# MODELING AND CONTROL OF LOGICAL DISCRETE EVENT SYSTEMS

**Ratnesh Kumar**
*Department of Electrical Engineering*
*University of Kentucky*
*Lexington, KY 40506-0046*

**Vijay K. Garg**
*Department of Electrical and Computer Engineering*
*University of Texas at Austin*
*Austin, TX 78712-1084*

Springer
Science+Business Media, LLC

# MODELING AND CONTROL OF LOGICAL DISCRETE EVENT SYSTEMS

# THE KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE

## DISCRETE EVENT DYNAMIC SYSTEMS

*Consulting Editor*

**Yu-Chi Ho**
Harvard University

**GRADIENT ESTIMATION VIA PERTURBATION ANALYSIS**, P. Glasserman
ISBN: 0-7923-9095-4

**PERTURBATION ANALYSIS OF DISCRETE EVENT DYNAMIC SYSTEMS**
Yu-Chi Ho and Xi-Ren Cao
ISBN: 0-7923-9174-8

**PETRI NET SYNTHESIS FOR DISCRETE EVENT CONTROL OF MANUFACTURING SYSTEMS**, MengChu Zhou and Frank DiCesare
ISBN: 0-7923-9289-2

# MODELING AND CONTROL OF LOGICAL DISCRETE EVENT SYSTEMS

**Ratnesh Kumar**
*Department of Electrical Engineering*
*University of Kentucky*
*Lexington, KY 40506-0046*



**Vijay K. Garg**
*Department of Electrical and Computer Engineering*
*University of Texas at Austin*
*Austin, TX 78712-1084*

SPRINGER SCIENCE+BUSINESS MEDIA, LLC

ISBN 978-1-4613-5931-9 ISBN 978-1-4615-2217-1 (eBook)
DOI 10.1007/978-1-4615-2217-1

---

**Library of Congress Cataloging-in-Publication Data**

A C.I.P. Catalogue record for this book is available
from the Library of Congress.

---


**Originally published by Kluwer Academic Publishers in** 1995
**Softcover reprint of the hardcover 1st edition** 1995


*Printed on acid-free paper.*

Dedicated to our parents:

Nagina Prasad and Premshila Gupta
Saran Lal and Laxmi Devi Garg

# CONTENTS

# PREFACE

The field of discrete event systems has emerged to provide a formal treatment of many of the man-made systems such as manufacturing systems, communication networks, automated traffic systems, database management systems, and computer systems that are event-driven, highly complex, and not amenable to the classical treatments based on differential or difference equations. Discrete event systems is a growing field that utilizes many interesting mathematical models and techniques. In this book we focus on a high level treatment of discrete event systems, where the *order* of events, rather than their occurrence *times*, is the principal concern. Such treatment is needed to guarantee that the system under study meets desired *logical* goals. In this framework, discrete event systems are modeled by formal languages or, equivalently, by state machines.

The field of logical discrete event systems is an interdisciplinary field—it includes ideas from computer science, control theory, and operations research. Our goal is to bring together in one book the relevant techniques from these fields. This is the first book of this kind, and our hope is that it will be useful to professionals in the area of discrete event systems since most of the material presented has appeared previously only in journals. The book is also designed for a graduate level course on logical discrete event systems. It contains all the necessary background material in formal language theory and lattice theory. The only prerequisite is some degree of "mathematical maturity". Several examples and exercise problems are included in each chapter to facilitate classroom teaching. Our treatment is based on the graduate courses given at the University of Kentucky at Lexington and The University of Texas at Austin.

Chapter 1 presents the fundamentals of language theory. Much of the discussion is centered around the notion of regular languages. Regular languages are useful from a practical viewpoint since they capture the finite nature of most real-life plants and controllers. There is also a discussion of more general languages but only to the extent of its usefulness to the theory of discrete event systems.

The set of all possible languages over a given event set forms a lattice, which is the subject of Chapter 2. Some of the results in this chapter are classical, such as Tarski's fixed point theorem, while other results concerning extremal solutions of a system of inequations are new. The main motivation behind these results is to derive the controllers discussed in later chapters by solving inequations. This way of deriving controllers unifies many earlier results on supervisory synthesis under complete and partial observation.

Chapter 3 discusses the control of a discrete event system under the complete observation of events so that the controlled system meets the desired qualitative goals. The concepts of controllability (needed for the design of control-compatible supervisors), relative-closure (needed for the design of non-blocking supervisors), and modularity (needed for the modular design of supervisors) are introduced. Techniques for verifying these properties as well as for designing optimal (maximally permissive) supervisors are presented.

Chapter 4 describes the control of a discrete event system under the partial observation of events. Partial observations arise due to the lack of sufficient event sensors. The concepts of observability (needed for the design of observation-compatible supervisors), normality (needed for the design of "local" supervisors), and co-observability (needed for the decentralized control of a distributed plant) are introduced. Techniques for verifying these properties as well as for designing optimal supervisors are also given.

Chapters 3 and 4 deal with the control of *safety* properties of a discrete event system. The richer treatment of non-terminating behavior is needed for capturing *progress* properties and is the subject of the final chapter (Chapter 5). The notions of $\omega$-languages and Buchi machines are presented for describing non-terminating behaviors. The concepts of $\omega$-controllability and relative $\omega$-closure, which are needed for the design of supervisors, are introduced. Techniques for verifying these properties and designing optimal supervisors are also discussed.

The authors thank the Department of Electrical Engineering at the University of Kentucky, and the Department of Electrical and Computer Engineering at The University of Texas at Austin, where they were given the opportunity to develop courses on discrete event systems. The first author (Ratnesh Kumar) acknowledges the support of UT Austin through M.C.D. Fellowship and a Research Assistantship during his Ph.D. research on discrete event systems, and the support of the Institute of Systems Research, University of Maryland at College Park during his several visits to the Institute. During the development of this book the first author has been supported by the National Science Foundation and the Center for Robotics and Manufacturing at the University

of Kentucky, and the second author (Vijay K. Garg) has been supported by the National Science Foundation, a TRW faculty assistantship, an IBM grant, and a General Motors Centennial fellowship. We also thank Kluwer Academic Publishers, especially Ken and Alex, for their help.

We are grateful to Prof. Steven I. Marcus who introduced both of us to this exciting field. His guidance and continued mentoring has benefitted us immensely. We owe special thanks to Darren Cofer for carefully reading the manuscript of the book. We are also thankful to our colleagues—Mark Shayman, Larry Holloway, Stephane Lafortune, Michael Heymann, Feng Lin, Pravin Varaiya, Bruce Krogh, Murray Wonham, Stanley Young and all the students who took the graduate courses at UK and UT Austin—with whom we had many fruitful discussions. Finally, we thank our wives Sudha and Meenakshi whose love, support, and understanding kept us inspired in accomplishing this goal.

Ratnesh Kumar
Vijay K. Garg

# MODELING AND CONTROL OF LOGICAL DISCRETE EVENT SYSTEMS

# 1

# INTRODUCTION TO FORMAL LANGUAGE THEORY

In this chapter we review concepts from formal language theory which form a basis for the supervisory control theory initiated by Ramadge and Wonham, and subsequently extended by other researchers. A familiarity with basic set theoretic notions is assumed.

## 1.1 INTRODUCTION

A Discrete Event System (DES) is a dynamical system which evolves according to asynchronous occurrence of certain discrete changes, called *events*. For example, an event may be the arrival of a customer in a queue, completion of a task or failure of a machine in a manufacturing system, transmission of a message in a communication network, termination of a computer program, variation of a set point in a control system, etc. Thus examples of DESs include many man-made systems such as computer and communication networks, robotics and manufacturing systems, computer programs, and automated traffic systems. A DES has a discrete set of states which, unlike a physical system, may take symbolic values rather than real values; for example, a machine is either idle, working, or broken. State transitions in such systems occur at asynchronous discrete instants of time in response to events, which may also take symbolic values. Unlike most physical systems, the relationships between state transitions and events are highly irregular and usually cannot be described using differential or difference equations.

Consider for example an elevator which moves between the ground floor (floor 0), first floor (floor 1) and second floor (floor 2), and executes only two types

of motions—"up" , which results in movement to the next higher floor, and "down", which results in movement to the next lower floor. Suppose the elevator is initially at the ground floor, and the following sequence of motions occur: "up" at time $t_1$, "up" and time $t_2$, "down" at time $t_3$, and "up" at time $t_4$, where $t_1 < t_2 < t_3 < t_4$. This behavior of the elevator is shown in Figure 1.1. We can view the floor at which the elevator is at any given point of time

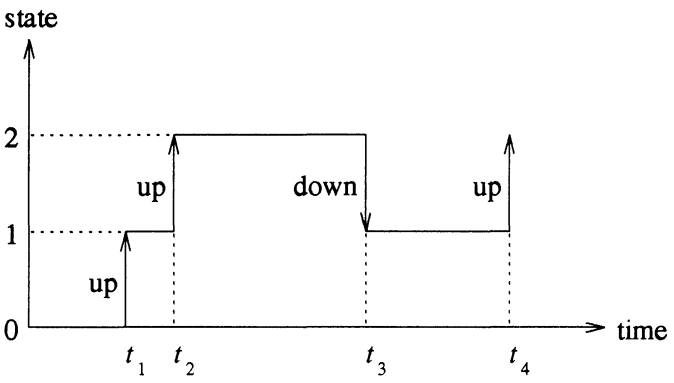



**Figure 1.1** Typical trajectory of a DES

as its state, and the motions it executes as its events. Thus the elevator has three states—0,1 and 2, and two events "up" and "down". These states of the elevator evolve according to asynchronous occurrence of the two events. The piece-wise constant "state-time trajectory" of Figure 1.1 is typical of a DES.

