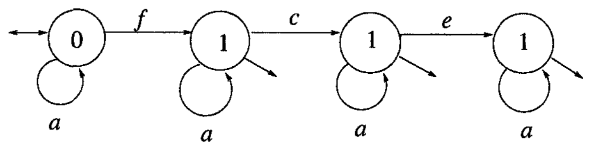


Figure 2.8: Nonblocking supervisor S.

Thus, the plant under supervision is nonblocking and satisfies the desired specification. The closed-loop system S/G is shown in Figure 2.9.

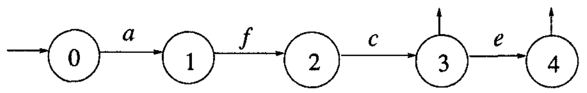


Figure 2.9: System under supervision S/G.

2.8 TTCT

TTCT software is used for verification and synthesis of supervisory control of discrete-event systems. A brief review of some useful commands and procedures which are used in some computations in this thesis is provided in the following.

Meet: Parallel product of two DES generators is calculated by **meet** command. This command produces G_3 with event set $\Sigma_3 = \Sigma_1 \cap \Sigma_2$:

$$G_3 = \mathbf{meet}(G_1, G_2)$$

Sync: In TTCT software, one can compute the synchronous product of two DES models by **sync** command. The generator G_3 is the synchronous product of two DES models G_1 and G_2 with event set $\Sigma_3 = \Sigma_1 \cup \Sigma_2$:

$$G_3 = \textbf{sync}(G_1, G_2)$$

Condat: In order to check the admissibility of a designed supervisor, one can use the command **condat** in TTCT. This command provides a list of all plant events disabled at each supervisor state. Therefore, if this list does not include any uncontrollable events, the supervisor is admissible.

Nonconflicting: The nonconflicting property of two DES models can be verified using the **nonconflict** command in TTCT.

$$\textbf{nonconflict}(G_1, G_2) = \mathit{true}? \tag{2.4}$$

The response “true” indicates that

$$\overline{L_m(G_1) \cap L_m(G_2)} = L(G_1) \cap L(G_2)$$

meaning that there is no blocking in any reachable state of the product automaton $\textbf{meet}(G_1, G_2)$. Therefore, using $\textbf{nonconflict}(G_1, G_2)$ command, one can investigate whether $L_m(G_1)$ and $L_m(G_2)$ are nonconflicting.

Chapter 3

Overview of Hybrid Systems

3.1 Introduction

In Broad terms, almost all control systems today have computer control structure and issue logical as well as continuous control commands. We refer to these as hybrid systems. Generally, the term hybrid refers to a mixture of two fundamentally different types of objects or methods. Hybrid control systems form a class of systems that are richer than ordinary control systems from the modeling, analysis, and verification points of view. In every system which includes continuous and discrete dynamics, the continuous dynamics are affected not only by the continuous control, but also by the discrete mode. Similarly, the discrete dynamics are affected by both discrete control actions and, indirectly, by the continuous dynamics. Thus, a strong interaction between analogue and logical dynamics exists. In addition to control inputs in the original system, disturbances can be considered as a mixture of discrete and continuous signals which add to the complexity of modeling of the original system. Therefore, hybrid control systems can be more complicated in comparison with purely discrete or continuous systems.

A common example of supervisory hybrid systems is found whenever a computer is used to supervise the behavior of a continuous-valued process. The continuous process may be a closed loop control system whose mathematical representation has the form of an ordinary differential equation. The computer program may be seen as a supervisor which controls the control loop by selecting various reference inputs. One can notice that the state of the program can evolve over a discrete set and the dynamics of the discrete-event process can be modeled using language theoretic framework. Therefore one can observe that a computer supervised system is a hybrid system since continuous and discrete dynamics interact.

In traditional approaches to the analysis of hybrid systems, their hybrid nature is often neglected, and the system is considered either as purely discrete dynamics or as purely continuous ones. The current hybrid framework, a combination of differential equations and finite automata, study continuous and discrete behavior simultaneously. There are numerous reasons to use hybrid control framework for analysis, verification, and design of different control systems. The primary motivation is the interaction between the continuous and discrete parts of a system, like the discrete planning of continuous processes. The other reason for using hybrid modeling is the fast increasing need for hierarchical organization of control in many complex applications, such as manufacturing processes or air traffic management systems.

Many complex engineering systems can be modeled using hybrid control framework. This kind of modeling can even be applied to systems with multiple time scales, where fast dynamics can be considered as discrete changes which has effect on slower dynamics; thus, a system with multiple time scales can be considered a hybrid system in which well-known methods of design and analysis can be employed for synthesis and verification purposes. For example, in a manufacturing process, one can model the processing of individual machines by their service times. Therefore, the composed discrete-event system for the whole production line can be represented by a Petri net and analyzed using queuing theory. In this paradigm, the control performance of each machine is modeled using continuous dynamics. The continuous feedback control depends on the service time specifications. If they are satisfied, the higher-level discrete-event system is a suitable model for the overall dynamics of the manufacturing system. The separation of asynchronous and synchronous controls will, in many cases, lead to a very conservative design. In the manufacturing example, this could result in large buffer sizes and inefficient use of machines. On the other hand, if all the dynamics and the interactions in the manufacturing process are captured within the hybrid model, it is possible to simply optimize the overall behavior and achieve a high-performance design. Tools in hybrid control systems address this type of problems.

There has been a large amount of research regarding the modeling of complex control systems encompassing continuous as well as discrete dynamics. The general motivation is to develop an equational framework which can be used to model all possible system behaviors such as chattering, switching, and autonomous jumping. In this paradigm, one challenge is to consider the switching nature of discrete-event processes present in such systems as well as their continuous behavior. From the modeling point of view, equational representations familiar to most system scientists cannot provide a framework which captures both continuous and discrete dynamics of such systems at the same time. In other words, developing a modeling paradigm which provides greater perspective toward the discrete-event dynamics of hybrid systems seems to be necessary in order to advance the hybrid control framework.

In general, the study of hybrid control systems is useful in designing intelligent control systems with a high degree of autonomy. Problems related to hybrid control systems (such as design of switching controllers for continuous processes) have been studied for many years. However, from different points of view, analysis of hybrid systems per se has evolved into a new research discipline over the years due to advances in the field of discrete-event systems and the availability of softwares for modeling and simulation of complex systems with mixed continuous and discrete dynamics. This has captured the attention of many researchers in control engineering and computer science.

One of the primary motivations for the interest in hybrid systems is rapid advances in computer and networking technology which have accelerated the development of large scale supervisory systems. Examples of such large scale systems include traffic control systems, communication networks, and power distribution systems. For example, in the case of traffic control systems, the hybrid nature of the overall system can be verified by considering the fact that the supervision process, which can be thought of as a discrete process, is combined with dynamics of vehicles which are continuous processes.

Current methods for the design, modeling, and verification of hybrid systems depend heavily on simulation testing which is a costly and time-consuming process. Moreover, this procedure of analysis cannot provide provable guarantees for safe operation of the system. The hope is that hybrid systems theory will provide a systematic framework for system engineers which reduces the cost of large scale system design while also improving the system reliability.

3.2 Review of Works in Hybrid Systems

Considerable research has been dedicated to modeling, analysis and synthesis of hybrid systems based on different mathematical paradigms which can be characterized along several directions. In broad terms, approaches differ with respect to the emphasis on, or the complexity of, the continuous and discrete dynamics; see for example [6, 7, 8, 9, 10, 11, 12, 13, 14, 15]. From a continuous-time perspective, there are several equational models which deal with traditional control problems related to hybrid systems such as stability and optimal control [16, 1, 17]. On the other hand, from a discrete-event perspective, there are different methodologies for describing real-time embedded systems [18, 19, 20, 21, 6, 22]. For example, Antsaklis *et al.* [13, 2, 23, 24, 12] used a discrete-event dynamical systems approach to model systems composed of interaction between Ordinary Differential Equations (ODEs) and finite automata. Nerode and Kohn [25] used an automata theoretic approach to model complex hybrid dynamical systems while taking timing considerations into account. Alur *et al.* [18, 19, 26, 10] used hybrid automata, an extension of timed automata [27], to find appropriate models for hybrid dynamical systems. Several researchers [28, 24] have used Petri nets to model the discrete aspects of hybrid systems. Several attempts have been made to apply supervisory control ideas to hybrid control systems, such as Stiver approach [12, 29, 30], Franke approach [31], and Raisch approach [32, 33]. In Koutsoukos *et al.* [2] continuous state space is partitioned by hypersurfaces in order to represent the approximated continuous plant by a DES abstract model via a nondeterministic finite automaton. In addition, a plant symbol may be generated each time the continuous system trajectory crosses a hypersurface.

From a discrete point of view toward hybrid systems, there has been two major research directions. On one side, primary use of formal graph theoretic models for computer processes led to the development of a framework with very high potential for practical purposes by the computer science community. Finite state machines and Petri nets represent two well known examples of such models. While powerful computational tools are developed for the simulations of such formal models, it can be noticed that in dealing with real-time applications, an extension of these traditional methodologies seems to be essential. This need has led to an attempt to apply traditional and highly successful model checking frameworks for finite state machines to real-time systems. The result is timed and hybrid automaton [14].

On the other side, a hybrid system model dealing with discrete and continuous dynamics, proposed by the control community, can be found in [2]. In this case, the hybrid system is viewed as a logical discrete-event supervisor connected to a continuous subsystem. This work suggested a logical discrete-event system approach to hybrid controller synthesis which is similar to traditional approaches to sampled-data control. The approach suggested to extract an equivalent discrete-event model of the continuous subsystem which can then be supervised using extensions of the Ramadge-Wonham supervisory control theory [34].

The hybrid systems studied in this thesis are based on the outlined modeling that hybrid systems consist of mixture of continuous-time and discrete-event dynamics, which typically possess a hierarchical structure, characterized by continuous dynamics at the lowest level of abstraction and discrete supervisor at the highest level of abstraction. The discrete and continuous systems are connected through an interface that transforms continuous-valued measurements into discrete event signals and vice versa. The interface dynamic is somehow similar to A/D and D/A in sampled-data systems paradigm. Generally, the continuous process is described by a set of nonlinear differential equations while logical decision-making part is a DES supervisor modeled by a finite automaton. Hybrid systems also arise naturally whenever logical decision-making is mixed with the generation of continuous control laws.

3.3 Applications

There are numerous applications for hybrid control systems ranging from embedded real-time systems to large-scale manufacturing facilities, from aerospace control to traffic control management, from chemical process control to communication networks and finally from engine control to robotics. Hybrid system methodologies are also applicable to any kind of switched systems where the system switches between various set points or operational modes in order to extend its effective operating range. Such applications are found in aerospace and power systems.

