This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
SUPERVISORY CONTROL
OF DISCRETE EVENT DYNAMICAL SYSTEMS
WITH PARTIAL OBSERVATIONS

by
ALIREZA HAJI-VALIZADEH

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

Thesis Advisor: Professor Kenneth A. Loparo

Department of Systems Engineering
CASE WESTERN RESERVE UNIVERSITY
January, 1995
CASE WESTERN RESERVE UNIVERSITY

GRADUATE STUDIES

We hereby approve the thesis of

Alireza Haji-Valizadeh

candidate for the Ph.D. degree.*

(signed)  
Kenneth A. Regan

(chair)  

date

*We also certify that written approval has been obtained for any proprietary material contained therein.
SUPERVISORY CONTROL
OF DISCRETE EVENT DYNAMICAL SYSTEMS
WITH PARTIAL OBSERVATIONS

Abstract

by

ALIREZA HAJI-VALIZADEH

In this work, we first define sufficient observation and work spaces for supervisors of a class of Discrete Event Dynamical Systems (DEDS) in the Ramadge-Wonham framework. A sufficient observation space is a collection of events whose observation by a supervisor is enough to realize a given desired behavior. A sufficient work space is a collection of events (sufficient observation space) for which the control action of the supervisor is limited to the controllable elements of the work space. We construct algorithms to evaluate a sufficient work (or observation) space with a rather small cardinality. Reduction of work (or observation) spaces results in reducing the event detectors, communication and command channels between the supervisor and the plant. The algorithms are based on testing a normality and an observability condition. We also develop two effective polynomial time algorithms to test the normality and the observability conditions for regular languages.

Next, we study the Supervisory Control and Observation Problem (SCOP)
language $K$, the SCOP has a solution if and only if $K$ is both observable and controllable. It is well known that the supremal observable and controllable sublanguage of $K$, i.e. the optimal solution for SCOP, does not exist if $K$ is not observable. By introduction of an extended normality property, we construct a suboptimal solution for the SCOP. We prove that our suboptimal solution contains, in the set theory sense, those reported in the literature and thus is an improved suboptimal solution for the SCOP.

Then, we introduce the concept of partially decentralized supervisory control of DEDSs when the control objectives are locally specified. We also study the problem of decentralized supervisory control of DEDSs when the control objective is given in terms of predicates defined on the state space. We introduce notions of statically and rationally decentralizable predicates. When a predicate is decentralizable, we construct decentralized supervisors that can realize the predicates that define the control objective.
To my late father
Hassan Haji-Valizadeh
Acknowledgements

I would like to express my deepest appreciation to my advisor Professor Kenneth A. Loparo for his continuous encouragement, guidance, technical advice and financial support. Without his help this work would not be possible. My special thanks also go to my co-advisor Professor Mark Buchner, whose excellent course in discrete event systems got me interested in the research area. I also thank him for helping me to get off-campus jobs on a number of occasions. I thank my thesis committee members Professor Howard J. Chizeck and Professor Charles Wells for their careful review of my thesis and invaluable suggestions. Special thanks also go to Mrs. Patty Matteucci and Mrs. Sharon K. Spinks, secretaries of the department, for their continuous help and special kindness that they have shown to me, day in and day out. I thank my dear wife and friend, Maryam Rezvani, for being patient with me and for putting up with my many late night stays in the department. I also thank the Department of Systems Engineering for awarding me a teaching assistantship. Finally, I thank the government of Iran whose many invaluable privileges for Iranian students studying abroad helped me to successfully complete my Ph.D. program.
# Table of Contents

Abstract .......................................................... ii

Dedication .......................................................... iv

Acknowledgements .................................................. v

**Chapter 1: Introduction** ........................................ 1

1.1 Dissertation overview ......................................... 1

1.2 Background .................................................... 5

1.2.1 DEDSs modeling and decidability issues .................. 5

1.2.2 Automata-based modeling of DEDSs and control objectives 9

1.2.3 Supervisor structure for DEDSs ............................ 14

1.2.4 Supervisory control of DEDSs ............................. 16

1.3 Literature survey .............................................. 16

**Part A) Supervisory control under partial observations**

**Chapter 2: Minimizing cardinality of events set for supervisors** 23

of discrete event dynamical systems

2.1 Introduction .................................................. 24

2.2 Observability and normality .................................. 28

2.3 Work spaces .................................................. 32

2.4 Observation spaces ........................................... 52

2.5 Example ...................................................... 62

2.6 Applications ................................................ 65

2.7 Conclusions ................................................ 67
control and observation problem

3.1 Introduction .............................................. 70
3.2 An improved suboptimal solution for SCOP ................ 76
3.3 An algorithm for the suboptimal solution ................. 80
3.4 Example .................................................. 85
3.5 Conclusions .............................................. 87

Chapter 4: Testing the normality and observability conditions

4.1 Introduction .............................................. 88
4.2 An algorithm to test the normality condition ............ 103
4.3 An algorithm to test the observability condition ......... 105
4.4 Conclusion ............................................... 113

Part B) Decentralized supervisory control

Chapter 5: Partially decentralized supervisory control of discrete event systems

5.1 Introduction .............................................. 115
5.2 Partially decentralized supervisory control ............... 119
5.3 Example .................................................. 121
5.4 Conclusions .............................................. 126

Chapter 6: Decentralized supervisory predicate control of discrete event systems

6.1 Introduction .............................................. 127
6.2 Predicates and predicate transformers in DEDSs ........ 129
Chapter 7: Summary

7.1 Contributions of this dissertation ............... 156
7.2 Limitations of supervisory control ............... 158
7.3 Future Work ......................................... 160

References ............................................. 161
1.1. Dissertation overview

In this dissertation we study supervisory control of discrete event dynamical systems (DEDSs). A DEDS is a system that evolves with occurrence of some physical events. We assume that physical events occur abruptly, non-deterministically and at possibly irregular points of time.

Many real world systems can be modeled as DEDSs at some level of abstraction. Manufacturing systems, communication networks, vehicular traffic systems and computer networks are a few examples. A few examples for the events in these systems are, respectively, completion of a task, receipt of a message, change of a traffic signal and a request for a computer resource. Occurrence of the events changes the state of the system. Thus a trace of the events that are executed in the system characterizes the logical behavior of the system.

A supervisor is a controller that observes the behavior of the system and regulates some controllable attributes of the system such that the closed loop system behaves as desired. The supervisor may observe states of the system or the events that are executed in the system. Here, we assume that the controllable attribute of the system is a subset of the system's events. That is
The goals of supervision that we are interested in are of a qualitative nature. For example, the goal may be enforcing a safety measure in a vehicular traffic system or enforcing a specific ordering among the events, for example assigning shared computer resources to active processes in an operating system.

This dissertation includes two major parts. In part A we study the supervisory control of DEDSs, where the system and the objectives of supervision are given in terms of some formal regular languages that are defined over the set of events. In Chapter 2, we address the problem of minimizing the cardinality of the event sets of the supervisors. To implement a supervisor, one must devise event sensors and actuators in the DEDS. Given models of the system and the objectives, we determine what events must be detected and what controllable events must be regulated by the supervisor in order to realize the supervision goals. We develop a few algorithms for evaluating the set of necessary or sufficient sensor and actuator events. We also discuss a supervisor structure based on the information and control structure provided by the detectors and the actuators, respectively. Minimizing the number of sensors and actuators is an appealing engineering objective. In addition to this advantage, if applied, our study yields a new and more robust supervisor for cases where all events are detected. We show that supervisors based on a reduced set of detectors are less sensitive to failures in the detection system.

Our results in Chapter 2 can be interpreted in the context of the supervisory control and observation problem (SCOP). SCOP addresses supervisory control of DEDSs where some events are not detected. One may address such a problem from two different points of view. One possible approach is to enhance the information obtained from the system. Given a supervision objective, our
order that the goal becomes realizable. Of course, we pursue the objective of having the smallest number of additional detectors required to achieve the supervisory goal.

In Chapter 3, we pursue the second, and more traditional, approach to the study of the SCOP. Given an objective that is not realizable using available information (or observations of the system), the traditional approach is to evaluate a subobjective that is realizable based on the partial observation of the system, and then to design a supervisor that realizes this subobjective. Obviously, it is desirable to have the evaluated subobjective as close as possible to the original objective. In Chapter 3, we give algorithms that improve existing solutions to the SCOP in this approach.

In Chapter 4, we provide algorithms for testing a normality and an observability condition for a given regular language that represents the supervision objective. Normality and observability are two algebraic conditions of formal languages that we extensively use in Chapters 2 and 3.

In part B of this dissertation we study the problem of decentralized supervisory control. Many discrete event dynamical systems are spatially distributed systems. Thus implementation of a centralized supervisor may require a large amount of communication and command links between the DEDS and the supervisor. To remedy this difficulty, the problem of decentralized supervisory control has been addressed in the control literature. A decentralized supervisor consists of a number of individual local supervisors such that each local supervisor receives information from nearby sensors and commands nearby actuators. In the control literature, an algebraic condition, called normality, that an objective must satisfy in order to implement a supervisor in a decentralized
ity condition is not satisfied by the objective given the original assignment of sensors and actuators (work spaces) to different local supervisors. Instead of implementing a centralized supervisor, our approach is to construct a partially decentralized supervisor. Given a primary set of work spaces, that are collections of sensors and actuators (presumably, in close proximity to each other), we determine additional information and control that a local supervisor needs in order to achieve its assigned or local objective.

Before this work, decentralized supervisory control of DEDSs have been formulated and addressed only in the context of formal languages. In Chapter 6, we formulate and address the same problem in the context of state predicates. In this chapter we assume that the objectives are given in terms of conditions on the state space and each of the local supervisors partially observes the state of the system and controls a subset of controllable events.

Finally in Chapter 7, we conclude this dissertation by summarizing the contributions of this work and discussing future work. A list of references used in this work is included in the final section of this dissertation.

Each chapter of this dissertation is written to be self contained. We believe that this should be beneficial to readers interested in only selected material from the dissertation. Each chapter begins with an introduction and a mathematical preliminary section. For each chapter, we have also included a conclusion section that discusses contributions of the chapter. Each chapter also contains an abstract like subsection that relates the chapter to the subjects discussed up to that point.

In the rest of this introductory chapter we provide a general background for
1.2. Background

1.2.1. DEDSs modeling and decidability issues

A Discrete Event Dynamical System (DEDS) is a dynamic system that evolves in accordance with the occurrence of some physical events. We assume that the physical events occur abruptly at unknown irregular instants of time. Generally, state space of a DEDS is a discrete space. Thus a typical state trajectory of the system is a piecewise constant function of time. Fig. (1.2.1) depicts a typical state trajectory for a DEDS.

![State trajectory of a DEDS](image)

Fig. (1.2.1): State trajectory of a DEDS.

The state transitions of a DEDS are called events. As in Fig. (1.2.1), state transitions may be labeled by the elements of some alphabet set.
A few examples are manufacturing systems, communication networks, vehicular traffic systems and concurrent computer programs. Events in these systems, respectively, could be completion of a task, receiving a message, change of a traffic signal and a request for a computer resource by a process of the program.

Problems related to DEDSs could have quantitative as well as qualitative aspects. In a queuing network, as a quantitative measure, it is of interest to evaluate and possibly control the average number of customers in a queue. On the other hand, in a concurrent program consisting of a few processes that share resources, as a qualitative property for the program, it is of interest to guarantee deadlock free computations. In the past, DEDSs have usually been sufficiently simple that ad hoc solutions to various problems have been adequate. Nowadays, the increasing complexity of man-made DEDSs requires a comprehensive theoretical basis for analyzing and controlling DEDSs.

Different tools have been used for analyzing problems of interest in the context of DEDSs [10]. For quantitative analysis of DEDSs, among other techniques, researchers have used min-max algebras [20, 21], perturbation analysis [9], generalized Markov processes [27] and simulation techniques for discrete event systems [46]. On the other hand, automata [75, 80], Petri nets [32, 33, 38], temporal logic [64, 65, 70, 71] and recursive communicating processes [29, 36, 37], are a few tools that are used for qualitative analysis of DEDSs. In this work we are concerned with qualitative analysis and control of DEDSs.

Methods for qualitative analysis of the behavior of a DEDS can be classified into two basic categories. In one category, the state of the system is
the system can be represented by the conditions that all state trajectories satisfy. Preventing a DEDS from entering a forbidden region in the state space could be a control goal in this approach. In the other category, the events of the system are considered the fundamental components of the DEDS. Here, the set of all sequences of the events that can happen in the DEDS captures the behavior of the system. Coordination to ensure orderly flow of the events is a typical control objective in this approach.

As we mentioned before, in a DEDS, state transitions are labeled by the elements of an alphabet. Thus a sequence of the events, i.e. an event trajectory, models an instance of the DEDS evolution. Such an event trajectory corresponds to a set of state trajectories. If the DEDS is deterministic then each event trajectory corresponds to a unique state trajectory. A measure of the expressive power of a modeling technique, could be the richness of the event trajectories that the set of all models can capture.

Formally, let $\Sigma$ represent a finite set of events and $\Sigma^*$ stand for the set of all finite length sequences of the events. A subset $L \subseteq \Sigma^*$ is called a formal language over the alphabet $\Sigma$. A particular DEDS can be algebraically represented by a formal language $L$, where $L$ contains all event sequences that can happen in the DEDS. Given a modeling methodology for DEDSs, one can define a language that is generated by a typical model. Then the class of languages that all models can generate is a measure of the modeling power of the methodology. Among the modeling techniques that we mentioned here, finite automata based techniques have the weakest modeling power. The set of all finite automata generate (or recognize) the set of regular languages [34]. At the other end of the spectrum, the modeling techniques based on commu-
all models in this modeling framework generate languages of Turing machines [18]. The modeling power of Petri nets resides somewhere between these two extremes [26, 45, 69]. Thus communicating sequential processes can model more DEDSs than Petri nets or finite automatons. Likewise, Petri nets can model more DEDSs than finite automatons.

As is the case in all engineering problems, there is a trade off involved in selecting a particular modeling technique. As the modeling power of a particular technique increases, more problems posed for the models of that kind become undecidable. To be more specific, consider the problem of deciding if a given event trajectory belongs to the set of all event trajectories of a given model. This problem is directly related to the controllability problem in DEDSs that we will introduce later. It is known that this problem is undecidable in the framework of communicating sequential processes and Petri nets. However, the same problem is decidable in the context of finite automatons. There are also some state reachability problems that are decidable for finite automatons and Petri nets but not for communicating sequential processes. The decidability issue is very important from a control theory perspective. The objective of supervisory control is not just designing a controller for a particular DEDS, but it is to automate the procedure of constructing a controller for any given DEDS. Thus the decidability problem is of crucial importance.

Undecidability of a problem in a modeling framework does not mean all is lost. Even though we cannot devise a technique that is applicable for all models within the framework, we may still use simulation techniques for studying particular systems. In special cases, certain attributes specific to the particular system may enable us to find a solution to the problem at hand.
In this work we mainly consider models based on finite automata. In order to survey different supervisory control problems that are considered in the control literature and introduce the objectives of this work, we now introduce languages that are generated or recognized by finite automata.

A finite automata is a five tuple $\mathbb{A} = (Q, \Sigma, \delta, q_0, Q_m)$. $Q$ is a set of states, $\Sigma$ is a finite set of events, $\delta : \Sigma \times Q \rightarrow Q$ is a partial transition function, $q_0$ is the initial state and $Q_m$ is the set of final states. Let $\Sigma^*$ denote the set of finite length strings (or words) of events in $\Sigma$. The null string is a string with length equal zero and is denoted by $\epsilon$. The transition function $\delta$ is extended to the domain $\Sigma^* \times Q$ as follows and is denoted by $\hat{\delta}$.

$$\begin{align*}
\hat{\delta}(\epsilon, q) &:= q; \\
\hat{\delta}(s\sigma, q) &:= \delta(s, \hat{\delta}(s, q)), \quad \text{if } \hat{\delta}(s, q) \text{ is defined}; \\
\text{Undefined} &\quad \text{otherwise.}
\end{align*}$$

Here, $q \in Q$, $s \in \Sigma$ and $s \in \Sigma^*$. From now on we use $\delta$ to denote both the original and extended transition functions. We also use the notation $\delta(s, q)!$ to denote the fact that the transition from state $q$ under string $s$ is defined. An string of events $t$ is said to be generated or recognized by $\mathbb{A}$ if $\delta(t, q_0)!$ and $\delta(t, q_0) \in Q_m$. We define two languages for an automaton $\mathbb{A}$ as follows.

$$L(\mathbb{A}) := \{ t \in \Sigma^* \mid \delta(t, q_0)! \},$$

$$L_m(\mathbb{A}) := \{ t \in L(\mathbb{A}) \mid \delta(t, q_0) \in Q_m \}.$$

As an example of an automaton based model of a discrete event dynamical system consider the following model that represents a material processing piece of machinery.
Here $Q := \{i, w, d\}$, $q_0 = i$, and $\Sigma = \{\alpha, \beta, \gamma, \eta\}$ which stand for
i: idle state,
w: working state,
d: down state,
$\alpha$: start processing,
$\beta$: machine completes processing,
$\gamma$: a failure occurs,
$\eta$: the machine is repaired.

For this model we have

$$L(G) = (\alpha(\beta + \gamma\eta))^*,$$

$$L_m(G) = (\alpha(\beta + \gamma\eta))^*.$$

In the above characterization of $L(G)$ and $L_m(G)$ we have used concatenation, addition, "*" and prefix closure operators of strings and languages. For completeness these are defined next.

For two strings $s_1$, $s_2$ in $\Sigma^*$, $s_1s_2 \in \Sigma^*$ is a new string that is obtained by concatenating $s_1$ and $s_2$. $s_1 + s_2$ is the language \(\{s_1, s_2\}\). Given a language $L$,
a finite number of concatenation operations on the words of $L$. $L^*$ also includes the null string. Let $s_1s_2 := t$, where $s_1, s_2, t \in \Sigma^*$. Then $s_1$ is a prefix of $t$ and $s_2$ is a suffix of $t$. $\bar{t}$ denotes the language of all prefixes of $t$, including the null string and $\bar{t}$ is called the prefix closure of $t$. For a language $L$, $\bar{L}$ is the union of prefix closures of words in $L$. Thus for the previous languages we have

$$L(G) = \{\varepsilon, a, a\beta, a\gamma, a\gamma\alpha, a\beta\alpha, a\beta\alpha\gamma, \cdots\}.$$  

$$L_m(G) = \{\varepsilon, a, a\beta, a\gamma\alpha, a\beta\alpha\gamma\eta, \cdots\}.$$  

$L_m(G)$ denotes the sequences of events that represent completed tasks. $L(G)$, on the other hand, denotes sequences of events that may happen in the DEDS. $G$ is called trim if $L_m(G) = L(G)$. If $G$ is trim, then every sequence of events can be extended to a complete task.

To introduce a control structure for DEDSs we distinguish two types of events, called controllable and uncontrollable and denote them by $\Sigma_c$, and $\Sigma_u$ respectively. In the previous example, we may prevent execution of $\alpha$ (i.e. do not start) or $\eta$ (i.e. do not repair). We call these events controllable. On the other hand, if there is no preventive maintenance on the machine, $\gamma$ (machine is failed) is unpreventable, thus $\gamma \in \Sigma_u$. In the above example we have

$$\Sigma_c = \{\alpha, \eta\} \quad \Sigma_u = \{\beta, \gamma\}.$$  

Ramadge and Wonham [80] were the first to introduce the above control structure for DEDSs. Not all DEDSs can be decomposed according to this control structure. However, there are many DEDSs that yield to such a characterization.
The DEDS $G$ consists of two machines that operate concurrently. Here we have

$$L(G) = L(G_1) || L(G_2),$$

$$L_m(G) = L_m(G_1) || L_m(G_2),$$

where "||" represents the shuffle product operator, which models concurrency by interleaving. To formally define the shuffle product we need to introduce the projection operation. Let $\Sigma_1$ and $\Sigma_2$ be two events set such that $\Sigma_2 \subseteq \Sigma_1$.

Define the projection operator $P_{\Sigma_1, \Sigma_2} : \Sigma_1^* \rightarrow \Sigma_2^*$ as follows,

$$P_{\Sigma_1, \Sigma_2}(\epsilon) = \epsilon,$$

for $s \sigma \in \Sigma_1^*$

$$P_{\Sigma_1, \Sigma_2}(s \sigma) = \begin{cases} P_{\Sigma_1, \Sigma_2}(s), & \text{if } \sigma \notin \Sigma_2; \\ (P_{\Sigma_1, \Sigma_2}(s))\sigma, & \text{if } \sigma \in \Sigma_2. \end{cases}$$

$P_{\Sigma_1, \Sigma_2}$ deletes from the argument all events that belong to $\Sigma_1$ but not to $\Sigma_2$, i.e. events in $\Sigma_1 \setminus \Sigma_2$. Now, let $\Sigma_1$ and $\Sigma_2$ denote the event sets of $G_1$ and $G_2$ respectively, and define $\Sigma := \Sigma_1 \cup \Sigma_2$. $\Sigma$ is the event set of $G$. We have

$$L(G_1)||L(G_2) = \{u \in \Sigma^* | P_{\Sigma, \Sigma_1}(u) \in L(G_1) \text{ and } P_{\Sigma, \Sigma_2}(u) \in L(G_2)\}.$$
Let $G := (Q^\parallel, \Sigma^\parallel, \delta^\parallel, q_0^\parallel, Q_m^\parallel)$ represent the automaton depicted in Fig. (1.2.4).

Two typical control objectives are listed below:

1) **Objective 1**) Only one machine must be working at any time, i.e. $(w_1, w_2)$ is a forbidden state in $G$. This is an example of a state-based objective.

2) **Objective 2**) When both machines are down, higher priority must be given to repairing machine 1. This is an example for an event-based objective. The desired language that captures this objective can be characterized as $K := L \cap L_m(G)$, where

$$L = \{s \in L(G) \mid P_{\Sigma, \{\eta_1, \eta_2, \eta_3\}}(s) \in \overline{\{\eta_1 \eta_1, \eta_2 \eta_2, \eta_1 \eta_2 \eta_1 \eta_2, \eta_2 \eta_1 \eta_1 \eta_2\}}\}.$$
A Supervisor is a dynamic controller that observes the output of the controlled DEDS and after each observation disables some controllable events in order to realize the control objectives. What constitutes the output of the DEDS depends on the sensors used in the system. This, in turn, determines whether we must use state-based or event-based characterization of the control objectives. Fig. (1.2.5) depicts the feedback structure that is used in controlling DEDSs.

![Diagram of System Model and Supervisor](image)

Fig. (1.2.5). Feedback structure used in supervisory control.

In order to realize objective 1 in the previous example, we can construct a supervisor whose observation is the current state of the system as follows,

\[
f : Q^\parallel \rightarrow 2^\Sigma \\
f(q) := \begin{cases} 
\Sigma \setminus \{\alpha_1\}, & \text{if } q = (i_1, w_1); \\
\Sigma \setminus \{\alpha_2\}, & \text{if } q = (w_1, i_1); \\
\Sigma, & \text{Otherwise.}
\end{cases}
\]

Supervisor "f" prevents events \(\alpha_1\) and \(\alpha_2\) at states \((i_1, w_1)\) and \((w_1, i_1)\) respectively, Fig. (1.2.4). Notice that \(\alpha_1\) and \(\alpha_2\) are controllable and thus
Now consider objective 2. For this language based objective we use a different supervisor structure. Here, our supervisor observes traces of the events executed in $\mathcal{G}$. The supervisor is a pair $S = (S, \varphi)$, where $S = (X, \Sigma, \xi, x_0, X_m)$ is a finite automaton that plays the role of an observer and $\varphi : X \rightarrow 2^\Sigma$ is a feedback map that determines what events must be disabled. $S$ is initialized at the state $x_0$. After $S$ observes that an event has occurred in $\mathcal{G}$, the state in $S$ is changed. At the new state, $\varphi$ determines the events that must be disabled. Fig. (1.2.6) depicts the automaton $S$. The feedback map is defined as follows,

$$
\varphi : X \rightarrow 2^\Sigma, \quad \varphi(x) = \begin{cases} 
\Sigma \setminus \{\eta_2\}, & \text{if } z = x_8; \\
\Sigma, & \text{otherwise.}
\end{cases}
$$

![Diagram](image.png)

Fig. (1.2.6). Automaton $S$.

The languages generated by the closed loop system are denoted by $L(S/\mathcal{G})$ and $L_m(S/\mathcal{G})$. It can be readily verified that these languages are equal to $L$
It is important to notice that the supervisor does not force the execution of an event in $G$. From the supervisor's perspective, events in $G$ are simply executed by some means and the supervisor can only inhibit a controllable event from potentially being the next event.

1.2.4. Supervisory control of DEDSs

Supervisory control is a technical discipline that aims to automate the construction of a supervisor for a given model of a DEDS and a given control objective.

1.3. Literature survey

In this subsection we survey a number of supervisory control problems that have been addressed in the control literature

Controllability of an objective in DEDSs: Given a DEDS and an objective, the objective is said to be controllable if there exists a supervisor that realizes the objective. Notice that in the context of DEDSs, controllability is a property of the objective rather the DEDS. Thus depending on the type of objective, different criterions for checking controllability have been developed. [80] defines language controllability. Predicate controllability is defined in [42, 48]. [24, 41, 51, 75] characterize controllability in the context of their formulation of DEDSs and their objectives.

Supremal controllable sub-objective for a DEDS: Suppose that the desired behavior of the closed loop system is given in terms of a language. If the de-
the language that is controllable. This subset is called the supremal controllable sublanguage. [99] gives algorithms to construct the supremal controllable sublanguage of a language. Likewise, in the context of state-based characterization of the objective, [42] presents algorithms for constructing the weakest controllable predicate that is stronger than a given predicate.

*Observability of an objective in DEDSs:* In many DEDSs we only have partial observation of the behavior of the DEDS. In the language-based formulation, there may be some events that are not detected. In state-based formulation, it may be the case that only a partial observation of the state is available. An objective is said to be observable if there exists a supervisor that realizes the objective based on the available information. [17, 58] discusses observability of languages. [74] considers observability of state-based objectives where it is assumed that events are the output of the system. In [42, 48] observability of predicates is studied.

*State observability in DEDSs:* [78] considers the problem of estimating the current state of the DEDSs based on partial observations of the events that the system executes.

*Invertibility of DEDSs:* [72] studies the problem of reconstructing the trace of events using a partial observation of the events that are generated by the system.

*Modular supervisory control of DEDSs:* In many cases the desired language is given as the intersection of a number of sublanguages. Each sublanguage characterizes a specific goal. Modular supervisory control is concerned with designing a number of supervisors each of which realizes one of the sublan-
of the supervisors. Modular supervisors are easily changed to cope with newly added objectives. Also, construction of modular supervisors is computationally simpler. [100] considers the theoretical issues related to modular supervisory control.

Decentralized supervisory control of DEDSs: Many DEDSs are spatially distributed system. Thus implementation of a centralized supervisor requires numerous communication and command links between the system and the supervisor. Decentralized supervisory control aims to remedy this problem. Here, it is desirable to have a number of supervisors, distributed over the system, such that each supervisor has access to the information provided by the sensors and commands the nearby actuators that are near to the supervisor. Decentralized supervisory control addresses this problem by developing conditions that an objective must satisfy in order to be decentralizable. [56, 59, 60, 61, 68, 84] address a number of technical issues related to the decentralized supervisory control where the DEDS and the objectives are characterized by formal languages. In Chapter 6 of this dissertation we address this problem when the system and the objectives are state-based.

Hierarchical supervisory control of DEDSs: In the problems that we have introduced up to now, a supervisor is assumed to have low-level information about the system behavior. For example, a supervisor observes an event generated by the DEDS. Even though the supervisor may only have a partial observation, it is assumed that the observation is an event or a system state. In many DEDSs, due to the complexity of the system, it is preferred to communicate with the supervisor with higher level information. Consider a supervisor that receives information about the completion of a task and determines the
represent sequence of events. Hierarchical supervisory control of DEDSs studies objectives that can be realized in a hierarchical structure. [103] studies the problem in the context of formal languages. A similar problem in the context of state-based formulation of supervisory control has been addressed in [73].

Supervisory control of concurrent DEDSs: A basic assumption that we have made in our formulation of supervisory control is that the events are executed one at a time. There may be cases that this assumption is violated. Concurrent DEDSs are systems that may execute a number of events at the same time. Occurrence of simultaneous events considerably complicates the supervision of DEDSs. [50, 95] address technical issues related to the supervision of concurrent DEDSs.

Optimal supervisory control of DEDSs: A largely accepted measure for optimality of a supervisor is the dimensionality of the closed loop language that the supervisor realizes. Consider the supervisory control problem that requires the design of a supervisor such that the closed loop language contains a minimum desired behavior \( L_d \) and is contained in a maximum allowable behavior \( \mathcal{L}_a \supseteq L_a \). There are many languages \( \mathcal{K} \) that satisfy the condition \( L_a \subseteq \mathcal{K} \subseteq \mathcal{L}_a \). Thus there are many solutions to this type of supervisory control problem. Among two possible supervisors, we say one is more optimal (or preferred) if the language that it realizes contains the language of the second supervisor. [54] shows that accepting such an optimality measure can be interpreted as accepting closed loop throughput (speed of generating events) as the optimality measure. In this context, optimal supervisory control pertains to designing a supervisor that realizes the supremal controllable sublanguage of \( \mathcal{L}_a \) that contains \( L_a \). [76, 86] studies optimal supervisory control from a
A supervisor is designed (based on heuristic search procedures) such that the generated sequence of events attain minimum cost of execution.

Robust and adaptive supervisory control of DEDSs: If there are uncertainties in a DEDS, then a single automaton model may not be adequate to capture the behavior of the system. In [55], it is assumed that the model of the system belongs to a set of models. The objective of robust supervision is to design a supervisor to achieve the desired behavior for any model belonging to the set. [55] develops a necessary and sufficient condition for the existence of such a robust supervisor. Adaptive supervisory control, in the above context, has the objective of reducing uncertainties about the system as supervision progresses. In [55], this is achieved by eliminating a model from the set of possible models after observing an event, if the sequence of events observed up to that point could not be generated by the model. After such an elimination, the supervisor's structure is changed to the robust supervisor based on the new reduced set of possible models of the DEDS.

Supervisor reduction: Implementation of a supervisor has two aspects. Firstly, one must devise sufficient sensors and actuators for the DEDS. Secondly, a module that processes the information provided by the sensors and commands actuators must be implemented. Thus supervisor reduction can be pursued in two directions. One may attempt to reduce the memory requirements of the supervisor module. This supervisor reduction is addressed in [97]. A second type of supervisor reduction pertains to reducing the number of sensors and actuators in the DEDS. This supervisor reduction is one of the main topics of this dissertation and will be discussed in detail later.

Complexity of decision problems arising in supervisory control of DEDSs:
A few examples are determining if a language is controllable, or observable. Complexity of these decision problems are measured in terms of the cardinalities of state spaces of automata representing the system and the objective as well as the cardinality of the events set. Except a few basic decision problems, most of the problems in logical DEDSs are NP complete problems. [82, 91] study computational complexity of a number of problems which arise in supervisory control.

*Formal languages arising in supervisory control of DEDSs:* In [14, 15, 40, 43, 83, 99], there are numerous algorithms for evaluating languages that arise in different problems of supervisory control. Some examples of these languages are supremal controllable sublanguage of a given language, maximal observable sublanguage of a given language, infimal superlanguage of a language and supremal normal sublanguage of a language.

*Real time supervisory control of DEDSs:* The initial formulation of supervisory control lacked the notion of time. [5, 39, 49, 70, 71, 85] are a few works that have tried to incorporate timing issues into the supervisory control of DEDSs. This area of research is still in its initial stages. Many problems that are already solved for logical DEDSs are yet to be addressed for timed DEDSs.

*Alternative modeling and formulation of supervisory control of DEDSs:* Many researchers have introduced alternative modeling and problem formulation frameworks for supervisory control of DEDSs. We discussed a number of these alternatives in the previous section. A few additional alternatives are as follows. [24] introduces a framework that allows for the existence of forced events. Here, a supervisor is able to force some events in the system. This is an
controllable events. [79] formulates the supervisory control for Buchi automata.
An input–output interpretation of supervisory control is discussed in [1, 2].
Here, controllable events are assumed to be commands to the system and un-
controllable events are considered responses from the system. An application
of the input–output modeling and supervisory control is also given in [2].

A list of literature in supervisory control of discrete event systems is
provided in the References section. With the background provided in this
chapter, we next present the main results of this dissertation.
Minimizing the Cardinality of an Events Set for Supervisors of Discrete Event Dynamical Systems.

In this chapter we study sufficient observation and work spaces for supervisors of a class of Discrete Event Dynamical Systems (DEDS). We use finite state automata to model a DEDS. The finite automata generates a formal language defined over the set of events in the DEDS. A supervisor is a feedback system that observes a generated trace of events and dynamically disables (or enables) a subset of controllable events such that the closed loop system behaves as desired. We model the desired behavior of the DEDS by a sublanguage defined over the set of events. A sufficient observation space is a collection of events whose observation by a supervisor is enough to realize a given desired behavior. A sufficient work space is a sufficient observation space such that the control action of the supervisor is limited to only the controllable elements of the work space.

In many cases, there exists a sufficient work (or observation) space that is smaller than the set of all events. In this chapter, we construct algorithms to evaluate a sufficient work (or observation) space.

The reduction of work (or observation) spaces results in reducing the
the supervisor and the plant.

2.1. Introduction

Discrete Event Dynamical Systems (DEDS) have received considerable attention in the control literature recently. We use Ramadge and Wonham's [80, 99] framework for modeling and supervisory control of DEDS. In this framework, a DEDS is modeled by a generator $G := (Q, \Sigma, \delta, q_0, Q_m)$ where $Q$ is the state set. $\Sigma$ is a finite alphabet or set of event symbols, $\delta : \Sigma \times Q \rightarrow Q$ is a partial transition function, $q_0$ is the initial state and $Q_m$ is the set of final states which can represent, for example, the completion of some tasks in the plant. While under no external control, a generator starts at the initial state and generates events in the course of transiting from one state to the other governed by the transition function. The behavior of a generator can be described by the set of all event sequences that it can generate and the set of all event sequences that transfer the initial state of the plant to a final, or marked, state.

Formally, let $\Sigma^*$ stand for the set of finite length strings of the events in $\Sigma$, including the null string denoted by $\epsilon$. The extension of the transition function to $\Sigma^* \times Q$, denoted also by $\delta$, is defined as follows:

$$
\delta(\epsilon, q) = q, \quad \delta(\sigma \sigma', q) = \delta(\sigma, \delta(s, q)) \quad \text{for} \quad \sigma \in \Sigma, s \in \Sigma^* \text{ and } q \in Q.
$$

We use the notation $\delta(s, q)!$ to denote that the transition from state $q$ under the string $s$ is defined. Then the open loop behavior of the plant is modeled by the following formal languages.