Letting $X$ denote the set of states and $\Sigma$ denote the set of events of a DES, its behavior can be described by a collection of sequences of triples of states, events, and time instants of the form:

$$(x_0, \sigma_1, t_1)(x_1, \sigma_2, t_2)\ldots,$$

where $x_0 \in X$ is the initial state, and for each $i \geq 1$, $x_i \in X$ denotes the $i$th state, $\sigma_i \in \Sigma$ denotes the $i$th event, and $t_i \in \mathcal{R}$ denotes the instant of occurrence of $i$th state transition. Models of DESs can be classified into untimed and timed models based on whether or not they ignore the timing information. Thus untimed models have information about *order* of state and event pairs, but not about their *timing*. Untimed models are used to control and coordinate *orderly* occurrence of states and events so that the system under study meets certain *qualitative* goals. A requirement that a message sequence must be received in the same sequence as it was sent, or that a buffer in a manufacturing system should never overflow are examples of qualitative goals. Timed models are used

when the system under study needs to be optimized for *quantitative* goals, such as average delay in a communication network, or production rate of a manufacturing system.

In this book, we are concerned with studying only qualitative behaviors of DESs, and developing techniques for controlling such behaviors.

## 1.2 LANGUAGES

At the qualitative or logical level of abstraction, the behavior of a DES is described by the set of all possible sequences of state and event pairs of the type:

$$(x_0, \sigma_1)(x_1, \sigma_2)\ldots,$$

where the symbols have the same meaning as above. A DES is said to be *deterministic* if, given the current state and an event that occurs in that state, the next state is *uniquely* determined. We primarily consider deterministic DESs in this book. Clearly, the (qualitative) behavior of a deterministic DES can be equivalently described by the set of all possible sequences of events:

$$\sigma_1\sigma_2\ldots,$$

and the initial state $x_0$. Each such event sequence is called a *trace* or *string* of the system, and a collection of traces is called a *language*. Let $\Sigma^*$ denote the set of all finite length traces consisting of events from $\Sigma$, including the zero length trace denoted $\epsilon$. Then a language is a subset of $\Sigma^*$. Symbols $H, K$ etc. are used for denoting languages. Given a string $s \in \Sigma^*$, $|s|$ is used to denote its length; if $t \in \Sigma^*$ is a *prefix* of $s$, then it is denoted as $t \leq s$; if $t \leq s$ and $|t| < |s|$, then $t$ is said to be a *proper prefix* of $s$, denoted as $t < s$.

**Example 1.1** Consider for example a buffer of capacity one; it has two different states: empty and full. When an *arrival* event occurs in the empty state, then the buffer becomes full; and when a *departure* event occurs in the full state, then the buffer becomes empty. No other state transition can occur in the buffer. Suppose initially the buffer is empty. Then the language of the buffer consists of all possible sequences of the type:

$$\text{arrival.departure.arrival.departure}\ldots,$$

where "." denotes the operation of *concatenation*. ■

### 1.2.1 Operation on Languages

languages operations onNext we consider various operations on languages which allow us to describe complex DESs in terms of simpler ones.

#### *Binary Operations*

Consider two languages $K_1, K_2 \subseteq \Sigma^*$. The *intersection* of $K_1$ and $K_2$, denoted as $K_1 \cap K_2$, is the language

$$K_1 \cap K_2 := \{s \in \Sigma^* \mid s \in K_1 \text{ and } s \in K_2\}.$$

The *difference* between $K_1$ and $K_2$, denoted as $K_1 - K_2$, is the language

$$K_1 - K_2 := \{s \in \Sigma^* \mid s \in K_1 \text{ and } s \notin K_2\}.$$

The *choice* between $K_1$ or $K_2$, denoted $K_1 + K_2$, is the language

$$K_1 + K_2 := \{s \in \Sigma^* \mid s \in K_1 \text{ or } s \in K_2\} = K_1 \cup K_2.$$

The *concatenation* of $K_1$ and $K_2$, denoted $K_1.K_2$ (or simply $K_1K_2$), is the language

$$K_1.K_2 := \{s.t \in \Sigma^* \mid s \in K_1 \text{ and } t \in K_2\}.$$

The *quotient* of $K_1$ with respect to $K_2$, denoted $K_1/K_2$, is the language

$$K_1/K_2 := \{s \in \Sigma^* \mid \exists t \in K_2 \text{ s.t. } st \in K_1\}.$$

The language $K_1$ *after* $K_2$, denoted $K_1 \backslash K_2$, is defined as

$$K_1 \backslash K_2 := \{s \in \Sigma^* \mid \exists t \in K_2 \text{ s.t. } ts \in K_1\}.$$

Some of the basic properties of the above binary operations is summarized in Table 1.1.

#### *Unary Operations*

Consider a language $K \subseteq \Sigma^*$. The *complement* of $K$, denoted $K^c \subseteq \Sigma^*$, is the language

$$K^c := \Sigma^* - K.$$

The *Kleene closure* of $K$, denoted $K^*$, is the language

$$K^* := \bigcup_{n \in \mathcal{N}} K^n,$$

| operation | notation | commutative | associative | identity | zero |
|---|---|---|---|---|---|
| intersection | $\cap$ | yes | yes | $\Sigma^*$ | $\emptyset$ |
| difference | $-$ | no | no | $\emptyset$ | none |
| choice | $+$ | yes | yes | $\emptyset$ | none |
| concatenation | . | no | yes | $\{\epsilon\}$ | $\emptyset$ |
| quotient | / | no | no | $\{\epsilon\}$ | $\emptyset$ |
| after | \ | no | no | $\{\epsilon\}$ | $\emptyset$ |

**Table 1.1** Properties of binary language operations

where $K^0 := \{\epsilon\}$ and for each $n \geq 0$, $K^{n+1} := K^n.K$ . The *prefix closure* of $K$, denoted $pr(K) \subseteq \Sigma^*$, is the language

$$pr(K) := \{s \in \Sigma^* \mid \exists t \in K : s \leq t\}.$$

The *extension closure* of $K$, denoted $ext(K) \subseteq \Sigma^*$, is the language

$$ext(K) := \{s \in \Sigma^* \mid \exists t \in K \mid t \leq s\}.$$

The *reverse* of $K$, denoted $K^R \subseteq \Sigma^*$, is the language

$$K^R := \{s^R \in \Sigma^* \mid s \in K\},$$

where $s^R$ denotes the string obtained by reversing the string $s$, i.e.,

$$\epsilon^R := \epsilon; \quad \forall s \in \Sigma^*, \sigma \in \Sigma : (s\sigma)^R := \sigma s^R.$$

The *projection* of $K$ on an event set $\hat{\Sigma} \subseteq \Sigma$, denoted $K\uparrow\hat{\Sigma} \subseteq \hat{\Sigma}^*$, is the language

$$K\uparrow\hat{\Sigma} := \{s\uparrow\hat{\Sigma} \mid s \in K\},$$

where $s\uparrow\hat{\Sigma}$ is inductively defined as:

$$\epsilon\uparrow\hat{\Sigma} := \epsilon; \quad \forall s \in \Sigma^*, \sigma \in \Sigma : (s\sigma)\uparrow\hat{\Sigma} := \begin{cases} (s\uparrow\hat{\Sigma})\sigma & \text{if } \sigma \in \hat{\Sigma} \\ s\uparrow\hat{\Sigma} & \text{otherwise.} \end{cases}$$

$K$ is said to be *Kleene closed* if $K^* = K$; it is said to be *prefix closed* if $pr(K) = K$; it is said to be *extension closed* if $ext(K) = K$. Some of the basic properties of the above unary operations is summarized in Table 1.2, where an operator $f(\cdot)$ defined on the set of languages is said to be *idempotent* if $f^2(\cdot) = f(\cdot)$; it is said to be *self-dual* if $f^2(\cdot) = (\cdot)$; and it is said to be *monotone* if whenever $K_1 \subseteq K_2 \subseteq \Sigma^*$, then $f(K_1) \subseteq f(K_2)$.

| operation | notation | idempotent | self-dual | monotone |
|---|---|---|---|---|
| complement | $(\cdot)^c$ | no | yes | no |
| Kleene closure | $(\cdot)^*$ | yes | no | yes |
| prefix closure | $pr(\cdot)$ | yes | no | yes |
| extension closure | $ext(\cdot)$ | yes | no | yes |
| reverse | $(\cdot)^R$ | no | yes | yes |
| projection | $(\cdot)\uparrow\hat{\Sigma}$ | yes | no | yes |

**Table 1.2** Properties of unary language operations

### 1.2.2 Language Models

Let $K \subseteq \Sigma^*, K \neq \emptyset$ be a language consisting of all traces that can occur in a given DES. For a sequence of events to occur in a DES, all of its prefixes must occur first; hence $K = pr(K)$. The language $K$ is called the *generated language* of such a DES. Let $K_m \subseteq K$ be the language consisting of those traces in the generated language of the DES whose execution implies completion of a certain task. The language $K_m$ is called the *marked language* of the DES. Thus a pair of languages $(K_m, K)$ such that $K_m \subseteq K = pr(K) \neq \emptyset$ models the behavior of a DES. Such a language pair is called a *language model* of a DES. Given two language models $(K_m, K)$ and $(H_m, H)$, they are said to be equal if $K_m = H_m$ and $K = H$.