It is important to capture the hybrid behavior of a control system when the continuous and the discrete components of the system interact such that their behavior influences the overall system performance. This is the case in many applications, particularly when performance measures should be improved under safety constraints. The hybrid nature of some of the outlined complex applications are illustrated below:

- Communication networks: In order to reduce the model complexity, large data flows are modeled using continuous variables, while traffic control mechanisms such as routing are considered as discrete dynamics.
- Embedded control: A micro-computer embedded in a physical device has discrete behavior because of its finite-precision computations and the process of quantization of the signals, but it interacts with a continuous-time environment through actuators and sensors.
- Robotics: A manipulator is accurately governed by continuous dynamics, but impacts and load shifting causes discrete and asynchronous changes.

Chapter 4

Supervisory Control of DES and Hybrid Systems

This chapter tries to make a bridge between concepts of supervisory control of DES and hybrid systems so that supervisory control framework can be used to design discrete-event supervisors for hybrid systems. In this chapter, we first present an extensive overview of supervisory control of hybrid systems, proposed by [2], where continuous dynamics of a system can be modeled as a discrete-event system using the proposed procedure of extraction of DES plant model. Second, we present the details of discrete-event systems with output, followed by presenting the main results of supervisory control of DES with output. Necessary and sufficient conditions for the existence of nonblocking input supervisory control as well as nonblocking output supervisory control are described at length. Applicability of the results is illustrated through several examples.

4.1 Supervisory Control of Hybrid Systems

In supervisory control of hybrid systems, proposed by [2], the control loop consists of three components: *continuous process*, *discrete-event controller* and *interface*. The dynamics of the continuous process (*plant*), which could include continuous controllers, is governed by differential equations. The controller (*supervisor*) is an event-driven, asynchronous discrete-event system modeled by a finite automaton. The interface enables continuous plant and discrete-event controller to communicate with each other, connecting them via a feedback loop. This control architecture is shown in Figure 4.1.

In general, this modeling framework is useful in control analysis of hybrid systems in which separation of continuous and discrete parts is feasible. However, for certain hybrid systems in which the dynamics of continuous and discrete parts cannot be separated, this representation can still be used as a mathematical tool to study the system's behavior and to identify its properties rather than to implement control strategies for the system.

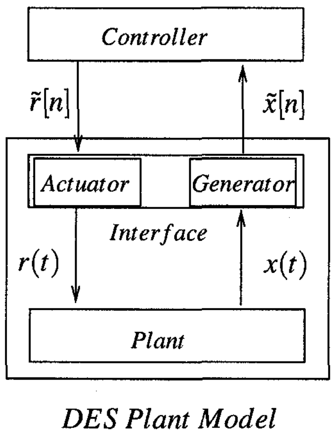


Figure 4.1: Hybrid control system.

In hybrid systems, the plant is considered as a nonlinear, time-invariant system represented by the differential equation

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{r}(t))$$

where $\mathbf{x}(t) \in X \subset \mathbb{R}^n$ and $\mathbf{r}(t) \in R \subset \mathbb{R}^m$ are the state and input vectors, respectively, and $f : X \times R \rightarrow \mathbb{R}^n$ is a controlled vector field. It is assumed that the function $f(\cdot, \mathbf{r}(t)) : X \rightarrow \mathbb{R}^n$ is continuous in X and all conditions for existence and uniqueness of its solutions are satisfied. This representation is general and can describe a large class of continuous systems. The supervisor is a finite automaton which can be represented by a five tuple $S = (\tilde{S}, \tilde{X}, \tilde{R}, \delta, \phi)$, where $\tilde{S}$ is the set of *states*, $\tilde{X}$ is the set of *plant symbols*, $\tilde{R}$ is the set of *controller symbols*, $\delta : \tilde{S} \times \tilde{X} \rightarrow \tilde{S}$ is the *transition function* which determines the next state of the controller based on its current state and plant symbol received from the interface, and $\phi : \tilde{S} \rightarrow \tilde{R}$ is the *output function* which assigns a controller command to each state of the controller.

Since the plant and the controller utilize different types of signals, they do not have the ability to communicate with each other directly. Therefore, an interface is essential to convert continuous-time signals to sequences of symbols and vice versa. The interface consists of two subsystems, *the actuator* and *the generator*. Generator and actuator play roles somewhat similar to, but more general than, A/D and D/A in digital control systems. The actuator feeds the appropriate control signals to the plant by converting the sequence of controller symbols to continuous-time staircase signals. The generator, on the other hand, converts the continuous-time output of the plant to the symbolic sequence of inputs for the controller. In this paradigm, generator's function includes two procedures: first, a triggering process which will determine when a plant symbol must be generated, and second, a selection process in which it will be determined which particular plant symbol is to be generated.

The outlined triggering mechanism is based on the idea of plant events in the system. In the hybrid control framework, a plant event is generated when a hypersurface is crossed in a predefined direction. Hypersurfaces are used to partition the plant's state space into disjoint sets. Crossing these hypersurfaces by the state trajectory results in the occurrence of plant events. The fact behind this idea is that an event can be considered as the realization of a specified condition. In case of hybrid systems, this condition is assumed to be an open region of the state space, separated from the remainder of the state space by a group of hypersurfaces. Now, if the state trajectory crosses one of these hypersurfaces and enters into a new open region, then the relevant plant event will be generated by the system.

Mathematically, a set of smooth functionals $\{h_i : \mathbb{R}^n \rightarrow \mathbb{R}, i \in I\}$ are defined on the state space of the plant. It is required that

$$\nabla_x h_i(\xi) \neq 0, \forall \xi \in \mathcal{N}(h_i) = \{\xi \in \mathbb{R}^n : h_i(\xi) = 0\}$$

in which case the null space of the functional will form a hypersurface that separates the state space. Considering the outlined definitions, a sufficient condition for the generation of a plant event is:

$$h_i(x(t)) = 0,\ d/dt(h_i(x(t))) \neq 0.$$

From the discrete supervisor's point of view, the behavior of plant and interface combined can be modeled by a discrete-event system. As shown in Figure 4.1, this combination generates the sequence of discrete outputs by the generator and accepts the sequence of discrete inputs from the controller. The discrete-event system which represents continuous plant together with generator and actuator can be modeled by a nondeterministic finite automaton called the *DES plant model.* The DES plant is represented by a five-tuple $(\tilde{P}, \tilde{X}, \tilde{R}, \psi, \lambda)$ where $\tilde{P}$ is the discrete set of states, $\tilde{X}$ represents the set of plant symbols and $\tilde{R}$ is the set of control commands. For a given pair of control command and DES plant state, the *state transition function* $\psi : \tilde{P} \times \tilde{R} \rightarrow 2^{\tilde{P}}$ determines the set of possible next DES plant states. Moreover, the current and next states are mapped to a set of plant symbols via the *output function* $\lambda : \tilde{P} \times \tilde{P} \rightarrow 2^{\tilde{X}}$. Note that the output function can be equivalently written as a function $\omega : \tilde{P} \times \tilde{R} \rightarrow 2^{\tilde{X}} : (\tilde{p}, \tilde{r}) \mapsto \bigcup_{\tilde{p}' \in \psi(\tilde{p}, \tilde{r})} \lambda(\tilde{p}, \tilde{p}')$. Every state of the DES plant is associated with an open region in the state space of the plant, which is bounded by a number of hypersurfaces. Assume that the DES plant is in state $\tilde{p}$, the discrete controller is in state $\tilde{s}$, and the input symbol $\tilde{r} = \phi(\tilde{s})$ is supplied by the controller. Then as the continuous state variable crosses a hypersurface, the system enters a new open region, or equivalently, a new state $\tilde{p}' \in \psi(\tilde{p}, \tilde{r})$ of the DES plant. A plant symbol $\tilde{x} \in \lambda(\tilde{p}, \tilde{p}')$ will be generated if the hypersurface is crossed in the prespecified direction. In this case the new plant symbol $\tilde{x}$ is observed by the discrete controller, which will cause it to move to a new state $\tilde{s}' = \delta(\tilde{s}, \tilde{x})$. A new control command $\tilde{r}' = \phi(\tilde{s}')$ is then issued by the controller which is fed back to the plant via the actuator, and the whole process repeats.

4.2 Extraction of DES Plant Model

In order to put it in a mathematical framework, an equivalence relation $\equiv_p$ is defined on the set $\{\xi \in \mathbb{R}^n : h_i(\xi) \neq 0, i \in I\}$ as:

$$\xi_1 \equiv_p \xi_2 \text{ iff } h_i(\xi_1)h_i(\xi_2) > 0, \forall i \in I$$

Each equivalence class of $\equiv_P$ corresponds to a unique state of the DES plant. Binary vectors are used to index the set of states $\tilde{P}$. For a binary vector b, the state $\tilde{p}_b \in \tilde{P}$ is associated with the open region $\{\xi \in \mathbb{R}^n : b_i = 1 \Leftrightarrow h_i(\xi) < 0\}$. The DES plant model changes state when a hypersurface is crossed by the state trajectory.

Thus, the set of DES plant states $\tilde{P}$ is defined as:

$$\tilde{P} = \{\xi \in \mathbb{R}^n : h_i(\xi) \neq 0, i \in I\} / \equiv_p$$

$$\tilde{p}_b = \{\xi \in \mathbb{R}^n : b_i = 0 \Rightarrow h_i(\xi) > 0, b_i = 1 \Rightarrow h_i(\xi) < 0\}$$

As an example, $\tilde{p}_{1001}$ is a DES plant state associated with the open region satisfying the following inequalities:

$$h_1(x) < 0$$
$$h_2(x) > 0$$
$$h_3(x) > 0$$
$$h_4(x) < 0$$

Definition 2 *([2]) Two DES plant states, $\tilde{p}_a$ and $\tilde{p}_b$, are adjacent at $(i \in I, \xi \in \mathcal{N}(h_i))$ if for all $j \in I$*

$$\begin{aligned}
&\mathcal{N}(h_j) = \mathcal{N}(h_i) \Rightarrow a_i \neq b_i \\
&\mathcal{N}(h_j) \neq \mathcal{N}(h_i) \Rightarrow a_i = b_i \\
&\xi \in \overline{\tilde{p}_a} \cap \overline{\tilde{p}_b}
\end{aligned}$$

where a and b are the binary vectors associated with $\tilde{p}_a$ and $\tilde{p}_b$, respectively, while $\overline{\tilde{p}_x}$ is the closure of $\tilde{p}_x$.

Proposition 1 *([2]) Given a hybrid control system, with f and h_i continuously differentiable, then $\tilde{p}_a \in \psi(\tilde{p}_b, \tilde{r}_k)$ if and only if there exist $i \in I$ and $\xi \in \mathcal{N}(h_i)$ such that the following conditions are satisfied:*

$$\begin{aligned}
&\tilde{p}_a \text{ and } \tilde{p}_b \text{ are adjacent at } (i, \xi) \\
&b_i = 0 \Rightarrow \nabla_x h_i(\xi).f(\xi, \gamma(\tilde{r}_k)) < 0 \\
&b_i = 1 \Rightarrow \nabla_x h_i(\xi).f(\xi, \gamma(\tilde{r}_k)) > 0
\end{aligned}$$

The procedure of extraction of DES plant model with respect to definitions of hypersurfaces, adjacent plant states and the outlined proposition can be illustrated via the following example.