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

$$
L_m(G) = \{ s \in L(G) \mid \delta(s, q_0) \in Q_m \}.
$$
When a plant is trim every sequence of events in \( L(G) \) can be completed to a finished task, i.e. a string in \( L_m(G) \). Here, for a given language \( L \), \( \hat{L} \) stands for the prefix closure of \( L \), i.e. \( \hat{L} \) contains all prefixes of all words in \( L \). For a word \( s \in \Sigma^* \), \( \bar{s} \) denotes \( \{s\} \). Also, for a given language \( L \subseteq \Sigma^* \) we denote the set of all events in \( L \) by \( \mathcal{E}(L) \). That is:

\[
\mathcal{E}(L) := \bigcup_{s \in L} \{ \sigma \in \Sigma \mid \exists t_1 \in \bar{s} \text{ such that } t_1 \sigma \in \bar{s} \}.
\]

For a string \( t \), \( \mathcal{E}(t) \) denotes the events of the language \( \{t\} \). Obviously, for \( L_1 \subseteq L_2 \subseteq \Sigma^* \), we have \( \mathcal{E}(L_1) \subseteq \mathcal{E}(L_2) \subseteq \Sigma \).

The set \( \Sigma \) is partitioned into two sets \( \Sigma_c \) and \( \Sigma_u \). \( \Sigma_c \) is the set of controllable events whose occurrences could be prevented by a supervisor. \( \Sigma_u \) denotes the set of uncontrollable events. In this thesis, the desired behavior for a DEDS is characterized by a sublanguage \( K \subseteq L_m(G) \).

A supervisor \( S = (S, \varphi) \) consists of a finite automata

\[
S := (X, \Sigma, \xi, x_0, X_m)
\]

and a feedback map \( \varphi : X \rightarrow 2^\Sigma \) where \( 2^\Sigma \) denotes the power set of \( \Sigma \). When a generator is coupled with a supervisor, the supervisor receives any event generated by \( G \). This executes a state transition in the supervisor automata \( S \). In the new state of the supervisor, \( \varphi \) determines a new subset of events. This subset determines all possible next events that the generator can generate while complying with the transition function \( \delta \). In this way the supervisor \( S \) controls the behavior of \( G \). Since elements of \( \Sigma_u \) cannot be prevented from happening, we require \( \Sigma_u \subseteq \varphi(x) \) for all \( x \in X \). A supervisor is called proper if it can respond to any string that the closed loop system can generate.
By introducing the notion of language-controllability, Ramadge and Wonham have developed a necessary and sufficient condition for the existence of a supervisor [80]. This condition requires $K$ to be controllable. $K$ is controllable if

$$\bar{K} \Sigma_u \cap L(G) \subseteq \bar{K}. \quad (2.1.1)$$

When condition (1.1) holds, one approach is to design a supervisor such that $L_m(S) = K$ and $\varphi(x) = \{\sigma \in \Sigma \mid \xi(\sigma, x)\}$. Then we have $L_m(S/G) = K$ and $L(S/G) = \bar{K}$.

Supervisor $S$ can be reduced or simplified in two ways. In one way, the objective is to reduce the cardinality of $X$ denoted by $|X|$. Notice that $|X|$ is a measure for the amount of memory needed in implementing $S$. This supervisor reduction is addressed in [97]. In the other way, the objective is to reduce the events set of the supervisor. In other words, the objective is to construct a supervisor $\hat{S} = (\hat{S}, \hat{\varphi})$ where $\hat{S} := (\hat{X}, \hat{\Sigma}, \hat{\xi}, \hat{x}_0, \hat{x}_m)$ and $\hat{\varphi} : \hat{X} \to 2^\Sigma$ such that $|\hat{\Sigma}| \leq |\Sigma|$ and for all $\hat{x} \in \hat{X}$ we have $(\Sigma_u \cup (\Sigma \setminus \hat{\Sigma})) \subseteq \varphi(\hat{x})$. We also require that $K$ to be the closed loop behavior. Notice that the supervisor $\hat{S}$ does not need to observe the events in $\Sigma \setminus \hat{\Sigma}$ nor does it need to disable events in $\Sigma_c \cap (\Sigma \setminus \hat{\Sigma})$. This in turn implies that there is no need for any detecting systems for, command channels to and communication channels from these events. However, implementation of this type of reduced supervisor may require more memory. That is $|\hat{X}| \geq |X|$. This type of supervisor reduction was first addressed in [13]. In this chapter, we address this type of supervisor reduction using a more concise approach. We will also compare our approach with that of [13] later in the chapter.
attempt to find sufficient information in order that the supervisor can achieve the desired closed loop behavior. Thus the results could be used in the context of supervision under partial observation.

The problem of supervisory control under partial observation arises when some events are not detected. This problem is investigated in [17, 58]. By introducing the notion of language-observability, a necessary and sufficient condition for the existence of a solution to the problem has been developed. Given a desired language $K$, there exists a supervisor with partial observation that realizes $K$ if and only if $K$ is both controllable and observable.

One could take two approaches in addressing the supervisory control problem where $K$ is controllable but not observable. In one approach, a sublanguage of $K$ that is both controllable and observable is evaluated and realized. The *supremal controllable and normal* sublanguage of $K$ and a *maximal controllable and observable* sublanguage of $K$ are two candidate solutions in this approach [14, 15, 17, 58]. The other approach involves enhancing the observation of the system such that $K$ itself becomes an observable language. That is, one must devise some additional detecting systems so that the observability condition is satisfied for $K$. It is relevant to ask the question: "What other events must be detected in order to have $K$ observable?" This question is closely related to the problem of event space reduction for supervisors. Thus the results of this chapter can be used to address this question too.

The main results of this chapter pertain to classifying work and observation spaces and developing search methods for characterizing these spaces

\footnote{in the context of general observation maps. normality is generally called recognizability.}
and observation spaces. Some applications of the results are also discussed.

The chapter is organized as follows: In section 2.2, observability and normality conditions are given in detail. These conditions are extensively used in this chapter. In section 2.3, we characterize work-spaces. In section 2.4, we characterize observation-spaces. In section 2.5, we give an example that illustrates the results. In section 2.6, we comment on the application of the results developed in this chapter. Finally, in section 2.7, we present a summary of the contributions of this chapter.

2.2. Observability and Normality

Observability and normality properties of languages are extensively used in this chapter. This section summarizes these properties.

In this chapter, we use a projection to model a partial observation. Formally, given $\Sigma_1$ and $\Sigma_2$ where $\Sigma_2 \subseteq \Sigma_1 \subseteq \Sigma$, let $P_{\Sigma_1, \Sigma_2} : \Sigma_1^* \rightarrow \Sigma_2^*$ be a projection defined by:

$$P_{\Sigma_1, \Sigma_2} \epsilon = \epsilon;$$

$$P_{\Sigma_1, \Sigma_2} \sigma = \epsilon, \quad \text{for } \sigma \in \Sigma_1 \setminus \Sigma_2;$$

$$P_{\Sigma_1, \Sigma_2} \sigma = \sigma, \quad \text{for } \sigma \in \Sigma_2;$$

$$P_{\Sigma_1, \Sigma_2}(s\sigma) = (P_{\Sigma_1, \Sigma_2}s)(P_{\Sigma_1, \Sigma_2}\sigma), \quad \text{for } s \in \Sigma_1^*, \sigma \in \Sigma_1.$$  

That is, the projection $P_{\Sigma_1, \Sigma_2}$ simply erases those events of $s$ that are not in $\Sigma_2$. Given the projection $P_{\Sigma_1, \Sigma_2}$, the inverse projection $P_{\Sigma_1, \Sigma_2}^{-1} : \Sigma_2^* \rightarrow \Sigma_1^*$ is defined as:

$$\text{for } s \in \Sigma_2^*, \quad P_{\Sigma_1, \Sigma_2}^{-1}s = \{ t \in \Sigma_1^* \mid P_{\Sigma_1, \Sigma_2}t = s \}.$$
of the projections (or inverse projections) of all words in the language. If a language is closed (or non-closed) then its projection as well as its inverse projection are closed (or non-closed).

Given a language $K$ and a projection $P_{\Sigma, \Sigma}$. Lin and Wonham [58] have developed the notion of language-observability. $K$ is called $P_{\Sigma, \Sigma}$-observable or simply observable if and only if

$$\ker P_{\Sigma, \Sigma} \leq \text{act}_{K, L_m(G)} \quad \text{or} \quad (s_1, s_2) \in \ker P_{\Sigma, \Sigma} \implies (s_1, s_2) \in \text{act}_{K, L_m(G)}.$$  

(2.2.1)

Where we have

$$(s_1, s_2) \in \ker P_{\Sigma, \Sigma} \iff P_{\Sigma, \Sigma} s_1 = P_{\Sigma, \Sigma} s_2.$$  

(2.2.2)

and

$$(s_1, s_2) \in \text{act}_{K, L_m(G)} \iff \begin{cases} A) \quad A_{K, L(G)} (s_1) \cap \text{IA}_{K, L(G)} (s_2) = \emptyset = \text{IA}_{K, L(G)} (s_1) \cap A_{K, L(G)} (s_2), \\
B) \quad s_1 \in K \cap L_m(G) \text{ and } s_2 \in K \cap L_m(G) \implies (s_1 \in K \iff s_2 \in K). \end{cases}$$  

(2.2.3)

Here, the active set and the inactive set of $s \in \Sigma^*$ with respect to $K$ and $L(G)$ are denoted by $A_{K, L(G)} (s)$ and $\text{IA}_{K, L(G)} (s)$ respectively, and are defined by

$$A_{K, L(G)} (s) := \begin{cases} \{ \sigma \mid s \sigma \in K \}, & \text{if } s \in K; \\
\emptyset, & \text{otherwise}. \end{cases}$$  

(2.2.4)

$$\text{IA}_{K, L(G)} (s) := \begin{cases} \{ \alpha \mid \alpha \sigma \in L(G) \setminus K \}, & \text{if } s \in K; \\
\emptyset, & \text{otherwise}. \end{cases}$$  

(2.2.5)

The following Theorem is taken from [58].
of $L_m(G)$. There exists a proper supervisor $S$ such that $L_m(S/G) = K$ if and only if $K$ is both controllable and observable.

\[ \triangle \]

Given a sublanguage $K \subseteq L_m(G)$ and a projection $P_{\Sigma,\Sigma}$, we say the language $K$ is $(P_{\Sigma,\Sigma}, L_m(G))$-normal if the following two relations hold,

- \[ A) \quad \hat{K} = L(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} \hat{K} \quad \text{and} \quad B) \quad K = L_m(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K. \] (2.2.6)

If $K$ is closed, then the normality condition reduces to only (2.2.6.A). To see this, suppose that (2.2.6.A) holds. Then

$$L_m(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K \subseteq L(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} \hat{K} = \hat{K} = K.$$ 

$L_m(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K \supseteq K$ always holds and thus (2.2.6.B) is implied. Therefore, for closed languages, condition (2.2.6.A) suffices for the normality of $K$.

The following proposition establishes the relationship between the normality and the observability conditions for a controllable language.

**Proposition (2.2.2).** $A \iff B$, where

- \[ A) \begin{cases} 1) \quad K = L_m(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K, \\ \text{and} \\ 2) \quad \hat{K} = L(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} \hat{K}, \\ \text{and} \\ 3) \quad \hat{K} \Sigma_u \cap L(G) \subseteq \hat{K}, \end{cases} \]

and

- \[ B) \begin{cases} 1) \quad \ker P_{\Sigma,\Sigma} \leq \text{act}_{K,L_m(G)}, \\ \text{and} \\ 2) \quad \hat{K}(\Sigma_u \cup (\Sigma \setminus \hat{\Sigma})) \cap L(G) \subseteq \hat{K}. \end{cases} \]

Theorem (2.2.1) and Proposition (2.2.2) suggest that if the controllability and the normality conditions are satisfied for $K$, then there exists a supervisor
\[ \Sigma_c \cap \tilde{\Sigma}. \]

**Proof:** Acknowledging the fact that a partial proof of the proposition is given in \([17, 56, 58]\), here we give a complete proof for the proposition.

Proof for \( A \implies B \).

(i). \([(A.1) \text{ and } (A.2)] \implies (B.1)\):

Select \((s_1, s_2) \in \ker P_{\Sigma,\Sigma}\). Assume that \(s_1 \text{ and } s_2 \in \tilde{K}\), otherwise \((s_1, s_2) \in \text{act}_{K,L_m(G)}\) and \((B.1)\) holds. To show \((2.2.3.A)\), suppose the contrary, that is there exists \(\alpha \in A_{K, L(G)}\) \((s_1) \cap IA_{K, L(G)}(s_2)\). Then \(s_1\alpha \in \tilde{K}\), \(s_2\alpha \in L(G) \setminus \tilde{K}\) and \(P_{\Sigma,\Sigma}s_2\alpha = P_{\Sigma,\Sigma}s_1\alpha\). Thus \(s_2\alpha \in L(G) \cap P_{\Sigma,\Sigma}^{-1}P_{\Sigma,\Sigma}s_1\alpha\). By \((A.2)\) we conclude that \(s_2\alpha \in \tilde{K}\), this contradicts \(s_2\alpha \in L(G) \setminus \tilde{K}\). Thus we must have \(A_{K, L(G)}(s_1) \cap IA_{K, L(G)}(s_2) = \emptyset\). Similarly \(IA_{K, L(G)}(s_1) \cap A_{K, L(G)}(s_2) = \emptyset\).

To show \((2.2.3.B)\) assume that \(s_1 \in K \cap L_m(G)\) and \(s_2 \in \tilde{K} \cap L_m(G)\). We need to show that \(s_2 \in K\). From \(P_{\Sigma,\Sigma}s_2 = P_{\Sigma,\Sigma}s_1\), we conclude that \(s_2 \in L_m(G) \cap P_{\Sigma,\Sigma}^{-1}P_{\Sigma,\Sigma}s_1\). By \((A.1)\) we have \(s_2 \in K\). Similarly, \(s_1 \in K \cap L_m(G)\) and \(s_2 \in K \cap L_m(G)\) together imply \(s_1 \in K\).

(ii). \([(A.2) \text{ and } (A.3)] \implies (B.2)\):

Consider \(t \in \tilde{K}\) and \(\beta \in \Sigma\) such that \(t\beta \in L(G)\). If \(\beta \in \Sigma_u\), then \((B.2)\) is identical to \((A.3)\) and the result holds. Thus assume that \(\beta \in \Sigma \setminus \tilde{\Sigma}\). In this case, we have \(P_{\Sigma,\Sigma}t\beta = P_{\Sigma,\Sigma}t\). Then, \(t\beta \in L(G) \cap P_{\Sigma,\Sigma}^{-1}P_{\Sigma,\Sigma}t\). By \((A.2)\) we conclude that \(t\beta \in \tilde{K}\). Thus, \((B.2)\) holds.

(i) and (ii) complete the proof for \( A \implies B \).

Proof for \( B \implies A \).
(iv). \([B.1]\) and \((B.2)\) \(\implies\) \((A.2)\): By induction on the length of words we show that if \(u \in L(G) \cap P^{-1}_{\Sigma,\Sigma} P_{\Sigma,\Sigma} \tilde{K}\), then \(u \in \tilde{K}\).

**Induction basis:** \(u = \varepsilon\). The result holds because \(\varepsilon \in \tilde{K}\).

**Induction hypothesis:** \([u \in L(G) \cap P^{-1}_{\Sigma,\Sigma} P_{\Sigma,\Sigma} \tilde{K}] \implies [u \in \tilde{K}]\).

**Induction step:** Take \(u \gamma \in L(G) \cap P^{-1}_{\Sigma,\Sigma} P_{\Sigma,\Sigma} \tilde{K}\).

By the induction hypothesis \(u \in \tilde{K}\). We distinguish two cases: (1) \(\gamma \in [\Sigma_u \cup (\Sigma \setminus \hat{\Sigma})]\) and (2) \(\gamma \notin [\Sigma_u \cup (\Sigma \setminus \hat{\Sigma})]\). In case (1), the result is immediate from \((B.2)\). Note that we have \(\gamma \in \hat{\Sigma}\) in case (2). Thus, we must have \(u \gamma \in \tilde{K}\) such that \((u, v) \in \ker P_{\Sigma,\Sigma}\). Because \(\gamma \in A_{\tilde{K}, L(G)}(v)\), using \((B.1)\), we conclude that \(\gamma \in A_{\tilde{K}, L(G)}(u)\). That is \(u \gamma \in \tilde{K}\).

(v). \([(B.1)\) and \((B.2)\) \(\implies\) \((A.1)\):

Take \(w \in L_m(G) \cap P^{-1}_{\Sigma,\Sigma} P_{\Sigma,\Sigma} \tilde{K}\). Then there exists \(x \in \tilde{K}\) such that \((w, x) \in \ker P_{\Sigma,\Sigma}\). Because \(\tilde{K} \subseteq \tilde{K}\) and \(L_m(G) \subseteq L(G)\), we can write \(w \in L(G) \cap P^{-1}_{\Sigma,\Sigma} P_{\Sigma,\Sigma} x\). By \((A.2)\) we have that \(w \in \tilde{K}\). Thus we have \(w \in \tilde{K} \cap L_m(G)\) and \(x \in \tilde{K} \cap L_m(G)\). By \((B.1)\) we conclude that \(w \in K\).

\[\triangle\]

2.3. **Work Spaces**

Consider a generator \(G = (Q, \Sigma, \delta, q_0, Q_m)\) and a desired controllable sublanguage \(K \subseteq L_m(G)\). One can design a supervisor for the generator to
The choice of a supervisor to achieve $K$ is not unique. One can start with $(A, \phi)$ and use techniques in [97] to design another supervisor that has smaller state space cardinality.

Given a subalphabet $\hat{\Sigma} \subseteq \Sigma$, suppose that $K$ is $(P_{\Sigma, \hat{\Sigma}}, L_m(G))$-normal, that is

$$K = L_m(G) \cap P^{-1}_{\Sigma, \hat{\Sigma}}P_{\Sigma, \hat{\Sigma}}K \quad \text{and} \quad \bar{K} = L(G) \cap P^{-1}_{\Sigma, \hat{\Sigma}}P_{\Sigma, \hat{\Sigma}}\bar{K} \quad (2.3.1)$$

Then, Theorem (2.2.1) and Proposition (2.2.2) suggest that we can design a supervisor that observes only $\hat{\Sigma}$ and disables only $\hat{\Sigma} \cap \Sigma_c$ such that the closed loop system generates $K$. In such a case, the supervisor $B = (B, \varphi)$ with the following characteristics is also a candidate supervisor.

$$B = (Y, \hat{\Sigma}, \zeta, y_0, Y_m) \quad \text{where} \quad L(B) = P_{\Sigma, \hat{\Sigma}}K;$$

$$\varphi(y)(\sigma) = \begin{cases} 0, & \text{if } \exists s \in P_{\Sigma, \hat{\Sigma}}\bar{K} \text{ such that } y = \zeta(s, y_0) \text{ and } s\sigma \in P_{\Sigma, \hat{\Sigma}}L(G) \setminus P_{\Sigma, \hat{\Sigma}}\bar{K}; \\ 1, & \text{Otherwise.} \end{cases}$$

We can compare supervisors $A$ and $B$ with respect to the cardinality of their state space and events set. For example, it is likely that $|X| \leq |Y|$. Thus computer implementation of supervisor $A$ would be simpler. On the other hand, $B$ does not need any information about $\Sigma \setminus \hat{\Sigma}$. Therefore there is
communication channels between the plant and the supervisor are needed for implementing B. There may be some cases that supervisor B is preferable to A. We call \( \hat{\Sigma} \) a sufficient work space for \( K \) because there exists a supervisor, here B, that works only with events in \( \hat{\Sigma} \) to realize \( K \). The above discussion suggests that to reduce the event space of a supervisor for \( K \), one must find a subset \( \hat{\Sigma} \subseteq \Sigma \) such that the normality condition (2.3.1) is satisfied.

Consider the power set of \( \Sigma \), i.e., \( 2^\Sigma \). Every element of \( 2^\Sigma \) induces a projection. The set of all projections together with the binary composition operator is an Abelian semigroup, where \( P_{\Sigma,\Sigma} \) is the identity element. Given a controllable sublanguage \( K \subseteq L_m(G) \), among all projections we are particularly interested in projections of the following type.

**Definition (2.3.1). Class of Sufficient Work Spaces:**

\[
CSWS(K, L_m(G)) \quad (2.3.2)
\]

\[
:= \{ \Sigma \subseteq \Sigma \mid K = L_m(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K \text{ and } \bar{K} = L(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} \bar{K} \}.
\]

In the sequel, when the underlying languages are clear from the context we write \( CSWS \) instead of \( CSWS(K, L_m(G)) \).

\( CSWS \) is not empty since \( \Sigma \in CSWS \). Given any sufficient work space \( \hat{\Sigma} \), as we discussed, it is possible to design a local supervisor such that the closed loop system generates the language \( K \).

The following proposition states that in the partially ordered system

\[
\text{(CSWS; \subseteq)}
\]

there exists many chains.
Proof: Take \( s \in L_m(G) \cap P_{\Sigma, \Sigma_1}^{-1} P_{\Sigma, \Sigma_1} K \). Then, \( \exists t \in K \) such that \( P_{\Sigma, \Sigma_1} s = P_{\Sigma, \Sigma_1} t \). Thus \( P_{\Sigma, \Sigma_1} P_{\Sigma, \Sigma_2} s = P_{\Sigma, \Sigma_1} P_{\Sigma, \Sigma_2} t \) or \( P_{\Sigma, \Sigma_1} s = P_{\Sigma, \Sigma_1} t \) and it follows that \( s \in P_{\Sigma, \Sigma_1}^{-1} P_{\Sigma, \Sigma_1} K = K \). Therefore \( s \in L_m(G) \cap P_{\Sigma, \Sigma_1}^{-1} P_{\Sigma, \Sigma_1} K = K \). This means that \( L_m(G) \cap P_{\Sigma, \Sigma_1}^{-1} P_{\Sigma, \Sigma_1} K \subseteq K \). Because it is always true that \( L_m(G) \cap P_{\Sigma, \Sigma_1}^{-1} P_{\Sigma, \Sigma_1} K \geq K \), we conclude that \( K = L_m(G) \cap P_{\Sigma, \Sigma_1}^{-1} P_{\Sigma, \Sigma_1} K \).

Similarly we can prove that \( K = L(G) \cap P_{\Sigma, \Sigma_2}^{-1} P_{\Sigma, \Sigma_2} K \). Therefore \( \Sigma_2 \in CSWS \).

\( \Delta \)

Corollary (2.3.3). CSWS is closed under the union operation and its supremal element is \( \Sigma \).

\( \Delta \)

Note that CSWS need not be closed under the intersection operation and consequently, CSWS may not have an infimal element. However, by Zorn’s Lemma we know that every chain in the partially ordered system \((CSWS, \subseteq)\) has a minimal element with respect to the cardinality metric. We shall refer to such a minimal element as a proper work space. Then, a proper work space is a sufficient work space that does not properly contain any other sufficient work spaces. Formally we have:

Definition (2.3.4). Class of Proper Work Spaces:

\[
CPWS(K, L_m(G)) := \{ \Sigma_P \in CSWS \mid \Sigma_S \not\subseteq \Sigma_P \text{ for all } \Sigma_S \in CSWS \}.
\]

(2.3.3)

Among the elements in \( CPWS \) there is at least one element with the least
Definition (2.3.5). Class of Minimal Work Spaces:

\[ CMWS(K, L_m(G)) := \{ \Sigma_M \in CSWS \mid |\Sigma_M| \leq |\Sigma_S| \text{ for all } \Sigma_S \in CSWS \}. \]  

(2.3.4)

There are those events whose observation is essential for any local supervisor. We call these necessary events and define them according to:

Definition (2.3.6). Class of Necessary Work Spaces:

\[ CNWS(K, L_m(G)) := \{ \Sigma_N \subseteq \Sigma \mid \Sigma_N \subseteq \Sigma_S \text{ for all } \Sigma_S \in CSWS \}. \]  

(2.3.5)

Lemma (2.3.7). \( CNWS \) is closed under intersections and unions and thus contains infimal and supremal elements. In particular, we have.

\[
\inf CNWS = \emptyset \quad \sup CNWS = \bigcup_{\Sigma_N \in CNWS} \Sigma_N = \bigcap_{\Sigma_S \in CSWS} \Sigma_S \quad (2.3.6)
\]

Proof: Immediate from the definitions.

\[ \triangle \]

Obviously \( \sup CNWS \) is unique. Given the above classes of subalphabets, we have the following lemma.

Lemma (2.3.8). If \( \sup CNWS \in CSWS \), then

(i) \( \sup CNWS \in CMWS \)

(ii) \( |CMWS| = 1 \)

(iii) \( CPWS = CMWS \)
Thus for all $\Sigma_S \in CWS$, $|\sup CNWS| \leq |\Sigma_S|$ holds. Because $\sup CNWS \in CWS$ we conclude that $\sup CNWS \in CMWS$.

(ii) By contradiction: Suppose that $|CMWS| > 1$ and let $\Sigma_1, \Sigma_2$ be elements of $CMWS$ such that $\Sigma_1 \neq \Sigma_2$. W.l.o.g., using (i) assume that $\Sigma_1 = \sup CNWS$. Then by the uniqueness of $\sup CNWS$ we have $\Sigma_1 \subset \Sigma_2$. Therefore $|\Sigma_1| < |\Sigma_2|$. But this contradicts $\Sigma_2 \in CMWS$.

(iii) It follows from the definitions that $CMWS \subseteq CPWS$. Suppose that $CMWS \not\supset CPWS$. Then there exists $\Sigma_1 \in CPWS$ such that $\Sigma_1 \not\in CMWS$. By (ii) we conclude that $\Sigma_1 \neq \sup CNWS$. Then from (2.3.6) we must have $\sup CNWS \subset \Sigma_1$. By hypothesis $\sup CNWS \in CWS$ and this implies that $\Sigma_1 \not\in CPWS$, which is a contradiction. Therefore $CPWS \subseteq CMWS$ and we obtain the desired result that $CPWS = CMWS$.

$\Delta$

In general, however, $\sup CNWS \not\in CWS$.

If for a given initial work space the normality condition fails, it may be necessary to include some additional detectors, communication and command lines in the system. Then, from an engineering design perspective, evaluation of $CMWS$ is a reasonable objective. For this preliminary analysis we assume that the cost of including an event in a work space is uniform over $\Sigma$. In what follows we try to establish a search method to obtain a solution to the question that we posed in introduction. Through our development, we also gain some insight into the structure of the work space classes that we have defined.

**Proposition (2.3.9).** Evaluation of $\sup CNWS$ requires a search over $\Sigma$ and
\[ \sup_{\sigma \in \Sigma} \{ \sigma \mid K \text{ is not } (P_{\Sigma,(\Sigma-\sigma)}, L_m(G))-normal \} \]

**Proof:** First we show

\[ \sup_{\sigma \in \Sigma} \{ \sigma \mid K \text{ is not } (P_{\Sigma,(\Sigma-\sigma)}, L_m(G))-normal \} \supseteq \bigcup_{\sigma \in \Sigma} \{ \sigma \mid K \text{ is not } (P_{\Sigma,(\Sigma-\sigma)}, L_m(G))-normal \}. \tag{2.3.7} \]

Assume the contrary, that is, \( \exists \alpha \in \Sigma \) such that \( K \) is not \( (P_{\Sigma,(\Sigma-\alpha)}, L_m(G))-normal \) and \( \alpha \notin \sup_{\sigma \in \Sigma} \{ \sigma \mid K \text{ is not } (P_{\Sigma,(\Sigma-\sigma)}, L_m(G))-normal \} \). Then \( \exists \Sigma_{\alpha} \in CSWS \) such that \( \alpha \notin \Sigma_{\alpha} \). We have \( \Sigma_{\alpha} \subseteq \Sigma \) and \( \alpha \notin \Sigma_{\alpha} \), therefore \( \Sigma_{\alpha} \subseteq (\Sigma - \alpha) \). By Proposition (2.3.2) we have \( (\Sigma - \alpha) \in CSWS \) which implies that

\[ \alpha \notin \bigcup_{\sigma \in \Sigma} \{ \sigma \mid K \text{ is not } (P_{\Sigma,(\Sigma-\sigma)}, L_m(G))-normal \}. \]

This is the desired contradiction. Next we show that

\[ \sup_{\sigma \in \Sigma} \{ \sigma \mid K \text{ is not } (P_{\Sigma,(\Sigma-\sigma)}, L_m(G))-normal \} \subseteq \bigcup_{\sigma \in \Sigma} \{ \sigma \mid K \text{ is not } (P_{\Sigma,(\Sigma-\sigma)}, L_m(G))-normal \}. \tag{2.3.8} \]

Assume the contrary, that is, \( \exists \alpha \in \sup_{\sigma \in \Sigma} \{ \sigma \mid K \text{ is not } (P_{\Sigma,(\Sigma-\sigma)}, L_m(G))-normal \} \) is

\( (P_{\Sigma,(\Sigma-\alpha)}, L_m(G))-normal. \)

Then by definition, \( (\Sigma - \alpha) \in CSWS \). This implies that \( \alpha \notin \bigcap_{\Sigma_{\alpha} \in CSWS} \Sigma_{\alpha} = \sup_{\sigma \in \Sigma} \{ \sigma \mid K \text{ is not } (P_{\Sigma,(\Sigma-\sigma)}, L_m(G))-normal \} \), which yields the desired contradiction. Thus (2.3.7) and (2.3.8) establish the proposition.

\[ \Delta \]

There are some events in \( \sup_{\sigma \in \Sigma} \{ \sigma \mid K \text{ is not } (P_{\Sigma,(\Sigma-\sigma)}, L_m(G))-normal \} \) that can be found without checking the normality condition. The following proposition characterizes a subset of such events.
Proof: Suppose $t \in \hat{K}$ and $t\sigma \in L(G) \setminus \hat{K}$. This implies that the event $\sigma$ must be disabled, at least once, by any supervisor. We need to show that $\forall \Sigma_\ast \in CSWS, \sigma \in \Sigma_\ast$. To prove this, we assume the contrary, that is for some $\Sigma_\ast \in CSWS, \sigma \not\in \Sigma_\ast$. It follows that $P_{\Sigma,\Sigma_\ast}t\sigma = P_{\Sigma,\Sigma_\ast}t$ and we have that $t\sigma \in P_{\Sigma,\Sigma_\ast}^{-1} P_{\Sigma,\Sigma_\ast} t \subseteq P_{\Sigma,\Sigma_\ast}^{-1} P_{\Sigma,\Sigma_\ast} \hat{K}$. Then $t\sigma \in L(G) \cap P_{\Sigma,\Sigma_\ast}^{-1} P_{\Sigma,\Sigma_\ast} \hat{K} = \hat{K}$. However, this contradicts $t\sigma \in L(G) \setminus \hat{K}$. Therefore, $\forall \Sigma_\ast \in CSWS$ we have that $\sigma \in \Sigma_\ast$. In other words, $\sigma \in \bigcap_{\Sigma_\ast \in CSWS} \Sigma_\ast = \sup CNWS$.

$\triangle$

In many situations there are some events in the open loop system that do not appear in the specification of the closed loop system. In such a case, we can modify the automaton representation of the open loop system so that the required normality tests are computationally less involved.

Given an automaton $A$, and a subalphabet $\hat{\Sigma} \subseteq \Sigma$, let $A_{mod,\hat{\Sigma}}$ denote the automaton obtained from $A$ by deleting all transitions under events of $\hat{\Sigma}$, and then finding the accessible part of the resultant automaton.

Let $G = (Q, \Sigma, \delta, q_0, Q_m)$ be the generator representing the open loop system and $K \subseteq L_m(G)$ is the desired behavior. We have the following proposition.

Proposition (2.3.11). Consider $\hat{\Sigma}$ such that $\hat{\Sigma} \subseteq \sup CNWS(K, L_m(G)) \cap [\mathcal{E}(K)]^c$. We have

\[
CSWS(K, L_m(G)) = \{ \Sigma' \mid \Sigma' = (\hat{\Sigma} \cup \hat{\Sigma}) \text{ for some } \hat{\Sigma} \in CSWS(K, L_m(G_{mod,\Sigma})) \}.
\]
Notice that $\hat{\Sigma}$ represents those events that do not appear in the desired language and a supervisor must prevent them of happening. Prevention of the events in $\hat{\Sigma}$ may result in blocking of many other events too. Thus there is no need to include these blocked events in the work space of a supervisor. As the statement of the proposition suggests, all such blocked events can be discarded by finding $G_{\text{mod},\hat{\Sigma}}$ and evaluating the work spaces with respect to this modified plant. Also, $G_{\text{mod},\hat{\Sigma}}$ would have a smaller state space than $G$. Therefore evaluation of $CSWS(K, L_m(G_{\text{mod},\hat{\Sigma}}))$ is computationally simpler.

**Proof:** Consider $\Sigma''$ that is disjoint union of $\hat{\Sigma}$ and $\hat{\Sigma}$, i.e. $\Sigma'' = (\hat{\Sigma} \cup \hat{\Sigma}) \subseteq \Sigma$, where
\[
\hat{\Sigma} \subseteq \sup C.N.W.S(K, L_m(G)) \cap [E(K)]^c.
\]
Then we have
\[
\hat{\Sigma} \cap E(K) = \emptyset.
\]
\[
\Rightarrow \hat{\Sigma} \cap E(P_{\Sigma', \Sigma''}^{-1} P_{\Sigma, \Sigma''} K) = \emptyset,
\]
\[
\Rightarrow \hat{\Sigma} \cap E(L_m(G) \cap P_{\Sigma', \Sigma''}^{-1} P_{\Sigma, \Sigma''} K) = \emptyset,
\]
Therefore,
\[
L_m(G) \cap P_{\Sigma', \Sigma''}^{-1} P_{\Sigma, \Sigma''} K = L_m(G_{\text{mod},\hat{\Sigma}}) \cap P_{\Sigma', \Sigma''}^{-1} P_{\Sigma, \Sigma''} \hat{K}. \quad (2.3.9)
\]
Similarly, we have
\[
L(G) \cap P_{\Sigma', \Sigma''}^{-1} P_{\Sigma, \Sigma''} \hat{K} = L(G_{\text{mod},\hat{\Sigma}}) \cap P_{\Sigma', \Sigma''}^{-1} P_{\Sigma, \Sigma''} \hat{K}. \quad (2.3.10)
\]
If $\Sigma'' \in CSWS(K, L_m(G))$ then, using (2.3.9) and (2.3.10) we conclude that
\[
\hat{\Sigma} := (\Sigma'' \setminus \hat{\Sigma}) \in CSWS(K, L_m(G_{\text{mod},\hat{\Sigma}})).
\]
Then we conclude that
\[
CSWS(K, L_m(G)) \subseteq \{ \Sigma' | \Sigma' = (\hat{\Sigma} \cup \hat{\Sigma}) \text{ for some } \hat{\Sigma} \in CSWS(K, L_m(G_{mod, \Sigma})) \}.
\]

On the other hand, if for some \( \hat{\Sigma} \subseteq \Sigma_{mod} \) we have
\[
\hat{\Sigma} \in CSWS(K, L_m(G_{mod, \Sigma})),
\]
by (2.3.9) and (2.3.10) we conclude that \( \Sigma'' := (\hat{\Sigma} \cup \hat{\Sigma}) \in CSWS(K, L_m(G)) \).