**Example 1.2** Consider the buffer of Example 1.1. Let $a, d$ denote the arrival, departure events respectively. Then the generated language of the buffer is $pr((a.d)^*)$. Suppose a trace $s \in pr((a.d)^*)$ corresponds to completion of a task if and only if its execution results in the empty state of the buffer. Then the marked language of the buffer equals $(a.d)^*$. ■

## 1.3 STATE MACHINES

state machineIn this section we introduce the notion of a state machine which provides an alternative way of representing a language model. A state machine consists of a state set, a finite set of events, a state transition function which describes the state(s) that are reached when a certain event occurs in a particular state, an initial state, and a set of special states, called the set of marked

states. Such a state machine starts in its initial state and changes its state upon occurrence of events according to its state transition function. A string belongs to the *generated* language of such a state machine if it is executable from the initial state; it belongs to the *marked* language if the state reached upon its execution is a marked state. Next we define this precisely.

A state machine, denoted $G$, is a quintuple:

$$G := (X, \Sigma, \alpha, x_0, X_m),$$

where $X$ denotes the set of states of $G$, $\Sigma$ is the (finite) event set of $G$, $\alpha : X \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^X$ is the partial state transition function of $G$ (it is a partial function since it is generally defined on a subset of $X \times (\Sigma \cup \{\epsilon\})$), $x_0 \in X$ is the initial state of $G$, and $X_m \subseteq X$ denotes the set of marked or accepting states of $G$. A state transition on $\epsilon$ represents a *hidden* transition, also called an $\epsilon$-*move*. Note that the state transition function does not uniquely determine the resulting state. Hence $G$ is referred as a *non-deterministic state machine with* $\epsilon$-*moves* ($\epsilon$-NSM) . $G$ is simply said to be a NSM if its state transition function can be written as a partial map $\alpha : X \times \Sigma \rightarrow 2^X$, i.e., if there are no hidden transitions in $G$. $G$ is said to be a *deterministic state machine* (DSM) if its state transition function can be written as a partial map $\alpha : X \times \Sigma \rightarrow X$, i.e, if there are no hidden transitions, and the transition function uniquely determines the resulting state.

**Example 1.3** Consider the buffer of Examples 1.1 and 1.2 with language model $[(ad)^*, pr((ad)^*)]$. The directed graph shown in Figure 1.2 represents a DSM $G := (X, \Sigma, \alpha, x_0, X_m)$ for the buffer, where $X = \{\text{empty}, \text{full}\}$; $\Sigma = \{a, d\}$; $x_0 = \text{empty}$; $X_m = \{\text{empty}\}$; and $\alpha(\text{empty}, a) = \text{full}$, $\alpha(\text{full}, d) = \text{empty}$. Note that $\alpha(\text{empty}, d)$ and $\alpha(\text{full}, a)$ are not defined; hence the transition function is a partial map. (A node in the graph represents a state; a label on a node represents the name of the corresponding state; a directed edge represents a state transition; a label on a directed edge represents the name of the corresponding event; an arrow entering a node represents an initial state; and a circled node represents a marked state.) ■

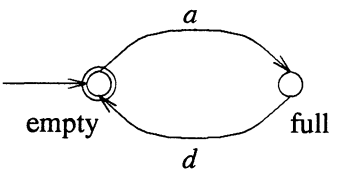



**Figure 1.2** Graph representing a DSM

Given a state machine $G$, the *epsilon-closure* of $x \in X$, denoted $\epsilon_G^*(x) \subseteq X$, is defined recursively as:

$$x \in \epsilon_G^*(x); \quad x' \in \epsilon_G^*(x) \Rightarrow \alpha(x', \epsilon) \subseteq \epsilon_G^*(x).$$

In other words, epsilon-closure of $x$ is the set of states reached from $x$ on zero or more $\epsilon$-moves. Note that if $G$ is a NSM (or a DSM), then $\epsilon_G^*(x) = \{x\}$.

Using the epsilon-closure map, the transition function of $G$ is extended from events to traces, denoted with a slight abuse of notation as $\alpha : X \times \Sigma^* \to 2^X$, which is inductively defined as $(x \in X, s \in \Sigma^*, \sigma \in \Sigma)$:

$$\begin{aligned} \alpha(x, \epsilon) &:= \epsilon_G^*(x) \\ \alpha(x, s\sigma) &:= \epsilon_G^*(\alpha(\alpha(x, s), \sigma)) = \left[ \bigcup_{x' \in \alpha(x,s)} \left\{ \bigcup_{x'' \in \alpha(x', \sigma)} \epsilon_G^*(x'') \right\} \right] \end{aligned}$$

**Example 1.4** Consider the $\epsilon$-NSM $G$ of Figure 1.3. Then $\epsilon_G^*(1) = \{1, 2, 3\}$,

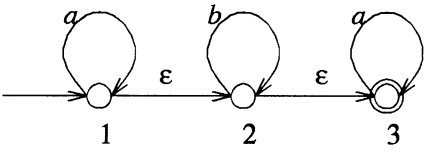



**Figure 1.3** Diagram illustrating an $\epsilon$-NSM

$\epsilon_G^*(2) = \{2, 3\}$, and $\epsilon_G^*(3) = \{3\}$.

So $\alpha(1, \epsilon) = \epsilon_G^*(1) = \{1, 2, 3\}$; $\alpha(1, a) = \epsilon_G^*(\alpha(\alpha(1, \epsilon), a)) = \epsilon_G^*(\alpha(\{1, 2, 3\}, a) = \epsilon_G^*(\{1, 3\}) = \{1, 2, 3\}$; $\alpha(1, ab) = \epsilon_G^*(\alpha(\alpha(1, a), b)) = \epsilon_G^*(\alpha(\{1, 2, 3\}, b) = \epsilon_G^*(\{2\}) = \{2, 3\}$, etc. ■

The extended transition function is used to define the generated and marked language of $G$, denoted $L(G)$ and $L_m(G)$ respectively, as:

$$L(G) := \{s \in \Sigma^* \mid \alpha(x_0, s) \neq \emptyset\}; \quad L_m(G) := \{s \in L(G) \mid \alpha(x_0, s) \cap X_m \neq \emptyset\}.$$

$L_m(G)$ is also called the language *accepted* by $G$. Clearly, $(L_m(G), L(G))$ is a language model, i.e., $L_m(G) \subseteq L(G) = pr(L(G)) \neq \emptyset$. Given two state machines $G_1$ and $G_2$, they are said to be *language equivalent* if $(L_m(G_1), L(G_1)) = (L_m(G_2), L(G_2))$.

**Remark 1.1** We have shown that the marked and generated language pair of a state machine is a language model. Conversely, given a language model $(K_m, K)$, there exists a state machine $G$ such that $(L_m(G), L(G)) = (K_m, K)$. Consider for example the state machine $G := (X, \Sigma, \alpha, x_0, X_m)$, where $X := \{s \in \Sigma^* \mid s \in K\}$, $x_0 := \epsilon$, $X_m := \{s \in \Sigma^* \mid s \in K_m\}$, and $\alpha : X \times \Sigma \to X$ is defined as:

$$\forall s \in X, \sigma \in \Sigma : \alpha(s, \sigma) := \begin{cases} s\sigma & \text{if } s\sigma \in K \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Clearly, $(L_m(G), L(G)) = (K_m, K)$.

Thus it follows that given a language model $(K_m, K)$, there exists a deterministic state machine $G$ such that $(L_m(G), L(G)) = (K_m, K)$. Hence there is no loss of generality in assuming that a given state machine is also deterministic. It also follows that given a state machine (possibly an $\epsilon$-NSM), there exists a language equivalent DSM for it. In the following subsection it is shown that if the state machine has finitely many states, then it is possible to construct a language equivalent DSM which *also* has finitely many states. ■

### 1.3.1 Operations on State Machines

We consider several operations on state machines that are useful in representation of complex DESs in terms of simpler ones, and development of many of the algorithms.