Example 5 *Consider a thermostat/furnace system in which thermostat is set at* 70 *degrees Fahrenheit. The dynamics of the system can be described as follows: if the temperature is below* 70, *then the furnace will be turned* ***ON*** *until the room temperature reaches* 75 *degrees. At this temperature, the furnace will be turned* ***OFF*** *automatically. The model for the thermostat/furnace hybrid control system can be represented by:*

$$\dot{x} = 0.0042(T_0 - x) + .1r$$

where r stands for the voltage across the furnace circuit which can be 12 *when the furnace is* ***ON*** *and* 0 *when the furnace is* ***OFF****; x stands for the room temperature; and T_0 stands for the outside temperature which is assumed to be* 60 *degrees Fahrenheit. Note that this model is a simplified representation of a real hybrid control system.*

This hybrid control system consists of two functionals:

$$h_1(x) = x - 75$$
$$h_2(x) = 70 - x$$

which partitions the state space into three open regions: $(-\infty, 70)$, $(70, 75)$ and $(75, \infty)$. Whenever the room temperature exceeds 75 *degrees, $h_1(x) > 0$, thus the hypersurface is crossed and the corresponding plant symbol x_1 will be generated; on the other hand, when the temperature falls below* 70 *degrees, then $h_2(x) > 0$, indicating that the hypersurface is crossed by the state trajectory in the specified direction, thus the corresponding plant symbol x_2 is generated.*

$T_0 = 60$

$h_1(x) = x - 75$

$h_2(x) = 70 - x$

$\delta(r_1) = 0$

$\delta(r_2) = 12$

$p_{11} \in \psi(p_{10}, r_2)$

p_{11} *and* p_{10} *are adjacent*

$i = 2, h_2(x) = 70 - x \Rightarrow \mathbb{N}(h_2) = 70 = \xi$

$f(\xi, r_2) = 0.0042(T_0 - x) + 0.1r > 0$

$b_2 = 0 \Rightarrow -1 * f < 0 \Rightarrow$ *conditions hold*

$p_{10} \in \psi(p_{11}, r_1)$

p_{11} *and* p_{10} *are adjacent*

$i = 2, h_2(x) = 70 - x \Rightarrow \mathbb{N}(h_2) = 70 = \xi$

$f(\xi, r_1) = 0.0042(T_0 - x) + 0.1r < 0$

$b_2 = 1 \Rightarrow -1 * f > 0 \Rightarrow$ *conditions hold*

$p_{01} \in \psi(p_{11}, r_2)$

p_{11} *and* p_{01} *are adjacent*

$i = 1, h_1(x) = x - 75 \Rightarrow \mathbb{N}(h_1) = 75 = \xi$

$f(\xi, r_2) = 0.0042(T_0 - x) + 0.1r > 0$

$b_1 = 1 \Rightarrow -1 * f > 0 \Rightarrow$ *conditions hold*

$p_{11} \in \psi(p_{01}, r_1)$

p_{11} *and* p_{01} *are adjacent*

$i = 1, h_1(x) = x - 75 \Rightarrow \mathbb{N}(h_1) = 75 = \xi$

$f(\xi, r_1) = 0.0042(T_0 - x) + 0.1r < 0$

$b_1 = 0 \Rightarrow -1 * f < 0 \Rightarrow$ *conditions hold*

The DES plant model corresponding to the hybrid control system is shown in Figure 4.2.

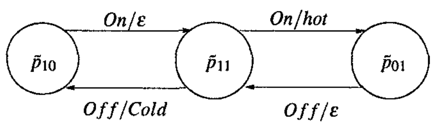


Figure 4.2: DES plant model for the thermostat/furnace system.

It should be noted that the DES plant is an overapproximation of the continuous system: a behavior is accepted by the continuous system only if it is accepted by the DES plant. Given a region $\tilde{p}$ and a discrete input $\tilde{r}$, the image $\psi(\tilde{p}, \tilde{r})$ should include all regions the continuous state might enter from some initial state $\mathbf{x}_0 \in \tilde{p}$ and by applying some input $\mathbf{r}(\cdot)$ supplied by the actuator. Thus, the DES plant is in general nondeterministic. Several researchers have studied the problem of supervisory control of non-deterministic DES [35, 36]; the theory developed in this work can be used to control such systems when the specification is defined on the language of output events.

In this thesis we assume that the extraction of the DES plant model for a hybrid control system is complete, and an input/output automaton representing the DES plant is available for the problem of supervisory control.

4.3 DES with Output

When a DES plant is deterministic it can be modeled by a Mealy automaton, which is a finite automaton in which each transition is labeled with a pair of input and output events. When a transition from q to q' is labeled i/o, the input symbol i received at state q makes the automaton move to state q' while the output symbol o is generated. The sequences of outputs will be controlled via controlling the corresponding sequences of inputs. Given the alphabets of input and output events Σ_i and Σ_o, respectively, the behavior of a Mealy automaton G can be represented by a triple $(L_m(G), L(G), \theta)$, where $L_m(G) \subseteq L(G) \subseteq \Sigma_i^*$, $L(G)$ is prefix-closed and $\theta : L(G) \longrightarrow \Sigma_o^*$ is an output map modeling the interaction between sequences of outputs and inputs of the system. The output map θ is recursively defined as follows: for all $s \in L(G)$ and $\sigma \in \Sigma_i$

$$\theta(\varepsilon) = \varepsilon; \qquad \theta(s\sigma) = \begin{cases} \theta(s) & or, \\ \theta(s)\tau & \end{cases} \quad \text{for some } \tau \in \Sigma_o \,.$$

The output map θ is prefix-preserving, that is, if $s \leq s'$ then $\theta(s) \leq \theta(s')$. In fact, θ is a causal map that assigns a sequence of outputs to every sequence of inputs, based on the DES behavior.

The problem of supervisory control of hybrid systems, which is a special class of supervisory control of DES with output, is to control the system by enabling or disabling an appropriate set of controllable input events based on the observation of its output sequences, such that the output of the supervised system satisfies a desired specification on outputs.

4.4 Supervisory Control of Discrete-Event Systems with Output

In supervisory control problem, the controller's objective is to restrict the operation of the uncontrolled system in such a way that some undesirable behavior is excluded from the plant's closed-loop behavior. For discrete-event systems, when the desirable behavior is specified by a formal language over the alphabet of input events, classical Supervisory Control Theory (SCT) of DES [34, 5, 4] can be employed to design a supervisor so that the behavior of the supervised system satisfies the specification.

When a DES generates outputs, which can be modeled by a Mealy automaton, the desired behavior of the system may be specified by a language over the alphabet of output events. Then the objective of supervisory control is to restrict the behavior of the plant by appropriately disabling a subset of controllable input events in such a way that sequences of output events generated by the supervised system satisfy the output specification. As usual, denote the disjoint sets of controllable and uncontrollable input events by $\Sigma_{i,c}$ and $\Sigma_{i,u}$, respectively, where $\Sigma_i = \Sigma_{i,c} \cup \Sigma_{i,u}$. Controllable input events can be disabled by a supervisor. As a result, an output event can be prevented from occurring if the input event that leads to its generation is controllable and can therefore be disabled. We call a supervisor an *input supervisor* if it decides which controllable input events to disable by observing sequences of input events. Similarly, we call a supervisor an *output supervisor* if it decides which controllable input events to disable by observing sequences of output events.

Non-blocking input supervisory control

Suppose the plant is given by a triple $G = (L_m(G), L(G), \theta)$, where $L_m(G)$ and $L(G)$ are the marked and closed languages of G, respectively. The languages $L_m(G)$ and $L(G)$ are defined over Σ_i with the property that $L_m(G) \subseteq L(G)$ and $L(G) = \overline{L_m(G)}$. The output function $\theta : L(G) \longrightarrow \Sigma_o^*$ is a causal map which represents the internal relation between output and input sequences in the discrete-event system. As shown in the previous section, G can be modeled by a Mealy automaton. Denote the set of control patterns with Γ, where:

$$\Gamma = \{\gamma \subseteq \Sigma_i | \ \gamma \supseteq \Sigma_{i,u}\}$$

A control pattern contains all uncontrollable input events, indicating that a supervisor does not have the ability to disable uncontrollable input events.

An *input supervisory control map* for DES G is any map $V_i : L(G) \longrightarrow \Gamma$. The pair (G, V_i)

will be written as V_i/G to suggest 'G under the supervision of V_i'. An input supervisor has access to all system inputs while at the same time it can observe the corresponding sequences of outputs through the system's output map. The system under supervision is denoted by $(L_m(V_i/G), L(V_i/G), \theta|_{L(V_i/G)})$. The language generated by the closed-loop system (V_i/G) is defined recursively as follows:

1. $\varepsilon \in L(V_i/G)$
2. $\forall s \in \Sigma_i^*, \sigma \in \Sigma_i.\ s\sigma \in L(V_i/G) \Leftrightarrow s\sigma \in L(G) \wedge \sigma \in V_i(s)$.

The language $L_m(V_i/G)$ is the collection of plant marked strings that survive the supervision of V_i:

$$L_m(V_i/G) = L_m(G) \cap L(V_i/G).$$

Finally, $\theta|_{L(V_i/G)}$ denotes the restriction of θ to $L(V_i/G)$. We call an input supervisory control map *nonblocking* if $\overline{L_m(V_i/G)} = L(V_i/G)$. In input supervisory control, the decision-making process by a supervisor is based on observing the entire sequences of inputs generated by the system, and comparing the corresponding output sequences with the desired output specification. In fact, the supervisor enables an input event if its corresponding output does not violate the output specification language.

The input supervisory control problem is formulated as follows: given a specification $K \subseteq \theta(L_m(G))$ on the outputs of the system, design a supervisor $V_i : L(G) \rightarrow \Gamma$ such that the system under supervision implements the specification on outputs, in other words, $\theta(L_m(V_i/G) = K$.

Non-blocking output supervisory control

In output supervisory control, a supervisor does not have access to input sequences generated by the system; rather, it can only observe sequences of the system outputs. Compared to an input supervisor, an output supervisor has to be more conservative because the decision as to whether to enable or disable an input event can only be based on the observation of output sequences: an output supervisor has to make the same control decision after observing all input sequences that generate the same output sequence. As a result, the class of specifications that can be satisfied by output supervisory control is more restricted.

Formally, an *output supervisory control map* for a DES G is any map $V_o : \theta(L(G)) \rightarrow \Gamma$. The supervised system, denoted by V_o/G, is defined as before. In particular, for $s \in L(V_o/G)$ we have $s\sigma \in L(V_o/G)$ if and only if $s\sigma \in L(G)$ and $\sigma \in V_o(\theta(s))$. The output supervisory control problem is formulated as follows: given a specification $K \subseteq \theta(L_m(G))$ on the outputs of the system, design a supervisor $V_o : \theta(L(G)) \rightarrow \Gamma$ such that the system under supervision implements the specification on outputs; in other words, $\theta(L_m(V_o/G) = K$.