That is
\[
CSWS(K, L_m(G)) \supseteq \{ \Sigma' | \Sigma' = (\hat{\Sigma} \cup \hat{\Sigma}) \text{ for some } \hat{\Sigma} \in CSWS(K, L_m(G_{mod, \Sigma})) \}.
\]

This completes the proof.

\[\triangle\]

The following corollaries are obtained from Proposition (2.3.11) and used in the forthcoming algorithm.

Corollary (2.3.12).
\[
\sup CNWS(K, L_m(G)) = \hat{\Sigma} \cup \sup CNWS(K, L_m(G_{mod, \Sigma})).
\]

Proof:

\[
\sup CNWS(K, L_m(G)) = \bigcap_{\Sigma_S \in CSWS(K, L_m(G))} \Sigma_S,
\]
\[
= \bigcap_{\hat{\Sigma} \in \text{CSWS}(K, L_m(G_{mod, \Sigma}))} (\hat{\Sigma} \cup \hat{\Sigma}),
\]

\[
= \hat{\Sigma} \cup \left( \bigcap_{\hat{\Sigma} \in \text{CSWS}(K, L_m(G_{mod, \Sigma}))} \hat{\Sigma} \right),
\]

\[
= \hat{\Sigma} \cup \sup \text{CSWS}(K, L_m(G_{mod, \Sigma})).
\]

\[\triangle\]

**Corollary (2.3.13).** If \( \hat{\Sigma} \in \text{CMWS}(K, L_m(G_{mod, \Sigma})) \) then

\[
\hat{\Sigma} \cup \hat{\Sigma} \in \text{CMWS}(K, L_m(G)).
\]

**Proof:** We prove this by contradiction. Take \( \Sigma_1 \in \text{CMWS}(K, L_m(G_{mod, \Sigma})) \). By Proposition (2.3.11) we have \( \Sigma^1 := \hat{\Sigma} \cup \Sigma_1 \in \text{CSWS}(K, L_m(G)) \). Assume that \( \Sigma^1 \) is not a minimal work space. Then we must have \( \Sigma^2 \subset \Sigma^1 \) such that \( \Sigma^2 \in \text{CMWS}(K, L_m(G)) \). By proposition (2.3.11) we have \( \Sigma^2 = \hat{\Sigma} \cup \Sigma_2 \) where \( \Sigma_2 \in \text{CSWS}(K, L_m(G_{mod, \Sigma})) \). Therefore \( \hat{\Sigma} \cup \Sigma_2 = \Sigma^2 \subset \Sigma^1 = \hat{\Sigma} \cup \Sigma_1 \). \( \hat{\Sigma} \cap \Sigma_1 = \hat{\Sigma} \cap \Sigma_2 = \emptyset \) because \( \Sigma_1, \Sigma_2 \not\subseteq \mathcal{E}(L_m(G_{mod, \Sigma})) \). Thus \( \Sigma^1 - \Sigma^2 = \Sigma_1 - \Sigma_2 \neq \emptyset \) and \( |\Sigma_2| < |\Sigma_1| \). This contradicts \( \Sigma_1 \in \text{CMWS}(K, L_m(G_{mod, \Sigma})) \), which is the desired contradiction.

\[\triangle\]

Notice that for each event in \( \hat{\Sigma} \), sequential application of proposition (2.3.11) can be used to provide a characterization of \( \text{CSWS}(K, L_m(G)) \), by working sequentially, a reduced order automaton is obtained at each stage.

At this stage of our development, a typical search algorithm to evaluate—say—a minimal work space can be summarized as follows:
Step 0: Define the automata $G$ and $S$ such that
\[ K = L_m(S) \subseteq L_m(G). \]

Step 1: Assign $\Sigma_{mod} = \Sigma$, $G_{mod} = G$ and
\[ \sup C.N.W.S(K, L_m(G)) = \sup C.N.W.S(K, L_m(G_{mod})) = \emptyset. \]

Step 2: For all $\sigma \in \Sigma$
\{ If $\sigma \in \Sigma_{mod}$
\{ If $K$ is not $(P_{\Sigma_{mod}, (\Sigma_{mod} - \sigma)}, L_m(G_{mod}))$-normal
\{ $\sup C.N.W.S(K, L_m(G)) = \sup C.N.W.S(K, L_m(G)) \cup \{\sigma\}$
If $\sigma \notin \mathcal{E}(K)$
\{ $G_{mod} = (G_{mod})_{mod, \sigma}$
\$\Sigma_{mod} = \mathcal{E}(L_m(G_{mod}))$
\}
\}Else
\{ $\sup C.N.W.S(K, L_m(G_{mod}))$
\[ = \sup C.N.W.S(K, L_m(G_{mod})) \cup \{\sigma\}\]
\}EndIf
\}EndIf
\}EndFor
\[
\begin{align*}
  \text{If } K \text{ is } (P_{\text{mod}}, \{C_{\text{mod}}, \sup C_{\text{NWS}}(K, L_m(G_{\text{mod}}))\}, L_m(G_{\text{mod}}))\text{-normal}\}
  \\
  \quad \{\hat{\Sigma} \cup \sup C_{\text{NWS}}(K, L_m(G_{\text{mod}}))\} \in C_{\text{MWS}}(K, L_m(G_{\text{mod}}))
  \\
  \quad \{\hat{\Sigma} \cup \sup C_{\text{NWS}}(K, L_m(G))\} \in C_{\text{MWS}}(K, L_m(G))
  \\
  \text{STDP.}
  \end{align*}
\]

EndIf

EndFor

EndFor

To relate Algorithm (2.3.A) to Proposition (2.3.11) and corollaries (2.3.12) and (2.3.13), notice that at the end of step 2, \( G_{\text{mod}} = G_{\text{mod}, \Sigma} \) and

\[
[\sup C_{\text{NWS}}(K, L_m(G)) \setminus \sup C_{\text{NWS}}(K, L_m(G_{\text{mod}}))] = \hat{\Sigma}.
\]

At the end of step 3,

\[
\{\hat{\Sigma} \cup \sup C_{\text{NWS}}(K, L_m(G_{\text{mod}}))\}
\]

and

\[
\{\hat{\Sigma} \cup \sup C_{\text{NWS}}(K, L_m(G))\}
\]

are equivalent to \( \hat{\Sigma} \) and \( \hat{\Sigma} \cup \tilde{\Sigma} \) of corollary (2.3.13), respectively.

Testing the normality condition is the basic operation in the Algorithm (2.3.A). Chapter 4 presents an algorithm to test the normality condition. Computational complexity of the algorithm to test normality is polynomial with respect to the state space cardinality of the automatons representing the generator and the desired closed loop language as well as the cardinality of the events set. However, computational complexity of Algorithm (2.3.A) is polynomial with respect to the cardinality of state spaces and exponential with
algorithm (2.3.A). If \( \text{sup} \mathcal{CNSW}(K, L_m(G)) \in \mathcal{CSWS}(K, L_m(G)) \) the algorithm will skip step 3 and stop in polynomial time. If \( \text{sup} \mathcal{CNSW}(K, L_m(G)) = \emptyset \), step 3 will involve a search over \( 2^\Sigma \) to evaluate a minimal work space.

In many cases, detection of some events, if not impossible, is very costly. If these events belong to \( \text{sup} \mathcal{CNSW}(K, L_m(G)) \), then it is impossible to design a supervisor without detecting these events. If these events are not necessary, it is of interest to see what is implied by the undetectibility of these events. The following propositions address this issue.

**Proposition (2.3.14).** For any \( \Sigma' \subseteq \hat{\Sigma} \in \mathcal{CSWS}(K, L_m(G)) \),

\[
\Sigma' \in \mathcal{CSWS}(P_{\Sigma', \Sigma} K, P_{\Sigma', \Sigma} L_m(G)) \iff \Sigma' \in \mathcal{CSWS}(K, L_m(G)).
\]

**Proof:**

(i) Proof of \( \implies \).

\[
\Sigma' \in \mathcal{CSWS}(P_{\Sigma', \Sigma} K, P_{\Sigma', \Sigma} L_m(G)),
\]
\[
\implies P_{\Sigma', \Sigma} K = P_{\Sigma', \Sigma}^{-1} P_{\Sigma', \Sigma} P_{\Sigma', \Sigma} K \cap P_{\Sigma', \Sigma} L_m(G),
\]
\[
\implies P_{\Sigma', \Sigma}^{-1} P_{\Sigma', \Sigma} K = P_{\Sigma', \Sigma}^{-1} \{P_{\Sigma', \Sigma}^{-1} P_{\Sigma', \Sigma} P_{\Sigma', \Sigma} K \cap P_{\Sigma', \Sigma} L_m(G)\},
\]
\[
\implies P_{\Sigma', \Sigma}^{-1} P_{\Sigma', \Sigma} K \supseteq P_{\Sigma', \Sigma}^{-1} P_{\Sigma', \Sigma} K \cap P_{\Sigma', \Sigma} L_m(G),
\]

because \( \hat{\Sigma} \in \mathcal{CSWS}(K, L_m(G)) \), we have

\[
\implies K = L_m(G) \cap P_{\Sigma', \Sigma}^{-1} P_{\Sigma', \Sigma} K \supseteq L_m(G) \cap P_{\Sigma', \Sigma}^{-1} P_{\Sigma', \Sigma} K,
\]

because \( K \subseteq L_m(G) \cap P_{\Sigma', \Sigma}^{-1} P_{\Sigma', \Sigma} K \), we have

\[
\implies K = L_m(G) \cap P_{\Sigma', \Sigma}^{-1} P_{\Sigma', \Sigma} K.
\]
\[ \hat{K} = L(G) \cap P_{\Sigma, \hat{\Sigma}}^{-1} P_{\Sigma, \hat{\Sigma}} \hat{K}. \]

Therefore,

\[ \Sigma' \in CSWS(K, L_m(G)). \]

(ii) Proof of \( \iff \). We must show that for any \( \Sigma' \subseteq \hat{\Sigma} \) such that \( \Sigma', \hat{\Sigma} \in CSWS(K, L(G)) \) we have

\[ A) P_{\Sigma, \Sigma} K = P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} P_{\Sigma, \Sigma} K \cap P_{\Sigma, \Sigma} L_m(G) \]

and

\[ B) P_{\Sigma, \hat{\Sigma}} \hat{K} = P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} P_{\Sigma, \hat{\Sigma}} \hat{K} \cap P_{\Sigma, \Sigma} L(G). \quad (2.3.11) \]

We first prove (2.3.11.A), relation (2.3.11.B) can be proved in a similar way. It is always true that

\[ P_{\Sigma, \Sigma} K \subseteq P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} P_{\Sigma, \Sigma} K \cap P_{\Sigma, \Sigma} L_m(G), \]

so it is enough to show

\[ P_{\Sigma, \Sigma} K \supseteq P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} P_{\Sigma, \Sigma} K \cap P_{\Sigma, \Sigma} L_m(G). \]

Rewrite the above relation as follows

\[ P_{\Sigma, \Sigma} K \supseteq P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} K \cap P_{\Sigma, \Sigma} L_m(G). \]

To prove this, take

\[ s \in P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} K \cap P_{\Sigma, \Sigma} L_m(G). \]
\[ \exists t \in K \quad \text{such that} \quad P_{\Sigma, \Sigma'} s = P_{\Sigma, \Sigma'} t, \]  
(2.3.12)

and

\[ \exists u \in L_m(G) \quad \text{such that} \quad s = P_{\Sigma, \Sigma'} u. \]  
(2.3.13)

Using (2.3.12) in (2.3.13), we obtain

\[ P_{\Sigma, \Sigma'}(P_{\Sigma, \Sigma'} u) = P_{\Sigma, \Sigma'} t \quad \implies \quad P_{\Sigma, \Sigma'} u = P_{\Sigma, \Sigma'} t. \]

Therefore,

\[ u \in P_{\Sigma, \Sigma'}^{-1} P_{\Sigma, \Sigma'} t \subseteq P_{\Sigma, \Sigma'}^{-1} P_{\Sigma, \Sigma'} K. \]

Using (2.3.13), we have

\[ u \in L_m(G) \cap P_{\Sigma, \Sigma'}^{-1} P_{\Sigma, \Sigma'} K. \]

Since \( \Sigma' \in CSWS(K, L_m(G)) \), we conclude that \( u \in K \). Thus

\[ s = P_{\Sigma, \Sigma} u \in P_{\Sigma, \Sigma} K. \]

This completes the proof of (2.3.11.A) and the proposition.

\[ \triangle \]

**Proposition (2.3.15).** If \( \hat{\Sigma} \in CSWS(K, L_m(G)) \), then

\[ \sup CNWS(K, L_m(G)) \subseteq \sup CNWS(P_{\Sigma, \Sigma} K, P_{\Sigma, \Sigma} L_m(G)) \subseteq \hat{\Sigma}. \]

**Proof:**

\[ \sup CNWS(K, L_m(G)) = \cap \{ \Sigma' \subseteq \Sigma \quad | \quad \Sigma' \in CSWS(K, L_m(G)) \}, \]

\[ \subseteq \cap \{ \Sigma' \subseteq \hat{\Sigma} \quad | \quad \Sigma' \in CSWS(K, L_m(G)) \}, \]

by Proposition (2.3.14), we have
Proposition (2.3.15) states that if a set of unnecessary events, here $(\Sigma \setminus \hat{\Sigma})$, is not detected, then we must include additional events in any sufficient work space to compensate for the absence of the undetected events. However this is not the case if we have a necessary and sufficient work space as the following proposition suggests.

**Proposition (2.3.16).** If $\Sigma' = \sup CNWS(K, L_m(G))$ and

$$
\Sigma', \hat{\Sigma} \in CSWS(K, L_m(G)),
$$

then

$$
\Sigma' = \sup CNWS(K, L_m(G)) = \sup CNWS(P_{\Sigma, \Sigma} K, P_{\Sigma, \Sigma} L_m(G)).
$$

**Proof:** By the hypothesis we have $\Sigma' \subseteq \hat{\Sigma}$, and $\Sigma', \hat{\Sigma} \in CSWS(K, L_m(G))$, so according to Proposition (2.3.14), $\Sigma' \in CSWS(P_{\Sigma, \Sigma} K, P_{\Sigma, \Sigma} L_m(G))$. This means that $\Sigma' \supseteq \sup CNWS(P_{\Sigma, \Sigma} K, P_{\Sigma, \Sigma} L_m(G))$. The reverse inclusion follows from Proposition (2.3.15) and the proof is complete.

\[ \Delta \]

Algorithm (2.3.B) utilizes Propositions (2.3.14) to evaluate a member of $CPWS(K, L_m(G))$. 

Step 0: Define the automata $G$ and $S$ such that

\[ K = L_m(S) \subseteq L_m(G). \]

Step 1: Assign $\Sigma_P = \Sigma, PK = K, PL_m(G) = L_m(G)$ and
\[ PL(G) = \overline{PL_m(G)}. \]

Step 2: For all $\sigma \in \Sigma$

\[
\begin{cases}
\text{If } PK \text{ is } (P_{\Sigma_P \setminus \{\Sigma_P - \sigma\}}, PL_m(G))\text{-normal}
\{ \\
PK = P_{\Sigma_P \setminus \{\Sigma_P - \sigma\}}(PK)
\}
\end{cases}
\]

\[ PL_m(G) = P_{\Sigma_P \setminus \{\Sigma_P - \sigma\}}(PL_m(G)) \text{ and } PL(G) = \overline{PL_m(G)} \]

\[ \Sigma_P = \Sigma_P - \sigma \]

\}

EndIf

EndFor

STOP.

\[ \triangle \]

**Proposition (2.3.17).** When Algorithm (2.3.B) stops we have

\[ \Sigma_P \in CPWS(K, L_m(G)). \]

**Proof:** Let $r := |\Sigma|$ and $\Sigma'_P, P^iK$ and $P^iL_m(G)$ represent $\Sigma_P, PK$ and
\[ PL_m(G) \] respectively, after the $i$th event is tested in step 2 of the algorithm. Apparently $\Sigma'_P = \Sigma, P^0K = K, P^0L_m(G) = L_m(G)$ and $P^0L(G) = L(G) = \overline{P^0L_m(G)}$. Also $\Sigma'_P = \Sigma_P$. Then for $i = 1, \ldots, r$ we have

\[ \Sigma'_P \subseteq \Sigma'_P^{i-1}, \tag{2.3.14} \]

\[ P^iK = P_{\Sigma'_P^{i-1} \setminus \Sigma'_P} P^{i-1}K, \tag{2.3.15} \]

\[ P^iL_m(G) = P_{\Sigma'_P^{i-1} \setminus \Sigma'_P} P^{i-1}L_m(G), \tag{2.3.16} \]
\[ \Sigma'_p \in \text{CSWS}(P^{i-1}K, P^{i-1}L_m(G)), \quad (2.3.18) \]

and by recursive application of (2.3.15), (2.3.16) and (2.3.18) we have

\[ \Sigma'_p \in \text{CSWS}(P_{\Sigma, \Sigma'_p}, K, P_{\Sigma, \Sigma'_p}, L_m(G)). \quad (2.3.19) \]

First we use induction on \( i \) to show that

\[ \Sigma'_p \in \text{CSWS}(K, L_m(G)) \quad \text{for} \quad i = 1, \ldots, r. \quad (2.3.20) \]

**Induction basis:** \( i = 0. \) \( \Sigma'_0 := \Sigma \in \text{CSWS}(K, L_m(G)). \)

**Induction hypothesis:** \( \Sigma'_i \in \text{CSWS}(K, L_m(G)) \) for some \( i \in \{0, \ldots, r - 1\} \).

**Induction step:** Consider \( \Sigma'^{i+1}_p \). Then by (2.3.14) and (2.3.19) we have

\[ \Sigma'^{i+1}_p \subseteq \Sigma'_p \in \text{CSWS}(K, L_m(G)) \]

and

\[ \Sigma'^{i+1}_p \in \text{CSWS}(P_{\Sigma, \Sigma'_p}, K, P_{\Sigma, \Sigma'_p}, L_m(G)). \]

Using Proposition (2.3.14) we conclude that \( \Sigma'^{i+1}_p \in \text{CSWS}(K, L_m(G)) \). This proves (2.3.20).

Using (3.20) for the particular value \( i = r \) we obtain

\[ \Sigma_p = \Sigma'_p \in \text{CSWS}(K, L_m(G)). \]

In order to complete the proof it is enough to show that for any \( \alpha \in \Sigma'_p, \) \((\Sigma'_p - \alpha) \notin \text{CSWS}(K, L_m(G))\). Since \( r = |\Sigma|, \) every event, including \( \alpha, \) is tested once in the For loop of the algorithm. Assume that \( \alpha \) was tested in the \((n + 1)\)th run of the For loop. Since \( \alpha \in \Sigma'_p \subseteq \Sigma'^{n+1}_p, \alpha \) must have failed the If test in the algorithm and \( \Sigma'^{n+1}_p = \Sigma_n^p \). This implies that

\[ \Sigma'_p - \alpha \notin \text{CSWS}(P_{\Sigma, \Sigma'_p}, K, P_{\Sigma, \Sigma'_p}, L_m(G)). \]
\[ \Sigma_\rho^\omega - \alpha \not\in \text{CSWS}(K, L_m(G)). \]

Using the fact that \(\Sigma_\rho^\omega - \alpha \subseteq \Sigma_\rho^\omega - \alpha\) and Proposition (2.3.2) we conclude that \(\Sigma_\rho^\omega - \alpha \not\in \text{CSWS}(K, L_m(G))\). Therefore

\[ \Sigma_P = \Sigma_\rho^\omega \in \text{CPWS}(K, L_m(G)). \]

Algorithm (2.3.B) will terminate with the necessary and sufficient work space, if it exists. If such a necessary and sufficient work space does not exist, the algorithm yields a proper work space. However, there is no guarantee that the result is a minimal work space.

Computational complexity of Algorithm (2.3.B) is polynomial with respect to \(|\Sigma|, |Q|\) and exponential with respect to \(|X|\), where \(Q\) and \(X\) represent the state spaces of automata \(G\) and \(S\), respectively. To see this notice that the test of the algorithm can be alternatively written as follows:

\[ PK = PL_m(G) \cap P_{\Sigma_P,(\Sigma_P - \sigma)}^{-1} P_{\Sigma_P,(\Sigma_P - \sigma)} P K, \tag{2.3.21} \]

\[ \iff PK \supseteq PL_m(G) \cap P_{\Sigma_P,(\Sigma_P - \sigma)}^{-1} P_{\Sigma_P,(\Sigma_P - \sigma)} P K, \]

\[ \iff (\Sigma_P^* \setminus PK) \cap PL_m(G) \cap P_{\Sigma_P,(\Sigma_P - \sigma)}^{-1} P_{\Sigma_P,(\Sigma_P - \sigma)} P K = \emptyset. \]

We also have

\[ P K = PL(G) \cap P_{\Sigma_P,(\Sigma_P - \sigma)}^{-1} P_{\Sigma_P,(\Sigma_P - \sigma)} P K, \tag{2.3.22} \]

\[ \iff (\Sigma_P^* \setminus PK) \cap PL(G) \cap P_{\Sigma_P,(\Sigma_P - \sigma)}^{-1} P_{\Sigma_P,(\Sigma_P - \sigma)} P K = \emptyset. \]

It is possible to construct nondeterministic automata that represent the languages \(PL_m(G), P_{\Sigma_P,(\Sigma_P - \sigma)}^{-1} P_{\Sigma_P,(\Sigma_P - \sigma)} P K\), and their closed language coun-
2.4. Observation Spaces

Among the events in a proper work space of a supervisor (CPWS), there may be some events that need not be observed. Consider the subset of supC,NWS that was characterized in Proposition (2.3.10). The events in this subset are included in the sufficient work space because the assumption that a local supervisor can disable the controllable events in its work space is implicit in the normality condition. If we relax this requirement, i.e. we let a local supervisor disable any controllable event, including those that the supervisor does not observe, then by removing some controllable events from a given sufficient work space we can construct a smaller sufficient observation space.

Given a controllable sublanguage $K \subseteq L_m(G)$, consider following class of subalphabets.

**Definition (2.4.1).** *Class of Sufficient Observation Spaces:*

$$CSOS(K, L_m(G)) := \{ \Sigma_S \subseteq \Sigma \mid \ker P_{\Sigma_S} \subseteq \text{act}_{K, L_m(G)} \}.$$  \hfill (2.4.1)
\[ \Sigma \], by Theorem (2.2.1) we know that it is possible to design a supervisor with partial observation such that the closed loop system generates the language \( K \).

Proposition (2.4.2), which follows, states that in the partially ordered system \((CSOS; \subseteq)\) there exists many chains.

**Proposition (2.4.2).** (Proposition 2 of [13]). Given \( \Sigma_1 \subseteq \Sigma_2 \subseteq \Sigma \), \( \Sigma_1 \in CSOS \) implies \( \Sigma_2 \in CSOS \).

**Proof:** \( \ker P_{\Sigma_2} \leq \ker P_{\Sigma_1} \leq \text{act}_{K.L_m(G)} \).

\[ \Delta \]

**Corollary (2.4.3).** \( CSOS \) is closed under the union operation and its supremal element is \( \Sigma \).

\[ \Delta \]

However, \( CSOS \) need not be closed under the intersection operation and consequently, \( CSOS \) may not have an infimal element. Every chain in the partially ordered system \((CSOS, \subseteq)\) has a minimal element with respect to the cardinality metric and this minimal element does not properly contain any other sufficient observation space. Define the **Class of Proper Observation Spaces** (CPOS), the **Class of Minimal Observation Spaces** (CMOS) and the **Class of Necessary Observation Spaces** (CNOS) similar to those for work spaces.

The problem of designing a sufficient observation space was studied first by H. Cho [13]. He used the concept of control equivalence to construct a sufficient observation space for a given closed language. Let \( \mathcal{S} = (S, \varphi) \) be a supervisor
to be control-equivalent, written $x \sim y$, if for all $s \in \Sigma^*$, $\varphi(\xi(s, x)) = \varphi(\xi(s, y))$. Cho constructs a sufficient observation space by eliminating those events that produce transitions only among control-equivalent states of $X$. For example, if $\alpha \in \Sigma$ appears only as loop transitions, i.e. $x = \xi(\alpha, x)$ if $\xi(\alpha, x)$!, then $\alpha$ need not be observed by a supervisor because transitions under $\alpha$ do not change the control decisions. In order to find a smaller sufficient observation space, Cho suggests minimizing the cardinality of the state space of a given supervisor using techniques reported in [97] and then removing from $\Sigma$ all events that appear only as self loop transitions. The premise of this approach is that if a supervisor has a smaller state space, it is more likely to have more events appearing only as self loop transitions. Thus there is a chance to obtain a smaller sufficient observation space. However, this approach of constructing a sufficient observation space suffers from the following shortcomings:

(1) There is no guarantee that minimization of the state space of a supervisor renders more self loop transitions. Even though every projection induces a cover [97] on the state space of a supervisor, there is no guarantee that the cardinality of that cover is smaller than $|X|$. Indeed, it may be the case that the automaton representing $P_{\Sigma, \Sigma_M} K$, where $\Sigma_M$ is a minimal observation or work space, has a state space cardinality on the order of $2^{|X|}$. This means that to obtain a supervisor that has a large number of events appearing as only self loops, one must use a cover whose cardinality is on the order of $2^{|X|}$.

(2) There is no guarantee that the observation space obtained using the above procedure is a minimal or a proper observation space.

(3) There is no control over the events that will appear as self loop transitions after the state space reduction. That is, after reducing the state
(4) The above procedure does not have any computational advantage when compared to the search methods presented in this thesis. This is a result of the computational complexity of finding a minimal cover in the state reduction of a supervisor which is on the order of $2^{|X|}$ [97].

Our approach for constructing a sufficient observation space is based on examining the normality and the observability conditions. Observation spaces obtained by these algorithms are guaranteed to be at least proper. Because there is no restriction over the order of events that are tested in the algorithms, there is some control over the structure of the observation space which is obtained using these algorithms. To see this, assume that observing $\alpha \in \Sigma$ is very costly. Then we would prefer to test $\alpha \in \text{sup } CNOS$ first. If $\alpha$ happens to be unnecessary, we can reduce the event space and look for only those observation spaces that do not include $\alpha$. We also treat the general class of non-closed languages.

It is possible to develop an algorithm similar to Algorithm (2.3.A), for finding a minimal observation space. However, we intend to construct an observation space by first constructing a work space, and then reducing the work space to an observation space. This method has two advantages. First, not only will we know what events must be detected but also what controllable events must be regulated. Second, testing the normality condition is generally simpler than testing the observability condition.

If we assume $\Sigma_u = \emptyset$, Proposition (2.2.2) reduces to the following rela-
Thus the normality condition implies the observability condition. Therefore a sufficient work space is also a sufficient observation space. The question that we address is whether the procedure of finding a limit point of a chain in \((CSOS, \subseteq)\) can be decomposed into two steps: step 1) finding the limit point of a chain in \((CSWS, \subseteq)\) and step 2) removing those unnecessary events that at some times must be disabled, i.e. those events that we captured in Proposition (2.3.10)? The following proposition answers this question.

**Proposition (2.4.4).** Consider \(\hat{\Sigma} \in CPWS(K, L_m(G))\) and \(\alpha \in \hat{\Sigma}\). If \(\ker P_{\Sigma,(\hat{\Sigma} - \alpha)} \leq act_{K, L_m(G)}\) then the event \(\alpha\) must be disabled, at least once, by any supervisor.

**Proof:** The proof is contradiction. Assume on contrary that \(\alpha\) is never disabled. Then we have

\[
\hat{K}(\Sigma \setminus (\hat{\Sigma} - \alpha)) \cap L(G) \subseteq \hat{K}. \tag{2.4.3}
\]

Using (2.4.2), from (2.4.3) and the hypothesis \(\ker P_{\Sigma,(\hat{\Sigma} - \alpha)} \leq act_{K, L_m(G)}\), we conclude that

\[
\hat{K} = L_m(G) \cap P_{\Sigma,(\hat{\Sigma} - \alpha)}^{-1} P_{\Sigma,(\hat{\Sigma} - \alpha)} \hat{K}, \quad \text{and} \quad \hat{K} = L(G) \cap P_{\Sigma,(\hat{\Sigma} - \alpha)}^{-1} P_{\Sigma,(\hat{\Sigma} - \alpha)} \hat{K}.
\]

That is \((\hat{\Sigma} - \alpha) \in CSWS\). But this contradicts \(\hat{\Sigma} \in CPWS\) and the proof is complete.

\(\Delta\)
space that can be eliminated while the observability condition is still satisfied. every event in that subset is a controllable event that is disabled by any supervisor at some time. Therefore by finding first a limit point in \((CSWS, \subseteq)\), i.e. a proper work space, and then by removing a subset of events as characterized in Proposition \((2.3.10)\), we can obtain a limit point of \((CSOS, \subseteq)\), i.e. a proper observation space. In Algorithm \((2.4.A)\) we use the observability test to search for these events.

**Algorithm \((2.4.A)\).**

*Step 0:* Define the automatons \(G\) and \(S\) such that
\[ K = L_m(S) \subseteq L_m(G). \]

*Step 1:* Using Algorithm \((2.3.A)\) or \((2.3.B)\) obtain
\[ \Sigma_P \in CPWS(K, L_m(G)). \]

*Step 2:* Evaluate \(\Sigma_d\), where \(\Sigma_d \subseteq \sup CNWS \subseteq \Sigma_P\) as characterized in Proposition \((2.3.10)\).

*Step 3:* Assign \(\Sigma_O := \Sigma_P\).

*Step 4:* For \(n = |\Sigma_d|\) to 0

\[
\{ \text{For all } \hat{\Sigma} \in 2^{\Sigma_d} \text{ such that } |\hat{\Sigma}| = n \}
\{ \text{If } \ker P_{\Sigma, (\Sigma + \hat{\Sigma})} \leq \text{act}_K, L_m(G) \}
\{ \Sigma_O = \Sigma_O - \hat{\Sigma} \}
\text{STOP.}
\}
\]

*EndIf*

*EndFor*

*EndFor*

\[\Delta\]

When Algorithm \((2.4.A)\) stops, \(\Sigma_O\) will be a proper observation space. The computational complexity of testing the observability condition is polynomial
condition. If Algorithm (2.3.A) is used in step 1, the computational complexity of Algorithm (2.4.A) will be polynomial with respect to $|Q|$ and $|X|$ and exponential with respect to $|\Sigma|$.

In Algorithm (2.3.B), we eliminated an event $\alpha$ from the event space once $\alpha$ is found to be unnecessary using the normality condition. Then we continued the search in the reduced event space. It is of interest to see if we can do the same in Algorithm (2.4.A). The following proposition states that if $\alpha$ is unnecessary for normality, then we may reduce the event space by eliminating $\alpha$ and looking for an observation space in the reduced event space.

**Proposition (2.4.5).** If $\Sigma' \subseteq \hat{\Sigma} \in CSWS(K, L_m(G))$, Then

$$\Sigma' \in CSOS(K, L_m(G)) \iff \Sigma' \in CSOS(P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L_m(G)).$$

**Proof:**

(i) Proof of ($\iff$ direction):

Suppose $\Sigma' \not\in CSOS(P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L_m(G))$. Then there exist $s_1$ and $s_2$ in $\hat{\Sigma}$ such that

$$P_{\Sigma,\Sigma}s_1 = P_{\Sigma,\Sigma}s_2, \quad \text{and} \quad (s_1, s_2) \notin act_{P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L_m(G)}.$$

(Given (2.4.4), then either (2.4.5.A) or (2.4.5.B) is satisfied.

$$\sigma \in A_{P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L(G)}(s_1) \cap IA_{P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L(G)}(s_2), \quad \text{(2.4.5.A)}$$

$$s_1 \in P_{\Sigma,\Sigma}K \quad \text{and} \quad s_2 \in (P_{\Sigma,\Sigma}K \cap P_{\Sigma,\Sigma}L_m(G)) \setminus P_{\Sigma,\Sigma}K. \quad \text{(2.4.5.B)}$$

Suppose (2.4.5.A) is true, then

$$s_1 \sigma \in P_{\Sigma,\Sigma}K \quad \text{and} \quad s_2 \sigma \in P_{\Sigma,\Sigma}L(G) \setminus P_{\Sigma,\Sigma}K.$$
\[ w_1 \sigma \in P_{\Sigma, \Sigma'}^{-1}(s_1 \sigma) \cap \tilde{K} \quad \text{and} \quad w_2 \sigma \in P_{\Sigma, \Sigma'}^{-1}(s_2 \sigma) \cap L(G). \]

We then have \( P_{\Sigma, \Sigma'}(w_1) = P_{\Sigma, \Sigma'}(w_2) \), and \( w_1 \sigma \in \tilde{K}, \ w_2 \sigma \in L(G) \setminus \tilde{K} \). Thus \( \Sigma' \notin C\Sigma O S(K, L_m(G)) \).

On the other hand, suppose (2.4.5.B) is true. Then choose

\[ w_1 \in P_{\Sigma, \Sigma'}^{-1}s_1 \cap \tilde{K} \quad \text{and} \quad w_2 \in P_{\Sigma, \Sigma'}^{-1}s_2 \cap \tilde{K} \cap L_m(G). \quad (2.4.6) \]

From (2.4.5.B) choosing \( w_2 \) as in (2.4.6) is possible because

\[ \exists u \in \tilde{K} \setminus \tilde{K}, \ \exists v \in L_m(G) \setminus \tilde{K} \quad \text{such that} \quad P_{\Sigma, \Sigma'}u = P_{\Sigma, \Sigma'}v = s_2. \]

Then because \( \tilde{\Sigma} \in C\Sigma W S(K, L_m(G)) \), we have

\[ v \in L_m(G) \cap P_{\Sigma, \Sigma'}^{-1}P_{\Sigma, \Sigma'} \tilde{K} \subseteq L(G) \cap P_{\Sigma, \Sigma'}^{-1}P_{\Sigma, \Sigma'} \tilde{K} = \tilde{K}. \]

That is \( v \in \tilde{K} \cap L_m(G) \) and \( w_2 \) could be chosen as the string \( v \). Here we have \( w_1 \in \tilde{K}, \ w_2 \in (\tilde{K} \cap L_m(G)) \setminus \tilde{K} \) and \( P_{\Sigma, \Sigma'}w_1 = P_{\Sigma, \Sigma'}w_2 \). Thus \( \Sigma' \notin C\Sigma O S(K, L_m(G)) \). This completes the proof of \( \Longrightarrow \).