#### *Binary Operations*

Given SMs $G_1 := (X_1, \Sigma_1, \alpha_1, x_{0,1}, X_{m,1})$ and $G_2 := (X_2, \Sigma_2, \alpha_2, x_{0,2}, X_{m,2})$, *synchronous composition* of $G_1$ and $G_2$, denoted $G_1 \| G_2 := (X, \Sigma, \alpha, x_0, X_m)$, is defined as $X := X_1 \times X_2$; $\Sigma := \Sigma_1 \cup \Sigma_2$; $x_0 := (x_{0,1}, x_{0,2})$; $X_m := X_{m,1} \times X_{m,2}$; and for each $x = (x_1, x_2) \in X, \sigma \in \Sigma$:

$$\alpha(x, \sigma) := \begin{cases} (\alpha_1(x_1, \sigma), \alpha_2(x_2, \sigma)) & \text{if } \alpha_1(x_1, \sigma), \alpha_2(x_2, \sigma) \text{ defined}, \sigma \in \Sigma_1 \cap \Sigma_2 \\ (\alpha_1(x_1, \sigma), x_2) & \text{if } \alpha(x_1, \sigma) \text{ defined}, \sigma \in \Sigma_1 - \Sigma_2 \\ (x_1, \alpha_2(x_2, \sigma)) & \text{if } \alpha(x_2, \sigma) \text{ defined}, \sigma \in \Sigma_2 - \Sigma_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

Thus if an event belongs to the common event set $\Sigma_1 \cap \Sigma_2$, then it occurs synchronously in the two systems; otherwise, it occurs asynchronously. It is readily seen that

$$L_m(G_1 \| G_2) \quad = \quad \{s \in \Sigma^* \mid s \uparrow \Sigma_1 \in L_m(G_1) \text{ and } s \uparrow \Sigma_2 \in L_m(G_2)\},$$

*Note 1.1.* $d_1$ 是共享事件，因此，(idle,empty) 状态下，$d_1$ 不能发生，仅发生 $a_1$ 事件; (working,empty) 状态下，$d_1$ 在 M 和 B 中的转移函数均有定义，因此该共享事件可以发生该共享事件。非共享事件 $(a_1, d_2)$ 在 M 或 B 中的转移函数有一个有定义，即可发生。

$$L(G_1 \| G_2) \quad = \quad \{s \in \Sigma^* \mid s\uparrow\Sigma_1 \in L(G_1) \text{ and } s\uparrow\Sigma_2 \in L(G_2)\}.$$

Clearly, if $\Sigma_1 = \Sigma_2$, then $L_m(G_1 \| G_2) = L_m(G_1) \cap L_m(G_2)$ and $L(G_1 \| G_2) = L(G_1) \cap L(G_2)$.

**Example 1.5** Consider for example a manufacturing production line consisting of a machine (M) and a buffer (B) of capacity one operating in synchrony as shown in Figure 1.4. The event set $\Sigma_1$ of M consists of events $a_1$ representing arrival into the machine, and $d_1$ representing departure from the machine; whereas the event set $\Sigma_2$ of B consists of events $d_1$ representing departure from the machine, and $d_2$ representing departure from buffer. The synchronous composition of the two systems is also shown in Figure 1.4. ■

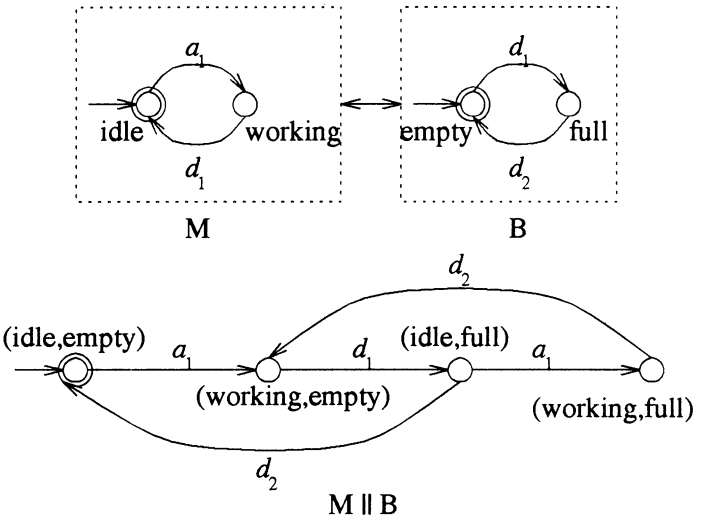



**Figure 1.4** Diagram illustrating synchronous composition of DFSMs

## *Unary Operations*

Let $G := (X, \Sigma, \alpha, x_0, X_m)$ be a DSM. The *complementation* of $G$ is defined to be the state machine $G^c := (X, \Sigma, \alpha, x_0, X - X_m)$, which is obtained by marking the unmarked states of $G$ and unmarking the marked states of $G$. It is easy to see that $G^c$ is a DSM, $L_m(G^c) = L(G) - L_m(G)$ and $L(G^c) = L(G)$. The *completion* of $G$, denoted $\overline{G} := (\overline{X}, \Sigma, \overline{\alpha}, x_0, X_m)$, is the state machine

G的补集：complementation of G;　G的全集：completion of G

obtained by "completing" the state transition function of $G$ so that whenever a transition on a certain event in a certain state is not defined in $G$, then its execution leads to a "dump" state in $\overline{G}$. Formally, $\overline{X} := X \cup \{x_D\}$, with $x_D \notin X$ ($x_D$ denotes the dump state), and

$$\forall \overline{x} \in \overline{X}, \sigma \in \Sigma : \overline{\alpha}(\overline{x}, \sigma) := \begin{cases} \alpha(\overline{x}, \sigma) & \text{if } \overline{x} \in X, \alpha(\overline{x}, \sigma) \text{ defined} \\ x_D & \text{otherwise} \end{cases}$$

It is easily seen that $\overline{G}$ is a DSM, $L_m(\overline{G}) = L_m(G)$ and $L(\overline{G}) = \Sigma^*$. $G$ is said to be *complete* if $\overline{G} = G$. Finally, the *reverse* of $G$ is defined to be the state machine $G^R := (X \cup \{x_0^R\}, \Sigma, \alpha^R, x_0^R, \{x_0\})$, where $x_0^R \notin X$ is the initial state of $G^R$, and the transition function $\alpha^R : (X \cup \{x_0^R\}) \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^{X \cup \{x_0^R\}}$ is defined to be ($x \in X \cup \{x_0^R\}, \sigma \in \Sigma \cup \{\epsilon\}$):

$$\alpha^R(x, \sigma) := \begin{cases} \{x' \in X \mid \alpha(x', \sigma) = x\} & \text{if } x \in X, \sigma \in \Sigma \\ X_m & \text{if } x = x_0^R, \sigma = \epsilon \\ \emptyset & \text{otherwise} \end{cases}$$

Then $G^R$ is an $\epsilon$-NSM with $L_m(G^R) = (L_m(G))^R$.

**Example 1.6** Completion and reverse of the DSM of Figure 1.6(b) are shown in Figure 1.5(a) and 1.5(b) respectively. The state labels have been changed

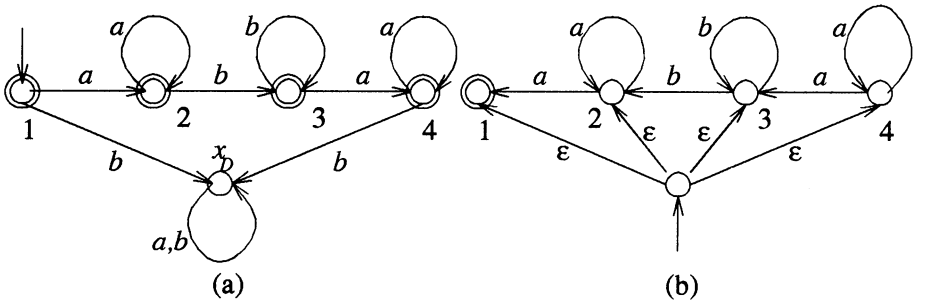



**Figure 1.5** Diagram illustrating completion and reverse operations

for future use. ■

## 1.3.2 Finite State Machines

A state machine $G := (X, \Sigma, \alpha, x_0, X_m)$ is said to be a finite state machine (FSM) if $X$ is a finite set. In general a finite state machine is an $\epsilon$-NSM; so it

is called an $\epsilon$-NFSM. NFSM (non-deterministic finite state machine without $\epsilon$-moves) and DFSM (deterministic finite state machine) are defined in an obvious way. In this subsection we show that given an $\epsilon$-NFSM, there exists a language equivalent NFSM; and given a NFSM, there exists a language equivalent DFSM.

**Theorem 1.1** Let $G := (X, \Sigma, \alpha, x_0, X_m)$ be an $\epsilon$-NFSM. Then there exists a language model equivalent NFSM $G' := (X, \Sigma, \alpha', x_0, X'_m)$, i.e., $L_m(G') = L_m(G)$ and $L(G') = L(G)$.