It should be noted that an output supervisor is 'memoryless', in the sense that it does not have any recollection of the history of events enabled by the supervisor in the past. In closing, we would like to point out that the notions of input and output supervisory control of DES are somewhat similar to the notions of state- and output-feedback control in classical control theory, respectively. In input supervisory control and its classical counterpart state-feedback control, the designer has access to the system's internal behavior (states in state feedback or inputs in input supervisory control). This makes the design of a controller more flexible and accurate in comparison to the output supervisory control or its classical counterpart output-feedback control, where controller must be designed based only on observing system outputs regardless of the system's internal behavior. While output supervisory control/output-feedback control is simple, input supervisory control/state-feedback control is more powerful. In what follows, we characterize the class of languages that can be implemented by input and output supervisory control maps.

Definition 3 *A language $H \subseteq L_m(G)$ is said to be* output-consistent *if*

$$\forall s_1, s_2 \in \overline{H}, \sigma \in \Sigma_i : \ \theta(s_1) = \theta(s_2) \wedge s_2\sigma \in L(G) \wedge s_1\sigma \in \overline{H} \implies s_2\sigma \in \overline{H}.$$

In plain words, a language H is output-consistent if every pair of strings in $\overline{H}$ with identical outputs have consistent one-step continuations with respect to $\overline{H}$.

Theorem 2 *Given a specification on outputs $K \subseteq \theta(L_m(G))$:*

1. *There exists a nonblocking input supervisory control map $V_i : L(G) \to \Gamma$ such that $\theta(L_m(V_i/G)) = K$ if and only if there exists a language $H \subseteq \theta^{-1}(K) \cap L_m(G)$ such that $\theta(H) = K$, H is controllable with respect to G and it is $L_m(G)$-closed.*

2. *There exists a nonblocking output supervisory control map $V_o : \theta(L(G)) \to \Gamma$ such that $\theta(L_m(V_o/G)) = K$ if and only if in addition to the conditions of part 1, H is output-consistent.*

Proof. Part 1 follows directly from the main result in Ramadge-Wonham supervisory control theory. Let controllable and $L_m(G)$-closed $H \subseteq \theta^{-1}(K) \cap L_m(G)$ be such that $\theta(H) = K$. Since $H \subseteq L_m(G)$, it follows that there exists a nonblocking input supervisory control map $V_i : L(G) \to \Gamma$ such that $L_m(V_i/G) = H$, and hence $\theta(L_m(V_i/G)) = \theta(H) = K$. Conversely, let nonblocking input supervisory control map $V_i : L(G) \to \Gamma$ be such that $\theta(L_m(V_i/G)) = K$. Define $H := L_m(V_i/G)$. It follows that H is controllable and $L_m(G)$-closed, and $\theta(H) = \theta(L_m(V_i/G)) = K$.

Next, we show that if the condition in part 2 is satisfied and V_i is a nonblocking input supervisory control map such that $L_m(V_i/G) = H$, then as shown in Figure 4.3 V_i *factors through* θ: that is, there exists an output supervisory control map V_o such that $V_i = V_o \circ \theta$.

$$\begin{tikzcd}
\theta(L(G)) \arrow[rd, dashed, "\exists V_o"] & \\
L(G) \arrow[u, "\theta"] \arrow[r, "V_i"'] & \Gamma
\end{tikzcd}$$

Figure 4.3: Supervisory control can be based on the observation of outputs if and only if the input supervisory control map V_i factors through θ.

Define an output supervisory control map $V_o : \theta(L(G)) \to \Gamma$ according to:

$$\forall s \in L(G).\ V_o(\theta(s)) := V_i(s).$$

We show that the map V_o is well-defined, that is, for $s_1, s_2 \in L(G)$ if $\theta(s_1) = \theta(s_2)$ then $V_o(\theta(s_1)) = V_o(\theta(s_2))$. We prove this by contradiction: assume in the nontrivial case where

$s_1, s_2 \in L(V_i/G)$ that $\theta(s_1) = \theta(s_2)$ but $V_i(s_1) = V_o(\theta(s_1)) \neq V_o(\theta(s_2)) = V_i(s_2)$. Then there must exist $\sigma \in \Sigma_i$ such that, for example, $\sigma \in V_i(s_1)$ but $\sigma \notin V_i(s_2)$. Assume without loss of generality that $s_1\sigma$ and $s_2\sigma$ are both in $L(G)$ (otherwise for $s \in L(G)$ if we have $s\sigma \notin L(G)$ then σ can be safely added to or removed from $V_i(s)$ without changing the closed-loop behavior). It follows from the definition that $s_1\sigma \in L(V_i/G)$ but $s_2\sigma \notin L(V_i/G)$, which contradicts the condition in part 2 since $L(V_i/G) = \overline{H}$.

We conclude that V_o is well-defined. It follows from the definition that $L(V_o/G) = L(V_i/G) = \overline{H}$ and $L_m(V_o/G) = L_m(V_i/G) = H$, and thus $\theta(L_m(V_o/G)) = \theta(H) = K$.

■

Let

$$\mathscr{H}_i(K) = \{H \subseteq \theta^{-1}(K) \cap L_m(G) \mid \theta(H) = K,\ H \text{ is controllable and } L_m(G)\text{-closed}\}$$

and

$$\mathscr{H}_o(K) = \{H \subseteq \theta^{-1}(K) \cap L_m(G) \mid \theta(H) = K,\ H \text{ is controllable, } L_m(G)\text{-closed and output-consistent}\}$$

When the type of supervisory control is not specified (input vs. output), we simply write $\mathscr{H}(K)$.

According to Theorem 2, for a nonblocking supervisory control map V such that $\theta(L_m(V/G)) = K$ to exist, it is necessary and sufficient that $\mathscr{H}(K) \neq \emptyset$. It is straightforward to verify that $\mathscr{H}_i(K)$ is closed under arbitrary union and therefore one can always find a minimally restrictive input supervisor when one exists (i.e. $\mathscr{H}_i(K) \neq \emptyset$). However, as Example 7 at the end of this section suggests, $\mathscr{H}_o(K)$ is not closed even under finite union, and therefore a minimally restrictive output supervisor does not in general exist.

A natural candidate for H in Theorem 2 is $\theta^{-1}(K) \cap L_m(G)$. As the following result suggests, since $\theta(\theta^{-1}(K) \cap L_m(G)) = K$ one only needs to check $\theta^{-1}(K) \cap L_m(G)$ for controllability and $L_m(G)$-closeness (and output-consistency in case of output supervisory control).

Proposition 2 *For $K \subseteq \theta(L_m(G))$ we have*

$$\theta(\theta^{-1}(K) \cap L_m(G)) = K.$$

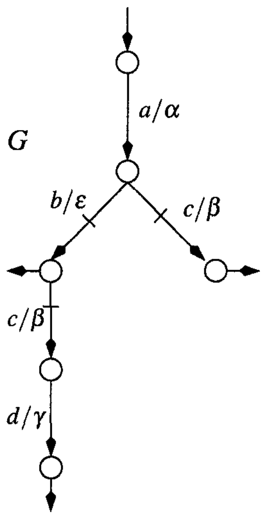
Proof. ($\subseteq$) We have:

$$\begin{aligned}
\theta(\theta^{-1}(K) \cap L_m(G)) &\subseteq \theta(\theta^{-1}(K)) \cap \theta(L_m(G)) \\
&\subseteq K \cap \theta(L_m(G)) \\
&= K
\end{aligned}$$

($\supseteq$) Let $t \in K$. Since $K \subseteq \theta(L_m(G))$, it follows that $t \in \theta(L_m(G))$, i.e. there exists $s \in L_m(G)$ such that $\theta(s) = t$. Since $\theta(s) = t \in K$, it follows that $s \in \theta^{-1}(K) \cap L_m(G)$ and therefore $\theta(s) = t \in \theta(\theta^{-1}(K) \cap L_m(G))$. ■

Example 6 *Consider the system shown in Figure 4.5, together with a specification language K of the desired output behavior. We represent a language M by an automaton* **M** *generating marked and closed languages M and $\overline{M}$, respectively. Note that $K \subseteq \theta(L_m(G))$.*

The language $H_1 = \theta^{-1}(K) \cap L_m(G)$ shown in Figure 4.5(a), despite being controllable and $L_m(G)$-closed, does not satisfy the condition in part 2 of Theorem 2: $a, ab \in \overline{H_1}$, $\theta(a) = \theta(ab)$, $abc \in L(G)$, $ac \in \overline{H_1}$ but $abc \notin \overline{H_1}$.



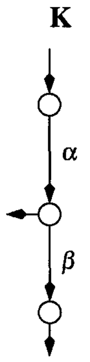
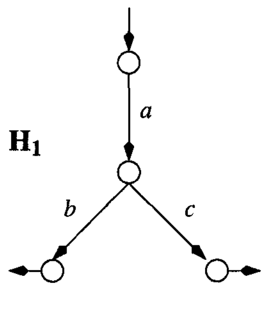


Figure 4.4: (a) DES G with output. (b) Specification K on the desired output behavior.



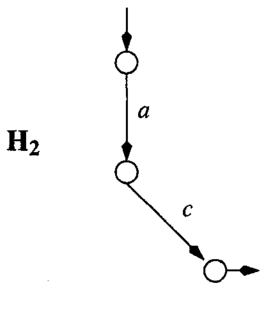


Figure 4.5: (a) The language $H_1 = \theta^{-1}(K) \cap L_m(G)$. (b) The language $H_2 \subseteq H_1$ where $\theta(H_1) = \theta(H_2) = K$.

However, it is not difficult to see that the language H_2 shown in Figure 4.5(b) belongs to $\mathscr{H}_o(K)$, and by Theorem 2 there exists an output supervisory control map $V_o : \theta(L(G)) \longrightarrow \Gamma$ such that $\theta(L_m(V_o/G)) = K$. The supervisor simply disables the input event b after observing the output event α. $\diamond$

Example 7 *Consider the system shown in Figure 4.6, together with a specification language K over the output alphabet and the language $H_1 = \theta^{-1}(K) \cap L_m(G)$. Although controllable and $L_m(G)$-closed, H_1 does not satisfy the condition in part 2 of Theorem 2. Intuitively, after observing α the output supervisor does not know for sure whether or not to disable c.*

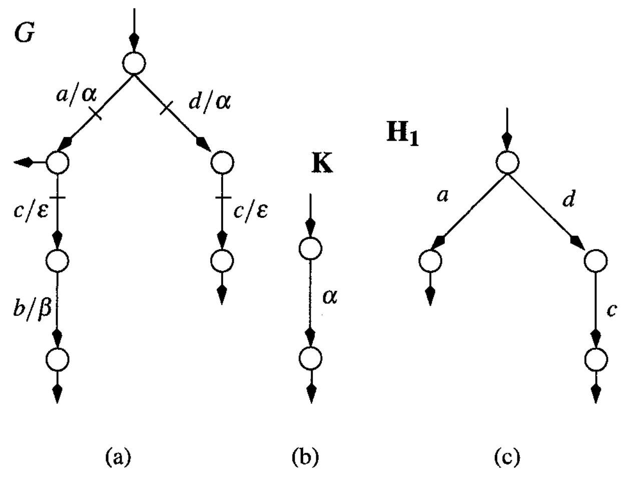


Figure 4.6: (a) DES G with output. (b) Specification K of the desired output behavior. (c) $H_1 = \theta^{-1}(K) \cap L_m(G)$.