(ii) Proof of (\( \Longleftarrow \) direction):

Suppose \( \Sigma' \notin C\Sigma O S(K, L_m(G)) \). Then there exist \( w_1 \) and \( w_2 \) in \( \Sigma^* \) such that

\[ P_{\Sigma, \Sigma'}w_1 = P_{\Sigma, \Sigma'}w_2 \quad \text{and} \quad (w_1, w_2) \notin \text{act}_{K, L_m(G)}. \quad (2.4.7) \]

Given (2.4.7), then either (2.4.8.A) or (2.4.8.B) is satisfied.

\[ \sigma \in A_{K, L(G)}(w_1) \cap IA_{K, L(G)}(w_2), \quad (2.4.8.A) \]

\[ w_1 \in \tilde{K} \quad \text{and} \quad w_2 \in (\tilde{K} \cap L_m(G)) \setminus \tilde{K}, \quad (2.4.8.B) \]
that \( \sigma \in \Sigma \), or otherwise \( P_{\Sigma,\Sigma}(w_2 \sigma) = P_{\Sigma,\Sigma}(w_2) \), which implies \( w_2 \sigma \in \bar{K} \) because \( \hat{\Sigma} \in \text{CSWS}(K, L_m(G)) \). Let \( s_1 := P_{\Sigma,\Sigma}(w_1) \) and \( s_2 := P_{\Sigma,\Sigma}(w_2) \). Then \( P_{\Sigma,\Sigma}(s_1) = P_{\Sigma,\Sigma}(s_2) \). Furthermore, since \( \sigma \in \hat{\Sigma} \),

\[
s_1 \sigma = P_{\Sigma,\Sigma}(w_1 \sigma) \in P_{\Sigma,\Sigma} \bar{K} \quad \text{and} \quad s_2 \sigma = P_{\Sigma,\Sigma}(w_2 \sigma) \in P_{\Sigma,\Sigma} L(G),
\]

and since \( \hat{\Sigma} \in \text{CSWS}(K, L_m(G)) \),

\[
s_2 \sigma = P_{\Sigma,\Sigma}(w_2 \sigma) \notin P_{\Sigma,\Sigma} \bar{K}.
\]

This means that \( \Sigma' \notin \text{CSOS}(P_{\Sigma,\Sigma} K, P_{\Sigma,\Sigma} L_m(G)) \).

On the other hand, if (2.4.8.B) is satisfied, then we can choose

\[
s_1 := P_{\Sigma,\Sigma} w_1 \quad \text{and} \quad s_2 := P_{\Sigma,\Sigma} w_2.
\]

We have \( s_1 \in P_{\Sigma,\Sigma} K \) and

\[
s_2 \in P_{\Sigma,\Sigma} (K \cap L_m(G)) \subseteq P_{\Sigma,\Sigma} \bar{K} \cap P_{\Sigma,\Sigma} L_m(G).
\]

Also \( s_2 \notin P_{\Sigma,\Sigma} K \), otherwise by \( \hat{\Sigma} \in \text{CSWS}(K, L_m(G)) \) we could conclude that \( w_2 \in L_m(G) \cap P_{\Sigma,\Sigma} P_{\Sigma,\Sigma} K = K \), which contradicts (2.4.8.B). Therefore

\[
\Sigma' \notin \text{CSOS}(P_{\Sigma,\Sigma} K, P_{\Sigma,\Sigma} L_m(G)).
\]

This completes the proof.

\[\Delta\]

Proposition (2.4.5) implies that if an event \( \alpha \) is not necessary for the normality condition, i.e. \( \hat{\Sigma} = (\Sigma - \alpha) \in \text{CSWS}(K, L_m(G)) \), the event space can be reduced and one can search for an observation space contained in this
condition $\Sigma \in \text{CSWS}(K, L_m(G))$ can be replaced by $\Sigma \in \text{CSOS}(K, L_m(G))$.

However, this is not the case. To see this, suppose $L_m(G) = \overline{L(G)} = \overline{\alpha^* \beta \alpha^*}$ and $K = \overline{K} = \overline{\alpha^* \beta}$, where both events are controllable. It is obvious that $\{\beta\} \in \text{CSOS}(K, L_m(G))$. If the event space is reduced by only observing $\beta$, then $P_{\{\alpha, \beta\}, \{\beta\}} L_m(G) = \overline{\beta}$ and $P_{\{\alpha, \beta\}, \{\beta\}} K = \overline{\beta}$. That is

$$0 \in \text{CSOS}(P_{\{\alpha, \beta\}, \{\beta\}} K, P_{\{\alpha, \beta\}, \{\beta\}} L_m(G)).$$

But contrary to what is expected $0 \notin \text{CSOS}(K, L_m(G))$. The reason is that observing $\beta$ is necessary so that the event $\alpha$ can be disabled at the right time. The reduced event space does not have any information about the need to disable $\alpha$. As a result, the observation of $\beta$ appears to be unnecessary. Of course this is incorrect.

According to Proposition (2.4.5) the If statement in Algorithm (2.4.1) can be changed to the following statement

If $\ker P_{\Sigma^p, \Sigma^p - \overline{\Sigma}} \leq \text{act} P_{\Sigma, \Sigma} K, P_{\Sigma, \Sigma} L_m(G)$.

However this change does not remove the need for a search over $2^{2^d}$ in the algorithm. As such, this change does not generally have any computational advantage.

To complete our discussion of observation spaces, we now present some relationships that hold among observation spaces.

**Proposition (2.4.6).** If $\hat{\Sigma} \in \text{CSWS}(K, L_m(G))$, then

$$\sup \text{CNOS}(K, L_m(G)) \subseteq \sup \text{CNOS}(P_{\Sigma, \Sigma} K, P_{\Sigma, \Sigma} L_m(G)) \subseteq \hat{\Sigma}.$$

**Proof:**

$$\sup \text{CNOS}(K, L_m(G)) = \cap \{\Sigma' \subseteq \Sigma \mid \Sigma' \in \text{CSOS}(K, L_m(G))\},$$
by Proposition (2.4.5), we have
\[ \cap \{ \Sigma' \subseteq \hat{\Sigma} \mid \Sigma' \in CSOS(P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L_m(G)) \}, \]
\[ = \sup CNOS(P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L_m(G)). \]

\[ \Delta \]

**Proposition (2.4.7).** If \( \hat{\Sigma} \in CSWS(K, L_m(G)) \) and
\[ \Sigma' := \sup CNOS(K, L_m(G)) \in CSOS(K, L_m(G)). \]
then
\[ \Sigma' := \sup CNOS(K, L_m(G)) = \sup CNOS(P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L_m(G)). \]

**Proof:** By Proposition (2.4.5) we have \( \Sigma' \in CSOS(P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L_m(G)) \).
Thus \( \sup CNOS(P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L_m(G)) \subseteq \Sigma' \). By Proposition (2.4.6)
\[ \sup CNOS(P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L_m(G)) \supseteq \Sigma'. \]
Therefore
\[ \sup CNOS(P_{\Sigma,\Sigma}K, P_{\Sigma,\Sigma}L_m(G)) = \Sigma'. \]

\[ \Delta \]

### 2.5. Example

To illustrate the major results of this chapter, let
\[ \Sigma = \Sigma_c = \{ \alpha, \beta, \gamma, \theta, \eta \}. \]
Figure (2.5.1) and Figure (2.5.2) are diagrams representing the graphs. The final states are denoted by double circles in the figures.

Figure (2.5.1): Generator for $L_m(G)$.

Figure (2.5.2): Recognizer for $K$.

For this example, it is easy to check that the work spaces defined in this work are given by the following:

\[
\sup \text{CNWS}(K, L_m(G)) = \{ \eta \};
\]

\[
\text{CPWS}(K, L_m(G)) = \{ \{\beta, \theta, \eta\}, \{\gamma, \eta\} \};
\]

\[
\text{CMWS}(K, L_m(G)) = \{ \{\gamma, \eta\} \};
\]

\[
\text{CSWS}(K, L_m(G)) = \{ \hat{\Sigma} \in 2^\Sigma \mid \{\beta, \theta, \eta\} \subset \hat{\Sigma} \text{ or } \{\gamma, \eta\} \subset \hat{\Sigma} \}.
\]
We also have

\[
\sup CNWS(P_{E,(E-\beta)}K, P_{E,(E-\beta)}L_m(G)) = \{\gamma, \eta\};
\]
\[
\sup CNWS(P_{E,(E-\gamma)}K, P_{E,(E-\gamma)}L_m(G)) = \{\beta, \theta, \eta\}.
\]

We need only to check the event \( \eta \) to see if we can reduce any of the above proper work spaces to obtain a smaller observation space. This is the case because \( \eta \) is the only controllable event that is disabled. Consider
On the other hand $P_{\Sigma_{(\Sigma-\eta)}} \delta_1 = P_{\Sigma_{(\Sigma-\eta)}} \delta_2$. This means that

$$\ker P_{\Sigma_{(\Sigma-\eta)}} \not\subseteq \text{act}_{K,L_m(G)}.$$ 

We conclude that $\eta$ must be observed by any supervisor. Therefore we have

$$CPOS(K,L_m(G)) = CPWS(K,L_m(G)).$$

### 2.6. Applications

As we discussed in section (2.1), the main application of the results of this chapter are for reducting the number of event sensors, communication and command channels that are required between a plant and supervisor, to guarantee a desired closed loop behavior.

The results in this chapter could also be used in the context of supervisory control under partial observation (SCOP) [58]. In SCOP, the desired behavior is described by any language $K$ satisfying $\emptyset \neq L_a \subseteq K \subseteq L_g \subseteq L_m(G)$, where $L_a$ and $L_g$ model the minimum acceptable and maximum allowable behaviors respectively. SCOP has a solution if the infimal observable superlanguage of $L_a$ [83], which is denoted by $\inf \mathcal{O}(L_a)$, is contained in the supremal controllable sublanguage of $L_g$ [99], denoted by $\sup \mathcal{C}(L_g)$. If the controllability condition ($L_a \subseteq \sup \mathcal{C}(L_g)$) holds, there exists a supervisor which realizes the desired behavior if a sufficient set of observations is available. If the observability condition ($\inf \mathcal{O}(L_a) \subseteq \sup \mathcal{C}(L_g)$) fails and sufficient observations are not available, the required supervisor cannot be implemented. In this case, the supervisor design phase must be proceeded by a system alteration phase. Particularly for the open loop system, one must devise some additional detecting
information space. Our results can be used to determine an additional set of events that must be detected in order to have \( L_a = \inf \mathcal{O}(L_a) \). If this condition is satisfied then the supervisory control problem has a solution. Also, it is often desirable to realize exactly \( \sup C(L_g) \). This may require that additional events are detected so that \( \sup C(L_g) \) is an observable language. This requirement can be dealt with by using the results of this chapter.

Various types of decentralized supervision problem have been investigated in the literature. Using notions of normality, decomposability, and coobservability, necessary and sufficient conditions for the existence of decentralized supervision schemes have been developed [56, 59, 60, 61, 84]. If the desired local behaviors are such that the necessary and sufficient conditions for decentralized supervision do not hold, then partially decentralized schemes [Chapter 5] are preferred when compared to centralized schemes. To realize the desired behavior using a partially decentralized supervisor, it is necessary to determine those events for which communication to and command from a particular local supervisor is necessary. This problem is addressed in Chapter 5.

Even in cases where all events are observable and spatial constraints allow centralized supervision, those supervisors that observe just a subset of events—and yet supervise the same desired closed loop behavior—are generally more robust and fault-tolerant. Here, robustness means being less sensitive to failures of the detection system. Such reduced-event supervisors are tolerant of a failure of the detection system for an event which does not belong to the work-space or observation-space of the reduced-event supervisors. Consider the previous example and suppose that all events are detected. A usual procedure in designing a supervisor is to select a recognizer of \( K \) as the automaton part
Consider Figure (2.5.2). Suppose the current state of the DEDS is 1, and for some reason the detector of event $\beta$ does not work. If the next event happens to be $\beta$, the supervisor will fail because as far as the supervisor knows, no events have happened. Thus it waits for a signal to decide if it must go to state 2, or 5. However, based on our previous analysis we know that the observation of $\beta$ is not necessary. In Figure (2.5.4), a supervisor based on the proper work space $\{\gamma, \eta\}$ is depicted. This supervisor is tolerant with respect to failures of any event detector except the detectors for $\gamma$ and $\eta$. Thus supervisors based on proper observation spaces can have improved characteristics with respect to failures in the detection system. Indeed if we could know what detector fails and when it fails, then by switching among proper observation spaces, we could achieve the maximum possible fault tolerance for all the detecting systems except those that detect elements of $\text{sup CNOS}$. However, the assumption of having access to such information may not be very realistic.

### 2.7. Conclusions

In this chapter, we have defined several classes of work spaces and observation spaces for a supervisor. We have developed search procedures for determining work spaces and observation spaces. We have investigated the relationship between work spaces and observation spaces and we have also discussed the usefulness of work and observation spaces in reducing event sensors, communication and command channels between a plant and its supervisor. Applications of the sufficient work and observation spaces in the context of supervision under partial observation, partially decentralized supervision and
An Improved Suboptimal Solution for Supervisory Control and Observation Problem.

In some Discrete Event Dynamical Systems (DEDS) only partial observations of the system behavior are available. The supervisory Control and Observation Problem (SCOP) is of particular importance in these applications. In the DEDS literature, a necessary and sufficient condition for the existence of a solution for SCOP has been developed. Given a desired closed loop language $\mathcal{K}$, SCOP has a solution if and only if $\mathcal{K}$ is both observable and controllable. There are two possible ways to deal with situations where $\mathcal{K}$ does not satisfy this necessary and sufficient condition. One approach is to enhance the information that is gathered from the DEDS such that the condition is satisfied given the new information set. This approach requires detecting more events in the DEDS, and we addressed this problem in the previous chapter. In the other more traditional approach, it is desirable to synthesize the largest observable and controllable sublanguage of $\mathcal{K}$. Unfortunately, the set of all observable and controllable sublanguages of $\mathcal{K}$, denoted by $\mathcal{DD}(\mathcal{K})$, is not closed under the set union operation. Thus in this approach, an optimal solution does not exist. An adopted method to deal with this problem is to consider the set of all $L_m(G)$-closed sublanguages of $\mathcal{K}$ whose prefix closures are normal and controllable, denoted by $\mathcal{DD}'(\mathcal{K})$. $\mathcal{DD}'(\mathcal{K})$ is a proper subset of $\mathcal{DD}(\mathcal{K})$ and is closed under the set union operation. Therefore $\mathcal{DD}'(\mathcal{K})$ contains a unique supremal
controllable, consequently, it is a suboptimal solution for SCOP.

In this chapter we introduce an extended normality property for non-closed languages. We use this property to construct an improved suboptimal solution for SCOP. We also construct an algorithm to evaluate such an improved suboptimal solution.

3.1. Introduction

Discrete Event Dynamic Systems (DEDS) have received considerable attention in the control literature recently. Ramadge and Wonham's [80, 99] framework for modeling and supervisory control of DEDS has proven to be effective. In this framework, a DEDS is modeled by a generator $G := (Q, \Sigma, \delta, q_0, Q_m)$ where $Q$ is the state set, $\Sigma$ is a finite alphabet or set of event symbols, $\delta : \Sigma \times Q \rightarrow Q$ is a partial transition function, $q_0$ is the initial state and $Q_m$ is the set of final states which can represent, for example, the completion of some tasks in the plant. While under no external control, a generator starts at the initial state and generates events in the course of transiting from one state to the other. This evolution is governed by the transition function. The behavior of a generator can be described by the set of all event sequences that it can generate and the set of all event sequences that transfer the initial state of the plant to a final, or marked, state.

Recall that $\Sigma^*$ denotes the set of finite length strings of the events in $\Sigma$, including the null string denoted by $\epsilon$. The extension of the transition function

---
\[ \delta(\epsilon, q) = q, \quad \delta(s\sigma, q) = \delta(\sigma, \delta(s, q)) \quad \text{for } \sigma \in \Sigma, s \in \Sigma^* \text{ and } q \in Q. \]

The notation \( \ell(s, q)! \) denotes that the transition from state \( q \) under the string \( s \) is defined. The open loop behavior of the plant is modeled by the formal languages

\[ L(G) = \{ s \in \Sigma^* \mid \delta(s, q_0)! \}, \]
\[ L_m(G) = \{ s \in L(G) \mid \delta(s, q_0) \in Q_m \}. \]

In this chapter we assume that the plant generator is trim, i.e. \( \overline{L_m(G)} = L(G) \). For a given language \( L \), \( \hat{L} \) stands for the prefix closure of \( L \), i.e. \( \hat{L} \) contains all prefixes of all words in \( L \). For a word \( s \in \Sigma^* \), \( \hat{s} \) denotes \( \{s\} \). When a plant is trim every sequence of events in \( L(G) \) can be completed to a finished task, i.e. a string in \( L_m(G) \).

The set \( \Sigma \) is partitioned into two sets \( \Sigma_c \) and \( \Sigma_u \). \( \Sigma_c \) is the set of controllable events whose occurrences could be prevented by a supervisor. \( \Sigma_u \) denotes the set of uncontrollable events. The desired behavior of a DEDS is generally characterized by a language \( K \) satisfying \( L_a \subseteq K \subseteq L_g \subseteq L_m(G) \subseteq \Sigma^* \). Where \( L_c \) and \( L_g \) model the smallest acceptable language and the largest allowable language, respectively.

A supervisor \( S = (S, \varphi) \) consists of a finite automata

\[ S := (X, \Sigma, \xi, x_0, X_m) \]

and a feedback map \( \varphi : X \to 2^\Sigma \) where \( 2^\Sigma \) denotes the power set of \( \Sigma \). When a generator is coupled with a supervisor, the supervisor receives events generated by \( G \), and these cause a state transition in the supervisor automata \( S \). In the
possible next events that the generator can generate while complying with the
transition function $\delta$. In this way the supervisor $S$ controls the behavior of
$G$. Since elements of $\Sigma_u$ cannot be prevented from happening, we require
$\Sigma_u \subseteq \varphi(x)$ for all $x \in X$. A supervisor is called proper if it can respond to any
string that the closed loop system can generate.

The closed loop system constructed in this way is itself an extended
generator whose behavior is denoted by the languages $L(S/G)$ and $L_m(S/G)$.

By introducing the notion of language-controllability, Ramadge and Won-
ham have developed a necessary and sufficient condition for the existence of a
supervisor [80]. Mathematically, this condition requires that $L_a$ is contained
in the supremal controllable sublanguage of $L_g$, denoted by $\sup \mathcal{C}(L_g)$. That is

$$L_a \subseteq \sup \mathcal{C}(L_g) := \sup \{ L \subseteq L_g \mid \hat{L}_\Sigma_u \cap L(G) \subseteq \hat{L} \}, \quad (3.1.1)$$

where in (3.1.1) the condition $\hat{L}_\Sigma_u \cap L(G) \subseteq \hat{L}$ pertains to the controllability
of the language $L$. When condition (3.1.1) holds, one approach is to design a
supervisor such that $L_m(S) = K := \sup \mathcal{C}(L_g)$ and $\varphi(x) = \{ \sigma \in \Sigma \mid \xi(\sigma, x)! \}$. Then we have $L_m(S/G) = K$ and $L(S/G) = K$.

In many cases, the supervisor cannot observe the occurrence of all events.
These cases can arise when detecting some events, if not impossible, is very
costly. The problem of supervision under partial observation (SCOP) is inves-
tigated in [17, 58]. In [58], a partial observation is modelled by a projection. In
this chapter, we also use a projection to model a partial observation. However,
the results of this chapter are also valid if a partial observation is modelled by
a general image map.

Formally, given $\Sigma_1$ and $\Sigma_2$ where $\Sigma_2 \subseteq \Sigma_1 \subseteq \Sigma$, let $P_{\Sigma_1,\Sigma_2} : \Sigma_1^* \longrightarrow \Sigma_2^*$ be
\[ P_{\Sigma_1, \Sigma_2} \epsilon = \epsilon; \]
\[ P_{\Sigma_1, \Sigma_2} \sigma = \epsilon, \quad \text{for } \sigma \in \Sigma_1 - \Sigma_2; \]
\[ P_{\Sigma_1, \Sigma_2} \sigma = \sigma, \quad \text{for } \sigma \in \Sigma_2; \]
\[ P_{\Sigma_1, \Sigma_2} (s \sigma) = (P_{\Sigma_1, \Sigma_2} s) (P_{\Sigma_1, \Sigma_2} \sigma), \quad \text{for } s \in \Sigma_1^*, \sigma \in \Sigma_1. \]

The projection \( P_{\Sigma_1, \Sigma_2} \) simply erases those events of \( s \) that are not in \( \Sigma_2 \). Given the projection \( P_{\Sigma_1, \Sigma_2} \), the inverse projection \( P_{\Sigma_1, \Sigma_2}^{-1} : \Sigma_2^* \rightarrow \Sigma_1^* \) is defined as:

\[ \text{for } s \in \Sigma_2^*, \quad P_{\Sigma_1, \Sigma_2}^{-1} s = \{ t \in \Sigma_1^* \mid P_{\Sigma_1, \Sigma_2} t = s \}. \]

The projection (or inverse projection) of a language is defined to be the union of the projections (or inverse projections) of all words in the language. If a language is closed (or non-closed) then its projection as well as its inverse projection are closed (or non-closed).

Given a language \( K \) and a projection \( P_{\Sigma, \Sigma} \), Lin and Wonham [58] have developed the notion of language-observability. \( K \) is called \( P_{\Sigma, \Sigma} \)-observable or simply observable if and only if

\[ \ker P_{\Sigma, \Sigma} \leq \text{act}_{K, \text{act}_m(G)} \quad \text{or} \quad (s_1, s_2) \in \ker P_{\Sigma, \Sigma} \implies (s_1, s_2) \in \text{act}_{K, \text{act}_m(G)}. \]  

(3.1.2)

Where we have

\[ (s_1, s_2) \in \ker P_{\Sigma, \Sigma} \iff P_{\Sigma, \Sigma} s_1 = P_{\Sigma, \Sigma} s_2, \]  

(3.1.3)

and

\[ (s_1, s_2) \in \text{act}_{K, \text{act}_m(G)} \iff (A) \text{ and } (B), \]

where

\[ \begin{align*}
A) \quad & A_{K, \text{act}_m(G)} (s_1) \cap \text{IA}_{K, \text{act}_m(G)} (s_2) = \emptyset = \text{IA}_{K, \text{act}_m(G)} (s_1) \cap A_{K, \text{act}_m(G)} (s_2), \\
& \quad \text{and} \\
B) \quad & s_1 \in \hat{K} \cap \text{act}_m(G) \text{ and } s_2 \in \hat{K} \cap \text{act}_m(G) \implies (s_1 \in K \iff s_2 \in K). 
\end{align*} \]

(3.1.4)
are denoted by \( A_{\mathcal{K},L(G)}(s) \) and \( IA_{\mathcal{K},L(G)}(s) \) respectively, and are defined by

\[
A_{\mathcal{K},L(G)}(s) := \begin{cases} 
\{ \sigma \mid s\sigma \in \mathcal{K} \}, & \text{if } s \in \mathcal{K}; \\
\emptyset, & \text{otherwise.}
\end{cases}
\quad (3.1.5)
\]

\[
IA_{\mathcal{K},L(G)}(s) := \begin{cases} 
\{ \alpha \mid s\alpha \in L(G) \setminus \mathcal{K} \}, & \text{if } s \in \mathcal{K}; \\
\emptyset, & \text{otherwise.}
\end{cases}
\quad (3.1.6)
\]

Given a projection \( P_{\Sigma,\Sigma} \) and \( L_a \subseteq L_g \subseteq L_m(G) \), the SCOP problem requires finding a language \( \mathcal{K} \) such that \( L_a \subseteq \mathcal{K} \subseteq L_g \) and \( \mathcal{K} \) is both controllable and \( P_{\Sigma,\Sigma} \)-observable. The SCOP has a solution if and only if

\[
\inf O(L_a) \subseteq \sup C(L_g).
\quad (3.1.7)
\]

where

\[
O(L_a) := \{ M \supseteq L_a \mid \ker P_{\Sigma,\Sigma} \leq \text{act}_{M,L_m(G)} \}.
\quad (3.1.8)
\]

Even though condition (3.1.7) is a necessary and sufficient condition for the existence of a solution for SCOP, it does not yield such a solution. Consider the following set

\[
\mathcal{D}(L_g) := \{ M \subseteq L_g \mid \hat{M} \Sigma_a \cap L(G) \subseteq \hat{M} \text{ and } \ker P_{\Sigma,\Sigma} \leq \text{act}_{M,L_m(G)} \}.
\quad (3.1.9)
\]

\( \mathcal{D}(L_g) \) is not closed under set union operation. Thus, in general, the optimal solution for the SCOP does not exist.

A number of suboptimal solutions for the SCOP are investigated in the literature [14, 15, 17, 58]. One procedure that can be used for both closed and non-closed languages is based on the \( \textit{normality} \) condition. Given a projection \( P_{\Sigma,\Sigma} \) and a language \( M \), \( M \) is called \( \textit{normal} \) if

\[
M = L(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} M.
\quad (3.1.10)
\]
If condition (3.1.11) holds, a supervisor can be designed such that the closed
loop system generates \( K := \sup \mathcal{DD}'(L_g) \). Important characteristics of such a
supervisor are: 1) It is \((\Sigma \setminus \hat{\Sigma})\)-null. In the other words, the supervisor needs
to observe only those events in \( \hat{\Sigma} \). 2) The supervisor needs to disable only the
events in \( \hat{\Sigma} \cap \Sigma_c \). We call such a supervisor a local supervisor with local work
space \( \hat{\Sigma} \). [14] presents algorithms to compute \( \sup \mathcal{DD}'(L_g) \).

In this chapter we develop an improved suboptimal solution for the SCOP.
This solution is developed by finding a controllable and observable language
that contains \( \sup \mathcal{DD}'(L_g) \). For this we use the extended normality condition
that we introduced in the previous chapter.

The reminder of the chapter is organized as follows: In section 3.2 we
review the extended normality property for non-closed languages and compare
it to the observability condition. Using this property, we develop an improved
suboptimal solution for the SCOP. In section 3.3, using two algorithms that
are currently in the supervisory control literature, we develop an algorithm
to evaluate the proposed suboptimal solution for the SCOP. In section 3.4 we
present an example demonstrating the results. A summary of the contributions
of this chapter is given in section 3.5.
Consider a non-closed language $K \subseteq L_m(G) \subseteq L(G)$ and a projection $P_{\Sigma, \Sigma}$. We introduce the following property for the language $K$.

**Definition (3.2.1).**

$$
\begin{align*}
&\text{(A) } \hat{K} = L(G) \cap P_{\Sigma, \Sigma}^{-1}P_{\Sigma, \Sigma}\hat{K}, \\
&\text{(B) } K = L_m(G) \cap P_{\Sigma, \Sigma}^{-1}P_{\Sigma, \Sigma}K.
\end{align*}
$$

($K$ is $(L_m(G), L(G), P_{\Sigma, \Sigma})$ - normal) $\iff$ Condition (3.2.1)

This condition implies that knowledge about $L(G)$ and $L_m(G)$ is sufficient for eliminating the ambiguity that results from the partial observation through the projection $P_{\Sigma, \Sigma}$ in the languages $\hat{K}$ and $K$. In general, conditions (3.2.1.A) and (3.2.1.B) are independent. However, if $K$ is closed condition (3.2.1.A) implies condition (3.2.1.B). To see this, suppose that (3.2.1.A) holds. Then

$$
L_m(G) \cap P_{\Sigma, \Sigma}^{-1}P_{\Sigma, \Sigma}K \subseteq L(G) \cap P_{\Sigma, \Sigma}^{-1}P_{\Sigma, \Sigma}\hat{K} = \hat{K} = K.
$$

$L_m(G) \cap P_{\Sigma, \Sigma}^{-1}P_{\Sigma, \Sigma}K \supseteq K$ always holds and thus (3.2.1.B) is obtained. Therefore, for closed languages, condition (3.2.1.A) suffices. Condition (3.2.1.A) has been extensively used in the DEDS literature. Both conditions (3.2.1.A) and (3.2.1.B) are used to derive the new results given in this chapter. Next, we establish the relationship between the normality condition given in (3.2.1) and those introduced in the literature.

**Proposition (3.2.2).** Given $K \subseteq L_m(G)$ and a projection $P_{\Sigma, \Sigma}$,

$$(L_m(G), L(G), P_{\Sigma, \Sigma})$$

is a stronger condition than $P_{\Sigma, \Sigma}$-observability.
implies $P_{\Sigma,\Sigma}$-observability. To complete the proof of this proposition we need to show that the converse is not true. To show this we give a counter example. Let $L(G) := \alpha \beta$, $L_m(G) := \alpha$, $K := \alpha$ and $\Sigma := \{\alpha\}$. For this example the observability condition holds. To see this, note that $K = \{\epsilon, \alpha\}$ and we have

$$A_{K,L(G)}(\epsilon) \cap I A_{K,L(G)}(\alpha) = \{\alpha\} \cap \{\beta\} = \emptyset,$$

$$I A_{K,L(G)}(\epsilon) \cap A_{K,L(G)}(\alpha) = \emptyset \cap \emptyset = \emptyset.$$

Thus (3.1.4.A) holds. (3.1.4.B) also holds because $K = L_m(G)$. Therefore the observability condition holds. But

$$L(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K = \alpha \beta \cap (\beta^*(\epsilon + \alpha)\beta^*) = \alpha \beta \neq K.$$ 

Therefore (3.2.1) does not hold. This completes the proof.

$$\Delta$$

In [58] it is shown that if $K = L(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K$ and $K = K \cap L_m(G)$ then $K$ is $P_{\Sigma,\Sigma}$-observable. This condition is used in [14, 58] to develop a suboptimal solution for the SCOP. The next proposition characterizes the relationship between these two conditions and the $(L_m(G), L(G), P_{\Sigma,\Sigma})$-normality condition given by (3.2.1).

**Proposition (3.2.3).** (i) $K = L(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K$ and (ii) $K = K \cap L_m(G)$ together are stronger conditions than $(L_m(G), L(G), P_{\Sigma,\Sigma})$-normality of $K$.

**Proof:** Condition (3.2.1.A) follows immediately from (i). To show that (3.2.1.B) is implied by (i) and (ii) we proceed as follows:

$$L_m(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K$$
\[ \subseteq L(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} \hat{K}, \]
\[ \subseteq \hat{K}, \quad \text{(by (i))} \quad (3.2.3) \]
\[ \subseteq \hat{K} \cap L_m(G), \quad \text{(by (3.2.2) and (3.2.3))} \]
\[ \subseteq K. \quad \text{(by (ii))} \]

Because it is always true that \( K \subseteq L_m(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K \), we conclude that
\[ K = L_m(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K. \]

To complete the proof we need to show that the two sets of conditions are not identical. For this, we provide the following counter example. Consider the case that \( L(G) = \alpha \beta \gamma \), \( L_m(G) = \alpha + \alpha \beta \gamma \) and \( K = \alpha \beta \gamma \). Let \( \hat{\Sigma} = \{ \alpha, \beta \} \). Then the \( (L_m(G), L(G), P_{\Sigma,\Sigma}) \)-normality condition is satisfied because
\[ L(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} \hat{K} = \alpha \beta \gamma \cap \gamma^* \alpha \gamma^* \beta \gamma^* = \alpha \beta \gamma = \hat{K}, \]
and
\[ L_m(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} K = (\alpha + \alpha \beta \gamma) \cap (\gamma^* \alpha \gamma^* \beta \gamma^*) = \alpha \beta \gamma = K. \]

However, we have
\[ \hat{K} = L(G) \implies \hat{K} \cap L_m(G) = L_m(G) \neq K. \]

Therefore \( K \) is not \( L_m(G) \)-closed. This completes the proof.

\( \triangle \)

Given \( L_g \subseteq L_m(G) \), define the following set of sublanguages.

\[ \mathcal{DD}_m(L_g) := \{ M \subseteq L_g \mid \tilde{M} \Sigma u \cap L(G) \subseteq \tilde{M}, \ M = L_m(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} M \text{ and } \tilde{M} = L(G) \cap P_{\Sigma,\Sigma}^{-1} P_{\Sigma,\Sigma} \tilde{M} \} \quad (3.2.4) \]
Proposition (3.2.4).

\[ \mathcal{DD}'(L_g) \subseteq \mathcal{DD}_m(L_g) \subseteq \mathcal{DD}(L_g). \]  

(3.2.5)

It is easy to see that \( \mathcal{DD}_m(L_g) \) is closed under the set union operation. All members of \( \mathcal{DD}_m(L_g) \) are sublanguages of \( L_g \). Therefore \( \mathcal{DD}_m(L_g) \) contains a unique supremal element defined by:

\[ \sup \mathcal{DD}_m(L_g) := \bigcup \{ M \mid M \in \mathcal{DD}_m(L_g) \} \in \mathcal{DD}_m(L_g). \]  

(3.2.6)

The following proposition characterizes a sufficient condition for the solvability of the SCOP.

Proposition (3.2.5). The SCOP has a solution if \( L_a \subseteq \sup \mathcal{DD}_m(L_g) \). Moreover, \( \sup \mathcal{DD}_m(L_g) \) is an improved solution to the SCOP, that is \( \sup \mathcal{DD}'(L_g) \subseteq \sup \mathcal{DD}_m(L_g) \).

Proof: We have \( L_a \subseteq \sup \mathcal{DD}_m(L_g) \subseteq L_g \) and \( \sup \mathcal{DD}_m(L_g) \in \mathcal{DD}_m(L_g) \subseteq \mathcal{DD}(L_g) \), i.e. \( \sup \mathcal{DD}_m(L_g) \) is both controllable and observable. Thus it is a solution for the SCOP. Furthermore, an immediate result of (3.2.5) is that

\[ \sup \mathcal{DD}'(L_g) \subseteq \sup \mathcal{DD}_m(L_g). \]  

(3.2.7)

Therefore \( \sup \mathcal{DD}_m(L_g) \) is an improved solution.

(3.2.8)
3.3. An algorithm for the suboptimal solution

In this section we develop an algorithm to evaluate \( \mathcal{DD}_m(L) \). We need to refer to the following sets of sublanguages.

\[
\mathcal{DD}_m(L) := \{ M \subseteq L \mid \bar{M} \Sigma_u \cap L(G) \subseteq \bar{M}, M = L_m(G) \cap P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} M \text{ and } \bar{M} = L(G) \cap P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} \bar{M} \}.
\]

\[
\mathcal{D}_m(L) := \{ M \subseteq L \mid M = L_m(G) \cap P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} M \}.
\]

\[
\mathcal{D}'(L) := \{ M \subseteq L \mid \bar{M} \Sigma_u \cap L(G) \subseteq \bar{M} \text{ and } \bar{M} = L(G) \cap P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} \bar{M} \}.
\]

Notice that each one of the above sets is closed under the set union operation and thus has a unique supremal element. [14] presents an algorithm for evaluating \( \sup \mathcal{D}'(L) \). [40] presents an algorithm for evaluating \( \sup \mathcal{D}_m(L) \). We combine these two algorithms to evaluate \( \sup \mathcal{DD}_m(L) \). Specifically, we will show that

\[
\sup \mathcal{DD}_m(L) = \sup \mathcal{D}'(\sup \mathcal{D}_m(L)).
\]

The following lemmas are required.