**Proof:** Define the transition function of $G'$ as follows:

$$\forall x \in X, \sigma \in \Sigma : \alpha'(x, \sigma) := \epsilon^*_G(\alpha(\epsilon^*_G(x), \sigma)) = \left[ \bigcup_{x' \in \epsilon^*_G(x)} \left\{ \bigcup_{x'' \in \alpha(x', \sigma)} \epsilon^*_G(x'') \right\} \right];$$

and the set of marked states of $G'$ is defined as:

$$X'_m := \begin{cases} X_m \cup \{x_0\} & \text{if } X_m \cap \epsilon^*_G(x_0) \neq \emptyset \\ X_m & \text{otherwise.} \end{cases}$$

Then it is easily shown that $(L_m(G'), L(G')) = (L_m(G), L(G))$. ■

**Example 1.7** Consider the $\epsilon$-NFSM $G$ shown in Figure 1.3. Then a language equivalent NFSM $G'$ obtained using the construction of outlined in Theorem 1.1

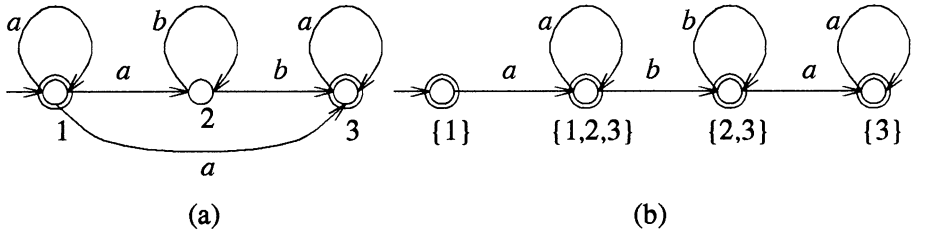



**Figure 1.6** Diagram illustrating $\epsilon$-NFSM to NFSM to DFSM conversion

is shown in Figure 1.6(a). Note that the epsilon-closure of the initial state includes a marked state in $G$, i.e., $\epsilon^*_G(1) = \{1, 2, 3\} \supset \{3\} = X_m$. Hence $X'_m = X_m \cup \{x_0\} = \{1, 3\}$. Also, $\alpha'(1, a) = \epsilon^*_G(\alpha(\epsilon^*_G(1), a)) = \epsilon^*_G(\alpha(\{1, 2, 3\}, a)) = \epsilon^*_G(\{1, 3\}) = \{1, 2, 3\}$; $\alpha'(2, b) = \epsilon^*_G(\alpha(\epsilon^*_G(2), b)) = \epsilon^*_G(\alpha(\{2, 3\}, b)) = \epsilon^*_G(\{2\}) = \{2, 3\}$; and $\alpha'(3, a) = \epsilon^*_G(\alpha(\epsilon^*_G(3), a)) = \epsilon^*_G(\alpha(\{3\}, a)) = \epsilon^*_G(\{3\}) = \{3\}$. ■

**Theorem 1.2** Let $G := (X, \Sigma, \alpha, x_0, X_m)$ be a NFSM. Then there exists a language model equivalent DFSM $\mathcal{G} := (\mathcal{X}, \Sigma, \hat{\alpha}, \{x_0\}, \mathcal{X}_m)$, i.e., $L_m(\mathcal{G}) = L_m(G)$ and $L(\mathcal{G}) = L(G)$.

**Proof:** Define $\mathcal{X} := 2^X$, $\mathcal{X}_m := \{\hat{X} \in \mathcal{X} \mid \hat{X} \cap X_m \neq \emptyset\}$, and

$$\forall \hat{X} \in \mathcal{X}, \sigma \in \Sigma : \hat{\alpha}(\hat{X}, \sigma) := \bigcup_{x \in \hat{X}} \alpha(x, \sigma).$$

Then it is easily shown that $(L_m(\mathcal{G}), L(\mathcal{G})) = (L_m(G), L(G))$. ■

Note that when an $\epsilon$-NFSM is converted to a language equivalent NFSM using the construction of Theorem 1.1, then the number of states remains unchanged. However, the number of states in the corresponding language equivalent DFSM obtained using the construction of Theorem 1.2 may equal $2^m$, where $m$ is the number of states in the given NFSM, as the state set of the DFSM thus obtained is the power set of the state set of the given NFSM. For this reason, the construction of Theorem 1.2 is also referred as the *power set construction.*

**Example 1.8** Consider the NFSM $G := (X, \Sigma, \alpha, x_0, X_m)$ shown in Figure 1.6(a). The language equivalent DFSM $\mathcal{G} := (\mathcal{X}, \Sigma, \hat{\alpha}, \{x_0\}, \mathcal{X}_m)$ obtained using the power set construction is shown in Figure 1.6(b). Note that $\mathcal{X} = \mathcal{X}_m = \{\{1\}, \{1,2,3\}, \{2,3\}, \{3\}\}$, as $X_m = \{1,3\}$, which has a nonempty intersection with each state in $\mathcal{X}$; $\hat{\alpha}(\{1\}, a) = \alpha(1, a) = \{1,2,3\}$; $\hat{\alpha}(\{1,2,3\}, a) = \alpha(1, a) \cup \alpha(2, a) \cup \alpha(3, a) = \{1,2,3\}$; $\hat{\alpha}(\{1,2,3\}, b) = \alpha(1, b) \cup \alpha(2, b) \cup \alpha(3, b) = \{2,3\}$; etc. ■

**Remark 1.2** It follows from Theorems 1.1 and 1.2 that if a language model $(K_m, K)$ can be represented as a finite state machine $G$, then there also exists a DFSM $G'$ such that $(L_m(G'), L(G')) = (K_m, K)$. Thus if we are only concerned with DESs that have finitely many states, then we can assume without loss of generality that they can be represented as DFSMs. We will see below that although the finiteness of states is not needed for most of the analysis, it is needed for developing all the decision algorithms.

However, it should be noted that although DFSMs are useful in developing decision algorithms, it is conceptually easier to obtain a NFSM from the given description of a language. For example, suppose $\Sigma = \{a, b\}$, and suppose we wish to represent the language with the property that every string in it must contain *aba* as a substring. A NFSM for the same is shown in Figure 1.7(a);

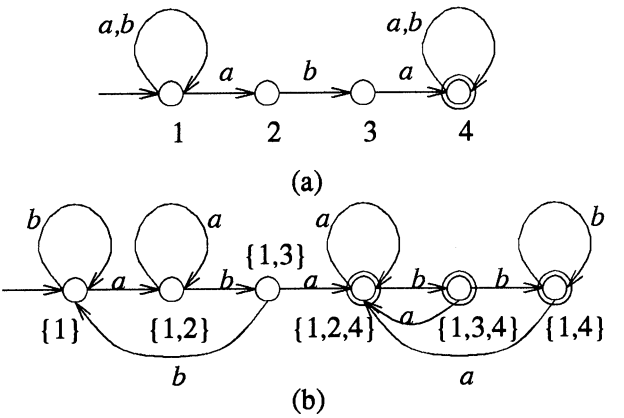



**Figure 1.7** NFSM accepting strings with $aba$ as a substring, and corresponding DFSM

corresponding DFSM obtained using the construction outlined in Theorem 1.2 is shown in Figure 1.7(b). ■

## 1.4 REGULAR LANGUAGES

regular languages Since $\Sigma^*$ is an infinite set, in general, a language is also an infinite set. We need to identify that class of languages which can be represented using finite state machines. This will allow us to develop desired algorithms for the languages of interest using their finite state machine representations. The next theorem due to Kleene establishes that a language admits a finite state machine representation if and only if it is *regular*.

**Definition 1.1** The *class of regular languages*, denoted $\mathcal{L}_R \subseteq 2^{\Sigma^*}$ is defined as:

- $\emptyset, \{\epsilon\} \in \mathcal{L}_R$
- $\forall \sigma \in \Sigma : \{\sigma\} \in \mathcal{L}_R$
- $K, K_1, K_2 \in \mathcal{L}_R \Rightarrow K_1 + K_2, K_1.K_2, K^* \in \mathcal{L}_R$

A language $K \subseteq \Sigma^*$ is said to be *regular* if $K \in \mathcal{L}_R$.

**Remark 1.3** For notational convenience we use the syntax of a *regular expression* for writing a regular language. $\emptyset, \epsilon$, and $\sigma$ (where $\sigma \in \Sigma$) are regular expressions; if $r_1$ and $r_2$ are regular expressions, then so are $r_1 + r_2$ and $r_1.r_2$; if $r$ is a regular expression, then so is $r^*$. Thus if $\Sigma = \{a, b\}$, then $(a+b)^*, a^*.b.a$, etc. are regular expressions, and $+1, 1.+3$ are not regular expressions. In order to avoid writing parenthesis, we associate highest binding power to the Kleene closure operation, and the lowest binding power to the choice operation.

Letting $L(r)$ denote the regular language represented by the regular expression $r$, it is defined inductively on number of operators in the regular expression as: $L(\emptyset) := \emptyset$; $L(\epsilon) := \{\epsilon\}$; for each $\sigma \in \Sigma$, $L(\sigma) := \{\sigma\}$; $L(r_1 + r_2) := L(r_1) + L(r_2)$; $L(r_1.r_2) := L(r_1).L(r_2)$; and $L(r^*) := L(r)^*$. Thus the regular expression $a + b^*$ represents the regular language $\{a\} + \{b\}^*$. ■

**Theorem 1.3** Given a DFSM $G := (X, \Sigma, \alpha, x_0, X_m)$, there exists a regular expression $r$ such that $L(r) = L_m(G)$. Conversely, given a regular expression $r$, there exists a DFSM $G := (X, \Sigma, \alpha, x_0, X_m)$ such that $L_m(G) = L(r)$.