Languages H_2 and H_3 shown in Figure 4.7 satisfy all conditions of Theorem 2 for the existence of output supervisory control. To implement H_2, a supervisor must disable d in its initial state and disable c after observing α. To implement H_3, a supervisor must just disable *a in its initial state. Note, however, that $H_1 = H_2 \cup H_3$ cannot be implemented by an output supervisory control map.* $\diamond$

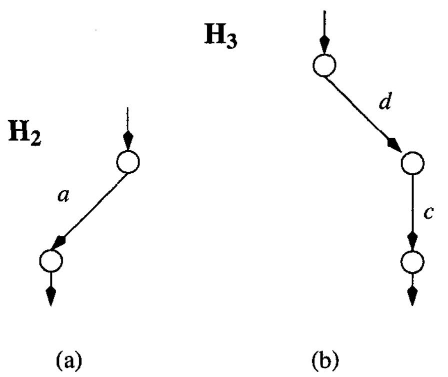


Figure 4.7: (a) H_2 and (b) H_3 satisfy all conditions of Theorem 2 for the existence of output supervisory control, but $H_1 = H_2 \cup H_3$ does not.

For $E \subseteq \theta(L_m(G))$, if $\mathscr{H}(E) = \emptyset$ then by Theorem 2 no nonblocking supervisory control map V can be found such that $\theta(L_m(V/G)) = E$. In this case, a natural question to ask is whether a largest subset $K^{\uparrow} \subseteq E$ can be found such that $\mathscr{H}(K^{\uparrow}) \neq \emptyset$. To this end we define

$$\mathscr{C}_i(E) = \{K \subseteq E \mid \mathscr{H}_i(K) \neq \emptyset\}$$

$$\mathscr{C}_o(E) = \{K \subseteq E \mid \mathscr{H}_o(K) \neq \emptyset\}$$

Once again, it can be readily verified that $\mathscr{C}_i(E)$ is closed under arbitrary union, while as illustrated by the next example $\mathscr{C}_o(E)$ is not. Thus, in general, minimally restrictive supervisory control based on the observation of outputs is not possible unless conditions that are yet to be determined are imposed on the class of plants and/or specifications.

Example 8 *Consider the system shown in Figure 4.8(a), where all states are marked and all events are controllable. Both specifications K_1 and K_2 shown in Figures 4.9(b) and 4.9(c) can be implemented by output supervisory control maps V_{o1} and V_{o2}, resulting in closed-loop languages shown in Figures 4.8(b) and 4.8(c), respectively. Supervisory control map V_{o1} disables e in its initial state, while V_{o2} disables b after observing the output event α.*

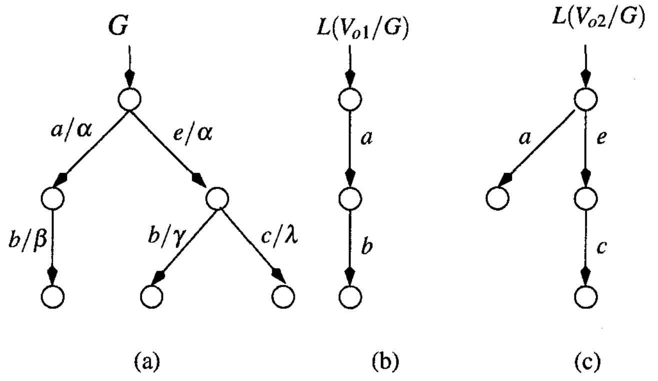


Figure 4.8: (a) DES G with output. (b) The system under supervision of V_{o1}. (c) The system under supervision of V_{o2}.

On the other hand, no output supervisory control map V_o can implement $K = K_1 \cup K_2$; if such a supervisor existed, then $\alpha\beta, \alpha\lambda \in K = \theta(L(V_o/G))$. Since $ab \in \theta^{-1}(\alpha\beta)$ and $ec \in \theta^{-1}(\alpha\lambda)$, we would have $ab, ec \in L(V_o/G)$. On the other hand, since $a, e \in L(V_o/G)$, $\theta(a) = \theta(e) = \alpha$, $ab \in L(V_o/G)$ and $eb \in L(G)$, by the condition in part 2 of Theorem (2) we would have $eb \in L(V_o/G)$, or $\theta(eb) = \alpha\gamma \in K$, which is a contradiction. Intuitively, it is not clear whether any such V_o should disable b after observing α: if it does, the output event β cannot be generated, while if it does not, the output event γ may be generated.

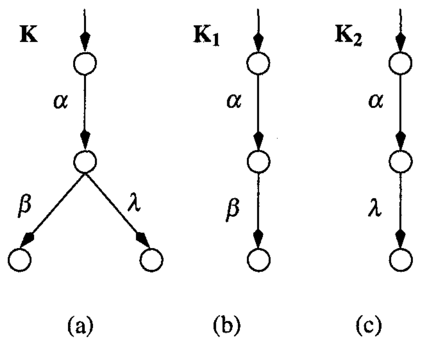


Figure 4.9: Specifications (a) $K = K_1 \cup K_2$, (b) K_1 and (c) K_2.

It is worth to notice that if the causal output map θ can be considered as projection, then the output consistency property is somewhat similar to the classical supervisory control definition of observability. Therefore, comparing our result regarding supervisory control of discrete-event systems with output with the method for partial observation proposed by Cieslak et al in [37], one can notice that the causal output map which we used to assign sequences of outputs to any sequences of inputs in discrete-event systems is the general form of mask function which Cieslak *et al* used in supervisory control with partial observation. In their work, observation of the supervisor is specified by a mask function

$$M : \Sigma \rightarrow \Delta \cup \{\varepsilon\}$$

in which $\delta = M(\alpha)$ is the symbol observed by the supervisor when the plant transition α happens. Similar to our case, the supervisor does not have the ability to distinguish between symbols α and α' if their mask function $M(\alpha)$ and $M(\alpha')$ are equal. The advantage of proposed causal output map over mask function is the generalization of sequences of outputs which can be assigned to sequences of inputs, while partial observation using mask function can only affect individual input symbols.

Chapter 5

Computation of Output-Consistent Languages and Application to Hybrid Systems

This chapter is divided into two parts. The first part presents a computational algorithm to determine whether the language of a closed-loop system is output-consistent. The algorithm makes it feasible to translate the language-based concept of output-consistency into automata framework, thus enabling one to check the output-consistency of the language of a closed-loop system using familiar operations in automata theory. In the second part, the supervisory control theory of DES with output is applied to DES abstractions of hybrid systems. The aim is to design discrete-event supervisors such that the overall hybrid system's behavior satisfies the desired specification.

5.1 Algorithm For Computation of Output-Consistency

Suppose the original system is modeled by a Mealy automaton G, and the system under supervision is modeled by S, where $L_m(S) \subseteq L_m(G)$. Our objective is to design an algorithm that outputs 'yes' if the language marked by S is output-consistent with respect to G.

Since checking output-consistency in the language-based framework requires consistency of control actions for strings generating identical output sequences, construction of output state machines for tracking outputs of the plant and the controlled system seems to be necessary. To obtain a deterministic output automaton G_o corresponding to the Mealy automaton G, first a nondeterministic ε-automaton $G_{\varepsilon,o}$ is obtained by labeling every transition in G with its generated output symbol (ε when no output symbol is generated), and converting the resulting nondeterministic ε-automaton to a deterministic one by following the procedure outlined below (adopted from [38]).

An ε-automaton is represented by a five-tuple $A = (Q, Q_m, q_0, \Sigma, \delta)$, where Q is a finite set of states, $Q_m \subseteq Q$ is a set of marker states, q_0 is an initial state, Σ is an alphabet of events, and $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow pwr(Q)$ is a transition function. To convert the ε-automaton into a Deterministic Finite Automaton (DFA), the notion of ε-closure of a state is introduced. Informally, ε-closure of a state q consists of all states that can be reached from q along a path whose every transition is labeled with ε. Formally, the ε-closure of a state q, denoted by $\bar{q}_\varepsilon$, is the smallest set with the property that $q \in \bar{q}_\varepsilon$, and $\delta(p,\varepsilon) \subseteq \bar{q}_\varepsilon$ for all $p \in \bar{q}_\varepsilon$. For any set of states $P \subseteq Q$ define $\overline{P}_\varepsilon = \bigcup_{p\in P} \bar{p}_\varepsilon$. The equivalent DFA of the nondeterministic ε-automaton A is denoted by $D = (R, R_m, r_0, \Sigma, \xi)$, where $R = pwr(Q)$, $R_m = \{r \in R |\ r \cap Q_m \neq \emptyset\}$, $r_0 = \overline{q_0}_\varepsilon$, and

$$\xi(r,\sigma) = \bigcup_{q\in r} \overline{\delta(q,\sigma)}_\varepsilon$$

Following the outlined procedure, denote by G_o and S_o the output automata of plant G and the controlled system S, respectively. In order to determine whether $L_m(S)$ is output-consistent with respect to G, we form the product of G, G_o, S and S_o to keep track of the system output when a controllable input event is disabled. Let $G = (Q, Q_m, q_0, \Sigma_i, \Sigma_o, \zeta, \omega)$ be a Mealy automaton where Q is a set of states, Q_m is a set of marked states, q_0 is an initial state, Σ_i is an alphabet of input events, Σ_o is an alphabet of output events, $\zeta : Q \times \Sigma_i \to Q$ is a transition function and $\omega : Q \times \Sigma_i \to \Sigma_o$ is an output function. Denote the corresponding output automaton by $G_o = (R, R_m, r_0, \Sigma_o, \delta)$. The product of G and G_o is a Mealy automaton $G \times G_o = (Q \times R, Q_m \times R_m, (q_0, r_0), \Sigma_i, \Sigma_o, \eta, \phi)$, where for all $(q,r) \in Q \times R$ and $\sigma \in \Sigma_i$

$$\eta((q,r),\sigma) = \begin{cases} (\zeta(q,\sigma), \delta(r, \omega(q,\sigma))) & ;\ \text{if all partial functions are defined,} \\ \text{undefined} & ;\ \text{otherwise} \end{cases}$$

and

$$\phi((q,r),\sigma) = \omega(q,\sigma).$$

Each state of $G \times S \times G_o \times S_o$ can be represented by a four-tuple, in which each component represents the state of the corresponding machine. Observe that the property of output-consistency is violated if conflicting control decisions are made at two strings where output sequences are identical. More precisely, the algorithm looks for two states of the form $(q_G, q_S, q_{G_o}, q_{S_o})$ and $(q'_G, q'_S, q_{G_o}, q_{S_o})$ of the product Mealy automaton where an event $\sigma \in \Sigma_i$ is eligible at q_G, enabled at q_S, but disabled at q'_S. Since the last two components (namely, q_{G_o} and q_{S_o}) are identical, the output sequences corresponding to any two strings $s, s' \in \Sigma_i^*$ reaching $(q_G, q_S, q_{G_o}, q_{S_o})$ and $(q'_G, q'_S, q_{G_o}, q_{S_o})$, respectively, could potentially be equal, and thus the algorithm returns 'no'. The language of the closed-loop system $L_m(S)$ is output-consistent if no such pair of states could be found.

```
Output Consistent(G,S)
input:  Mealy automata of plant G & controlled system S;
compute $G_{\varepsilon,o}$ and $S_{\varepsilon,o}$;
compute $G_o$ and $S_o$;
H:=$G \times S \times G_o \times S_o$
for all states $(q_G, q_S, q_{G_o}, q_{S_o})$ and $(q'_G, q'_S, q_{G_o}, q_{S_o})$ of H, all $\sigma \in \Sigma_i$;
  if $\sigma$ is eligible in $q_G$ and $q'_G$, enabled in $q_S$, disabled in $q'_S$
    return 'no';
  endif;
endfor;
return 'yes';
```

The following example illustrates the steps of the algorithm.