**Lemma (3.3.1).** \( \sup \mathcal{DD}_m(L) = \sup \mathcal{DD}(\sup \mathcal{D}_m(L)) \).

**Proof:** It is obvious that \( \mathcal{DD}_m(L) \subseteq \mathcal{D}_m(L) \). Thus \( \sup \mathcal{DD}_m(L) \subseteq \sup \mathcal{D}_m(L) \). Therefore \( \sup \mathcal{DD}_m(L) \in \mathcal{DD}_m(\sup \mathcal{D}_m(L)) \). Consequently

\[
\sup \mathcal{DD}_m(L) \subseteq \sup \mathcal{DD}_m(\sup \mathcal{D}_m(L)). \quad (3.3.1)
\]
\[ \sup \mathcal{D}_m(\sup \mathcal{D}_m(L)) \subseteq \sup \mathcal{D}_m(L). \quad (3.3.2) \]

(3.3.1) and (3.3.2) together establish the lemma.

\[ \triangle \]

**Lemma (3.3.2).** If \( L = L_m(G) \cap P_{\Sigma}^{-1} P_{\Sigma} L \), then

\[ \sup \mathcal{D}'(L) = L_m(G) \cap P_{\Sigma}^{-1} P_{\Sigma} \sup \mathcal{D}'(L). \]

**Proof:** Trivially we have

\[ \sup \mathcal{D}'(L) \subseteq L_m(G) \cap P_{\Sigma}^{-1} P_{\Sigma} \sup \mathcal{D}'(L). \quad (3.3.3) \]

Thus we need only to show

\[ \sup \mathcal{D}'(L) \supseteq L_m(G) \cap P_{\Sigma}^{-1} P_{\Sigma} \sup \mathcal{D}'(L). \quad (3.3.4) \]

To establish (3.3.4), take

\[ s \in L_m(G) \cap P_{\Sigma}^{-1} P_{\Sigma} \sup \mathcal{D}'(L). \quad (3.3.5) \]

Then

\[ s \in L(G) \cap P_{\Sigma}^{-1} P_{\Sigma} \mathcal{D}'(L) = \mathcal{D}'(L). \quad (3.3.6) \]

Thus we have

\[ s \in \mathcal{D}'(L). \quad (3.3.7) \]

Moreover, because \( \sup \mathcal{D}'(L) \subseteq L \), using (3.3.5), we conclude that

\[ s \in L_m(G) \cap P_{\Sigma}^{-1} P_{\Sigma} L, \quad (3.3.8) \]
\( s \in L. \) \hspace{1cm} (3.3.9)

Now, define the following language

\[ Q := \{s\} \cup \sup D'(L). \] \hspace{1cm} (3.3.10)

From (3.3.7) and (3.3.10) we have that

\[ Q \subseteq L \quad \text{and} \quad \bar{Q} = \sup D'(L). \] \hspace{1cm} (3.3.11)

Therefore \( Q \in D'(L) \) and this yields

\[ Q \subseteq \sup D'(L). \] \hspace{1cm} (3.3.12)

From (3.3.10) and (3.3.12) we conclude that

\[ Q = \sup D'(L) \quad \text{and} \quad s \in \sup D'(L). \] \hspace{1cm} (3.3.13)

(3.3.5) and (3.3.13) together prove (3.3.4), and (3.3.3) and (3.3.4) together complete the proof of the lemma.

\[ \triangle \]

Lemma (3.3.3). If \( L = L_m(G) \cap P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} L \), then

\[ \sup DD_m(L) = \sup D'(L). \]

Proof: Using the fact that \( \sup D'(L) \in D'(L) \) and Lemma (3.3.2) we conclude that

\[ \sup D'(L) \in DD_m(L). \] \hspace{1cm} (3.3.14)
\[ \sup \mathcal{D}'(L) \subseteq \sup \mathcal{D}_m(L). \tag{3.3.15} \]

On the other hand \( \mathcal{D}_m(L) \subseteq \mathcal{D}'(L) \). Thus

\[ \sup \mathcal{D}_m(L) \subseteq \sup \mathcal{D}'(L). \tag{3.3.16} \]

(3.3.15) and (3.3.16) together establish the proof of the lemma.

\[ \Delta \]

**Proposition (3.3.4).** \[ \sup \mathcal{D}_m(L) = \sup \mathcal{D}'(\sup \mathcal{D}_m(L)). \]

**Proof:** By Lemma (3.3.1) we have

\[ \sup \mathcal{D}_m(L) = \sup \mathcal{D}_m(\sup \mathcal{D}_m(L)). \tag{3.3.17} \]

By the definition of \( \mathcal{D}_m(L) \) we have

\[ \sup \mathcal{D}_m(L) = \mathcal{L}_m(G) \cap \mathcal{P}_\Sigma \mathcal{P}_\Lambda \sup \mathcal{D}_m(L). \tag{3.3.18} \]

By (3.3.18) and Lemma (3.3.3) we conclude that

\[ \sup \mathcal{D}_m(\sup \mathcal{D}_m(L)) = \sup \mathcal{D}'(\sup \mathcal{D}_m(L)). \tag{3.3.19} \]

Using (3.3.17) and (3.3.19) we have

\[ \sup \mathcal{D}_m(L) = \sup \mathcal{D}'(\sup \mathcal{D}_m(L)). \]

This completes the proof.

\[ \Delta \]

Proposition (3.3.5) provides some insight into the structure of \( \sup \mathcal{D}_m(L) \).
then $\Omega_m(L) = \sup D_m(L)$.

**Proof:** We first show that

$$\sup D_m(L) \subseteq \Omega_m(L). \tag{3.3.20}$$

Take $s \in \sup D_m(L)$, then

$$s \in L. \tag{3.3.21}$$

Also

$$L_m(G) \cap P_{\Sigma,\Sigma}^{-1}P_{\Sigma,\Sigma}s \subseteq L_m(G) \cap P_{\Sigma,\Sigma}^{-1}P_{\Sigma,\Sigma}\sup D_m(L) = \sup D_m(L) \subseteq L. \tag{3.3.22}$$

From (3.3.21) and (3.3.22) we conclude that $s \in \Omega_m(L)$.

Next we show that

$$\Omega_m(L) \subseteq \sup D_m(L). \tag{3.3.23}$$

We begin by showing that $\Omega_m(L) \in D_m(L)$. $\Omega_m(L) \subseteq L$ and trivially $\Omega_m(L) \subseteq L_m(G) \cap P_{\Sigma,\Sigma}^{-1}P_{\Sigma,\Sigma}\Omega_m(L)$, we need only to show that

$$\Omega_m(L) \supseteq L_m(G) \cap P_{\Sigma,\Sigma}^{-1}P_{\Sigma,\Sigma}\Omega_m(L). \tag{3.3.24}$$

Take

$$u \in L_m(G) \cap P_{\Sigma,\Sigma}^{-1}P_{\Sigma,\Sigma}\Omega_m(L).$$

This implies that $u \in L_m(G)$ and there exists $t \in \Omega_m(L)$ such that $P_{\Sigma,\Sigma}u = P_{\Sigma,\Sigma}t$. With $t \in \Omega_m(L)$ we have

$$u \in L_m(G) \cap P_{\Sigma,\Sigma}^{-1}P_{\Sigma,\Sigma}\Omega_m(L). \tag{3.3.25}$$

Also

$$L_m(G) \cap P_{\Sigma,\Sigma}^{-1}P_{\Sigma,\Sigma}u = L_m(G) \cap P_{\Sigma,\Sigma}^{-1}P_{\Sigma,\Sigma}t \subseteq L.$$
Therefore
\[ \Omega_m(L) \subseteq \sup D_m(L). \]

This completes the proof
\[ \triangle \]

Using Proposition (3.3.4) to evaluate \( \sup DD_m(L) \), we can first determine \( N := \sup D_m(L) \) and then evaluate \( \sup D'(N) \).

Given a language \( L \), [14] presents an algorithm that evaluates \( \sup D'(L) \). Also, an algorithm that evaluates \( \sup D_m(L) \) is presented in [40]. The combination of these two algorithms yields an algorithm for evaluation of \( \sup DD_m(L) \). If the languages are regular and therefore can be represented by Deterministic Finite Automata (DFA), the algorithms given in [14] and [40] are effective.

3.4. Example

In this example we consider the SCOP where \( L_m(G) = (\beta + \alpha \gamma)^\ast(\epsilon + \alpha \beta + \eta \eta) \). Assume all events are controllable, i.e. \( \Sigma_c = \Sigma = \{\alpha, \beta, \gamma, \eta\} \). Here, the event \( \gamma \) is not observed, i.e. \( \tilde{\Sigma} = \{\alpha, \beta, \eta\} \). Suppose that the minimum acceptable behavior is \( L_a := \eta \eta \) and the maximum allowable behavior is \( L_g := (\alpha \gamma)^\ast(\alpha \beta + \eta \eta) \). Figure (3.4.1) and Figure (3.4.2) depict recognizers for \( L_m(G) \) and \( L_g \), respectively. In the figures, final states are denoted by double circles.
Figure (3.4.1): $L_m(G)$ and $L(G) = \overline{L_m(G)}$.

Figure (3.4.2): $L_g$ and $\overline{L_g}$.

Note that for this example $\epsilon \in L_m(G) \setminus L_g$ and therefore no sublanguages of $L_g$ are $L_m(G)$-closed. Thus $\sup D\mathcal{D}'(L_g) = \emptyset$. That is,

$$\sup\{M \subseteq L_g \mid \bar{M} \cap L_m(G) = M\} = \emptyset,$$

$$\sup D\mathcal{D}'(L_g) = \emptyset.$$

However, we will see that $\sup D\mathcal{D}_m(L_g)$ is non-empty and consequently using the results of this thesis, the SCOP has a solution.

Notice that $L_g \neq L_m(G) \cap P_{\Sigma^*}^{-1}P_{\Sigma^*}L_g$. Thus we need to find the largest sublanguage of $L_g$ that satisfies the required condition (3.2.1). In Figure (3.4.3) we have depicted a recognizer for $\sup D_m(L_g)$. This recognizer can be effectively constructed using [Theorem 4.3, 40].
Figure (3.4.3). sup $D_m(L_g)$.

Because all events are controllable, the language $sup D_m(L_g)$ is also controllable. It can be shown that $sup D_m(L_g)$ satisfies the normality condition (3.2.1.A) with respect to $L(G)$ and $P_{\Sigma,E}$. Therefore $sup D_m(L_g)$ is a member of $DD_m(L_g)$ and must be identical to $sup DD_m(L_g)$. We also have $L_a \subset sup DD_m(L) \subset L_g$. Thus, from our results, the SCOP has a solution and $sup DD_m(L)$ is a suboptimal solution to this problem.

3.5. Conclusions

In this chapter, we have introduced an extended normality property for non-closed languages. We used this property to derive an improved sufficient condition for the existence of a solution for the SCOP. In the course of testing this sufficient condition, a suboptimal solution for the SCOP is also constructed. Using already existing algorithms, we can evaluate the new suboptimal solution. An example demonstrating the results was also provided.
Testing the Normality and Observability Conditions.

In this chapter we present two algorithms to test the normality and observability conditions for regular languages. We also show that the algorithms are of polynomial time complexity. We measure the complexity of an algorithm using the cardinality of the state space for the input finite automata and the events set.

4.1. Introduction

Let Σ be a nonempty finite set of events, called an alphabet. The set of all finite strings—words—of Σ is denoted by Σ*, where Σ* also includes the null string ε. For a word s ∈ Σ*, by |s|, and s we denote the number of events in s, and the set of all prefixes of string s, respectively. Thus |ε| = 0, and ε ∈ s for any string s. We also use |·| to denote set cardinality. A subset L of Σ* is called a formal language—or simply a language. A prefix-closure of L, denoted by $\bar{L}$, is a language containing all prefixes of words in L. A language L is said to be closed if $L = \bar{L}$. A language L is called regular if there is a finite automaton generating or recognizing L. Throughout this chapter we only consider regular languages.
\( G := (Q, \Sigma, \delta, q_0, Q_m) \),

where \( Q \) is the finite set of states, \( q_0 \in Q \) is the initial state and \( Q_m \) is the set of final states. The transition function is \( \delta : \Sigma \times Q \rightarrow 2^Q \). In general, \( \delta \), is a partial function, i.e. for some states and for some events no transition is defined. For \( \Sigma' \subseteq \Sigma \) and \( q \in Q \), let

\[
\delta(\Sigma', q) := \bigcup_{\sigma \in \Sigma'} \delta(\sigma, q).
\]

\( \delta \) is also extended to \( \Sigma^* \times Q \) in the standard way and the extended map is denoted by the same symbol, \( \delta \). For \( s \in \Sigma^* \), the notation \( \delta(s, q)! \) denotes the case that \( \delta(s, q) \neq \emptyset \). For graphical representation, sometimes we use \( q \xrightarrow{\sigma} \hat{q}\) (in \( G \)) to denote the fact that \( \hat{q} \in \delta(s, q) \). We also associate with an automaton \( G \) the next event map \( d : 2^Q \rightarrow 2^\Sigma \) defined by:

\[
d(\hat{Q}) := \bigcup_{q \in Q} \{ \sigma \in \Sigma \mid \delta(\sigma, q)! \},
\]

where \( \hat{Q} \subseteq Q \). \( G \) is said to be a Deterministic Finite Automaton (DFA) if for any \( \sigma \in \Sigma \) and \( q \in Q \), \( \delta(\sigma, q) \) is a singleton set when the transition is defined.

The language generated or recognized by \( G \) is defined by:

\[
L_m(G) := \{ w \in \Sigma^* \mid \delta(w, q_0) \cap Q_m \neq \emptyset \}.
\]

An automaton is said to be accessible if for all \( q \in Q \) there exists \( s \in \Sigma^* \) such that \( q \in \delta(s, q_0) \).

Formally, given \( \Sigma_1 \) and \( \Sigma_2 \) where \( \Sigma_2 \subseteq \Sigma_1 \subseteq \Sigma \), let \( P_{\Sigma_1, \Sigma_2} : \Sigma_1^* \rightarrow \Sigma_2^* \) be a projection defined by:

\[
P_{\Sigma_1, \Sigma_2} \epsilon = \epsilon;
\]

\[
P_{\Sigma_1, \Sigma_2} \sigma = \epsilon, \quad \text{for } \sigma \in \Sigma_1 - \Sigma_2;
\]

\[
P_{\Sigma_1, \Sigma_2} \sigma = \sigma, \quad \text{for } \sigma \in \Sigma_2;
\]

\[
P_{\Sigma_1, \Sigma_2}(s\sigma) = (P_{\Sigma_1, \Sigma_2}s)(P_{\Sigma_1, \Sigma_2}\sigma), \quad \text{for } s \in \Sigma_1^*, \ \sigma \in \Sigma_1.
\]
Given the projection $P_{\Sigma_1, \Sigma_2}$, the inverse projection $P_{\Sigma_1, \Sigma_2}^{-1} : \Sigma_2^* \rightarrow \Sigma_1^*$ is defined as:

$$\text{for } s \in \Sigma_2^*, \quad P_{\Sigma_1, \Sigma_2}^{-1}s = \{ t \in \Sigma_1^* \mid P_{\Sigma_1, \Sigma_2}t = s \}.$$ 

The projection (or inverse projection) of a language is defined to be the union of the projections (or inverse projections) of all words in the language. If a language is closed (or non-closed) then its projection as well as its inverse projection are closed (or non-closed).

In our development of some algorithms to test the normality and observability conditions, we need to use some fairly standard algorithms that have been presented in the automata and supervisory control literature. Here, we list these algorithms for later reference. An algorithm, operating on regular languages, is called effective if it operates on the attributes (i.e. states, events, ...) of the finite automaton representation of the regular languages. All of the algorithms that we present in this chapter are effective. We also discuss the computational time complexity of the algorithms that we develop. A unit of time in our discussion is the time that an algorithm spends in checking a state or an event. We give conservative estimates for the computational time complexity of the algorithms. Our main objective here is to show that the algorithms are all of polynomial time complexity.

Given NFA $G = (Q, \Sigma, \delta, q_0, Q_m)$, $\hat{Q} \subseteq Q$ and $\hat{\Sigma} \subseteq \Sigma$, define the reach of $\hat{Q}$ under $\hat{\Sigma}$ in $G$ as follows:

$$R(\hat{Q}, \hat{\Sigma}, G) := \{ q \in Q \mid \exists \hat{q} \in \hat{Q} \text{ and } \exists s \in \hat{\Sigma}^* \text{ such that } q \in \delta(s, \hat{q}) \}.$$ 

The following algorithm evaluates the reach of a subset of states in a NFA under transitions belonging to a subalphabet.
\[ R(\text{state-set } Q_0, \text{subalphabet } \Sigma_0, \text{NFA } G); \quad \text{return state-set} \]

\{

\text{Step 0: initialize } G := (Q, \Sigma, \delta, q_0, Q_m),
\quad Q_0 \subseteq Q, \quad \Sigma_0 \subseteq \Sigma.

\text{Step 1: let } R_0 = Q_0 = X_0,

\text{Step 2: iterate } R_{k+1} = R_k \cup \delta(\Sigma_0, Q_k),
\quad Q_{k+1} = R_{k+1} \cap (Q \setminus R_k),
\quad \text{If } \ (R_{k+1} = R_k)
\quad \{
\quad \text{return } R_{k+1},
\quad \text{STOP.}
\quad \}

\}

\[ \Delta \]

\textbf{Remark (4.1.1).} The computational time complexity of } R(Q_0, \Sigma_0, G) \text{ is } O(|\Sigma_0| |Q|^3) \text{ if } G \text{ is a NFA and } O(|\Sigma_0| |Q|^2) \text{ if } G \text{ is a DFA.}

\[ \Delta \]

To verify Remark (4.1.1), assume that in step 2 we iterate } m \text{-times. Each iteration requires } |\Sigma_0||Q_k| \text{ evaluations of } \delta(\cdot, \cdot) \text{ to completely specify } \delta(\Sigma_0, Q_k). \text{ Each such evaluation, in the worst case, renders a set with cardinality equal to } |Q| \text{ (the result of an evaluation is a singleton set in the case of DFAs). The results must be individually intersected with } R_k \text{ to evaluate } Q_{k+1} \text{ and } R_{k+1}. \text{ Such intersections require } |Q||R_k| \text{ (only } |R_k| \text{ in the case of DFAs). Thus the}
\[
\text{num. of visits} = \sum_{r=0}^{m} |\Sigma_0||Q_r||Q||R_k|.
\]

The following two relations also hold,

\[
|R_0| < |R_1| < \cdots \leq |R_m| \leq |Q|,
\]
\[
|Q_0| + |Q_1| + \cdots + |Q_m| \leq |Q|.
\]

Thus

\[
\text{num. of visits} \leq |\Sigma_0||Q|^2 \sum_{r=0}^{m} |Q_r|,
\]
\[
\leq |\Sigma_0||Q|^3.
\]

Frequently, operations that we perform on automata render an automaton that is not accessible. For later operations we need to evaluate the accessible part of the resultant automaton. The following algorithm evaluates the accessible part of a given NFA.

**Algorithm (4.1.B).** Evaluating the accessible part of an automaton.

```plaintext
accessible(NFA G); return NFA
{
  Step 0: initialize G := (Q, \Sigma, \delta, q_0, Q_m),
  Step 1: X := R(q_0, \Sigma, G),
  Step 2: return (X, \Sigma, \delta |_{X, q_0, Q_m \cap X}).
}
```

\(\triangle\)

**Remark (4.1.2).** Computational time complexity of accessible(G) is \(O(|\Sigma||Q|^3)\) if \(G\) is a NFA and \(O(|\Sigma||Q|^2)\) if \(G\) is a DFA.
Algorithm (4.1.C). Evaluating the intersection of two languages $L_1, L_2 \subseteq \Sigma^*$.

intersect (NFA $A_1$, NFA $A_2$); return NFA

{
Step 0: initialize $A_1 := (Q_1, \Sigma, \delta_1, q_{10}, Q_{1m})$, $L_m(A_1) := L_1,$

$A_2 := (Q_2, \Sigma, \delta_2, q_{20}, Q_{2m})$, $L_m(A_2) := L_2,$

Step 1: construct

$A_1 \times A_2 := (Q_1 \times Q_2, \Sigma, \delta_1 \times \delta_2, (q_{10}, q_{20}), Q_{1m} \times Q_{2m})$

as follows,

$(\delta_1 \times \delta_2)(\sigma, (q_1, q_2)) := \begin{cases} 
\delta_1(\sigma, q_1) \times \delta_2(\sigma, q_2), & \text{if } \delta_1(\sigma, q_1)! \text{ and } \delta_2(\sigma, q_2)!, \\
\text{undefined}, & \text{otherwise,}
\end{cases}$

Step 2: $A_1 \times A_2$ accessible $(A_1 \times A_2)$,

Step 3: return $A_1 \times A_2$.

}\n
Remark (4.1.3). The computational time complexity of intersect($A_1, A_2$) is $O(|\Sigma||Q_1|^3|Q_2|^3)$ if the automatons are NFAs and $O(|\Sigma||Q_1|^2|Q_2|^2)$ if they are DFAs.
complement (DFA $A$); return DFA

{
Step 0: initialize $A := (Q, \Sigma, \delta, q_0, Q_m)$, $L_m(A) := L$.
Step 1: construct
$$A^c := (Q \cup \{\star\}, \Sigma, \delta^c, q_0, (Q \setminus Q_m) \cup \{\star\})$$
as follows,
$$\delta^c(\sigma, \{\star\}) = \{\star\} \quad \text{for all } \sigma \in \Sigma,$$
$$\delta^c(\sigma, q) := \begin{cases} 
\delta(\sigma, q), & \text{if } \delta(\sigma, q) \notin L_m(A), \\
\{\star\}, & \text{otherwise},
\end{cases}$$
Step 2: $A^c = \text{accessible} (A^c)$,
Step 3: return $A^c$.

}$

Remark (4.1.4). The computational time complexity of $\text{complement}(A)$ is $O(|\Sigma|(|Q| + 1)^2)$.

Given a NFA $G$ as above, define the inverse automaton
$$G^{-1} := (Q, \Sigma, \delta^{-1}, q_0, Q_m)$$
where we simply reverse the direction of transitions in $G$. Notice that the initial state $q_0$ does not have any meaning in $G^{-1}$. We include $q_0$ just to keep track of the initial state in the original automaton. Similarly, $d^{-1}(\cdot)$ denotes the next events map associated with $G^{-1}$. The next algorithm inverts a given NFA.
invert (NFA A); return NFA
{
Step 0: initialize \( A := (Q, \Sigma, \delta, q_0, Q_m) \), \( L_m(A) := L \),
Step 1: construct
\[ A^{-1} := (Q, \Sigma, \delta^{-1}, q_0, Q_m) \] as follows,
\[ \delta^{-1}(\sigma, q) := \{ q' \in Q \mid q \in \delta(\sigma, q') \} \],
Step 2: return \( A^{-1} \).
}

Remark (4.1.5). The computational time complexity of \( \text{invert}(A) \) is \( O(|Q|^2) \).

The following algorithm modifies a NFA by deleting some transitions of the NFA.

Algorithm (4.1.F). Delete some transitions from a given NFA.

delete (subalphabet \( \Sigma_d \), NFA A); return NFA
{
Step 0: initialize \( A := (Q, \Sigma, \delta, q_0, Q_m) \), \( \Sigma_d \),
Step 1: construct
\[ A_{\text{mod.} \Sigma_d} := (Q, \Sigma \setminus \Sigma_d, \delta_{\Sigma \setminus \Sigma_d}, q_0, Q_m) \] as follows,
\[ \delta_{\Sigma \setminus \Sigma_d}(\sigma, q) := \begin{cases} 
\delta(\sigma, q), & \text{if } \delta(\sigma, q)! \text{ and } \sigma \notin \Sigma_d, \\
\text{undefined.} & \text{otherwise},
\end{cases} \]
**Remark (4.1.6).** The computational time complexity of $\text{delete}$$\left(\Sigma_d, A\right)$ is $O(\left|\Sigma \setminus \Sigma_d\right|\left|Q\right|^3)$ if $A$ is a NFA and $O(\left|\Sigma \setminus \Sigma_d\right|\left|Q\right|^2)$ if it is a DFA.

For our development we need the notion of a strict-subautomaton, which is introduced next.

Consider two DFAs $A = (Q_A, \Sigma, \delta_A, q_{A_0}, Q_{A_m})$ and $B = (Q_B, \Sigma, \delta_B, q_{B_0}, Q_{B_m})$ where following conditions hold:

1. $L(B) \subseteq L(A)$,
2. $L_m(B) \subseteq L_m(A)$,
3. $q_{A_0} = q_{B_0}$,
4. $Q_B \subseteq Q_A$.

$B$ is a subautomaton of $A$ if (i) for all $s \in L(B)$ we have $\delta_B(s, q_{B_0}) = \delta_A(s, q_{A_0})$.

Also, $B$ is a strict-subautomaton of $A$ if, in addition to (i), (ii) if $s \in L(A)$ and $s \notin L(B)$ then there exists $\hat{s} \in \hat{s}$ such that $\delta_A(\hat{s}, q_{A_0}) \notin Q_B$.

Given two DFAs $G = (Q, \Sigma, \delta, q_0, Q_m)$ and $S = (X, \Sigma, \xi, x_0, X_m)$ where $L(S) \subseteq L(G)$ and $L_m(S) \subseteq L_m(G)$, the following algorithm constructs $\hat{G} = (\hat{Q}, \Sigma, \hat{\delta}, \hat{q}_0, \hat{Q}_m)$ and $\hat{S} = (\hat{X}, \Sigma, \hat{\xi}, \hat{x}_0, \hat{X}_m)$ such that $L(\hat{G}) = L(G)$, $L_m(\hat{G}) = L_m(G)$, $L(\hat{S}) = L(S)$ and $L_m(\hat{S}) = L_m(S)$. Furthermore, $\hat{S}$ is a strict-subautomaton of $\hat{G}$.
subautomaton \((DFA \ G, DFA \ S)\); \return DFA \ \tilde{G}, DFA \ \tilde{S}

\{

Step 0: initialize \( G := (Q, \Sigma, \delta, q_0, Q_m) \),
\[ S := (X, \Sigma, \xi, x_0, X_m), \quad L_m(S) \subseteq L_m(G), \]

Step 1: construct \( \tilde{G} = (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{q}_0, \tilde{Q}_m) \) as follows,

\[ \tilde{Q} = Q \times (X \cup \{\ast\}), \quad \tilde{Q}_m = Q_m \times (X \cup \{\ast\}), \quad \tilde{q}_0 = (q_0, x_0), \]
\[ \tilde{\delta}(\sigma, (\tilde{q}, \tilde{x})) := \begin{cases} 
(\delta(\sigma, \tilde{q}), \xi(\sigma, \tilde{x})), & \text{if } \tilde{x} \in X \text{ and } \delta(\sigma, \tilde{q})! \text{ and } \xi(\sigma, \tilde{x})!; \\
(\delta(\sigma, \tilde{q}), \ast), & \text{if } (\tilde{x} \not\in X \text{ or } \sim \xi(\sigma, \tilde{x})!) \text{ and } \delta(\sigma, \tilde{q})!; \\
\text{undefined}, & \text{otherwise.} 
\end{cases} \]

Step 2: construct \( \tilde{S} = (\tilde{X}, \Sigma, \tilde{\xi}, \tilde{x}_0, \tilde{X}_m) \) as follows,

\[ \tilde{X} = Q \times X, \quad \tilde{X}_m = Q_m \times X_m, \quad \tilde{x}_0 = (q_0, x_0), \]
\[ \tilde{\xi}(\sigma, (\tilde{q}, \tilde{x})) := \begin{cases} 
(\delta(\sigma, \tilde{q}), \xi(\sigma, \tilde{x})), & \text{if } \delta(\sigma, \tilde{q})! \text{ and } \xi(\sigma, \tilde{x})!; \\
\text{undefined}, & \text{otherwise.} 
\end{cases} \]

Step 3: \return \tilde{G} and \tilde{S}.
\}

\[ \triangle \]

Remark (4.1.7). The computational time complexity of \text{subautomaton}(G, S) is \( O(|\Sigma||Q|(|X| + 1)) \).
Next we present an algorithm that evaluates the projection of a given regular language. The algorithm is initialized by a DFA recognizing the input language and a projection. The algorithm yields a NFA recognizing the projected language. The state space cardinality of the NFA is equal to that of the input DFA. This prevents subsequent algorithms from having to search an exponential state space that a deterministic projection procedure may yield.

**Algorithm (4.1.H).** Constructing the projection of a DFA.

```
project ( projection \( P_{\Sigma, \Sigma} \), DFA \( A \) ); return NFA
{
    Step 0: initialize \( A = (Q, \Sigma, \delta, q_0, Q_m) \) and projection \( P_{\Sigma, \Sigma} \).
    Step 1: If \( (R(q_0, \Sigma \setminus \hat{\Sigma}, A) \cap Q_m \neq \emptyset) \)
        
        \[
        Q_m = Q_m \cup \{q_0\},
        \]
        
    } EndIf
    Step 2: construct NFA \( A_{1, \Sigma} := (Q, \Sigma, \delta_{1, \Sigma}, q_0, Q_m) \) as follows,

\[
\delta_{1, \Sigma}(\sigma, q) := \begin{cases} 
    R(\delta(\sigma, q), \Sigma \setminus \hat{\Sigma}, A), & \text{if } \sigma \in \hat{\Sigma} \text{ and } \delta(\sigma, q); \\
    \delta(\sigma, q), & \text{if } \sigma \in \Sigma \setminus \hat{\Sigma} \text{ and } \delta(\sigma, q); \\
    \text{undefined,} & \text{otherwise.}
\end{cases}
\]

    Step 3: construct NFA \( A_{2, \Sigma} := (Q, \Sigma, \delta_{2, \Sigma}, q_0, Q_m) \) as follows,

\[
A_{2, \Sigma} = A_{1, \Sigma}^{-1},
= (Q, \Sigma, \delta_{1, \Sigma}^{-1}, q_0, Q_m),
= \text{invert}(A_{1, \Sigma}),
\]

    Step 4: construct NFA \( A_{3, \Sigma} := (Q, \Sigma, \delta_{3, \Sigma}, q_0, Q_m) \) as follows,
```
Step 5: construct NFA $A_{4,\Sigma} := (Q, \Sigma, \delta_{4,\Sigma}, q_0, Q_m)$ as follows,

$$A_{4,\Sigma} = A_{3,\Sigma}^{-1},$$

$$= (Q, \Sigma, \delta_{3,\Sigma}^{-1}, q_0, Q_m),$$

$$= \text{invert}(A_{3,\Sigma}).$$

Step 6: construct NFA

$$A_{n,\Sigma} := (Q, \Sigma, \delta_{4,\Sigma}, q_0, Q_m),$$

$$= \text{delete}(\Sigma, A_{4,\Sigma}).$$

Step 7: return NFA $A_{n,\Sigma}$.}

\[\triangle\]

Lemma (4.1.8). Project$(P_{\Sigma,\Sigma}, A)$ is a polynomial-time algorithm with complexity $O(|\Sigma|^2|Q|^4)$.

Proof: Steps 2 and 4 determine the complexity of the algorithm. In each step, at the worst case, we must evaluate $|\Sigma||Q|$-times the reach operation, where the worst case complexity is $O(|\Sigma||Q|^3)$. Thus the complexity of the algorithm is of the order of $O(|\Sigma|^2|Q|^4)$. However, the state space cardinality of the result is $|Q|$.

\[\triangle\]

Lemma (4.1.9). Given the DFA $A$ and the NFA $A_{n,\Sigma}$ as above, the following properties hold:

(i) $L(A_{n,\Sigma}) = P_{\Sigma,\Sigma}L(A)$. 

...
(iii) \( q_0 \xrightarrow{a} q(\text{in } A) \) and \( |P_{\Sigma,\Sigma}s| \geq 1 \) implies

\[ q_0 \xrightarrow{P_{\Sigma,\Sigma}a} q(\text{in } A_{n,\Sigma}). \]

(iv) \( q_0 \xrightarrow{t} q(\text{in } A_{n,P}) \) and \(|t| \geq 1 \) implies

\[ \exists s \in \Sigma^* \text{ such that } t = P_{\Sigma,\Sigma}s \text{ and } q_0 \xrightarrow{s} q(\text{in } A). \]

**Proof:** (i) and (ii) immediately follow from (iii), (iv) and Step 1 of the algorithm. We only present proofs of (iii) and (iv).

(iii) We use induction on the length of the words in \( L(A) \).

**Induction basis:** Take \( \sigma \in \Sigma \) such that \( P_{\Sigma,\Sigma}\sigma = \sigma \) and \( q_0 \xrightarrow{\sigma} q(\text{in } A) \). In all steps of Algorithm (4.1.H) we keep this transition. Thus \( q_0 \xrightarrow{\sigma = P_{\Sigma,\Sigma}\sigma} q(\text{in } A_{n,\Sigma}) \).

**Induction hypothesis:** Given \( q_0 \xrightarrow{a} q(\text{in } A) \) where \( |P_{\Sigma,\Sigma}s| \geq 1 \) and \(|s| \leq n\), then

\[ q_0 \xrightarrow{P_{\Sigma,\Sigma}a} q(\text{in } A_{n,\Sigma}). \]

**Induction step:** Take \( s\sigma \in L(A) \). We delineate the following four cases:

**Case 1:** \( P_{\Sigma,\Sigma}s = \epsilon \) and \( P_{\Sigma,\Sigma}\sigma = \sigma \).

We have

\[ q_0 \xrightarrow{a} q' \xrightarrow{\sigma} q''(\text{in } A), \]

\[ q_0 \xleftarrow{a} q' \xleftarrow{\sigma} q''(\text{in } A_{2,\Sigma}), \]

\[ q_0 \xrightarrow{\sigma} q''(\text{in } A_{3,\Sigma}), \]

thus

\[ q_0 \xrightarrow{\sigma} q''(\text{in } A_{4,\Sigma} \text{ and } A_{n,\Sigma}). \]
Then we can write $s = s_1 \alpha s_2$, where $|P_{\Sigma, \Sigma} s_1 \alpha| \geq 1$, $\alpha \in \hat{\Sigma}$ and $s_2 \in (\Sigma \setminus \hat{\Sigma})^*$.