**Proof:** First suppose a DFSM $G$ is given. Let $m$ be the number of states in $G$. We label the states of $G$ with numbers 1 through $m$. Without loss of generality, let the label of state $x_0$ be 1. For each $i, j \leq m$ inductively define the regular expression $r_{ij}^m$ as follows:

$$r_{ij}^0 = \begin{cases} +_{\{\sigma \in \Sigma | \alpha(i,\sigma) = j\}} \sigma & \text{if } i \neq j \\ (+_{\{\sigma \in \Sigma | \alpha(i,\sigma) = j\}} \sigma) + \epsilon & \text{otherwise} \end{cases}$$

$$r_{ij}^k = r_{ik}^{k-1} (r_{kk}^{k-1})^* r_{kj}^{k-1} + r_{ij}^{k-1}, \quad \forall k \leq m$$

Then it is easily verified that $L(r_{ij}^k) \subseteq \Sigma^*$ is the regular language consisting of strings which start from the state with label $i$, end at the state with label $j$, and only visit states with labels no larger than $k$, i.e.,

$$L(r_{ij}^k) = \{s \in \Sigma^* \mid \alpha(i, s) = j, \text{ and } \forall t < s, t \neq \epsilon : \alpha(i, t) \leq k\}.$$

Hence $L_m(G) = L(+_{j \in X_m} r_{1j}^m)$. Thus $r := +_{j \in X_m} r_{ij}^m$ is the required regular expression.

Next suppose a regular expression $r$ is given. In view of Theorems 1.1 and 1.2 it suffices to show that there exists an $\epsilon$-NFSM $G$ such that $L_m(G) = L(r)$. We prove this using induction on number of operators in the regular expression of $r$. If the number or operators in the $r$ equals zero, then either $r = \emptyset$, or $r = \epsilon$, or $r = \sigma_0$ for some $\sigma_0 \in \Sigma$. If $r = \emptyset$, then construct

$G := \{\{x_0, x_m\}, \Sigma, \alpha, x_0, \{x_m\})$ such that for each $\sigma \in \Sigma$, $\alpha(x_0, \sigma)$ as well as $\alpha(x_m, \sigma)$ is undefined. Then $L_m(G) = \emptyset = L(r)$. If $r = \epsilon$, then construct $G := (\{\{x_0\}, \Sigma, \alpha, x_0, \{x_0\})$ with for each $\sigma \in \Sigma$, $\alpha(x_0, \sigma)$ is undefined. Then $L_m(G) = \{\epsilon\} = L(r)$. If $r = \sigma_0$ for some $\sigma_0 \in \Sigma$, then construct $G := (\{x_0, x_m\}, \Sigma, \alpha, x_0, \{x_m\})$, with

$$\forall x \in \{x_0, x_m\}, \sigma \in \Sigma : \alpha(x, \sigma) := \begin{cases} x_m & \text{if } x = x_0, \sigma = \sigma_0 \\ \emptyset & \text{otherwise.} \end{cases}$$

Then $L_m(G) = \{\sigma_0\} = L(r)$. This completes the proof of the base step. The above constructions are illustrated in Figure 1.8.

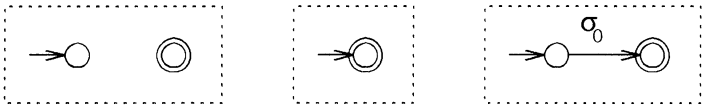



**Figure 1.8** Diagram illustrating DFSMs marking languages $\emptyset, \{\epsilon\}, \{\sigma_0\}$ respectively

In order to prove the induction step, first suppose $r = r_1 + r_2$, where $r_1$ and $r_2$ are regular expressions, then from induction hypothesis there exist $\epsilon$-NFSMs $G_1 := (X_1, \Sigma, \alpha_1, x_{0,1}, X_{m,1})$ and $G_2 := (X_2, \Sigma, \alpha_2, x_{0,2}, X_{m,2})$ such that $L_m(G_1) = L(r_1)$ and $L_m(G_2) = L(r_2)$. Define $G := (X, \Sigma, \alpha, x_0, X_m)$ such that $X := X_1 \cup X_2 \cup \{x_0\}$, $X_m := X_{m,1} \cup X_{m,2}$ and

$$\forall x \in X, \sigma \in \Sigma \cup \{\epsilon\} : \alpha(x, \sigma) := \begin{cases} \{x_{0,1}, x_{0,2}\} & \text{if } x = x_0, \sigma = \epsilon \\ \emptyset & \text{if } x = x_0, \sigma \neq \epsilon \\ \alpha_1(x, \sigma) & \text{if } x \in X_1 \\ \alpha_2(x, \sigma) & \text{if } x \in X_2 \end{cases}$$

Then $G$ is an $\epsilon$-NFSM with $L_m(G) = L_m(G_1) + L_m(G_2) = L(r_1) + L(r_2) = L(r)$.

Next suppose $r = r_1.r_2$, where $r_1$ and $r_2$ are regular expressions, then from induction hypothesis there exist $\epsilon$-NFSMs $G_1 := (X_1, \Sigma, \alpha_1, x_{0,1}, X_{m,1})$ and $G_2 := (X_2, \Sigma, \alpha_2, x_{0,2}, X_{m,2})$ such that $L_m(G_1) = L(r_1)$ and $L_m(G_2) = L(r_2)$. Define $G := (X, \Sigma, \alpha, x_{0,1}, X_{m,2})$ such that $X := X_1 \cup X_2$ and

$$\forall x \in X, \sigma \in \Sigma \cup \{\epsilon\} : \alpha(x, \sigma) := \begin{cases} \alpha_1(x, \sigma) \cup \{x_{0,2}\} & \text{if } x \in X_{m,1}, \sigma = \epsilon \\ \alpha_1(x, \sigma) & \text{if } x \in X_{m,1}, \sigma \neq \epsilon \\ \alpha_1(x, \sigma) & \text{if } x \in X_1 - X_{m,1} \\ \alpha_2(x, \sigma) & \text{if } x \in X_2 \end{cases}$$

Then $G$ is an $\epsilon$-NFSM with $L_m(G) = L_m(G_1).L_m(G_2) = L(r_1).L(r_2) = L(r)$.

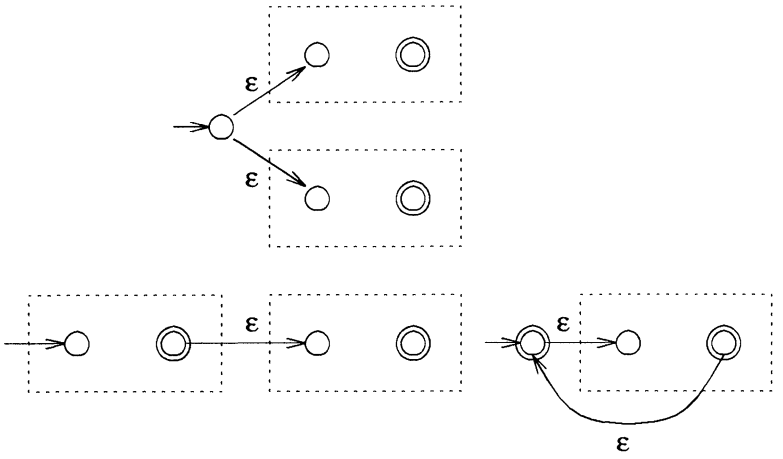



**Figure 1.9** Diagram illustrating $\epsilon$-NFSMs marking languages $L(r_1 + r_2)$, $L(r_1.r_2), L(r_1^*)$, respectively

Finally, suppose $K = r_1^*$, where $r_1$ is a regular expression, then from induction hypothesis there exists an $\epsilon$-NFSM $G_1 := (X_1, \Sigma, \alpha_1, x_{0,1}, X_{m,1})$ such that $L_m(G_1) = L(r_1)$. Define $G := (X, \Sigma, \alpha, x_0, X_m)$ such that $X := X_1 \cup \{x_0\}, X_m := X_{m,1} \cup \{x_0\}$ and

$$\forall x \in X, \sigma \in \Sigma \cup \{\epsilon\} : \alpha(x,\sigma) := \begin{cases} \alpha_1(x,\sigma) \cup \{x_0\} & \text{if } x \in X_{m,1}, \sigma = \epsilon \\ \alpha_1(x,\sigma) & \text{if } x \in X_{m,1}, \sigma \neq \epsilon \\ \alpha_1(x,\sigma) & \text{if } x \in X_1 - X_{m,1} \\ x_{0,1} & \text{if } x = x_0, \sigma = \epsilon \\ \emptyset & \text{if } x = x_0, \sigma \neq \epsilon \end{cases}$$

Then $G$ is an $\epsilon$-NFSM with $L_m(G) = (L_m(G_1))^* = L(r_1)^* = L(r)$. Figure 1.9 depicts the above constructions. ■