Example 9 *Consider the system G shown in Figure 5.1, where all states are marked and all input events are controllable; therefore, the conditions of controllability and $L_m(G)$-closeness are automatically satisfied for any automaton S with $L_m(S) = L(S) \subseteq L_m(G)$. We would like to find out if an output supervisor can be found such that the system under supervision can be represented by the Mealy automaton S in Figure 5.2. Since $L_m(S)$ is controllable and $L_m(G)$-closed, it follows from the theory developed in the previous section that this would be the case if $L_m(S)$ is output-consistent with respect to G, which we would like to verify using the algorithm presented in this section.*

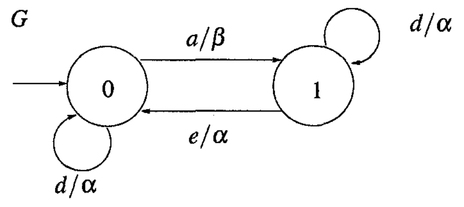


Figure 5.1: Plant G.

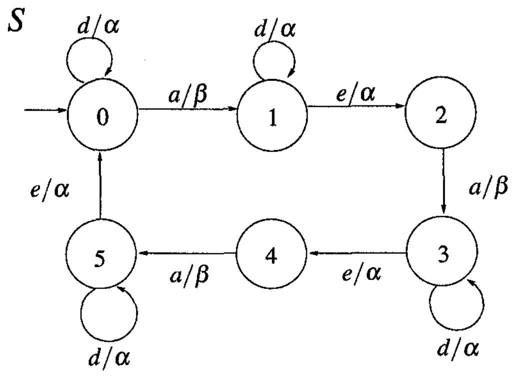


Figure 5.2: The system under supervision S.

In order to examine the output-consistency of $L_m(S)$, we first construct deterministic output machines for G and S. The corresponding output machines, denoted respectively by G_o and S_o, are shown in Figures 5.3 and 5.4, respectively. The Mealy automaton $G \times S \times G_o \times S_o$ is shown in Figure 5.5.

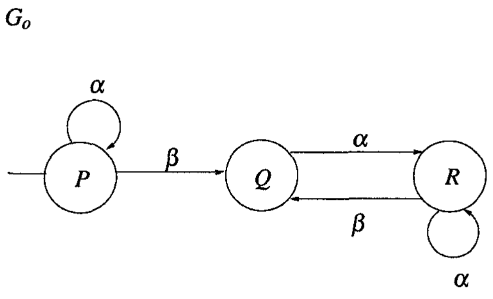


Figure 5.3: Deterministic output machine G_o.

S_o

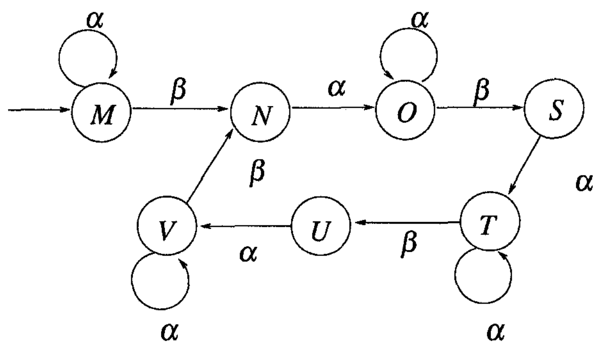


Figure 5.4: Deterministic output machine S_o.

Final Product

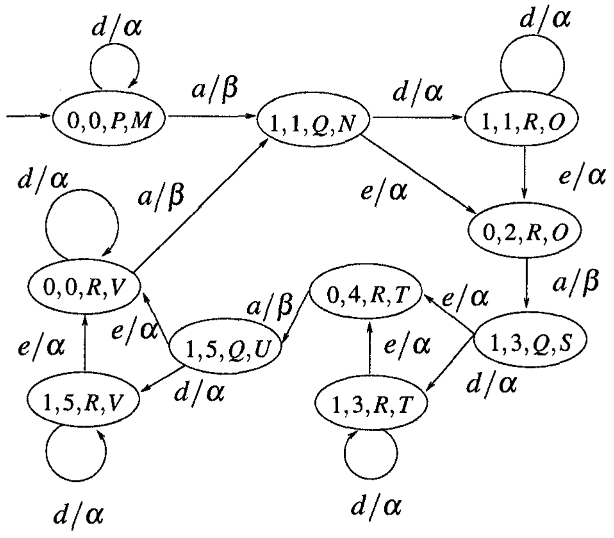


Figure 5.5: The product Mealy automaton $G \times S \times G_o \times S_o$.

In the product automaton, each state is labeled with four components corresponding to states in G, S, G_o and S_o, respectively. The algorithm presented in this section isolates three pairs of states of the product automaton in which outputs are potentially identical but conflicting control decisions are made: $((1,1,R,O),(0,2,R,O))$, $((0,4,R,T),(1,3,R,T))$, *and* $((1,5,R,V),(0,0,R,V))$. *For instance, in state* $(1,1,R,O)$, *the event d is enabled while in the other state of the pair, namely,* $(0,2,R,O)$, *this event is disabled. We conclude that* $L_m(S)$ *is not output-consistent with respect to G, hence no output supervisory control can be found to implement S. One can see that the definition of output-consistency is violated since* $ad, ae \in \overline{H}$, $\theta(ad) = \theta(ae) = \beta\alpha$, *aed* $\in L(G)$ *and add* $\in \overline{H}$, *but aed* $\notin \overline{H}$. $\diamond$

5.2 Application of Supervisory Control of DES with Output in Hybrid Systems

In this section, we use supervisory control theory of DES with output developed in Section 4.4 to design discrete-event supervisors for continuous dynamics approximated by DES with output. As mentioned in Section 4.4, continuous process along with its interface can be approximated by a DES plant model, a nondeterministic finite automaton which can be represented by a 6-tuple $(\tilde{P}, \tilde{P}_m, \tilde{X}, \tilde{R}, \psi, \lambda)$, where in addition to the model in [2], we now allow the user to specify a subset $\tilde{P}_m \subseteq \tilde{P}$ of marker states. We assume that $\tilde{X}$ includes a silent symbol ε corresponding to the case where crossing a hypersurface does not generate an output. In addition, as in [2] we assume that crossing a hypersurface generates at most one output symbol, and thus λ can be reduced to a partial function $\lambda : \tilde{P} \times \tilde{P} \rightarrow \tilde{X}$. Due to the nondeterminism of the DES plant model and specification of the desired behavior on output sequences in hybrid systems, the procedure to design discrete-event supervisors for hybrid systems is more involved. In the first step, to make the plant model deterministic, we rename identical transition labels originating from every state. The nondeterministic nature of the plant is preserved, as we require that a supervisor treat all transitions with identical labels (in the nondeterministic model) consistently, that is, to enable or disable all events which have identical corresponding symbols in the nondeterministic automaton. Using the above procedure for every event of the nondeterministic finite automaton, the result would be an automaton with the property of pseudo-determinism from the analytical point of view while the system is still nondeterministic in the eyes of the supervisor.

For an event $\tilde{r} \in \tilde{R}$, define the *degree of nondeterminism* of $\tilde{r}$, denoted by $n_{\tilde{r}}$, to be the maximum number of transitions, out of any state labeled with $\tilde{r}$:

$$n_{\tilde{r}} := \max_{\tilde{p}\in\tilde{P}} |\psi(\tilde{p},\tilde{r})|.$$

Define the set of *siblings* corresponding to $\tilde{r}$ according to:

$$Sbl(\tilde{r}) = \{\tilde{r}^i | i = 1,2,\ldots,n_{\tilde{r}}\}.$$

It is assumed that all siblings of a controllable or uncontrollable event are controllable or uncontrollable, respectively. In hybrid systems, input events are controller commands which are all controllable; therefore, all of their siblings are controllable, too. In other words, the property of controllability is always held in DES plant model which results in having a controllable automaton. We assume that controllability of $\tilde{r}$ is inherited by all siblings in $Sbl(\tilde{r})$. Let the nondeterministic DES plant be given by $G_{ndet} = (\tilde{P},\tilde{P}_m,\tilde{X},\tilde{R},\psi,\lambda)$. Define its corresponding deterministic Mealy automaton as $G_{det} = (\tilde{P},\tilde{P}_m,\tilde{X},\tilde{U},\zeta,\omega)$, where:

- $\tilde{U} = \bigcup_{\tilde{r}\in\tilde{R}} Sbl(\tilde{r})$

- For $\tilde{p}\in\tilde{P}$ and $\tilde{r}\in\tilde{R}$, if $\psi(\tilde{p},\tilde{r}) \neq \emptyset$ let $\psi(\tilde{p},\tilde{r}) = \{\tilde{p}^1,\ldots,\tilde{p}^k\}$, where $k \leq n_{\tilde{r}}$ is a natural number. Define the partial function $\zeta : \tilde{P}\times\tilde{U} \rightharpoonup \tilde{P}$ according to:

$$\forall i \in \{1,2,\ldots,k\}.\ \zeta(\tilde{p},\tilde{r}^i) := \tilde{p}^i.$$

- $\omega : \tilde{P}\times\tilde{U} \rightharpoonup \tilde{X} : (\tilde{p},\tilde{u}) \mapsto \lambda(\tilde{p},\zeta(\tilde{p},\tilde{u}))$.

We say $\tilde{u},\tilde{u}' \in \tilde{U}$ are *siblings*, denoted by $\tilde{u} \equiv_{sbl} \tilde{u}'$, if there exists an $\tilde{r}\in\tilde{R}$ such that $\tilde{u},\tilde{u}' \in Sbl(\tilde{r})$. An output supervisory control map V_o for G_{det} is also an output supervisory control map for G_{ndet} if for all $t \in \tilde{X}^*$ and $\tilde{u}, \tilde{u}' \in \tilde{U}$, we have

$$\tilde{u} \in V_o(t) \wedge \tilde{u} \equiv_{sbl} \tilde{u}' \implies \tilde{u}' \in V_o(t). \tag{5.1}$$

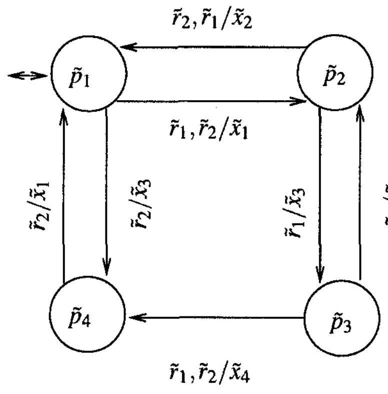
Intuitively, to a supervisor all siblings look identical, and if one is enabled (disabled), all other siblings should be enabled (respectively, disabled) by the supervisor as well.