We have

$q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2 \xrightarrow{\sigma} q_3 (\text{in } A)$,

$q_1 \xrightarrow{\sigma_1} q_2 \xrightarrow{\sigma} q_3 (\text{in } A_1, \Sigma)$,

$q_1 \xrightarrow{\sigma_2} q_2 \xrightarrow{\sigma} q_3 (\text{in } A_2, \Sigma)$,

$q_1 \xrightarrow{\sigma} q_3 (\text{in } A_3, \Sigma)$,

$q_1 \xrightarrow{\sigma} q_3 (\text{in } A_4, \Sigma \text{ and } A_n, \Sigma)$.

By the induction hypothesis we have

$q_0 \xrightarrow{P_{\Sigma, \Sigma} \sigma_1} q_1 (\text{in } A_n, \Sigma)$.

Thus

$q_0 \xrightarrow{(P_{\Sigma, \Sigma} \sigma_1) \sigma} q_3 (\text{in } A_n, \Sigma)$,

which results in

$q_0 \xrightarrow{P_{\Sigma, \Sigma} \sigma_2} q_3 (\text{in } A_n, \Sigma)$.

Case 3: $|P_{\Sigma, \Sigma} s| > 1$ and $P_{\Sigma, \Sigma} \sigma = \epsilon$.

Then we can write $s\sigma = s_1 \alpha s_2$, where $|P_{\Sigma, \Sigma} s_1 \alpha| \geq 1$, $\alpha \in \hat{\Sigma}$ and $s_2 \in (\Sigma \setminus \hat{\Sigma})^*$.

We have

$q_0 \xrightarrow{\sigma_1} \hat{q}_1 \xrightarrow{\alpha} \hat{q}_2 \xrightarrow{\sigma_2} \hat{q}_3 (\text{in } A)$,

$\hat{q}_1 \xrightarrow{\alpha} \hat{q}_3 (\text{in } A_1, \Sigma \text{ and } A_n, \Sigma)$.

By the induction hypothesis we have

$q_0 \xrightarrow{P_{\Sigma, \Sigma} \sigma_1} \hat{q}_1 (\text{in } A_n, \Sigma)$,

which results in

$q_0 \xrightarrow{(P_{\Sigma, \Sigma} \sigma_1) \alpha} \hat{q}_3 (\text{in } A_n, \Sigma)$,

or

$q_0 \xrightarrow{P_{\Sigma, \Sigma} \sigma_2} \hat{q}_3 (\text{in } A_n, \Sigma)$.

Case 4: $|P_{\Sigma, \Sigma} s| = 1$ and $P_{\Sigma, \Sigma} \sigma = \epsilon$. 


We have

\[ q_0 \xrightarrow{s_1} \tilde{q}_1 \xrightarrow{\alpha} \tilde{q}_2 \xrightarrow{s_2} \tilde{q}_3 \text{ (in } A_1, \Sigma) \],

\[ q_0 \xrightarrow{s_1} \tilde{q}_1 \xrightarrow{\alpha} \tilde{q}_3 \text{ (in } A_{1, \Sigma}) \],

\[ q_0 \xrightarrow{s_1} \tilde{q}_1 \xrightarrow{\alpha} \tilde{q}_3 \text{ (in } A_{2, \Sigma}) \],

\[ q_0 \xrightarrow{\alpha} \tilde{q}_3 \text{ (in } A_{3, \Sigma}) \],

\[ q_0 \xrightarrow{\alpha} \tilde{q}_3 \text{ (in } A_{4, \Sigma} \text{ and } A_{n, \Sigma}) \],

which yields \[ q_0 \xrightarrow{P_{\Sigma, \Sigma} \varepsilon} \tilde{q}_3 \text{ (in } A_{n, \Sigma}) \].

This completes the proof of (iii).

(iv) Notice that

If \[ q' \xrightarrow{\alpha} q'' \text{ (in } A_{n, \Sigma}) \]

then \[ \exists \{s_1, s_2\} \subset (\Sigma \setminus \hat{\Sigma})^* \text{ and } \exists \{q_1, q_2\} \subset Q \text{ such that} \]

\[ q' \xrightarrow{s_1} q_1 \xrightarrow{\alpha} q_2 \xrightarrow{s_2} q'' \text{ (in } A) \].

To see this consider the different cases that would result in the transition \[ q' \xrightarrow{\alpha} q'' \text{ (in } A_{n, \Sigma}) \]. This transition would be the result of an identical transition in \( A \). In this case \( s_1 = s_2 = \varepsilon \) and \( q_1 = q' \) and \( q_2 = q'' \). If the transition is created in step 2 of Algorithm (4.1.H), then \( s_1 = \varepsilon \) and \( q_1 = q' \). If the transition created in step 4 of Algorithm (4.1.H), then \( s_2 = \varepsilon \) and \( q_2 = q'' \). Finally, if the transition is a result of joint actions in step 2 and step 4, then we have the most general form. These are all possibilities for creation of a transition. This proves (iv).

\[ \triangle \]

Step 2 in Algorithm (4.1.H) is not necessary for (i) and (ii) of Lemma (4.1.9). However, for (iii) and (iv) of Lemma (4.1.9), which are frequently used later, we must use step 2 in the algorithm.
\textbf{inversepp} (projection $P_{\Sigma, \Sigma}$, DFA $A$); return NFA \\
\{ \\
\textit{Step 0}: initialize $A = (Q, \Sigma, \delta, q_0, Q_m)$ and projection $P_{\Sigma, \Sigma}$, \\
\textit{Step 1}: evaluate $A_{n, \Sigma} := (Q, \overset{\wedge}{\Sigma}, \delta_{n, \Sigma}, q_0, Q_m) := \text{project}(P_{\Sigma, \Sigma}, A)$, \\
\textit{Step 2}: construct NFA $A_{n, \Sigma} := (Q, \Sigma, \delta_{n, \Sigma}, q_0, Q_m)$ as follows, \\
\[
\delta_{n, \Sigma}(\sigma, q) := \begin{cases} \\
\delta_{n, \Sigma}(\sigma, q) & \text{if } \sigma \in \overset{\wedge}{\Sigma} \text{ and } \delta(\sigma, q) \text{ is defined}; \\
\{q\} & \text{if } \sigma \in \Sigma \setminus \overset{\wedge}{\Sigma}; \\
\text{undefined, otherwise.} & \\
\end{cases} \\
\]
\textit{Step 3}: return NFA $A_{n, \Sigma}$.
\} \\
\triangle

\textbf{Remark (4.1.10).} \textbf{inversepp}($P_{\Sigma, \Sigma}, A$) is a polynomial time algorithm with complexity $O(|\Sigma|^2 |Q|^4)$.

\triangle

Using the above algorithms, in the next section we develop a polynomial time algorithm to test the normality condition.

\section*{4.2. An algorithm to test the normality condition}

Testing the normality condition is studied in this subsection. The normality condition that we have used in this dissertation is composed of the following two conditions.
\[ B) \quad K = L(G) \cap P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} K, \]

Since a closed language is a special case of a non-closed language where all states are marked, i.e. \( Q_m = Q \), we only need to develop an algorithm for testing (4.2.1.A). Such an algorithm can also test (4.2.1.B). We also have the following relationships.

\[
K = L_m(G) \cap P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} K, \\
\iff K \supseteq L_m(G) \cap P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} K, \\
\iff \emptyset = L_m(G) \cap P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} K \cap (\Sigma^* \setminus K). \quad (4.2.2)
\]

We next devise an algorithm to test (4.2.2) by using the algorithms that we listed in the previous section. Let \( G := (Q, \Sigma, \delta, q_0, Q_m) \) and \( S := (X, \Sigma, \xi, x_0, X_m) \) represent recognizers for \( L_m(G) \) and \( K \), respectively.

**Algorithm (4.2.A).** An algorithm to test the normality condition. This algorithm returns YES if the normality condition holds. It returns NO otherwise.

```plaintext
test正常性( projection \( P_{\Sigma, \Sigma} \), DFA \( G \), DFA \( S \)); return YES or NO
{
Step 0: initialize \( G \), \( S \) and \( P_{\Sigma, \Sigma} \),
Step 1: evaluate \( A \) := \( (Y, \Sigma, \eta, y_0, Y_m) \) as follows,

\[ A \text{ = intersect}(G, \text{intersect}(\text{inversepp}(P_{\Sigma, \Sigma}, S), \text{complement}(S))) \],

Step 2: If\( (Y_m = \emptyset) \)

{ return YES. }
ElseIf

{ return NO. } EndIf
```
Proposition (4.2.1). The computational time complexity of Algorithm (4.2.A) is of the order of $O(|\Sigma||Q|^3|X|^6 + |\Sigma|^2|X|^4)$.

Proof: $\text{complement}(S)$ is computed in $O(|\Sigma|(|X|+1)^2)$ time and yields a DFA whose state space cardinality is $(|X|+1)$. $\text{inversep}(P_{\Sigma,\Sigma}, S)$ is computed in $O(|\Sigma|^2|X|^4)$ time and yields a NFA with state space cardinality $|X|$. Thus

$$\text{intersect}(\text{inversep} \ (P_{\Sigma,\Sigma}, S), \text{complement}(S))$$

can be computed in $O(|\Sigma||Q|^3(|X|+1)^3)$ time. The operation yields a NFA with state space cardinality equal $|X||(|X|+1)$. Therefore we can compute

$$\text{intersect}(G, \text{intersect}(\text{inversep} \ (P_{\Sigma,\Sigma}, S), \text{complement}(S)))$$

in $O(|\Sigma||Q|^3|X|^6)$ time. We conclude that overall computational time complexity of the algorithm is $O(|\Sigma||Q|^3|X|^6 + |\Sigma|^2|X|^4)$.

$\Delta$

4.3. An algorithm to test the observability condition

Testing the observability condition is studied in this subsection. A number of procedures to test this condition are discussed in the control literature. Cho and Marcus in [15] present an effective procedure to test the observability condition. Their procedure requires construction of the image of a given desired language recognizer and the enumeration of the image automaton's state space. If $|Q|$ is the cardinality of the state space of the desired language, then the DFA recognizing the image of this language could have $2^{|Q|}$ states.
and Wonham in [83] developed an effective procedure to compute the infimal observable language containing a given language \( K \), denoted by \( \text{inf} \, O(K) \). To examine if \( K \) is observable, an effective test procedure based on the equality 
\( K = \text{inf} \, O(K) \) would be required. However in the course of constructing the recognizer for \( \text{inf} \, O(K) \), it is necessary to construct the image of a language related to \( K \), and then enumerate the state space of the resulting automaton in subsequent computations. Thus, in general, this procedure cannot be accomplished in polynomial time either. However, Tsitsiklis [91] has shown that the observability condition can be tested in polynomial time. He has shown that testing the observability condition is equivalent to the nonexistence of any winning strategy in an appropriately defined game. Since dynamic programming can be used to resolve the existence of a solution to the winning strategy problem, he concludes that the observability condition can be tested in polynomial time. Next, we present a polynomial time test procedure which is based on familiar methods in automata theory.

Recall that \( K \) is called \( P_{\Sigma,\Sigma} \)-observable or simply observable if and only if

\[
\ker P_{\Sigma,\Sigma} \leq \text{act}_{K, L_m(G)} \quad \text{or} \quad (s_1, s_2) \in \ker P_{\Sigma,\Sigma} = (s_1, s_2) \in \text{act}_{K, L_m(G)}.
\]

(4.3.1)

Where we have

\[
(s_1, s_2) \in \ker P_{\Sigma,\Sigma} \iff P_{\Sigma,\Sigma} s_1 = P_{\Sigma,\Sigma} s_2,
\]

(4.3.2)

and

\[
(s_1, s_2) \in \text{act}_{K, L_m(G)} \iff (A) \text{ and } (B),
\]
\[
\begin{align*}
\begin{cases}
A) & \ A_{K,L(G)}(s_1) \cap I A_{K,L(G)}(s_2) = \emptyset = I A_{K,L(G)}(s_1) \cap A_{K,L(G)}(s_2), \\
B) & \ s_1 \in K \text{ and } s_2 \in \bar{K} \cap L_m(G) \implies s_2 \in K.
\end{cases}
\end{align*}
\]  

(4.3.3)

Here, the active set and the inactive set of \( s \in \Sigma^* \) with respect to \( \bar{K} \) and \( L(G) \) are denoted by \( A_{K,L(G)}(s) \) and \( I A_{K,L(G)}(s) \) respectively, and are defined by

\[
\begin{align*}
A_{K,L(G)}(s) := \begin{cases}
\{ \sigma \mid s\sigma \in \bar{K} \}, & \text{if } s \in \bar{K}; \\
\emptyset, & \text{otherwise.}
\end{cases}
\end{align*}
\]  

(4.3.4)

\[
\begin{align*}
I A_{K,L(G)}(s) := \begin{cases}
\{ \alpha \mid s\alpha \in L(G) \setminus \bar{K} \}, & \text{if } s \in \bar{K}; \\
\emptyset, & \text{otherwise.}
\end{cases}
\end{align*}
\]  

(4.3.5)

Thus to test the observability condition we must test the following two conditions,

\[
(s_1, s_2) \in \ker P_{\Sigma, \Sigma} \implies (4.3.3.A),
\]  

(4.3.6)

and

\[
(s_1, s_2) \in \ker P_{\Sigma, \Sigma} \implies (4.3.3.B).
\]  

(4.3.7)

We first devise an algorithm to test condition (4.3.7), which is equivalent to testing the following condition

\[
P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} K \cap \bar{K} \cap L_m(G) \cap (\Sigma^* \setminus \bar{K}) = \emptyset.
\]  

(4.3.8)

The following algorithm tests condition (4.3.8).

**Algorithm (4.3.A).** The algorithm to test condition (4.3.8) returns YES if the condition holds, and NO otherwise.
testobserve\( (\text{projection } P_{\Sigma,\Sigma}, \text{DFA } G, \text{DFA } S)\); return YES or NO

\[
\text{Step 0: } \text{initialize } G := (Q, \Sigma, \delta, q_0, Q_m), \quad S := (X, \Sigma, \xi, x_0, X_m) \text{ where } L_m(S) = K \subseteq L_m(G), \quad \tilde{S} := (X, \Sigma, \xi, x_0, X) \text{ where } L_m(\tilde{S}) = \tilde{K}, \\
P_{\Sigma,\Sigma}.
\]

\[
\text{Step 1: } \text{evaluate } A := (Y, \Sigma, \eta, y_0, Y_m) \text{ as follows, } \\
A = \text{intersect}(G, \text{intersect}(\tilde{S}, \\
\text{intersect(inversepp } (P_{\Sigma,\Sigma}, S), \text{complement}(S))))),
\]

\[
\text{Step 2: } \text{If}(Y_m = \emptyset) \\
\{ \text{return YES. } \} \\
\text{ElseIf} \\
\{ \text{return NO. } \}\text{ElseIf}
\]

\[
\triangle
\]

**Proposition (4.3.1).** The computational time complexity of Algorithm (4.3.A) is on the order of \(O(|\Sigma||Q|^3|X|^9 + |\Sigma|^2|X|^4).\)

**Proof:** Similar to the proof of Proposition (4.2.1).

\[
\triangle
\]

Next we present an algorithm to test condition (4.3.6).

**Algorithm (4.3.B).** The algorithm to test condition (4.3.6) returns YES if the
testobserveA(projection \( P_{\Sigma,\Sigma}, \text{DFA } G, \text{DFA } S \)); return \text{YES or NO}
{

\text{Step 0: } \text{initialize } G := (Q, \Sigma, \delta, q_0, Q_m),
\hspace{1cm} S := (X, \Sigma, \xi, x_0, X_m) \text{ where } L(S) = \tilde{K} \subseteq L(G), \hspace{1cm} P_{\Sigma,\Sigma},
\begin{align*}
\text{Step 1: } & \text{evaluate } [\tilde{G}, \tilde{S}] = \text{subautomaton}(G, S), \text{ where } \\
& \tilde{G} := (\tilde{Q}, \Sigma, \tilde{\delta}, \tilde{q}_0, \tilde{Q}_m), \text{ and } \\
& \tilde{S} := (\tilde{X}, \Sigma, \tilde{\xi}, \tilde{x}_0, \tilde{X}_m),
\end{align*}
\text{Step 2: } \text{construct map } \varphi(\cdot) \text{ as follows, } \forall(q, x) \in \tilde{X} \text{ and } \forall \sigma \in \Sigma,
\begin{align*}
\varphi((q, x))(\sigma) &= \begin{cases} 
1, & \text{if } \tilde{\xi}(\sigma, (q, x))!; \\
0, & \text{if } \tilde{\delta}(\sigma, (q, x))! \text{ and } \sim \tilde{\xi}(\sigma, (q, x))!; \\
dc, & \text{otherwise.}
\end{cases}
\end{align*}
\text{Step 3: } \text{construct } \tilde{S}_{n,\Sigma} := (\tilde{X}, \tilde{\Sigma}, \tilde{\xi}_{n,\Sigma}, \tilde{x}_0, \tilde{X}_m),
\hspace{1cm} = \text{project}(P_{\Sigma,\Sigma}, \tilde{S}),
\text{Step 4: } \text{construct }
\begin{align*}
\tilde{S}_{n,\Sigma} \times \tilde{S}_{n,\Sigma} &:= (\tilde{X} \times \tilde{X}, \tilde{\Sigma}, \tilde{\xi}_{n,\Sigma} \times \tilde{\xi}_{n,\Sigma}, (\tilde{x}_0, \tilde{x}_0), \tilde{X}_m \times \tilde{X}_m), \\
&= \text{intersect}(\tilde{S}_{n,\Sigma}, \tilde{S}_{n,\Sigma}),
\end{align*}
\text{Step 5: } \Void := \{((\tilde{x}_1, \tilde{x}_2) \in \tilde{X} \times \tilde{X} \mid \exists \sigma \in \Sigma \text{ such that } \varphi(\tilde{x}_1)(\sigma) = 1 \text{ and } \varphi(\tilde{x}_2)(\sigma) = 0\},
\text{Step 6: } \begin{align*}
A) \hspace{1cm} & R(\tilde{x}_0, \Sigma \setminus \tilde{\Sigma}, \tilde{S}) \quad \text{and} \\
B) \hspace{1cm} & R((\tilde{x}_0, \tilde{x}_0), \tilde{\Sigma}, \tilde{S}_{n,\Sigma} \times \tilde{S}_{n,\Sigma}),
\end{align*}
\text{Step 7: } \text{test the following two conditions}
\begin{itemize}
\item \( B \) \( \forall (\hat{x}_0, \hat{x}_0), \hat{\Sigma}, \hat{S}_{n, \Sigma} \times \hat{S}_{n, \Sigma}) \cap \text{Void} = \emptyset \),
\end{itemize}

\textbf{Step 8:} If both conditions of step 7 hold, return \textit{YES}.
Otherwise, return \textit{NO}.

\[ \triangle \]

We establish the correctness of Algorithm (4.3.B) in the following proposition.

\textbf{Proposition (4.3.2).} Algorithm (4.3.B) returns \textit{YES} if and only if condition (4.3.6) holds. Or otherwise stated, Algorithm (4.3.B) returns \textit{NO} if and only if condition (4.3.6) fails.

\textbf{Proof:}

\textit{Only if part:} Suppose that the algorithm returns \textit{NO}. We show that in this case condition (4.3.6) fails.

\textit{Case 1:} Suppose \( \exists (\hat{x}_1, \hat{x}_2) \in (R(\hat{x}_0, \Sigma \setminus \hat{\Sigma}, \hat{S}) \times R(\hat{x}_0, \Sigma \setminus \hat{\Sigma}, \hat{S})) \cap \text{Void} \neq \emptyset \).

Then

\[ \hat{x}_1 \in R(\hat{x}_0, \Sigma \setminus \hat{\Sigma}, \hat{S}) \Rightarrow \exists t_1 \in K \ni P_{\Sigma, \Sigma} t_1 = \epsilon \text{ and } \hat{x}_0 \xrightarrow{t_1} \hat{x}_1(\text{in } \hat{S}). \]

Also

\[ \hat{x}_2 \in R(\hat{x}_0, \Sigma \setminus \hat{\Sigma}, \hat{S}) \Rightarrow \exists t_2 \in K \ni P_{\Sigma, \Sigma} t_2 = \epsilon \text{ and } \hat{x}_0 \xrightarrow{t_2} \hat{x}_2(\text{in } \hat{S}). \]

Without loss of generality we can write

\[ (\hat{x}_1, \hat{x}_2) \in \text{Void} \Rightarrow \exists \alpha \in \Sigma \ni t_1 \alpha \in K \text{ and } t_2 \alpha \in L(G) \setminus K. \]

\[ \Rightarrow \alpha \in A_{K, L(G)}(t_1) \cap IA_{K, L(G)}(t_2) \neq \emptyset, \text{(4.3.9)} \]

We also have
Equations (4.3.9) and (4.3.10) imply that condition (4.3.6) does not hold.

**Case 2:** Suppose \((\bar{x}_1, \bar{x}_2) \in R((\bar{x}_0, \bar{x}_0), \tilde{S}_{n,\Sigma} \times \tilde{S}_{n,\Sigma}) \cap \text{Void} \neq \emptyset\). Then \(\exists s \in \tilde{\Sigma}^*\) such that

\[
\bar{x}_0 \xrightarrow{s} \bar{x}_1 \text{ (in } \tilde{S}_{n,\Sigma}) \quad \Longrightarrow \quad \exists s_1 \in \tilde{K} \ni s = P_{\Sigma,\Sigma}s_1 \text{ and } \bar{x}_0 \xrightarrow{s_1} \bar{x}_1 \text{ (in } \tilde{S}).
\]

Also

\[
\bar{x}_0 \xrightarrow{s} \bar{x}_2 \text{ (in } \tilde{S}_{n,\Sigma}) \quad \Longrightarrow \quad \exists s_2 \in \tilde{K} \ni s = P_{\Sigma,\Sigma}s_2 \text{ and } \bar{x}_0 \xrightarrow{s_2} \bar{x}_2 \text{ (in } \tilde{S}).
\]

Without loss of generality, we can write,

\[
(\bar{x}_1, \bar{x}_2) \in \text{Void} \quad \Longrightarrow \quad \exists \beta \in \Sigma \ni s_1\beta \in \tilde{K} \text{ and } s_2\beta \in L(G) \setminus \tilde{K},
\]

\[
\Rightarrow \quad \exists \ J \in A_{K, L(G)}(s_1) \cap IA_{K, L(G)}(s_2) \neq \emptyset,
\]

(4.3.11)

We also have

\[
P_{\Sigma,\Sigma}s_1 = P_{\Sigma,\Sigma}s_2 = s \quad \Rightarrow \quad (s_1, s_2) \in \ker P_{\Sigma,\Sigma}. \quad (4.3.12)
\]

Equations (4.3.11) and (4.3.12) imply that condition (4.3.6) does not hold. This completes the proof of the necessity part.

**If part:** We show that if condition (4.3.6) fails, then the algorithm returns \texttt{NO}.

Suppose \(\exists (s_1, s_2) \in \tilde{K} \times \tilde{K}\) such that \((s_1, s_2) \in \ker P_{\Sigma,\Sigma} \text{ and } (s_1, s_2)\) does not satisfy condition (4.3.3.A). Without loss of generality we can conclude that

\[
\exists \sigma \in \Sigma \text{ such that } s_1\sigma \in \tilde{K} \text{ and } s_2\sigma \in L(G) \setminus \tilde{K} \text{ and } s_1 \neq s_2.
\]

This implies

\[
\varphi(\bar{x}_1 := \tilde{\xi}(s_1, \bar{x}_0))(\sigma) = 1 \quad \text{and} \quad \varphi(\bar{x}_2 := \tilde{\xi}(s_2, \bar{x}_0))(\sigma) = 0.
\]
There are two cases:

**Case 1:** If \( P_{\Sigma, \Sigma} s_1 = P_{\Sigma, \Sigma} s_2 = \epsilon \), then

\[
\begin{align*}
  s_1 \in \hat{K} & \implies \hat{x}_0 \xrightarrow{\Sigma \setminus \hat{\Sigma}} \hat{x}_1 (\text{in } \hat{\mathcal{S}}). \\
  s_2 \in \hat{K} & \implies \hat{x}_0 \xrightarrow{\Sigma \setminus \hat{\Sigma}} \hat{x}_2 (\text{in } \hat{\mathcal{S}}).
\end{align*}
\] (4.3.13)

Therefore,

\[
(\hat{x}_1, \hat{x}_2) \in (R(\hat{x}_0, \Sigma \setminus \hat{\Sigma}, \hat{\mathcal{S}}) \times R(\hat{x}_0, \Sigma \setminus \hat{\Sigma}, \hat{\mathcal{S}})) \cap \text{Void} \neq \emptyset.
\]

In other words, the algorithm returns \( \emptyset \).

**Case 2:** Suppose \( P_{\Sigma, \Sigma} s_1 = P_{\Sigma, \Sigma} s_2 = t \neq \epsilon \). We have that

\[
\begin{align*}
  s_1 \in \hat{K} & \implies \hat{x}_0 \xrightarrow{P_{\Sigma, \Sigma} s_1 = t} \hat{x}_1 (\text{in } \hat{S}_{n, \Sigma}). \\
  s_2 \in \hat{K} & \implies \hat{x}_0 \xrightarrow{P_{\Sigma, \Sigma} s_2 = t} \hat{x}_2 (\text{in } \hat{S}_{n, \Sigma}).
\end{align*}
\] (4.3.15)

Therefore

\[
(\hat{x}_0, \hat{x}_0) \xrightarrow{P_{\Sigma, \Sigma} s_1 = P_{\Sigma, \Sigma} s_2 = t} (\hat{x}_1, \hat{x}_2) \in \text{Void} \ (\text{in } \hat{S}_{n, \Sigma} \times \hat{S}_{n, \Sigma}).
\]

Thus Algorithm (4.3.B) returns \( \emptyset \) because

\[
(\hat{x}_1, \hat{x}_2) \in R((\hat{x}_0, \hat{x}_0), \hat{\Sigma}, \hat{S}_{n, \Sigma} \times \hat{S}_{n, \Sigma}) \cap \text{Void} \neq \emptyset.
\]

\( \triangle \)

**Proposition (4.3.3).** Algorithm (4.3.B) is a polynomial-time algorithm with complexity on the order of \( O(|\Sigma||Q|^6|X|^6 + |\Sigma|^2|Q|^4|X|^4) \).
$\tilde{S}$ with state space cardinalities $|Q||X| + 1$ and $|Q||X|$, respectively. Step 2 is completed in $O(|\Sigma||Q||X|)$ time. Step 3 is accomplished in $O(|\Sigma|^2|Q|^4|X|^4)$ and yields NFA $\tilde{S}_{n,\Sigma}$ whose state space cardinality is $|Q||X|$. Step 4 is completed in $O(|\Sigma||Q|^6|X|^6)$ and yields NFA $\tilde{S}_{n,\Sigma} \times \tilde{S}_{n,\Sigma}$ whose state space cardinality is $|Q|^2|X|^2$. Step 5 is computed in $O(|\Sigma||Q|^2|X|^2)$ and yields Void with $|\text{Void}| \leq |Q|^2|X|^2$. Step 6.A is computed in $O(|\Sigma \setminus \tilde{S}||Q|^3|X|^3)$ and the reach has cardinality $|Q||X|$. Step 6.B is computed in $O(|\tilde{S}||Q|^6|X|^6)$ and the reach has cardinality $|Q|^2|X|^2$. Thus Step 7 is completed in $O(|Q|^4|X|^4)$. Finally we conclude that the complexity of the algorithm is in order of $O(|\Sigma||Q|^6|X|^6 + |\Sigma|^2|Q|^4|X|^4)$.

$\Delta$

Finally, combination of testobserveA(·,·,·) and testobserveB(·,·,·) yields a polynomial time algorithm to test the observability condition.

4.4. Conclusion

In this chapter we presented two polynomial time effective algorithms to test the normality and observability conditions for regular languages.
In this chapter, we revisit the problem of decentralized supervisory control of discrete event dynamical systems when the control objectives are locally specified. It is well known that if the normality condition is satisfied, then there exists a decentralized supervisor that is as permissive (optimal) as any optimal centralized supervisor. The structure of a local events set (a local work space) can change the result of testing the normality condition. Generally, the structure of a local work space is determined by the events that are involved in a task allocated to a specific system and/or some spatial considerations. By generalizing a theorem of Lin and Wonham, we show that it is always possible to find an appropriate local work space for a given local objective. If the structure of the local work space is determined by some spatial considerations, the main result of this chapter can be interpreted as an effort to construct a partially distributed supervisor. On the other hand, if the structure of the local work space is determined by a subtask, the result can be thought of as an attempt to find a set of events that need to be observed or controlled in order to realize the subtask by a decentralized supervisor.
In this chapter, we use the Ramadge and Wonham [80, 99] framework for modeling and supervisory control of Discrete Event Dynamical Systems (DEDS). A DEDS is modeled by an automaton $G := (Q, \Sigma, \delta, q_0, Q_m)$ where $Q$ is the states set, $\Sigma$ is a finite alphabet or set of event symbols, $\delta : \Sigma \times Q \rightarrow Q$ is a partial transition function, $q_0$ is the initial state and $Q_m$ is the set of final states. We restrict our attention to the case that $Q_m = Q$ and represent $G$ by the four tuple $G := (Q, \Sigma, \delta, q_0)$.

Recall, $\Sigma^*$ denote the set of all finite strings (words) of $\Sigma$, including the null string $\epsilon$. A subset $L$ of $\Sigma^*$ is called a formal language or simply a language. A prefix-closure of $L$, denoted by $\bar{L}$, is a language containing all prefixes of words in $L$. A language $L$ is said to be closed if $\bar{L} = L$.

The extended transition function for $G$, denoted by $\hat{\delta}$, is defined as follows,

$$\hat{\delta} : \Sigma^* \times Q \rightarrow Q,$$

for $s \in \Sigma^*$, $\sigma \in \Sigma$, $q \in Q$, $\hat{\delta}(\epsilon, q) := q$, $\hat{\delta}(s\sigma, q) := \delta(\sigma, \hat{\delta}(s, q))$.

From now on, we use $\delta$ to denote both the transition function and its extension.

Without external control, a generator starts at the initial state and generates events in the course of transiting from one state to another. $L(G)$ denotes the closed language generated by $G$ and is defined as follows,

$$L(G) = \{ s \in \Sigma^* \mid \delta(s, q_0) \text{ is defined} \}.$$

$\Sigma$ is partitioned into two sets, the controllable and uncontrollable events that are denoted by $\Sigma_c$ and $\Sigma_u$, respectively. Unlike uncontrollable events, controllable events can be disabled (inhibited) in a controlled discrete event
events, and disables some of the controllable events in \( \Sigma_c \), in a dynamic fashion, such that the closed loop system behaves as desired. In this chapter, we model the desired behavior by a closed sublanguage of \( L(G) \), i.e. \( E \subseteq L(G) \).

Ramadge and Wonham have introduced the notion of language controllability. A desired closed loop language \( E \) is controllable if \( E \Sigma_u \cap L(G) \subseteq E \). Language controllability of \( E \) is a necessary and sufficient condition for the existence of a supervisor for \( G \) that realizes \( E \). If \( E \) is not controllable, it is desirable to implement a supervisor that realizes the supremal controllable sublanguage of \( E \), that is

\[
K := \sup C_{\Sigma_u, L(G)}(E) = \sup \{ M \subseteq E \mid M = \bar{M} \text{ and } M \Sigma_u \cap L(G) \subseteq M \}.
\]

The subscripts denote the underlying uncontrollable events and the open loop system. The supervisory control problem has a solution if \( K \) is nonempty.

A candidate supervisor that realizes \( K \) is characterized as follows. The supervisor \( S = (S, \varphi) \) consists of a finite automata \( S := (X, \Sigma, \xi, x_0) \) and a feedback map \( \varphi : X \rightarrow 2^\Sigma \) such that \( L(S) = K \) and for all \( x \in X \), \( \varphi(x) = \Sigma_u \cup \{ \sigma \in \Sigma \mid \xi(\sigma, x) \text{ is defined} \} \). Here \( 2^\Sigma \) denotes the power set of \( \Sigma \). When a generator is coupled with a supervisor, the supervisor receives events of the generator. The events of the generator induce a state transition in the automata \( S \) according to \( \xi \). In the new state \( \varphi \) evaluates to a new subset of events. This subset determines all possible events that the generator can generate, in the next step, while complying with the transition function \( \delta \). In this way the supervisor controls the behavior of the generator. The closed loop language is denoted by \( L(S/G) \) and is equal to \( K \). The supervisor \( S \) is called a centralized supervisor because it can observe the occurrence of any event and it can disable any controllable event.
subtasks which define the desired behavior for local subsystems. For example, in a distributed system, \( G \) may represent a number of interactive processes for which we specify the desired behavior of each. Then, \( E \) is the global representation of all the local desired behaviors. Also, \( E \) may be given in terms of subtasks. Each subtask may represent a desired ordering among the events of a subset of \( \Sigma \). In such cases, it is desirable to implement the supervisor in a \textit{decentralized} fashion. One advantage of decentralized supervisors is they are more maintainable. Furthermore, they often require fewer communication and command channels between the plant and the supervisor. A decentralized supervisor consists of a number of local supervisors. Each local supervisor has partial observation of the plant \( G \) and can control a subset of controllable events. In decentralized supervisory control, we require that the joint action of the local supervisors be as equivalent to a centralized supervisor. The notions of \textit{normality}, \textit{decomposability}, and \textit{coobservability}, that guarantees the optimality of a decentralized supervisor have been developed in [17, 60, 84].