**Example 1.9** We now illustrate the inductive computation of a regular expression $r$ for a given DFSM $G$ so that $L(r) = L_m(G)$. Consider the DFSM of Figure 1.2 which represents a buffer of capacity one. It has two states—empty and full. Let the empty state be labeled 1, and the full state be labeled 2. Then the marked language of the buffer DFSM equals $L(r_{11}^2)$, where $r_{11}^2$ is computed inductively as follows:

$$\begin{aligned} r_{11}^0 &= \epsilon \\ r_{12}^0 &= a \end{aligned}$$

$$
\begin{aligned}
r_{21}^0 &= d \\
r_{22}^0 &= \epsilon \\
r_{11}^1 &= r_{11}^0 (r_{11}^0)^* r_{11}^0 + r_{11}^0 = \epsilon(\epsilon)^*\epsilon + \epsilon = \epsilon \\
r_{12}^1 &= r_{11}^0 (r_{11}^0)^* r_{12}^0 + r_{12}^0 = \epsilon(\epsilon)^* a + a = a \\
r_{21}^1 &= r_{21}^0 (r_{11}^0)^* r_{11}^0 + r_{21}^0 = d(\epsilon)^*\epsilon + d = d \\
r_{22}^1 &= r_{21}^0 (r_{11}^0)^* r_{12}^0 + r_{22}^0 = d(\epsilon)^* a + \epsilon = da + \epsilon \\
r_{11}^2 &= r_{12}^1 (r_{22}^1)^* r_{21}^1 + r_{11}^1 = a(da + \epsilon)^* d + \epsilon = a(da)^* d + \epsilon = (ad)^*
\end{aligned}
$$

■

The next corollary follows from Theorem 1.3. For a deterministic state machine $G$, we use

$$Re_G(x) := \{x' \in X \mid \exists s \in \Sigma^* \text{ s.t } \alpha(x, s) = x'\}$$

to denote the set of states *reachable* from a state $x \in X$ in $G$. $G$ is said to be *accessible* if $Re_G(x_0) = X$, i.e., if all states in $G$ are reachable from the initial state. The state machine obtained by removing from $G$ all states that are not reachable from the initial state is language equivalent to $G$. Hence it can be assumed without loss of generality that a given DFSM is accessible. $G$ is said to be *co-accessible* if for each $x \in X$, $Re_G(x) \cap X_m \neq \emptyset$, i.e., at least one marked state is reachable from each state of $G$. Thus if $G$ is accessible, then it is co-accessible if and only if $pr(L_m(G)) = L(G)$. (Refer to exercise problem 6.) Finally, $G$ is said to be *trim* if it is accessible as well as co-accessible.

**Corollary 1.1** Given a language model $(K_m, K)$, there exists a DFSM $G := (X, \Sigma, \alpha, x_0, X_m)$ such that $(L_m(G), L(G)) = (K_m, K)$ if and only if $K_m$ and $K$ are both regular.

**Proof:** Consider first the necessity part. It follows from the necessity part of Theorem 1.3 that $K_m = L_m(G)$ is regular. Moreover, if we redefine $X_m := X$, then $L_m(G) = L(G) = K$. Thus the necessity part of Theorem 1.3 implies that $K$ is also regular.

Next we consider the sufficiency part. It follows from the sufficiency part of Theorem 1.3 that there exist DFSMs $G_1 := (X_1, \Sigma, \alpha_1, x_{0,1}, X_{m,1})$ and $G_2 := (X_2, \Sigma, \alpha_2, x_{0,2}, X_{m,2})$ such that $L_m(G_1) = K_m$ and $L_m(G_2) = K$. Without loss of generality we can assume that $G_1$ and $G_2$ are trim. Then since $K$ is prefix closed, $L(G_2) = pr(L_m(G_2)) = pr(K) = K$. Define $G := \overline{G_1} \| G_2$. Then

$$L_m(G) = L_m(\overline{G_1}) \cap L_m(G_2) = L_m(G_1) \cap K = K_m \cap K = K_m,$$

$$L(G) = L(\overline{G_1}) \cap L(G_2) = \Sigma^* \cap K = K,$$

as desired. ■

### 1.4.1 Properties of Regular Languages

In this subsection, we describe some of the useful properties of regular languages.

#### *Closure Properties*

It follows from the definition of regular languages, that they are closed under choice, concatenation and Kleene closure. The following proposition states that they are also closed under boolean operations.

**Theorem 1.4** Suppose $K_1, K_2 \subseteq \Sigma^*$ are regular. Then $K_1 \cap K_2$ and $K_1^c$ are also regular.

**Proof:** Let $G_1 := (X_1, \Sigma, \alpha_1, x_{0,1}, X_{m,1})$ and $G_2 := (X_2, \Sigma, \alpha_2, x_{0,2}, X_{m,2})$ be DFSMs such that $L_m(G_1) = K_1$ and $L_m(G_2) = K_2$. That such state machines exist follows from Theorem 1.3. Consider $G_1||G_2$; then $L_m(G_1||G_2) = L_m(G_1) \cap L_m(G_2) = K_1 \cap K_2$. Since $G_1||G_2$ is a DFSM, it follows from Theorem 1.3 that $K_1 \cap K_2$ is regular.

Next consider $(\overline{G_1})^c$; then $L_m((\overline{G_1})^c) = L(\overline{G_1}) - L_m(\overline{G_1}) = \Sigma^* - L_m(G_1) = \Sigma^* - K_1 = K_1^c$. Since $(\overline{G})^c$ is a DFSM, it follows from Theorem 1.3 that $K_1^c$ is regular. ■

#### *Pumping Lemma*

We have seen different ways of establishing that a given language is regular. For example, we can show that it is written as a regular expression, or we can show that there exists a DFSM marking it. The following lemma, called *pumping lemma*, is useful in establishing that a given language is not regular.

**Lemma 1.1** Suppose $K \subseteq \Sigma^*$ is regular. Then there exists $m \in \mathcal{N}$ such that for each $s \in K$ with $|s| \geq m$, there exist $u, v, w \in \Sigma^*$ such that $s = uvw$,

$|uv| \leq m$, $|v| \geq 1$, and for each $i \geq 0$, $uv^i w \in K$. Furthermore, $m$ can be chosen to be smaller than the number of states in any DFSM marking $K$.

**Proof**: From Theorem 1.3 that there exists a DFSM $G := (X, \Sigma, \alpha, x_0, X_m)$ with $L_m(G) = K$. Set $m$ to be the number of states in $G$. Pick a string $s \in K$ with $|s| \geq m$. Let $t \leq s$ be a prefix of $s$ with $|t| = m$. Since the number of states in $G$ is $m$, and $|t| = m$, there exists at least one state of $G$ which is visited at least twice when $t$ is executed. Let $x \in X$ be the first such state. Define $u \leq t$ such that $\alpha(x_0, u) = x$, and $v \in \Sigma^*$ such that $uv \leq t$ and $\alpha(x, v) = x$. Finally define $w \in \Sigma^*$ such that $uvw = s$. Then assertions of Lemma 1.1 can be easily verified. ■

**Example 1.10** As an application of the pumping lemma (Lemma 1.1), consider the language $K := \{a^i b^i \mid i \geq 0\}$ defined over the event set $\{a, b\}$. We show by way of contradiction that $K$ is not regular. Suppose for contradiction that $K$ is regular so the result of pumping lemma can be applied. Pick $s = a^m b^m \in K$, where $m \in \mathcal{N}$ is as in pumping lemma. Clearly, $|s| = 2m \geq m$. Hence it follows from the pumping lemma that there exist $u, v, w \in \Sigma^*$ such that $uvw = s = a^m b^m$, $|uv| \leq m$, $|v| \geq 1$, and for each $i \geq 0$, $uv^i w \in K$. Since $|uv| \leq m$, it follows that $uv \leq a^m$, which implies that $u = a^j$ and $v = a^k$ with $j + k \leq m$. Then $w = a^{m-(j+k)} b^m$. Choose $i = 2m$. Then we conclude that $uv^m w = a^j (a^k)^{2m} a^{m-(j+k)} b^m = a^{m+2mk-k} b^m \in K$, which is a contradiction. ■

## *Myhill-Nerode Characterization*

Next we present another characterization of regular languages. We need to introduce the definition of equivalence relations on $\Sigma^*$ induced by a language, and by a DFSM.

**Definition 1.2** Given a language $K \subseteq \Sigma^*$, it induces an equivalence relation, denoted $R_K$, on $\Sigma^*$:

$$\forall s, t \in \Sigma^* : s \cong t (R_K) \Leftrightarrow [K \backslash \{s\} = K \backslash \{t\}].$$

For each $s \in \Sigma^*$, $[s](R_K) \subseteq \Sigma^*$ is used to denote the equivalence class containing the string $s$.