Given a language $H \subseteq L_m(G_{det})$ and a nonblocking output supervisory control V_o with $L_m(V_o/G_{det}) = H$, the language H satisfies the condition given by (5.1) if

$$\forall s \in \overline{H},\ \tilde{u}, \tilde{u}' \in \tilde{U}.\ s\tilde{u} \in \overline{H} \wedge s\tilde{u}' \in L(G_{det}) \wedge \tilde{u} \equiv_{sbl} \tilde{u}' \implies s\tilde{u}' \in \overline{H}. \tag{5.2}$$

Thus, when designing supervisory control for nondeterministic DES models of hybrid systems, the condition given by (5.2) must be checked along with all other conditions of Theorem 2. In the following example, we use the ideas developed in this section to illustrate the conversion of a nondeterministic automaton modeling a hybrid system's DES plant to a deterministic Mealy automaton with the same behavior, and examine whether a discrete-event supervisor for a specification on outputs can be designed.

Example 10 *Consider the system shown in Figure 5.6 as a DES plant model, where the initial state is marked and all input events are controllable. A common problem in supervisory control of hybrid systems is to supervise the system such that it never enters some undesirable states. Suppose in this example that we want to design a supervisor such that it prevents the state of the system from entering the unsafe region $\tilde{p}_4$. The specification K can be defined on the outputs of* *the system, represented by a finite automaton shown in Figure 5.7. This example is similar to Example 2 in [2], with the major difference being that in our DES plant model the initial state is marked.*



$\tilde{r}_2, \tilde{r}_1/\tilde{x}_2$
$\tilde{p}_1$
$\tilde{p}_2$
$\tilde{r}_1, \tilde{r}_2/\tilde{x}_1$
$\tilde{r}_2/\tilde{x}_1$
$\tilde{r}_2/\tilde{x}_3$
$\tilde{r}_1/\tilde{x}_3$
$\tilde{r}_1/\tilde{x}_2$
$\tilde{p}_4$
$\tilde{p}_3$
$\tilde{r}_1, \tilde{r}_2/\tilde{x}_4$

Figure 5.6: DES plant model G_{ndet}.

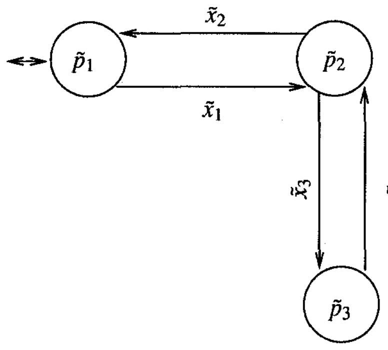


Figure 5.7: The desired sequence of outputs

In order to design a nonblocking output supervisory control for the DES plant to implement the desired specification, we first convert the nondeterministic model of the DES plant into a deterministic Mealy automaton. The deterministic model of the DES plant is shown in Figure 5.8.

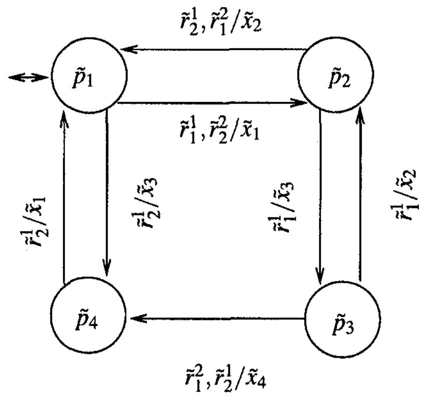

$\tilde{r}_2^1,\tilde{r}_1^2/\tilde{x}_2$
$\tilde{p}_1$
$\tilde{p}_2$
$\tilde{r}_1^1,\tilde{r}_2^2/\tilde{x}_1$
$\tilde{r}_2^1/\tilde{x}_1$
$\tilde{r}_2^1/\tilde{x}_3$
$\tilde{r}_1^1/\tilde{x}_3$
$\tilde{r}_1^1/\tilde{x}_2$
$\tilde{p}_4$
$\tilde{p}_3$
$\tilde{r}_1^2,\tilde{r}_2^1/\tilde{x}_4$

Figure 5.8: Equivalent deterministic DES plant model G_{det}.

As illustrated in the previous examples, a natural candidate for the solution of output supervisory control problem is $H = \theta^{-1}(K) \cap L_m(G_{det})$, which is represented by the automaton of Figure 5.9. Observe that H is controllable (since all events are controllable), $L_m(G_{det})$-closed and output-consistent. Therefore, by Theorem 2, there exists an output supervisory control V_o such that $L_m(V_o/G_{det}) = H$. However, H does not satisfy the condition given by (5.2), because $\tilde{r}_2^2 \in \overline{H}$, $\tilde{r}_2^1 \in L(G_{det})$ and $\tilde{r}_2^1 \equiv_{sbl} \tilde{r}_2^2$ but $\tilde{r}_2^1 \notin \overline{H}$. In state $\tilde{p}_1$, the output supervisory control map V_o requires disabling $\tilde{r}_2^1$ while enabling $\tilde{r}_2^2$, which is not possible because in the actual nondeterministic system $\tilde{r}_2^1$ and $\tilde{r}_2^2$ are indistinguishable.

We construct the output supervisory control map $\hat{V}_o$ from V_o by disabling, in addition, $\tilde{r}_2^2$ in state $\tilde{p}_1$ and $\tilde{r}_1^1$ in state $\tilde{p}_3$. The supervised system $\hat{V}_o/G_{det}$ is shown in Figure 5.10. There are two major problems with supervisor $\hat{V}_o$: the output generated by the supervised system is strictly smaller than K, and more important, the supervisor is blocking, *as the supervised system blocks in state $\tilde{p}_3$. We conclude that the supervisor $\hat{V}_o$ does not satisfy the output specification K due*

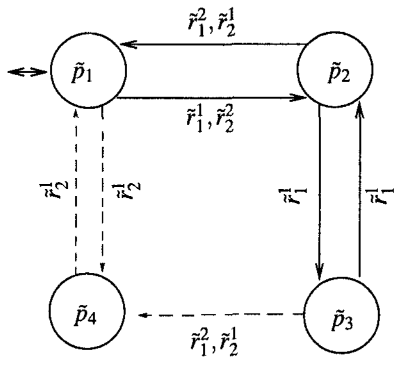


Figure 5.9: The language $H = \theta^{-1}(K) \cap L_m(G_{det})$ is controllable, $L_m(G_{det})$-closed and output-consistent.

to blocking in state $\tilde{p}_3$, and inability to produce the output event $\tilde{x}_2$ after observing the output event $\tilde{x}_3$.

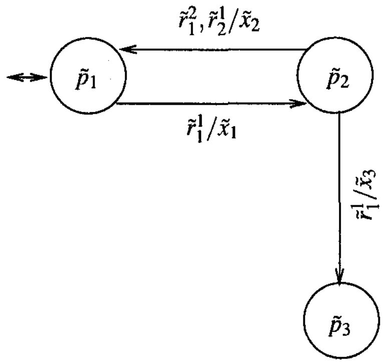

$\tilde{r}_1^2,\tilde{r}_2^1/\tilde{x}_2$
$\tilde{p}_1$
$\tilde{p}_2$
$\tilde{r}_1^1/\tilde{x}_1$
$\tilde{r}_1^1/\tilde{x}_3$
$\tilde{p}_3$

Figure 5.10: The supervised system $\hat{V}_o/G_{det}$.

Next, consider the output specification $K' \subseteq K$ shown in Figure 5.11, which in addition to the unsafe region $\tilde{p}_4$, prevents the system from entering the blocking region $\tilde{p}_3$. The language $H' = \theta^{-1}(K') \cap L_m(G_{det})$ shown in Figure 5.12 satisfies all conditions of Theorem 2, and therefore there exists a nonblocking output supervisory control map V_o' such that $L_m(V_o'/G_{det}) = H'$.

The supervisor V_o' simply disables $\tilde{r}_2^1$ in state $\tilde{p}_1$ and $\tilde{r}_1^1$ in state $\tilde{p}_2$. Again, H' does not satisfy the condition given by (5.2), while its subset $\hat{H}' \subseteq H'$, shown in Figure 5.13, satisfies all conditions for the existence of a nonblocking output supervisory control map $\hat{V}_o'$ such that $L_m(\hat{V}_o'/G_{det}) = \hat{H}'$ and $\theta(L_m(\hat{V}_o'/G_{det})) = K'$. The supervisor $\hat{V}_o'$ is obtained from V_o' by disabling all *siblings when* some *are disabled; thus, $\hat{V}_o'$ disables both $\tilde{r}_2^1$ and $\tilde{r}_2^2$ (i.e. disables $\tilde{r}_2$) in state $\tilde{p}_1$, and $\tilde{r}_1^1$ and $\tilde{r}_1^2$ (i.e. disables $\tilde{r}_1$) in state $\tilde{p}_2$.*

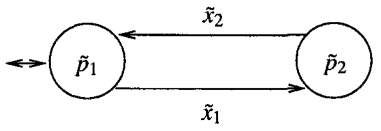


Figure 5.11: Specification $K' \subset K$ on outputs.

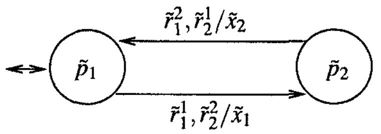

$\tilde{r}_1^2, \tilde{r}_2^1/\tilde{x}_2$
$\tilde{p}_1$
$\tilde{p}_2$
$\tilde{r}_1^1, \tilde{r}_2^2/\tilde{x}_1$

Figure 5.12: The supervised system V_o'/G_{det}, where $L_m(V_o'/G_{det}) = H'$.

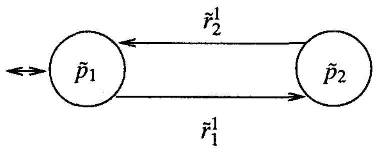


Figure 5.13: $\hat{H}' = L_m(\hat{V}_o'/G_{det})$.

The supervisor $\hat{V}_{o,ndet}'$ is represented by the automaton of Figure 5.14. The automaton has two states. A transition from the initial state is triggered when $\tilde{x}_1$ is observed, while a transition back to the initial state is triggered when $\tilde{x}_2$ is observed. Each state outputs the list of events that are enabled at that state. The supervised nondeterministic plant is shown in Figure 5.15.

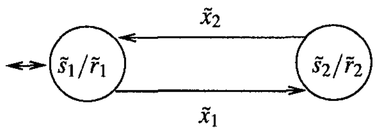


Figure 5.14: The nonblocking output supervisor $\hat{V}'_{o,ndet}$.