We use a \textit{projection} to model a local supervisor. Formally, given \( \Sigma_1 \) and \( \Sigma_2 \) such that \( \Sigma_2 \subseteq \Sigma_1 \subseteq \Sigma \) let \( P_{\Sigma_1, \Sigma_2} : \Sigma_1^* \rightarrow \Sigma_2^* \) be a projection defined by:

\[
P_{\Sigma_1, \Sigma_2} \varepsilon = \varepsilon;
\]

\[
P_{\Sigma_1, \Sigma_2} \sigma = \varepsilon, \quad \text{for } \sigma \in \Sigma_1 \setminus \Sigma_2;
\]

\[
P_{\Sigma_1, \Sigma_2} \sigma = \sigma, \quad \text{for } \sigma \in \Sigma_2;
\]

\[
P_{\Sigma_1, \Sigma_2}(s\sigma) = (P_{\Sigma_1, \Sigma_2} s)(P_{\Sigma_1, \Sigma_2} \sigma), \quad \text{for } s \in \Sigma_1^*, \, \sigma \in \Sigma_1.
\]

The projection \( P_{\Sigma_1, \Sigma_2} \) simply erases those events of \( s \) that are not in \( \Sigma_2 \) and the inverse projection \( P_{\Sigma_1, \Sigma_2}^{-1} : \Sigma_2^* \rightarrow \Sigma_1^* \) is defined as:

\[
\text{for } s \in \Sigma_2^*, \quad P_{\Sigma_1, \Sigma_2}^{-1} s = \{ t \in \Sigma_1^* \mid P_{\Sigma_1, \Sigma_2} t = s \}.
\]
Suppose \( \hat{\Sigma} \subset \Sigma \) captures a set of local events. \( \hat{\Sigma} \) could represent events involved in a subsystem or it may be the set of events involved in a subtask \( E \subseteq \hat{\Sigma}^* \). Let \( E \) denote the desired local behavior. In order to satisfy the local requirement, a centralized supervisor implements

\[
K := \sup_{\Sigma, L(G)} (L(G) \cap P_{\Sigma, \Sigma}^{-1} E).
\]

On the other hand, a local supervisor's view of the system is \( P_{\Sigma, \Sigma} L(G) \). Then, a locally optimal supervisor synthesizes

\[
\hat{K} := \sup_{\Sigma, L(G)} (P_{\Sigma, \Sigma} L(G) \cap E).
\]

The global effect of the above local supervision is

\[
\hat{K} := L(G) \cap P_{\Sigma, \Sigma}^{-1} \hat{K}.
\]

Lin and Wonham [60] have proved that normality of \( K \) with respect to \( L(G) \) and the projection \( P_{\Sigma, \Sigma} \) is a necessary and sufficient condition for \( \hat{K} = K \). \( K \) is said to be normal with respect to \( L(G) \) and \( P_{\Sigma, \Sigma} \) if

\[
K = L(G) \cap P_{\Sigma, \Sigma}^{-1} P_{\Sigma, \Sigma} K.
\]

In section 5.2 we generalize the result in [60] and discuss the contributions of this generalization. In section 5.3 we present an example that demonstrates the main result of this chapter. Finally, we provide a summary of the main contribution of this chapter in section 5.4.
Suppose that the event set $\Sigma$ is composed of a class of, not necessarily disjoint, local event sets $\{\Sigma_i \mid i \in I\}$ where $I$ is an arbitrary index set and $\bigcup_{i \in I} \Sigma_i = \Sigma$. Let $E_i \subset \Sigma_i^*$ be a nonempty closed sublanguage that captures the desired local behavior of the $i$th subsystem of the DEDS. At the global level, the corresponding desired behavior is the sublanguage

$$L(G) \cap \bigcap_{i \in I} P_{\Sigma_i}^{-1} E_i.$$

A centralized supervisor then synthesizes the sublanguage

$$K := \sup_{\Sigma} L(G) \left( L(G) \cap \bigcap_{i \in I} P_{\Sigma_i}^{-1} E_i \right),$$

$$= \bigcap_{i \in I} \sup_{\Sigma} L(G) \left( L(G) \cap P_{\Sigma_i}^{-1} E_i \right),$$

$$= \bigcap_{i \in I} K_i \subset \Sigma^*.$$

where

$$K_i := \sup_{\Sigma} L(G) \left( L(G) \cap P_{\Sigma_i}^{-1} E_i \right).$$

Suppose that a local supervisor $\hat{S}_i$ is able to observe the events in $\hat{\Sigma}_i \supseteq \Sigma_i$ and disable events in $\hat{\Sigma}_i \cap \Sigma_c$. In order to synthesize the desired local behavior given by $E_i$, $\hat{S}_i$ supervises the following sublanguage,

$$\hat{K}_i := \sup_{\Sigma} L(G) \left( L(G) \cap P_{\Sigma_i}^{-1} E_i \right) \subset \hat{\Sigma}_i^*.$$

Let $\hat{K}_i$ denote the global sublanguage synthesized by the local supervisor $\hat{S}_i$, then

$$\hat{K}_i := L(G) \cap P_{\Sigma_i}^{-1} \hat{K}_i \subset \Sigma^*.$$

While supervising concurrently, the class of supervisors implement the following language

$$\hat{K} := L(G) \cap \bigcap_{i \in I} P_{\Sigma_i}^{-1} \hat{K}_i \subset \Sigma^*.$$
Proposition (5.2.1). (i) \( \hat{K} = K \) if \( \hat{K}_i = K_i \) for all \( i \in I \).
(ii) \( \hat{K}_i = K_i \) if and only if \( K_i \) is \( (P_{\Sigma, L(G)}) \)-normal for any \( i \in I \).

Proof: A slight modification of the proof of Theorem (3.1) and (4.1) in [60].

A natural criteria for selecting the original local spaces (\( \Sigma_i \)'s) could be based on some spatial considerations or the set of events that appear in a particular subtask. If for a given \( i \in I \) we have \( \hat{\Sigma}_i = \Sigma_i \), then the local supervisor \( \hat{S}_i \) can implement the \( i \)th desired local behavior by working only with events in the \( i \)th local event space. This is the case that Lin and Wonham have presented in [60]. On the other hand, if for some \( j \in I, \hat{\Sigma}_j \supset \Sigma_j \), then \( \hat{S}_j \) needs to work with events in \( \hat{\Sigma}_j \setminus \Sigma_j \) in addition to \( \Sigma_j \). This means that the \( j \)th local objective cannot be achieved in the original \( j \)th local space. In this case, to achieve optimal supervision, the local supervisor needs more information (observing the behavior of \( G \) through \( P_{\Sigma, \Sigma_j} \) instead of \( P_{\Sigma, \Sigma_j} \)) or it needs more control over the system (controlling \( \hat{\Sigma}_j \cap \Sigma_e \) instead of \( \Sigma_j \cap \Sigma_e \)), or both.

Obviously, we would like to have the case that the cardinality of \( \hat{\Sigma}_i \setminus \Sigma_i \) is as small as possible for all \( i \in I \). In such a case, in order to realize the desired local behavior, each local supervisor needs to have information about and control over a few additional events besides their corresponding local events set. We refer to such a control configuration as a \textit{partially decentralized} supervision scheme.

It is always possible to find a \( \hat{\Sigma}_i \) for which condition (ii) of the proposition is satisfied. \( \hat{\Sigma}_i = \Sigma \) is a possible choice although it is not, in many cases, the
5.3. Example

Consider a simple train network that consists of two stations and three trains, Fig. (5.3.1). Initially, one train is at station 1 and two trains are at station 2. At any time only one train may travel from station 1 to station 2 or vice versa.

![Train network diagram]

Figure (5.3.1). Train network.

Events that are involved in the system are $\Sigma = \{d_1^2, d_2^1, a_1^1, a_1^2\}$ and are defined as follows:

- $d_1^2$: A train departs from station 1 toward station 2, controllable.
- $d_2^1$: A train departs from station 2 toward station 1, controllable.
- $a_1^1$: A train that departed from station 2 arrives at station 1, uncontrollable.
- $a_1^2$: A train that departed from station 1 arrives at station 2, uncontrollable.

Unsupervised behavior of the system is depicted in Fig. (5.3.2), where a pair $(m, n)$ denotes that $m$ trains are at station 1 and $n$ trains are at station 2.
Suppose that the capacity of station 1 is 2 and the capacity of station 2 is 3. Our supervision objective is to satisfy the capacity requirements. Due to obvious spatial considerations, we would like to have a supervisor at each station that controls the capacity requirement of the corresponding station. Thus we have two local event spaces as follows:

$\Sigma_1 := \{d_1^2, a_2^1\}$, local events at station 1.

$\Sigma_2 := \{d_2^1, a_2^2\}$, local events at station 2.

Fig. (5.3.3) and (5.3.4) depict the view of the system by the local supervisors at station 1 and 2, respectively.
Languages $E_1$ and $E_2$ whose recognizers are depicted in Fig (5.3.5) and (5.3.6) represent the local desired languages.

Figure (5.3.5). Recognizer for $E_1$,
satisfying the capacity requirement of station 1.

Figure (5.3.6). Recognizer for $E_2$,
satisfying the capacity requirement of station 2.

For this example we have

$$\sup_{E_1 \in \Sigma_n} \left( P_{E_1, E_2} \cdot L(G) \cap E_1 \right) = \emptyset,$$
This means that the capacity requirement of station 2 can be locally supervised but it is impossible to locally supervise the capacity requirement of station 1. Using the above local event spaces, the decentralized supervisory control problem does not have a solution.

Fig. (5.3.7) depicts the global language that satisfies the required local specifications.

Figure (5.3.7). Recognizer for $E_g := L(G) \cap P_{\Sigma_1}^{-1} E_1 \cap P_{\Sigma_2}^{-1} E_2$. global desired behavior.

A centralized supervisor realizes language $K := \sup C_{\Sigma_*,L(G)}(E_g)$. This language is depicted in Fig. (5.3.8).

Figure (5.3.8). Recognizer for $K := \sup C_{\Sigma_*,L(G)}(E_g)$.

is realized by a centralized supervisor.

Our generalization suggests that if we re-structure the local event spaces
To show this suppose we assign $\Sigma_1 := \{d_1^1, d_1^2, d_1^3\} \supset \Sigma_1$ as the local work space for the supervisor at station 1. Then the local desired behavior will be $\hat{E} := P_{\Sigma_1, \Sigma_1} L(G) \cap P_{\Sigma_1, \Sigma_1}^{-1} E_1$, which is depicted in Fig. (5.3.9).

![Diagram](image)

Figure (5.3.9). Recognizer for $\hat{E}$, the desired behavior representation under work space $\Sigma_1$.

Then, supervisor 1 realizes language $\hat{K} := \sup_{C_{\Sigma_1 \cap \Sigma_1}, P_{\Sigma_1, \Sigma_1}} L(G) (\hat{E})$ which is depicted in Fig. (5.3.10).

![Diagram](image)

Figure (5.3.10). Recognizer for $\hat{K} := \sup_{C_{\Sigma_1 \cap \Sigma_1}, P_{\Sigma_1, \Sigma_1}} L(G) (\hat{E})$.

It can be easily seen that the global effect of the local supervisions under the new event spaces is the same as that of the centralized supervisor scheme (Fig. (5.3.8)).

In the above example, the original work space $\Sigma_1$ captures the events that happen at the site of station 1. Also $\Sigma_1$ includes all events that appear in the
It could be the case that a local supervisor has enough control over the system but does not have enough information about the system behavior. Lack of information can result in a more conservative supervision by a local supervisor. Thus optimal supervision cannot be realized in a decentralized fashion.

5.4. Conclusions

In this chapter we generalized a Theorem by Lin and Wonham addressing the problem of decentralized supervisory control. We showed that by re-structuring the local event spaces it is always possible to realize a language that is as permissive as the one realized by a centralized supervisor. We call the supervision scheme partially decentralized when such a re-structuring of the local event spaces is required. In practice this means that a supervisor for a subsystem needs to observe or control some events in other subsystems. It is always possible to find local event spaces that yield a partially decentralized solution for the supervisory control problem. In Chapter 2 we addressed the problem of finding an appropriate work space for a given desired behavior.
Decentralized Supervisory Predicate Control of Discrete Event Dynamical Systems.

In this chapter we study the problem of decentralized supervisory control of discrete event dynamical systems where the control objective is given in terms of predicates defined on the state space. We introduce notions of \textit{statically} and \textit{rationally} decentralizable predicates. When a predicate is decentralizable, we construct decentralized supervisors that can realize the predicates that define the control objective. We also discuss the relationship between predicate observability and predicate decentralizability.

6.1. Introduction

Two basic methods for supervisory control of discrete event dynamical systems (DEDs) have been used in the control literature. In the first modeling paradigm, the events of the underlying DEDS are considered the fundamental components of the system and the control objective is stated in terms of the events; for example it may be desirable that any event trace that the controlled DEDS can generate satisfy a specific event sequencs. Formal language theory
framework. In the previous chapters we used the event based framework in our study. In the other modeling paradigm, the state of the underlying DEDS is considered as the fundamental concept and the control objective is expressed in the form of conditions that the state of the controlled system must satisfy [23, 42, 48, 51, 52, 75, 81]. In this chapter, we adopt the formalism based on the state of the DEDS where the behavior of the DEDS is described by its state trajectories. The control objective is given by a desired predicate that any sample trajectory must satisfy.

Ramadge and Wonham have considered the predicate invariance problem in [81]. Li and Wonham have addressed the observability and controllability of predicates in [48]. More recently, Kumar, et al., have addressed the problem of controllability and observability of predicates in a concise way [42]. In [42] they also developed an algorithm for computing the weakest controllable predicate that is stronger than a given predicate. Different notions of stability, stabilizability and state steering of a DEDS have also been addressed in [6, 8, 23, 75, 77].

The decentralized supervisory control problem formulated in the context of formal languages has been addressed in [60]. In this chapter the overall objective is given in the form of a desired predicate, and we investigate cases where the desired predicate can be realized by the joint action of a number of supervisors, each of which has a partial observation of the state and has limited control over the system.

This chapter is organized as follows: In section 6.2 we introduce predicates and predicate transformers in the context of DEDSs. In section 6.3 we present the main results of the chapter. Section 6.4 contains our concluding remarks
6.2. Predicates and predicate transformers in DEDSs

We consider a discrete event dynamical system (DEDS) that can be modeled as a state machine. The quadruple $G := (X, \Sigma, \delta, x_0)$ denotes a state machine that represents our DEDS. Here $X$ is the state space, which could be a countable set; $\Sigma$ is the event set that is represented as the disjoint union of a controllable event set $\Sigma_c$ and an uncontrollable event set $\Sigma_u$; $\delta : \Sigma \times X \rightarrow X$ is a partial function which describes the transition characteristics of the DEDS and $x_0$ is the initial state.

One can alternatively describe a DEDS using predicate transformers. A boolean valued function $\mathcal{P} : X \rightarrow \{0, 1\}$ is called a predicate. Given a predicate $\mathcal{P} : X \rightarrow \{0, 1\}$ and a state $x$, we say $x$ satisfies the predicate $\mathcal{P}$ if $\mathcal{P}(x) = 1$. $X_\mathcal{P}$ denotes all states that satisfy predicate $\mathcal{P}$, that is $X_\mathcal{P} = \{x \in X \mid \mathcal{P}(x) = 1\}$. Therefore associated with every predicate $\mathcal{P}$, there exists a subset of the state space $X_\mathcal{P}$. Conversely, we associate a predicate $\mathcal{Q}$ with a given subset $Q$ of $X$ such that the relation $X_\mathcal{Q} = Q$ holds. $TRUE$ and $FALSE$ are two special predicates defined to satisfy the relations $X_{TRUE} = X$ and $X_{FALSE} = \emptyset$. $\mathcal{P}_X$ will denote the set of all predicates on $X$.

Three basic operations of predicates: negation denoted by $\neg$, conjunction denoted by $\wedge$, and disjunction denoted by $\vee$, are defined as follows:
Given a predicate transformer $f$ is a predicate valued function of predicates, i.e.
$f : P_X \rightarrow P_X$. Similar to the case of predicates we define the operations of
negation, conjunction, and disjunction as follows:

Given $f : P_X \rightarrow P_X$ and $P \in P_X$,

$$(-f)(P) = \neg(f(P)).$$

Given an index set $I$, a class of predicate transformers $\{f_i : P_X \rightarrow P_X \mid i \in I\}$
and $P \in P_X$,

$$\left(\bigwedge_{i \in I} f_i\right)(P) = \bigwedge_{i \in I} (f_i(P)),$$

$$\left(\bigvee_{i \in I} f_i\right)(P) = \bigvee_{i \in I} (f_i(P)).$$
denoted by $C_\alpha$, as follows:

$$C_\alpha(x) = 1 \iff \delta(\alpha, x) \text{ is defined.}$$

Given the DEDS $G := (X, \Sigma, \delta, x_0)$ the \textit{weakest precondition} predicate transformer denoted by "wp" is defined as follows:

For $\alpha \in \Sigma$, $\mathcal{P} \in \mathcal{P}_X$ and $x \in X$,

$$wp_\alpha(\mathcal{P})(x) = \begin{cases} 1, & \text{if } C_\alpha(x) = 1 \text{ and } \mathcal{P}(\delta(\alpha, x)) = 1, \\ 0, & \text{otherwise}. \end{cases}$$

$$wp(\mathcal{P}) = \bigwedge_{\sigma \in \Sigma} wp_\sigma(\mathcal{P}).$$

The \textit{weakest liberal precondition} predicate transformer is denoted by "wlp" and is defined as

$$wlp(\mathcal{P}) = \bigwedge_{\sigma \in \Sigma} wlp_\sigma(\mathcal{P}),$$

$$wlp_\alpha(\mathcal{P}) = wp_\alpha(\mathcal{P}) \lor (\neg C_\alpha).$$

The \textit{strongest postcondition} predicate transformer is denoted by "sp" and is defined as follows:

For $\alpha \in \Sigma$, $\mathcal{P} \in \mathcal{P}_X$ and $x \in X$,

$$sp_\alpha(\mathcal{P})(x) = \begin{cases} 1, & \text{if for some } x' \in X_{\mathcal{P}}, C_\alpha(x') = 1 \text{ and } x = \delta(\alpha, x'). \\ 0, & \text{otherwise}. \end{cases}$$

$$sp(\mathcal{P}) = \bigvee_{\sigma \in \Sigma} sp_\sigma(\mathcal{P}).$$

A predicate transformer $f$ is said to be \textit{strict} if $f(\text{FALSE}) = \text{FALSE}$; \textit{monotone} if for $\mathcal{P}, \mathcal{Q} \in \mathcal{P}_X$, $\mathcal{P} \preceq \mathcal{Q} \implies f(\mathcal{P}) \preceq f(\mathcal{Q})$; \textit{disjunctive} if
be shown that \( sp(\cdot) \) is disjunctive and \( wp(\cdot) \) and \( wlp(\cdot) \) are conjunctive. All three predicate transformers are monotone. Given a predicate transformer \( f^* \), \( f^* \) is another predicate transformer defined by \( f^* = \bigvee_{k \geq 0} f^k \), where \( f^k \) for \( k \geq 1 \) denotes the \( k \)-fold composition of \( f \) and \( f^0 \) is defined to be the identity predicate transformer. Given a predicate transformer \( f \) and a predicate \( R \in \mathcal{P}_X \), \( (f \mid R) \) is the predicate transformer obtained by restricting the action of \( f \) to \( R \), that is for \( P \in \mathcal{P}_X \) we have \( (f \mid R)(P) := f(R \land P) \land R \).

For notions of duality for predicates and predicate equations refer to [42].

The calculus of predicates can be used to represent the DEDS under consideration. In particular, \( G := (\mathcal{P}_X, \Sigma, sp, I) \) is another representation of \( G \) where \( I \) is the initial predicate that is satisfied by any possible initial state of \( G \). The predicate-based representation of \( G \) has an advantage over the state-space representation because it can easily model infinite state DEDSs. Using predicate transformers and predicate equations, the notions of controllability and observability for infinite state DEDSs can be investigated [42, 48, 51, 52]. Throughout this chapter we will use predicate based representation of DEDSs.

### 6.3. Decentralized predicate control

Consider a discrete event dynamical system given by

\[
G := (\mathcal{P}_X, \Sigma, sp, I). 
\]

where \( X \) is the state space, \( \mathcal{P}_X \) is the set of all predicates defined on \( X \), \( \Sigma \) is the events set, \( sp : \mathcal{P}_X \rightarrow \mathcal{P}_X \) is the strongest postcondition predicate transformer induced by the transition characteristics of the DEDS, and \( I \) is
\( \mathcal{I} \subseteq \mathcal{R} \) and \( \Sigma \) is decomposed into controllable events \( \Sigma_c \) and uncontrollable events \( \Sigma_u \).

**Definition (6.3.1): Local Work Space**

A *local work space* is a triple \((\hat{Y}, \hat{M}, \hat{\Sigma})\) where \(\hat{Y}\) is the observation space, \(\hat{M} : X \rightarrow \hat{Y}\) is an image map from the state space \(X\) to the observation space, and \(\hat{\Sigma}\) is a subset of \(\Sigma\). The preimage map \(\hat{M}^{-1} : \hat{Y} \rightarrow 2^X\) is defined as follows: 

\[
\hat{M}^{-1}(\hat{y}) = \{ x \in X | \hat{M}(x) = \hat{y} \}.
\]

Given an image map \(\hat{M} : X \rightarrow \hat{Y}\), we define a predicate transformer \(\hat{M} : \mathcal{P}_X \rightarrow \mathcal{P}_Y\) such that for \(Q \in \mathcal{P}_X\) we have:

\[
\hat{M}(Q)(\hat{y}) = \begin{cases} 
1, & \text{if } M^{-1}(\hat{y}) \cap X_Q \neq \emptyset, \\
0, & \text{otherwise}. 
\end{cases}
\]

Similarly, a preimage map \(\hat{M}^{-1} : \hat{Y} \rightarrow 2^X\) induces a predicate transformer \(\hat{M}^{-1} : \mathcal{P}_Y \rightarrow \mathcal{P}_X\) such that for \(Q \in \mathcal{P}_Y\) we have:

\[
\hat{M}^{-1}(Q)(x) = \begin{cases} 
1, & \text{if } Q(M(x)) = 1, \\
0, & \text{otherwise}. 
\end{cases}
\]

**Definition (6.3.2): Static Local Supervisor**

A map \(\hat{S} : \hat{Y} \rightarrow 2^\Sigma\) is called a *static local supervisor* with the local work space \((\hat{Y}, \hat{M}, \hat{\Sigma})\). Given an observation \(\hat{y} \in \hat{Y}\), \(\hat{S}(\hat{y})\) includes all events in \(\hat{\Sigma}\) which are enabled. The remaining events in the local work space are disabled by the supervisor. We require that \(\hat{\Sigma} \cap \Sigma_u \subseteq \hat{S}(\hat{y})\) for any \(\hat{y} \in \hat{Y}\). More general forms of local supervisors like *dynamic local supervisors* can also be defined. However, in this chapter we only study decentralization via static local supervisors.
identity map, i.e. \( \mathcal{M}(x) = x \) and \( \Sigma = \Sigma \) then the supervisor \( S \) is called a centralized supervisor. The *Supervisory Predicate Control Problem* (SPCP) is the process of constructing a centralized supervisor \( S : X \rightarrow 2^\Sigma \) for the DEDS \( G \) such that \( sp^*_S(\mathcal{I}) = \mathcal{R} \), where by definition we have \( sp^*_s(\cdot) = \bigvee_{k \geq 0} sp^k_S(\cdot) \). It is proved in [42] that a necessary and sufficient condition for solvability of SPCP is controllability of the predicate \( \mathcal{R} \). Here, a predicate \( \mathcal{R} \) is called controllable if (1) \( sp_u(\mathcal{R}) \preceq \mathcal{R} \) for any \( u \in \Sigma_u \) and (2) \( \mathcal{R} = (sp_{|\mathcal{R}})^*(\mathcal{I}) \). When the SPCP is solvable a supervisor must disable any controllable event \( \alpha \in \Sigma_c \) wherever the predicate \( \mathcal{R} \land wp_{\alpha}(\neg \mathcal{R}) \) is equal to one. This will realize the predicate \( \mathcal{R} \) for the DEDS.

In most applications, the DEDS under consideration is composed of subsystems that are spatially distributed. In such a case, it is more desirable to realize \( \mathcal{R} \) in a decentralized fashion. For decentralized supervision, there are a number of local supervisors that each have partial information about the system and also control a subset of events. One, however, expects to realize the same desired predicate \( \mathcal{R} \) by the joint action of these local supervisors. In what follows we explain the construction of a decentralized supervision scheme to accomplish this objective. Because the local supervisors in this work are static, we refer to this scheme as Static Decentralized Supervision.

**Definition (6.3.3): Static Decentralized Supervision**

A static decentralized supervision scheme in the context of predicate control of a DEDS consists of a set of local supervisors, each of which regulates controllable events in their corresponding events set and a combination rule that determines the way that decisions of local supervisors are combined to regulate a common event. More precisely, let \( \mathcal{W} = \{(Y_j, M_j, \Sigma_j) \mid j \in J\} \) de-
consider a class of local supervisors \( \{ S_j \mid j \in J \} \) where the local work space of \( S_j \) is \((Y_j, M_j, \Sigma_j)\). Given \( x \in X \) as the current state of \( G \), \( y_j \) is the observation by \( S_j \). Every local supervisor evaluates a predicate \( \mathcal{P}^{S_i} \in \mathcal{P}_X \) so that any state \( x' \) satisfying the predicate \( \mathcal{P}^{S_i} \) could produce the observation \( y_j = M_j(x') = M_j(x) \). Predicate \( \mathcal{P}^{S_i} \) is the estimate of the current state of the system by the local supervisor \( S_j \). A local supervisor \( S_j \) disables an event \( \alpha_j \in \Sigma_j \cap \Sigma_c \) if \( sp_{\alpha_j}(\mathcal{P}^{S_i}) \not\in \mathcal{R} \).

If two or more supervisors regulate the same event, the combination rule amounts to enabling that event if at least one local supervisor enables the event. To rationalize this combination rule, suppose that at state \( x \in X \) there are observations \( y_j \) and \( y_i \) for the supervisors \( S_j \) and \( S_i \), respectively. Furthermore, suppose that an event \( \alpha \in \Sigma_c \) is regulated by both supervisors. If \( \alpha \) happens to be enabled by \( S_j \), that is \( \alpha \in S_j(y_j) \), then \( sp_{\alpha}(\mathcal{P}^{S_j}) \not\in \mathcal{R} \). Because \( \mathcal{P}^{S_j}(x) = 1 \) we conclude that \( \mathcal{P}(z) \not\in \mathcal{P}^{S_j} \) and \( sp_{\alpha}(\mathcal{P}(z)) \not\in \mathcal{R} \). Consequently, if \( \alpha \not\in S_i(y_i) \), i.e. \( S_i \) disables event \( \alpha \), \( \alpha \)-disablement is the result of the imperfect observation of \( x \) by \( S_i \). To ensure the correct decision, the combination rule must enable the union of subsets of events that the local supervisors can enable. In what follows we assume that every controllable event is regulated by at least one local supervisor. If this assumption does not hold, one could treat an unregulated controllable event as an uncontrollable event in the context of decentralized predicate control.

To characterize the predicates that can be realized using a static decentralized supervisor for \( G \) as explained above, consider the predicate transformer \( sp_g : \mathcal{P}_X \rightarrow \mathcal{P}_X \) where for \( Q \in \mathcal{P}_X \) and \( \sigma \in \Sigma \) we have

\[
(sp_g)_\sigma(Q) = \bigvee_{x \in \mathcal{X}_Q} (sp_g)_\sigma(\mathcal{P}(x)),
\]
Here, \( J_\sigma = \{ j \in J \mid \sigma \in \Sigma_j \} \) and \( \cdot|\cdot \) is used as an “everywhere” operator.
That is the operator \( \cdot|\cdot \) evaluates either to \( \text{FALSE} \) or \( \text{TRUE} \) predicates.

**Definition (6.3.4):** Statically Decentralizable Predicates

Given the system \( G = (\mathcal{P}_X, \Sigma, sp, I) \), a class of work spaces \( \mathcal{W} = \{ (Y_j, M_j, \Sigma_j) \mid j \in J \} \), and a desired predicate \( R \in \mathcal{P}_X \) such that \( I \setminus R \), the predicate \( R \) is said to be statically decentralizable w.r.t. \( \mathcal{W} \) if the following two conditions hold,

(i) \( sp_u(R) \leq R \) for any \( u \in \Sigma_u \),

(ii) \( (sp_\gamma)^*(I) = R \).

where \( sp_\gamma(\cdot) \) is defined in (6.3.1).

**Proposition (6.3.5):** Given the previous problem setting, if \( R \) is statically decentralizable w.r.t. \( \mathcal{W} \) then \( R \) is controllable.

**Proof:** It is enough to show that \( (sp_\gamma)^*(I) = R \). First we show that for any \( Q \in \mathcal{P}_X \) such that \( Q \preceq R \), we have \( sp_\gamma(Q) \preceq (sp_\gamma)(Q) \).

To this end consider

\[
(s_p_\gamma)(Q) = \bigvee_{\sigma \in \Sigma} \bigvee_{x \in X_Q} (sp_{\sigma}^p(\mathcal{P}_{\{x\}})),
\]

\[
= \bigvee_{\sigma \in \Sigma} \bigvee_{x \in X_Q} sp_{\sigma}(\mathcal{P}_{\{x\}}) \wedge [\exists j \in J_\sigma : sp_{\sigma}(R \wedge M_j^{-1} M_j \mathcal{P}_{\{x\}}) \preceq R]. \tag{6.3.2.A}
\]

Since \( Q \preceq R \) we have \( \mathcal{P}_{\{x\}} \preceq R \) for any \( x \in X_Q \). Therefore (6.3.2.A) becomes

\[
= \bigvee_{\sigma \in \Sigma} \bigvee_{x \in X_Q} sp_{\sigma}(R \wedge \mathcal{P}_{\{x\}}) \wedge [\exists j \in J_\sigma : sp_{\sigma}(R \wedge M_j^{-1} M_j \mathcal{P}_{\{x\}}) \preceq R]. \tag{6.3.2.B}
\]

Since \( sp_{\sigma}(R \wedge \mathcal{P}_{\{x\}}) \preceq sp_{\sigma}(R \wedge M_j^{-1} M_j \mathcal{P}_{\{x\}}) \), (6.3.2.B) can be written as

\[
= \bigvee_{\sigma \in \Sigma} \bigvee_{x \in X_Q} sp_{\sigma}(R \wedge \mathcal{P}_{\{x\}}) \wedge R \wedge [\exists j \in J_\sigma : sp_{\sigma}(R \wedge M_j^{-1} M_j \mathcal{P}_{\{x\}}) \preceq R] \]

Given

\[ \forall_{\sigma \in \Sigma} \forall_{x \in X} \quad (sp \ | R)(P_{\{x\}}) \]
\[ \leq (sp \ | R)(Q). \]

Using the definition of \((sp \ | R)(\cdot)\) we have

\[ sp_g(Q) \leq (sp \ | R)(Q) \leq R. \quad (6.3.2.C) \]

Replacing \(Q\) by \(I\) and applying (6.3.2.C) recursively, and using the property

\[ (sp_g)^*(I) = R, \]

we have

\[ R = (sp_g)^*(I) \leq (sp \ | R)^*(I) \leq R, \]

which is equivalent to \((sp \ | R)^*(I) = R\).

\[ \Delta \]

The converse of this proposition does not hold in general.

**Proposition (6.3.6):** Given the previous problem setting, if the predicate \(R\) is statically decentralizable then there exists a class of local supervisors \(\{S_j \mid j \in J\}\) that jointly realize the predicate \(R\).

**Proof:** The proof is by construction. Consider a class of local supervisors \(\{S_j : Y_j \rightarrow 2^{\Sigma_j} \mid j \in J\}\) where \(S_j\) is defined as follows:

\[ S_j(y_j) = \left\{ \sigma \in \Sigma_j \mid sp_\sigma(R \wedge P_{M_j^{-1}(y_j)}) \leq R \right\}, \quad \text{for any } y_j \in Y_j. \quad (6.3.3) \]

Let \(S_g : X \rightarrow 2^\Sigma\) be the global supervisor whose function is determined by the concurrent supervision of the local supervisors and the combination rule as discussed before. Then we can write

\[ S_g(x) = \bigcup_{j \in J} \{S_j(M_j(x))\}. \]
as follows:

\[ S_g / G := (\mathcal{P}_X, \Sigma, sp_g, \mathcal{I}), \]

where the \( \sigma \)-occurrence predicate in the closed loop system, denoted by \( C^g_\sigma \), can be characterized as:

\[
C^g_\sigma(x) = \begin{cases} 
1, & \text{iff } C_\sigma(x) = 1 \text{ and } \exists j \in J_\sigma \text{ such that} \\
sp_\sigma(\mathcal{R} \land M_j^{-1} M_j \mathcal{P}(x)) \preceq \mathcal{R}, \\
0, & \text{otherwise.}
\end{cases}
\]

Alternatively we can write

\[
C^g_\sigma(\cdot) = C_\sigma(\cdot) \land [\exists j \in J_\sigma :: sp_\sigma (\mathcal{R} \land M_j^{-1} M_j \mathcal{P}(x)) \preceq \mathcal{R}],
\]

and for \( Q \in \mathcal{P}_X \) we have

\[
(sp_g)_\sigma(Q) = \bigvee_{x \in X_Q} (sp_g)_\sigma(\mathcal{P}(x))
\]

\[
= \bigvee_{x \in X_Q} sp_\sigma(\mathcal{P}(x)) \land [\exists j \in J_\sigma :: sp_\sigma (\mathcal{R} \land M_j^{-1} M_j \mathcal{P}(x)) \preceq \mathcal{R}].
\]

For \( S_j \) to be a local supervisor we need to show that it does not disable an uncontrollable event in its local work space. To show this, take any \( \sigma \in \Sigma_j \cap \Sigma_u \) and \( y_j \in Y_j \). Using (i) of Definition (6.3.4) we can write

\[
sp_\sigma(\mathcal{R} \land \mathcal{P}_{M_j^{-1}(y_j)}) \preceq sp_\sigma(\mathcal{R}) \preceq \mathcal{R}.
\]

Thus \( \Sigma_j \cap \Sigma_u \subseteq S_j(y_j) \) for any \( y_j \in Y_j \). Furthermore, the local supervisors’ joint action realizes the predicate \( (sp_g)^*(\mathcal{I}) \). By (ii) of Definition (6.3.4) we conclude that this predicate is the desired predicate \( \mathcal{R} \).

\[ \triangle \]

**Example (6.3.7):** Consider the DEDS depicted in Figure (6.3.1).
Here $X = \{1, 2, \cdots, 7\}$, $\Sigma_c = \{\alpha, \beta, \gamma\}$, $\Sigma_u = \{\eta\}$ and $X_I = \{1\}$. Assume the desired predicate is given by $R$ where $X_R = \{1, 4, 5, 6, 7\}$. It is easy to check that $\forall \sigma \in \Sigma_u \ sp_\sigma(R) \leq R$ and $(sp|_R)^*(I) = R$. Thus $R$ is controllable. Now consider the following two work spaces:

$$(Y = \{y_1, y_2\}, M_1, \Sigma_1 = \{\alpha, \beta\}) \quad M_1 : \begin{cases} \{1, 7, 4\} \rightarrow y_1 \\ \{2, 3, 5, 6\} \rightarrow y_2 \end{cases}$$

$$(Z = \{z_1, z_2, z_3\}, M_2, \Sigma_2 = \{\alpha, \gamma, \eta\}) \quad M_2 : \begin{cases} \{1, 5\} \rightarrow z_1 \\ \{3, 4\} \rightarrow z_2 \\ \{2, 7, 6\} \rightarrow z_3 \end{cases}$$

To see if we can realize $R$ in a decentralized fashion using the above work spaces, it is enough to check if $(sp_\eta)^*(I) = R$. We begin by evaluating

$$sp_\eta(I) = sp_\eta(P_{\{1\}}),$$

$$= \bigvee_{\sigma \in \Sigma} sp_\sigma(P_{\{1\}}) \land \exists j \in \Sigma : sp_\sigma(R \land M_{j\sigma}^{-1} M_j, P_{\{1\}}) \leq R,$$

$$= (sp_\eta(P_{\{1\}}) \land FALSE) \lor (sp_{\alpha}(P_{\{1\}}) \land TRUE) = sp_{\alpha}(P_{\{1\}}) = P_{\{7\}}.$$

Continuing to evaluate $sp_\eta^k(I)$ for $k \geq 2$ we have

$$sp_\eta^2(I) = sp_\eta(P_{\{7\}}) = P_{\{4\}}, \quad sp_\eta^4(I) = sp_\eta(P_{\{5\}}) = P_{\{5\}},$$

$$sp_\eta^3(I) = sp_\eta(P_{\{4\}}) = P_{\{5\}}, \quad sp_\eta^5(I) = sp_\eta(P_{\{5\}}) = P_{\{1,7\}}.$$
\[ \mathcal{P}_{\{1,4,5,6,7\}} = \mathcal{R} = \bigvee_{k=0}^{\infty} (sp_g)^k(I) \preceq (sp_g)^*(I) \subseteq \mathcal{R}. \]

Therefore \( \mathcal{R} \) is statically decentralizable. Indeed, the following supervisors realize the predicate \( \mathcal{R} \).

\[
S_1: \begin{cases} 
  y_1 \rightarrow \emptyset, \\
  y_2 \rightarrow \{\alpha, \beta\}, \\
  z_1 \rightarrow \{\alpha, \gamma, \eta\}.
\end{cases}
\]

\[
S_2: \begin{cases} 
  z_2 \rightarrow \{\alpha, \eta\}, \\
  z_3 \rightarrow \{\gamma, \eta\}.
\end{cases}
\]

Notice that if \( M_2 \) is changed to

\[
M_2: \begin{cases} 
  \{1,7\} \rightarrow z_1, \\
  \{3,4\} \rightarrow z_2, \\
  \{2,5,6\} \rightarrow z_3,
\end{cases}
\]

then state 7 is not reachable in the DEDS and the predicate \( \mathcal{R} \) would not be statically decentralizable.

\[\triangle\]

Parallel to the development in [48], next we introduce the notion of rational supervision in the context of decentralized predicate control.