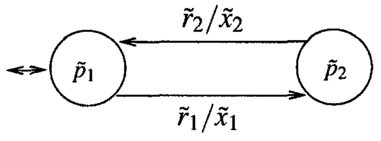


Figure 5.15: The nondeterministic system under supervision.

◇

It is worthwhile to note that in [2] a specification such as K of Figure 5.7 is called 'uncontrollable'. We believe this to be inconsistent with the standard definition of controllability in supervisory control theory: as all events are controller commands and thus controllable, every specification is trivially controllable according to the classical definition of controllability in [34]. As illustrated in the above example, the problem is that a supervisor designed to implement K causes the DES plant of Figure 5.6 to block in state $\tilde{p}_3$ by disabling both $\tilde{r}_1$ and $\tilde{r}_2$. The issue of blocking was not addressed in [2].

Chapter 6

Conclusion

In this thesis, the problem of supervisory control of discrete-event systems with output and its application to hybrid systems have been extensively studied. In order to introduce the discrete-event systems with outputs, a causal output map is used to correspond sequences of inputs with sequences of outputs. Then necessary and sufficient conditions are proposed for the existence of nonblocking input/output supervisory control such that the controlled system generates some desired specification language on outputs. An algorithm is proposed to extend the results of nonblocking input/output supervisory control theory when the prescribed specifications on outputs of the system are modeled in finite automaton framework. The result is applied to hybrid systems approximated by nondeterministic Mealy automata by requiring that in its every state a supervisor enables or disables all transitions carrying the same label. The idea of siblings is introduced to modify the nondeterministic DES plant model into a deterministic one such that the proposed theory and algorithm for discrete-event systems with outputs can be applicable to hybrid systems as well. The major contribution of this work has been the development of a theory for supervisory control of hybrid systems that is fully compatible with supervisory control theory of DES.

6.1 Future Research

In this section, we discuss the directions for future research.

- In this thesis, the problem of supervisory control of discrete-event systems with output is studied when such systems are represented as Mealy automata. It would be interesting to develop a similar algorithm for hybrid control systems modeled by petri nets which can be computationally more efficient for large concurrent systems.
- In this thesis, we focused on the problem of supervisory control of hybrid systems, considering the fact that the DES plant approximation is available. Although this DES plant model can be used to approximate the overall behavior of the continuous and interface parts of a hybrid control system, the simplification of the proposed hypersurfaces, mainly employed in the construction of the DES plant model, can result in imprecise modeling of complex hybrid systems. Thus, adapting our results with other approximations of continuous dynamics of hybrid systems seems to be useful.
- In section 5, we employed the idea of siblings in order to modify the nondeterministic

DES plant model of hybrid systems into the equivalent deterministic representations such that our proposed theory can be applicable to hybrid systems. It would be useful to enhance the proposed algorithm such that it deals with nondeterministic DES plant models of hybrid systems without any modifications.

- In addition, now that a link between supervisory control of hybrid systems and supervisory control of DES is established, further research is needed to import such concepts from supervisory control of DES as decentralized or hierarchal control, and to study the computational complexity of synthesizing supervisors for hybrid systems [39].

- Furthermore, in order to practice our results in the real world, it would be very interesting to apply our proposed nonblocking supervisory theorem for DES with output to a real-life application such that the applicability, efficiency, and usefulness of our results can be illustrated at length.

Bibliography

[1] A. Morse, "Supervisory control of families of linear set-point controllers—Part 1: Exact matching," *IEEE Transactions on Automatic Control*, vol. 41, pp. 1271–1281, 1996.

[2] X. Koutsoukos, P. Antsaklis, J. Stiver, and M. Lemmon, "Supervisory control of hybrid systems," *in Proceeding of the IEEE*, vol. 88, no. 7, pp. 1026–1049, 2000.

[3] P. Mahdavinezhad, P. Gohari, and A. G. Aghdam, "Supervisory control of discrete event systems with output: Application to hybrid systems," *in Proceedings of the 26th American Control Conference (ACC)*, pp. 4291–4296, 2007.

[4] P. Ramadge and W. Wonham, "The control of discrete event systems," *in Proceedings of the IEEE*, vol. 71, no. 1, pp. 81–98, 1989.

[5] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete-event systems," *SIAM Journal on Control and Optimization*, vol. 25, pp. 206–230, 1987.

[6] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, "Hybrid systems ii," *Lecture Notes in Computer Science*, vol. 999, pp. 982–992, 1995.

[7] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, "Hybrid systems iv," *Lecture Notes in Computer Science*, vol. 1273, pp. 1123–1159, 1997.

[8] P. Antsaklis and M. Lemmon, "Introduction to the special issue on hybrid systems," *Journal of Discrete Event Dynamic Systems: Theory and Applications, Special Issue on Hybrid Control Systems*, vol. 8, pp. 2–10, 1998.

[9] T. Moor and J. M. Davoren, "Robust controller synthesis for hybrid systems using modal logic," *Lecture Notes in Computer Science, Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control*, vol. 2034, pp. 433–446, 2001.

[10] R. Alur, T. Henzinger, and E. Sontag, "Hybrid systems iii, verification and control," *Lecture Notes in Computer Science*, vol. 1066, 1996.

[11] P. Tabuada, G. Pappas, and P. Lima, "Compositional abstractions of hybrid control systems," *in Proceeding of the 40th IEEE, Conference on Decision and Control*, pp. 352–357, 2001.

[12] J. Stiver and P. Antsaklis, "A novel discrete event system approach to modeling and analysis of hybrid control systems," *in proceedings of the 29th Annual Allerton Conference on Communication, control and computing, Univ. of Illinois at Urbana-Champaign.*

[13] P. Antsaklis and M. Lemmon, "Hybrid control systems: An introductory discussion to the special issue," *IEEE Transanctions on Automatic Control, Special Issue on Hybrid Control Systems*, vol. 43, no. 2, pp. 457–460, 1998.

[14] N. Lynch, R. Segala, and F. Vaandrager, "Hybrid i/o automata," *Information and Computation*, vol. 185, no. 53, pp. 105–157, 2003.

[15] T. Moor, J. M. Davoren, and A. Nerode, “Hybrid control loops, a/d maps, and dynamic specifications,” *Lecture Notes in Computer Science, in Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control*, vol. 2289, pp. 149–163, 2002.

[16] N. McClamroch, C. Rui, I. Kolmanovsky, and M. Reyhanoglu, “Hybrid closed loop systems: A nonlinear control perspective,” *in Proceedings of the 36th IEEE Conference on Decision and Control*, pp. 114–119, 1997.

[17] H. Ye, A. Michel, and L. Hou, “Stability theory for hybrid dynamical systems,” *IEEE Transactions on Automatic Control*, vol. 43, no. 4, pp. 461–474, 1998.

[18] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Oliveiro, J. Sifakis, , and S. Yovine, “The algorithmic analysis of hybrid systems,” *Theoretical and Computer Science*, vol. 138, pp. 3–34, 1995.

[19] R. Alur, T. Henzinger, G. Lafferriere, and G. Pappas, “Discrete abstrcation of hybrid systems,” *in Proceeding of the IEEE*, vol. 88, pp. 971–984, 2000.

[20] P. Tabuada and G. Pappas, “From discrete specification to hybrid control,” *in Proceeding of the 42nd IEEE, Conference on Decision and Control*, pp. 336–337, 2003.

[21] J. Davoren and A. Nerode, “Logics for hybrid systems,” *in Proceeding of the IEEE*, vol. 88, no. 7, pp. 985–1010, 2000.

[22] T. Henzinger, “The theory of hybrid automata,” *in Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pp. 278–292, 1996.

[23] P. Antsaklis, J. Stiver, and M. Lemmon, "Hybrid system modeling and autonomous control systems," *Lecture Notes in Computer Science*, vol. 736, pp. 366–392, 1993.

[24] X. Koutsoukos, K. He, M. Lemmon, and P. Antsaklis, "Timed petri nets in hybrid systems: Stability and supervisory control," *Journal of Discrete Event Dynamic Systems: Theory and Applications*, vol. 8, no. 2, pp. 137–173, 1998.

[25] A. Nerode and W. Kohn, "Models for hybrid systems: Automata, topologies, controllability, observability," *Lecture Notes in Computer Science*, vol. 736, pp. 317–356, 1993.

[26] R. Alur, C. Courcoubetis, T. Henzinger, and P.-H. Ho, "Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems," *Lecture Notes in Computer Science*, vol. 736, pp. 209–229, 1993.

[27] R. Alur and D. DILL, "The theory of timed automata," *Theoretical and Computer Science*, vol. 126, pp. 183–235, 1994.

[28] I. Demongodin and N. Koussoulas, "Differential petri nets: Representing continuous systems in a discrete event world," *IEEE Transactions on Automatic Control*, vol. 44, no. 3, pp. 573–579, 1998.

[29] J. Stiver, P. Antsaklis, and M. Lemmon, "An invariant based approach to the design of hybrid control systems," *IFAC, 13th triennial world congress, San Francisco, CA*, pp. 467–472, 1996.

[30] J. Stiver, P. Antsaklis, and M. Lemmon, "A logical DES approach to the design of hybrid control systems," *Mathematical Computer Modelling*, vol. 23, no. 11, pp. 55–76, 1996.

[31] D. Franke, T. Moor, and J. raisch, "Discrete supervisory control of switched linear systems," *Automatisierungstechnik, Special issue on Hybrid systems : Modeling and Control*, vol. 48, no. 9, pp. 460–469, 2000.

[32] J. Raisch and S. O'Young, "A totally ordered set of discrete abstractions for a given hybrid system," *Lecture Notes in Computer Science*, vol. 1273, pp. 342–360, 1997.

[33] J. Raisch and S. OYoung, "Discrete approximation and supervisory control of continuous systems," *IEEE Transaction on Automatic Control*, vol. 43, pp. 569–573, 1998.

[34] W. Wonham, "Supervisory control of discrete event systems." http://www.control.utoronto.ca/DES, July 2006.

[35] S. Park and J. Lim, "Robust and non-blocking supervisory control of nondeterministic discrete event systems using trajectory models," *IEEE Transaction on Automatic Control*, vol. 47, pp. 655–658, 2002.

[36] C. Zhou and R. Kumar, "Control of nondeterministic discrete event systems for bisimulation equivalence," *IEEE transaction on Automatic control*, vol. 51, no. 5, pp. 754–765, 2006.

[37] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya, "Supervisory control of discrete event processes with partial observation," *IEEE Transactions on Automatic Control*, vol. 33, no. 3, pp. 249–260, 1988.

[38] J. Hopcroft, R. Motwani, and J. Ullman, "Introduction to automata theory, languages, and computation," *Addition-Wesley*, 2007.

[39] P. Gohari and W. Wonham, "On the complexity of supervisory control design in the rw framework," *IEEE Transactions on Systems, Man and Cybernetics, Part B, Special Issue on DES*, vol. 30, no. 5, pp. 643–652, 2000.