**Definition (6.3.8): Rational decentralized predicate control**

Given the previous problem setting, consider a class of local supervisors \( \{S_j \mid j \in J\} \) whose concurrent supervision realizes the predicate \( \mathcal{R} \) in \( G \) and can be represented by a centralized supervisor \( S_g : X \rightarrow 2^X \). We call the supervision scheme rational if the following condition holds

\[
(\forall x \in X_R) \quad \mathcal{P}_x \preceq \mathcal{R} \land \mathcal{w}_\sigma(\mathcal{R}) \implies \sigma \in S_g(x).
\]
\( \mathcal{R} \wedge wp_\sigma(\mathcal{R}) \) evaluates to one. Therefore a rational supervision scheme does not unnecessarily disable any event.

Considering the type of decentralization that we suggested before, the following remarks are in order.

**Remark (6.3.9):** The decentralized supervision scheme suggested before may not be a rational one. To see this consider a case that at a state \( x \) the event \( \sigma \) is defined such that the execution of \( \sigma \) transfers \( x \) to \( x' \), where both \( x \) and \( x' \) are in \( X_\mathcal{R} \). It may happen that all \( \sigma \)-regulating local supervisors disable the event \( \sigma \) and yet \( x' \) is reached in the closed loop system via some other states and transitions. In Example (6.3.7) the decentralized supervision scheme is not rational because it unnecessarily disables event \( \beta \) at states 4 and 7.

\[ \triangle \]

**Remark (6.3.10):** The supervision problem was formulated using the predicate calculus. This calculus is used by many researchers because it promises a way of applying the results to infinite state DEDSs. However, the type of conditions that we have obtained up to now, i.e. the conditions of Definition (6.3.4), do not appear to be applicable to infinite state DEDSs. Anyhow, for finite state DEDSs, it can be shown that the conditions of Definition (6.3.4) can be tested in polynomial time, where complexity is measured w.r.t. the cardinality of the state space.

\[ \triangle \]

Next, we obtain another sufficient condition that guarantees rational predicate decentralization and can be tested for infinite state DEDSs using the
Assume that $|J_\sigma| = 1$ for any $\sigma \in \Sigma$. That is any event is regulated by just one local supervisor. Considering the type of condition that we will obtain, this assumption is not restrictive. We will discuss this point later on in the chapter. Furthermore, this assumption is in accord with the modular formulation of predicate control [81], where a supervisor is specified by its $\sigma$-components. Suppose $f : X \rightarrow 2^\Sigma$ is a centralized supervisor that realizes $\mathcal{R}$. A $\sigma$-component of $f$, denoted by $f_\sigma$, is defined as following:

$$f_\sigma : X \rightarrow \{0, 1\} \quad \text{such that} \quad f_\sigma(x) = 1 \iff \sigma \in f(x).$$

$f$ is completely characterized by its $\sigma$-components for any $\sigma \in \Sigma_c$. In a modular solution of the SPCP one constructs $\sigma$-components of the desired supervisor. If $\mathcal{R}$ is controllable, then the $f_\sigma$'s defined below characterize a rational centralized supervisor that realizes $\mathcal{R}$ [42, 48].

For any $\sigma \in \Sigma_c$, \quad $f_\sigma(x) = \begin{cases} 
0, & \text{if } (\mathcal{R} \land \wp_\sigma(\neg\mathcal{R})) (x) = 1, \\
1, & \text{otherwise.} 
\end{cases}$

In a modular characterization of the supervisor every $\sigma$-component of the supervisor has complete knowledge of the state of the system, i.e. $f_\sigma$ is a map from $X$ to $\{0, 1\}$. It may be the case, however, that a $\sigma$-component of a supervisor can make the correct decisions with only partial state observations. It is this aspect of $\sigma$-components that we examine next.

**Definition (6.3.11): Rationally Decentralizable Predicates**

Given the system $G = (\mathcal{P}_X, \Sigma, sp, I)$, a class of work spaces $\mathcal{W} = \{(Y_j, M_j, \Sigma_j) \mid j \in J\}$, and a desired predicate $\mathcal{R} \in \mathcal{P}_X$ where we assume
the following three conditions hold.

(i) $sp_u(\mathcal{R}) \leq \mathcal{R}$ for any $u \in \Sigma_u$,  \\
(ii) $(sp|\mathcal{R})^*(I) = \mathcal{R}$,  \\
(iii) $\mathcal{R} \wedge M_{j_\sigma}^{-1} M_{j_\sigma} (\mathcal{R} \wedge w\rho_\sigma(\mathcal{R})) \preceq (\mathcal{R} \wedge w\rho_\sigma(\mathcal{R}))$, for any $\sigma \in \Sigma_c$.

In (iii) $j_\sigma$ is the index of the local supervisor that regulates the controllable event $\sigma$. Conditions (i) and (ii) require that the predicate $\mathcal{R}$ be controllable. $\mathcal{R} \wedge w\rho_\sigma(\mathcal{R})$ is the predicate at which a rational supervisor must enable the event $\sigma$ and $\mathcal{R} \wedge w\rho_\sigma(\mathcal{R})$ is the predicate at which the $j_\sigma$th local supervisor does not disable event $\sigma$. Thus condition (iii) requires that partial observation of the state by the $j_\sigma$th local supervisor does not result in disabling event $\sigma$ if transition under $\sigma$ does not violate the desired predicate. Also notice that condition (iii) is always satisfied under perfect observations, in the case $M_{j_\sigma}(\cdot)$ is the identity map.

Proposition (6.3.12): Given the previous problem formulation, if $\mathcal{R}$ is rationally decentralizable then $\mathcal{R}$ is also statically decentralizable.

Proof: We will show that conditions (ii) and (iii) of Definition (6.3.11) together imply $(sp_g)^*(I) = \mathcal{R}$, where for any $Q \in P_X$ and $\sigma \in \Sigma$, $sp_g : P_X \rightarrow P_X$ is defined by:

$$(sp_g)_\sigma(Q) = \bigvee_{x \in X_Q} (sp_g)_\sigma(P_{\{x\}})$$

$$= \bigvee_{x \in X_Q} sp_\sigma(P_{\{x\}}) \wedge [sp_\sigma(\mathcal{R} \wedge M_{j_\sigma}^{-1} M_{j_\sigma} P_{\{x\}}) \preceq \mathcal{R}]. \quad (6.3.4)$$

Notice that the assumption $|J_\sigma| = 1$ has changed the argument inside $[\cdot]$ in (6.3.4) when compared to that of (6.3.1). We first show that

$$\forall \sigma \in \Sigma \text{ and } \forall Q \preceq \mathcal{R} \quad (sp_\sigma|\mathcal{R})(Q) \preceq (sp_g)_\sigma(Q). \quad (6.3.5)$$
Since \((R \land M_{j\sigma}^{-1} M_{j\sigma} P_{\{x\}}) \leq R\) using (i) and the monotonicity of \(sp(\cdot)\) we have
\[
sp_{\sigma} (R \land M_{j\sigma}^{-1} M_{j\sigma} P_{\{x\}}) \leq sp_{\sigma}(R) \leq R.
\]
Thus
\[
[sp_{\sigma} (R \land M_{j\sigma}^{-1} M_{j\sigma} P_{\{x\}}) \leq R] = TRUE.
\]
Therefore for any \(Q \leq R\) we have
\[
(sp_g)_{\sigma}(Q) = \bigvee_{x \in X_Q} (sp_g)_{\sigma}(P_{\{x\}})
\]
\[
= \bigvee_{x \in X_Q} sp_{\sigma}(P_{\{x\}}) \land [sp_{\sigma} (R \land M_{j\sigma}^{-1} M_{j\sigma} P_{\{x\}}) \leq R]
\]
\[
= \bigvee_{x \in X_Q} sp_{\sigma}(P_{\{x\}}) \land TRUE.
\]
We also have \(sp_{\sigma}(P_{\{x\}}) \leq sp_{\sigma} (R \land M_{j\sigma}^{-1} M_{j\sigma} P_{\{x\}}) \leq R\). Thus we can continue from the last equality as follows:
\[
= \bigvee_{x \in X_Q} sp_{\sigma}(P_{\{x\}} \land R) \land R
\]
\[
= \bigvee_{x \in X_Q} (sp_{\sigma} |R)(P_{\{x\}})
\]
\[
= (sp_{\sigma} |R)(Q).
\]

Case 2: \(\sigma \in \Sigma_c\)

Take \(x \in X_R\) such that \((sp_{\sigma} |R)(Q)(x) = 1\). Then there exists \(x' \in X_Q = X_R\) such that \(P_{\{x\}} = (sp_{\sigma} |R)(P_{\{x'\}}) \leq R\). Thus
\[
P_{\{x'\}} \leq R \land wp_{\sigma}(R),
\]
\( R \land M_{j_0}^{-1} M_{j_0} P_{\{x'\}} \preceq R \land M_{j_0}^{-1} M_{j_0} (R \land w p_\sigma(R)). \)

By (iii) of Definition (6.3.11) we conclude that

\[ R \land M_{j_0}^{-1} M_{j_0} (R \land w p_\sigma(R)) \preceq R \land w l p_\sigma(R). \]

Thus

\[ sp_\sigma(R \land M_{j_0}^{-1} M_{j_0} P_{\{x'\}}) \preceq sp_\sigma(R \land w l p_\sigma(R)), \]

\[ \preceq R. \]

Therefore

\[ [sp_\sigma(R \land M_{j_0}^{-1} M_{j_0} P_{\{x'\}}) \preceq R] = TRUE, \]

and

\[ (sp_\sigma | R)(P_{\{x'\}}) \preceq (sp_\sigma)_\sigma(Q). \]

This results in

\[ (sp_\sigma)_\sigma(Q)(x) = 1, \]

or

\[ (sp_\sigma | R)(Q) \preceq (sp_\sigma)_\sigma(Q). \tag{6.3.6} \]

Cases 1 and 2 establish (6.3.5). By definition we have that \((sp_\sigma | _\kappa)(\cdot) \preceq R\) and \((sp_\sigma)_\sigma(\cdot) \preceq R\). Therefore (6.3.5) can be written as:

\[ \forall Q \preceq R \quad (sp | R)(Q) \preceq (sp_\sigma)(Q) \preceq R. \tag{6.3.7} \]

Replacing \( Q \) by \( I \) in (6.3.7), applying (6.3.7) recursively and using (ii) of Definition (6.3.11) we obtain

\[ R = (sp | R)^*(I) \preceq (sp_\sigma)^*(I) \preceq R. \]

That is

\[ R = (sp_\sigma)^*(I). \]
Corollary (6.3.13): If predicate $\mathcal{R}$ is rationally decentralizable, then the decentralization scheme that suggested previously will realize $\mathcal{R}$.

\[ \Delta \]

Proposition (6.3.14): If the predicate $\mathcal{R}$ is rationally decentralizable, then there exists a rational decentralized supervision scheme that realizes $\mathcal{R}$.

Proof: We prove the proposition by constructing a rational decentralized supervision scheme. By Proposition (6.3.12) and Corollary (6.3.13) we know that the decentralized supervision scheme suggested previously realizes the predicate $\mathcal{R}$. Thus it suffices to show that this scheme is a rational one. Let $x \in \mathcal{X}_\mathcal{R}$ and $\sigma \in \Sigma_c$. Assume that $\mathcal{P}_{\{x\}} \preceq \mathcal{R} \land wp_\sigma(\mathcal{R})$. We must show that $\sigma \in S_g(x)$. We have

\[ \mathcal{P}_{\{x\}} \preceq \mathcal{R} \land wp_\sigma(\mathcal{R}), \]

\[ \mathcal{R} \land M^{-1}_{j_\sigma} M_{j_\sigma} \mathcal{P}_{\{x\}} \preceq \mathcal{R} \land M^{-1}_{j_\sigma} M_{j_\sigma} (\mathcal{R} \land wp_\sigma(\mathcal{R})). \]

Because the predicate $\mathcal{R}$ is rationally decentralizable, we can write

\[ \mathcal{R} \land M^{-1}_{j_\sigma} M_{j_\sigma} \mathcal{P}_{\{x\}} \preceq \mathcal{R} \land wlp_\sigma(\mathcal{R}), \]

\[ sp_\sigma(\mathcal{R} \land M^{-1}_{j_\sigma} M_{j_\sigma} \mathcal{P}_{\{x\}}) \preceq sp_\sigma(\mathcal{R} \land wlp_\sigma(\mathcal{R})) \preceq \mathcal{R}. \]

Using (6.3.3) we conclude that $\sigma \in S_{j_\sigma}(M_{j_\sigma}(x))$. Therefore, as a result of our combination rule we have $\sigma \in S_g(x)$.

\[ \Delta \]

Remark (6.3.15): We assumed that every event $\sigma$ is regulated by just one local supervisor whose index is $j_\sigma$. We may remove this assumption and require
satisfied for \( j_\sigma \). However, in this case the local supervisor indexed by \( j_\sigma \) will always regulate \( \sigma \) correctly. Therefore other \( \sigma \)-regulating local supervisors could be eliminated, thereby reducing the number of necessary communication and command channels. The assumption that \( |J_\sigma| = 1 \) is consistent with this.

\( \triangle \)

**Remark (6.3.16):** The set of conditions in Definition (6.3.11) are only sufficient. That is, it is possible to have a rational supervisor for a desired predicate even though that predicate does not satisfy the sufficient conditions. However, the conditions of Definition (6.3.11) are such that one can use the predicate calculus to test them. Therefore it may be possible to test these conditions even for infinite state DEDSs.

\( \triangle \)

**Example (6.3.17):** This is a modified version of an example reported in [48]. Consider a simple manufacturing system which consists of two types of feeders, two buffers and a number of assemblers. Feeders of type 1 and type 2 produce parts of type 1 and type 2, respectively. A supervisor can initiate production of a part but does not have any control on the time that the corresponding feeder finishes production and submits the finished part to its corresponding buffer. An assembler needs two parts of type 1 and one part of type 2 in order to assemble a target product. A supervisor has control over initiating an assembly but does not control the time that an assembler takes to finish its job. The following figure depicts the DEDS model of the manufacturing process.
Here,

\( x_1 \): number of idle feeders of type 1;

\( x_2 \): number of working feeders of type 1;

\( x_3 \): number of finished parts of type 1 in buffer 1;

\( x_4 \): number of idle feeders of type 2;

\( x_5 \): number of working feeders of type 2;

\( x_6 \): number of finished parts of type 2 in buffer 2;

\( x_7 \): number of idle assemblers;

\( x_8 \): number of working assemblers;

\( \alpha_1^1 \): a feeder of type 1 is initiated, a controllable event;

\( \alpha_1^2 \): a feeder of type 1 is finished and submits the part to buffer 1, an uncontrollable event;

\( \alpha_2^1 \): a feeder of type 2 is initiated, a controllable event;

\( \alpha_2^2 \): a feeder of type 2 is finished and submits the part to buffer 2, an uncontrollable event.
\( \alpha_1^3 \): an assembler is initiated, a controllable event;

\( \alpha_2^3 \): an assembler is finished assembling a target product, an uncontrollable event.

Suppose that buffers 1 and 2 have capacity limitations \( B_1 \) and \( B_2 \), respectively, and our objective is to keep the contents of the buffers under the capacity limits. In this example we do not impose any limitation on the overall number of feeders or assemblers. Thus our DEDS can be modeled as follows:

\[
G := (\mathcal{P}_{N^8}, \Sigma, sp, I)
\]

where \( N \) stands for natural numbers, \( x = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8] \) is a typical state and the initial predicate is

\[
I = ((x_1 \geq 1) \land (x_4 \geq 1) \land (x_7 \geq 1))
\]

\[
\land ((x_2 = 0) \land (x_5 = 0) \land (x_8 = 0))
\]

\[
\land ((x_3 = 0) \land (x_6 = 0)).
\]

The events occurrence predicates are

\[
C_{\alpha_1^1} = (x_1 \geq 1),
\]

\[
C_{\alpha_2^1} = (x_2 \geq 1),
\]

\[
C_{\alpha_1^2} = (x_4 \geq 1),
\]

\[
C_{\alpha_3^2} = (x_5 \geq 1),
\]

\[
C_{\alpha_4^2} = (x_3 \geq 2) \land (x_6 \geq 1) \land (x_7 \geq 1),
\]

\[
C_{\alpha_5^2} = (x_8 \geq 1).
\]

The transition characteristics of the system at state \( x \in N^8 \) are given by

\[
\delta(\alpha_1^1, x) = x + [-1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0],
\]
\[
\delta(\alpha_1^2, x) = x + [0 \ 0 \ 0 \ -1 \ 1 \ 0 \ 0 \ 0], \\
\delta(\alpha_2^2, x) = x + [0 \ 0 \ 0 \ 1 \ -1 \ 1 \ 0 \ 0], \\
\delta(\alpha_3^3, x) = x + [0 \ 0 \ -2 \ 0 \ 0 \ -1 \ -1 \ 1], \\
\delta(\alpha_2^3, x) = x + [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ -1].
\]

Finally, our objective predicate is

\[
\mathcal{P}_{obj} = (x_3 \leq B_1) \land (x_6 \leq B_2).
\]

Since \(\mathcal{P}_{obj}\) is not invariant under uncontrollable event transitions, i.e. \(sp_{\alpha_i^j}(\mathcal{P}_{obj}) \nless \mathcal{P}_{obj}\), we first find the weakest controllable predicate that is stronger than \(\mathcal{P}_{obj}\) [42] and denote this predicate by \(\mathcal{R}\). One can evaluate \(\mathcal{R}\) using the formulas given in [42]. However we avoid tedious computations and claim that \(\mathcal{R}\) is the following:

\[
\mathcal{R} = \big\{ \bigvee_{m=0}^{B_1} (x_2 \leq m \land x_3 \leq B_1 - m) \big\} \land \big\{ \bigvee_{n=0}^{B_2} (x_5 \leq n \land x_6 \leq B_2 - n) \big\}. \quad (6.3.8)
\]

To justify our claim we note that \(\mathcal{R}\) is weaker than \(\mathcal{P}_{obj}\), it is invariant under uncontrollable events transitions, and given any state satisfying \(\mathcal{R}\) there exists an initial state that satisfies the initial predicate and from which we can reach the given state. Also, any state that violates \(\mathcal{R}\) can reach to a state violating \(\mathcal{P}_{obj}\) only through uncontrollable events. Now consider the following work spaces:

\[
W_1 = (\mathcal{N}^3, M_1 = proj_{(x_1,x_2,x_3)}, \{\alpha_1^1, \alpha_2^1\}), \\
W_2 = (\mathcal{N}^3, M_2 = proj_{(x_4,x_5,x_6)}, \{\alpha_1^2, \alpha_2^2\}), \\
W_3 = (\mathcal{N}^3, M_3 = proj_{(x_7,x_8)}, \{\alpha_1^3, \alpha_2^3\}).
\]
\[ M_2[x_1 \ x_2 \ x_3 \ x_4 \ x_5 \ x_6 \ x_7 \ x_8] = [x_4 \ x_5 \ x_6]. \]

Since \( \mathcal{R} \) is controllable, to see if \( \mathcal{R} \) is rationally decentralizable it is enough to check

\[ \mathcal{R} \land M^{-1}_{j_0}(\mathcal{R} \land \mathcal{W}_p(\mathcal{R})) \leq \mathcal{R} \land \mathcal{W}_p(\mathcal{R}) \quad \text{for any} \quad \sigma \in \Sigma_c. \quad (6.3.9) \]

1) Event \( \alpha_1^i \):

\[
wp_{\alpha_1^i}(\mathcal{R}) = \bigvee_{m=0}^{B_1} (x_2 + 1 \leq m \land x_3 \leq B_1 - m) \\
\land \bigvee_{n=0}^{B_2} (x_5 \leq n \land x_6 \leq B_2 - n) \\
\land (x_1 \geq 1),
\]

\[ \mathcal{R} \land \mathcal{W}_p(\mathcal{R}) = \mathcal{W}_p(\mathcal{R}), \]

\[ M^{-1}_1 M_1(\mathcal{R} \land \mathcal{W}_p(\mathcal{R})) = \bigvee_{m=0}^{B_1} (x_2 + 1 \leq m \land x_3 \leq B_1 - m) \land (x_1 \geq 1), \]

\[ \mathcal{R} \land M^{-1}_1 M_1(\mathcal{R} \land \mathcal{W}_p(\mathcal{R})) = \bigvee_{m=0}^{B_1} (x_2 + 1 \leq m \land x_3 \leq B_1 - m) \\
\land \bigvee_{n=0}^{B_2} (x_5 \leq n \land x_6 \leq B_2 - n) \\
\land (x_1 \geq 1),
\]

\[ \mathcal{R} \land M^{-1}_1 M_1(\mathcal{R} \land \mathcal{W}_p(\mathcal{R})) = \mathcal{W}_p(\mathcal{R}), \]

On the other hand,

\[ \mathcal{R} \land M^{-1}_1 M_1(\mathcal{R} \land \mathcal{W}_p(\mathcal{R})) = \mathcal{W}_p(\mathcal{R}) = \mathcal{R} \land \mathcal{W}_p(\mathcal{R}) \leq \mathcal{R} \land \mathcal{W}_p(\mathcal{R}). \]

That is (6.3.9) holds for \( \alpha_1^i \).
Event $a_1^2$:

$$wp_{a_1^2}(R) = \bigvee_{m=0}^{B_1} (x_2 \leq m \land x_3 - 2 \leq B_1 - m) \land \bigvee_{n=0}^{B_2} (x_5 \leq n \land x_6 - 1 \leq B_2 - n) \land C_{a_1^2}.$$ 

$$R \land wp_{a_1^2}(R) = R \land ((x_3 \geq 2) \land (x_6 \geq 1) \land (x_7 \geq 1)),$$

$$M_3^{-1}M_3(R \land wp_{a_1^2}(R)) = x_7 \geq 1,$$

$$R \land M_3^{-1}M_3(R \land wp_{a_1^2}(R)) = R \land (x_7 \geq 1).$$

On the other hand,

$$R \land wlp_{a_1^2}(R) = R \land (\neg C_{a_1^2} \lor wp_{a_1^2}(R)),$$

$$= (R \land \neg C_{a_1^2}) \lor (R \land wp_{a_1^2}(R)),$$

$$= (R \land \neg C_{a_1^2}) \lor (R \land C_{a_1^2}),$$

$$= R \land (\neg C_{a_1^2} \lor C_{a_1^2}) = R \land TRUE = R.$$ 

Since

$$R \land (x_7 \geq 1) \leq R,$$

thus

$$R \land M_3^{-1}M_3(R \land wp_{a_1^2}(R)) \leq R \land wlp_{a_1^2}(R).$$

Therefore $R$ is rationally decentralizable over the given work spaces.

\[ \Delta \]

Condition (iii) of Definition (6.3.11) is closely related to the observability condition introduced in [48]. To characterize this relationship consider the
C1: Condition (iii) of Definition (6.3.11)
\[ \mathcal{R} \wedge M^{-1} M(\mathcal{R} \wedge \varpi(\mathcal{R})) \leq \mathcal{R} \wedge \varpi(\mathcal{R}). \]

C2: Condition of proposition (6.3.1) in [48]
\[(\forall x, x' \in X_\mathcal{R}) \quad (Mx = Mx' \text{ and } x, x' \in X_c \text{ and } \delta(\sigma, x) \in X_\mathcal{R}) \implies \delta(\sigma, x') \in X_\mathcal{R}. \]

C3: Assumption made in [48]
\[(\forall x, x' \in X_c) \quad Mx = Mx' \implies M\delta(\sigma, x) = M\delta(\sigma, x'). \]

C4: Assumption made in [48]
\[(\forall x, x' \in X_c) \quad M\delta(\sigma, x) = M\delta(\sigma, x') \implies Mx = Mx'. \]

C5: Observability condition in [48]
\[ sp(\mathcal{R}) \wedge M^{-1} M(\mathcal{R} \wedge sp(\mathcal{R})) \leq \mathcal{R}. \]

Proposition (6.3.18): C1 $\iff$ C2.

Proof: (i) C1 $\implies$ C2

Take $x, x' \in X_\mathcal{R}$ such that $Mx = Mx', x, x' \in X_c$ and $\delta(\sigma, x) \in X_\mathcal{R}$. We then have $(\mathcal{R} \wedge \varpi(\mathcal{R}))(x) = 1$. Therefore $(M^{-1} M(\mathcal{R} \wedge \varpi(\mathcal{R}))(x) = 1$. Since $Mx = Mx'$, we have $(M^{-1} M(\mathcal{R} \wedge \varpi(\mathcal{R}))(x') = 1$. We also have $x' \in X_\mathcal{R}$, thus $(\mathcal{R} \wedge M^{-1} M(\mathcal{R} \wedge \varpi(\mathcal{R}))(x') = 1$. Using C1 we have $(\mathcal{R} \wedge \varpi(\mathcal{R}))(x') = 1$. Since $x' \in X_c$, we conclude that
\[ (\mathcal{R} \wedge \varpi(\mathcal{R}))(x') = 1. \]
(ii) $C_2 \implies C_1$

Take $x \in X_R$ such that $(R \land M^{-1}M(R \land wlp(\sigma)(R))(x) = 1$. Then there exists $x' \in X_R$ such that $Mx = Mx'$ and $\delta(\sigma, x') \in X_R$. There are two cases to be considered. Case 1: Suppose $C_\sigma(x) = 0$. Then $wlp(\sigma)(x) = 1$, which implies that $(R \land wlp(\sigma)(R))(x) = 1$. Case 2: Suppose $C_\sigma(x) = 1$. Then by C2 we conclude that $\delta(\sigma, x) \in X_R$. That is, $(R \land wlp(\sigma)(R))(x) = 1$.

(i) and (ii) completes the proof.

Proposition (6.3.19): The following relations hold,

$C_2$ and $C_4 \implies C_5$

$C_5$ and $C_3 \implies C_2$

Proof: Refer to Lemma (3.1) and Proposition (3.1) in [48].

Propositions (6.3.18) and (6.3.19) characterize the relationship among conditions $C_1$ through $C_5$. Notice that we have not made any assumptions regarding $C_3$ and $C_4$ in our work. These assumptions, specifically assumption $C_4$, are very restrictive. In fact, most DEDSs do not satisfy assumption $C_4$ even under perfect observation, i.e. $M(\cdot)$ is the identity map.
In this chapter we investigated the decentralized supervisory predicate control problem. We introduced the notion of statically decentralized predicates and a sufficient condition for static decentralizability was derived. We also introduced the notion of rationally decentralized predicates. A sufficient condition for rationally decentralized predicates was given and the relationship between predicate decentralizability and observability was discussed.

The basic short coming of the conditions derived in this chapter are that they are only sufficient conditions. In order to obtain necessary and sufficient conditions for predicate decentralizability, it is necessary that dynamic decentralizability of predicates be considered. Here the local supervisors are allowed to be dynamic systems. There are two major obstacles when dealing with dynamic local supervisors. The first is that testing the conditions, most likely, must be done in terms of states rather than predicates. This increases the computational complexity and more importantly will result in a situation where the conditions cannot be applied to an infinite state DEDS. Secondly, we believe that in order to obtain necessary and sufficient conditions for predicate decentralizability, in addition to allowing the local supervisors to be dynamic, every local supervisor must be informed about the control decisions that the other local supervisors make. Local supervisors would need this information in order to have consistent estimates of the current state of the DEDS. Transmitting such information requires increased communications and one of the primary objectives of predicate decentralization is to reduce the number of communication channels.
7.1. Contributions of this dissertation

In part A of this work, we studied supervisory control of discrete event dynamical systems (DEDSs) where some events are not detected. We studied two basic approaches in dealing with such a situation. In one approach, we determined what events need to be detected in order to have a supervisor that realizes a given desired objective. For this, we introduced a generalized normality condition. Based on this concept of normality, several types of work spaces for supervisors for DEDSs were developed. A number of algorithms to find these different work spaces were developed. After we find a reduced work or observation space using these algorithms, we can design a supervisor that decides control actions based on the observation of only a subset of the detected events. We refer to this supervisor as a reduced events supervisor. Using an example, we demonstrated that reduced event supervisors are more robust with respect to failures of the detection system.

We also studied the supervisory control problem where the structure of the detection system for events is fixed and cannot be altered. Here, we approximated the desired objective with an objective that can be realized using
better than those reported in the literature. We also developed an algorithm to construct the approximation as well as the suboptimal supervisor based on the approximation.

Our treatment of the supervisory control problem under partial observation is strongly related to testing two algebraic conditions, namely the normality and observability conditions. We developed two polynomial time effective algorithms to test these conditions in the context of regular languages. A combination of the normality and observability algorithms and the algorithms to find a sufficient work or observation space, yields a comprehensive framework for automated construction of a supervisor when some events are not detected. For example, given an open loop DEDS and a desired language based objective, using the normality test algorithms, we first search for a sufficient work space. Once such a work space is evaluated, we synthesized a reduced supervisor based on the evaluated work space.

In part B of this dissertation, we studied the decentralized supervisory control problem for DEDSs. We first generalized an existing result, in the context of formal regular languages, by showing that decentralization is always possible if the local work spaces are appropriately designed. Our results can be used to design appropriate local work spaces for decentralized supervisory control. We also studied supervisor decentralization in the context of state predicates. Here, instead of using formal languages, we used state predicates and predicate transformers to model the objective and the DEDS, respectively. We defined a static local supervisor as well as a decentralized supervisor in this framework. We developed sufficient conditions that an objective must satisfy for the existence of a decentralized supervisor that realizes the objective,
the condition is satisfied. We also investigated two types of decentralized supervisors, namely rational and non rational decentralized supervisors. If an objective satisfies the sufficient condition we developed, a rational decentralized supervisor that realizes this objective can be designed.

7.2. Limitations of supervisory control

Recognizing the limitations of a new control discipline is always equally as important as understanding its advantages. As its advantages, supervisory control of DEDSs brings about a structured framework to deal with behavioral properties of a discrete event dynamical system. It distinguishes between a DEDS plant and its supervisor. This separation of a plant and its supervisor makes it possible to define notions of controllability, observability, optimal supervision, modular supervision, hierarchical supervision, decentralized supervision, etc. It also yields a framework to automatically synthesize supervisors for a DEDS. Nowadays systems, like a flexible manufacturing system, have the potential of achieving numerous objectives under appropriate control. Thus a framework for automating the construction of controllers can greatly increase the level of automation in man-made systems.

However, there are a number of limitations for the discipline. The one that we mentioned in the introductory chapter is the modeling limitation. Up to now, supervisory control has mainly been studied in the context of finite automatons. There are many DEDSs that can not be modeled in this way. Some efforts to extend the paradigm of supervisory control to Petri nets have been reported in the literature. However, as we discussed, many supervisory
Even in the context of finite automata, supervisory control techniques has to overcome a number of difficulties. Most of the supervisory control problems are NP complete. Considering the fact that DFA models of real world DEDSs are likely to have very large state spaces, computational problems related to the computation of a supervisor are formidable. Hopefully in the near future, faster computers and more readily available immense data storage system will remedy this problem.

Algorithms of supervisory control, in general, require representations of a DEDS and an objective using finite state automata. Still, there is no structured procedure to construct such models for real world systems.

Also, for supervisory control problem, a plant is thought of as a language generator while the supervisor is considered as a language recognizer that passively disables some controllable events. This modeling assumption has important consequences. One for example is the ability to construct supervisors that realize the supremal controllable sublanguages of a desired language. However, in real world applications, control modules provide inputs to the controlled plant and the plant responds. This discrepancy, makes it harder to model a real world problem using this supervisory control framework.

Another important limitation of current formulation of supervisory control is the lack of temporal notions in the modeling framework. Many researchers are considering the problem of real-time supervisory control. However to date, there is no single modeling framework for real-time supervisory control and the initial research results show that consideration of time immensely increases the complexity of the decision algorithms that are required.
Any research that alleviates some of the limitations of supervisory control, that we mentioned in the previous section, will contribute to the theory.

Another important area that needs to be worked on is the real world application of supervisory control techniques. While there is considerable amount of work on the theoretical aspects of supervisory control, only a few application oriented studies have been reported in the literature.

Any work on the implementation issues related to supervisory control would also contribute to the advancement of the field. A part of this dissertation that relates to the minimization of the event sensors and actuators in the plant could be considered as an effort to address some of the important implementation issues. Also, it is important to investigate what types of existing industrial control modules are more appropriate for the implementation of supervisors for DEDSs. We believe that Programmable Logic Controllers (PLCs) are a suitable candidate for the implementation of a supervisor in a manufacturing setting. A research and development effort that addresses issues related to a PLC based supervisor would be a valuable effort.

More closely related to the topics of this dissertation, the continuation of research on decentralized supervisory control, in the context of state predicates, is needed. In this dissertation, we only investigated statically decentralized predicate supervisors. More work needs to be done on the dynamic version of the decentralized supervisory control problem.


161