**Definition 1.3** Given a DFSM $G := (X, \Sigma, \alpha, x_0, X_m)$, it induces an equivalence relation, denoted $R_G$, on $\Sigma^*$:

$$\forall s, t \in \Sigma^* : s \cong t (R_G) \Leftrightarrow [\alpha(x_0, s) = \alpha(x_0, t)] \vee [\alpha(x_0, s), \alpha(x_0, t) \text{ undefined}].$$

For each $s \in \Sigma^*$, $[s](R_G) \subseteq \Sigma^*$ is used to denote the equivalence class containing the string $s$.

Note that $|R_G|$, the *index* of $R_G$, i.e., the number of equivalence classes of $R_G$, is one more than the number of states in $G$. (The set of all strings that do not belong to $L(G)$ belong to a single equivalence class of $R_G$.) It can be easily seen that the equivalence relation $R_G$ *refines* the equivalence relations $R_{L_m(G)}$ and $R_{L(G)}$. In other words,

$$\forall s, t \in \Sigma^* : s \cong t(R_G) \Rightarrow [s \cong t(R_{L_m(G)})] \wedge [s \cong t(R_{L(G)})].$$

An equivalence relation $R$ on $\Sigma^*$ is said to be *right invariant (with respect to concatenation)* if

$$\forall s, t, u \in \Sigma^* : s \cong t(R) \Rightarrow su \cong tu(R).$$

It is easy to verify that $R_K$ as well as $R_G$ defined above are right invariant. The following proposition is due to Myhill and Nerode:

**Theorem 1.5** Let $K \subseteq \Sigma^*$ be a language. Then the following are equivalent:

1. $K$ is regular.
2. $K$ can be written as union of some of the equivalence classes of a right invariant equivalence relation of finite index.
3. $R_K$ is of finite index.

**Proof:** Suppose $K$ is regular. Then there exists a DFSM $G$ such that $L_m(G) = K$. Then clearly $K$ can be written as union of the following equivalence classes of $R_G$:

$$\{[s](R_G) \mid s \in K\}.$$

This proves the first assertion implies the second assertion, as $R_G$ is right invariant.

Let $R$ be a right invariant equivalence relation of finite index such that $K$ can be written as union of some of the equivalence classes of $R$. In order to show that $R_K$ is of finite index it suffices to show that $R$ refines $R_K$. Pick $s, t \in \Sigma^*$ such that $s \cong t(R)$. Since $R$ is right invariant, for any $u \in \Sigma^*$, $su \cong tu(R)$. Since

$K$ equals union of some of the equivalence classes of $R$, this implies $su \in K$ if and only if $tu \in K$. In other words, $s \cong t(R_K)$, which proves that the second assertion implies the third assertion.

Finally, suppose $R_K$ is of finite index. Define a DFSM $\hat{G} := (\hat{X}, \Sigma, \hat{\alpha}, \hat{x}_0, \hat{X}_m)$ as follows: $\hat{X} := \{[s](R_K) \mid s \in \Sigma^*\}$; $x_0 := [\epsilon](R_K)$; $\hat{X}_m := \{[s](R_K) \mid s \in K\}$; and

$$\forall\, [s](R_K) \in \hat{X}, \sigma \in \Sigma : \hat{\alpha}([s](R_K), \sigma) := [s\sigma](R_K).$$

Then it is readily verified that for each $s \in \Sigma^*$, $\alpha(\hat{x}_0, s) = [s](R_k)$. Hence from definition of marked language we obtain that $s \in L_m(\hat{G})$ if and only if $\hat{\alpha}(\hat{x}_0, s) = [s](R_K) \in \hat{X}_m$, i.e., if and only if $s \in K$. Thus $L_m(\hat{G}) = K$. Since $\hat{G}$ is a DFSM (as $R_k$ is of finite index), this implies that $K$ is regular; so the third assertion implies the first assertion. ■

The construction of the DFSM $\hat{G}$ in the proof of Theorem 1.5 is known as the Myhill-Nerode construction. The following example illustrates such a construction.

**Example 1.11** Consider for example the marked language $K_m = (ad)^*$ of the buffer of capacity one of Example 1.2. Clearly, $K_m$ is a regular language. Hence it follows from Theorem 1.5 that $R_{K_m}$ is of finite index. It can be easily verified that $[\epsilon](R_{K_m}) = (ad)^* = K_m$, $[a](R_{K_m}) = (ad)^*a = pr(K_m) - K_m$, $[d](R_{K_m}) = \{a, d\}^* - pr(K_m)$; and these are the only equivalence classes of $R_{K_m}$. Hence Myhill-Nerode construction yields the DFSM $\hat{G} := (\hat{X}, \Sigma, \hat{\alpha}, \hat{x}_0, \hat{X}_m)$, where $\hat{X} = \{[\epsilon](R_{K_m}), [a](R_{K_m}), [d](R_{K_m})\}$; $\hat{x}_0 = [\epsilon](R_{K_m})$; $X_m = \{[\epsilon](R_{K_m})\}$; and

$$\begin{array}{lcl}
\hat{\alpha}([\epsilon](R_{K_m}), a) & = & [a](R_{K_m}); \\
\hat{\alpha}([\epsilon](R_{K_m}), d) & = & [d](R_{K_m}); \\
\hat{\alpha}([a](R_{K_m}), a) & = & [aa](R_{K_m}) = [d](R_{K_m}); \\
\hat{\alpha}([a](R_{K_m}), d) & = & [ad](R_{K_m}) = [\epsilon](R_{K_m}); \\
\hat{\alpha}([d](R_{K_m}), a) & = & [da](R_{K_m}) = [d](R_{K_m}); \\
\hat{\alpha}([d](R_{K_m}), d) & = & [dd](R_{K_m}) = [d](R_{K_m}).
\end{array}$$

■

**Remark 1.4** Given a regular language $K$, there always exists a DFSM $G$ such that $L_m(G) = K$. Hence there exists a *minimal* such DFSM (one with a minimal number of states). Let $G'$ be the DFSM obtained by removing the

state $[s](R_K)$ from $\hat{G}$ (and all transitions leading into/out of it), where $\hat{G}$ is the DFSM in the proof of Theorem 1.5 and $s \in \Sigma^*$ is such that $s \notin pr(K)$. Then it is easy to see that $L_m(G') = K$. In fact $G'$ is a minimal DFSM with marked language $K$. In order to see this first note that the number of states in $\hat{G}$ is $|R_K|$, the index of $R_K$. Since $G'$ is obtained by removing a single state from $\hat{G}$, the number of states in $G'$ equals $|R_K| - 1$. Let $G$ be any DFSM with $L_m(G) = K$. Then as noted above number of states in $G$ equals $|R_G| - 1$. Also, as noted above, $R_G$ refines $R_{L_m(G)} = R_K$, which implies that $|R_K| \leq |R_G|$. Thus $|R_K| - 1 \leq |R_G| - 1$, which proves that $G'$ is a minimal DFSM with marked language $K$. ■

The following proposition can be easily proved, the proof of which is left as an exercise (refer to exercise problem 16).

**Theorem 1.6** Suppose $K \subseteq \Sigma^*$ is a regular language. Then a trim DFSM $G := (X, \Sigma, \alpha, x_0, X_m)$ with marked language $K$ is minimal if and only if

$$\forall x, x' \in X, s \in \Sigma^* : (\alpha(x, s), \alpha(x', s)) \in X_m \times X_m \Rightarrow x = x'.$$

## 1.4.2 Algorithms for Regular Languages

In this subsection we present some decision algorithms for regular languages using their DFSM representations.

### *Emptiness and Containment*

Suppose $K \subseteq \Sigma^*$ is regular. Let $G := (X, \Sigma, \alpha, x_0, X_m)$ be a trim DFSM such that $L_m(G) = K$. It is easy to see that

$$K = \emptyset \Leftrightarrow Re_G(x_0) \cap X_m = \emptyset.$$

Thus emptiness of $K$ can be determined by checking whether the set of reachable states from $x_0$ in $G$ contains a marked state. The following algorithm can be used to efficiently compute $Re_G(x_0)$:

**Algorithm 1.1**

1. Initiation step:

$$Re_G^{-1}(x_0) := \emptyset; \quad Re_G^0(x_0) := \{x_0\}; \quad k := 0.$$

2. Iteration step:
   $Re_G^{k+1}(x_0) := Re_G^k(x_0) \cup$
   $\{x \in X - Re_G^k(x_0) \mid \exists x' \in Re_G^k(x_0) - Re_G^{k-1}(x_0), \sigma \in \Sigma \text{ s.t. } \alpha(x', \sigma) = x\}$.

3. Termination step:
   If $Re_G^{k+1}(x_0) = Re_G^k(x_0)$, then set $Re_G(x_0) := Re_G^k(x_0)$; else, set $k := k+1$, and go to step 2.

Clearly, for each $k$, the set $Re_G^k(x_0)$ is the set of states reachable from $x_0$ in $k$ or less transitions. If the number of states in $G$ is $m$, then Algorithm 1.1 computes $Re_G(x_0)$ in $O(m)$ steps[1].

Next consider two regular languages $K_1, K_2 \subseteq \Sigma^*$. Then $K_1 \subseteq K_2$ if and only if $K_1 \cap K_2^c = \emptyset$. Thus the problem of determining containment of $K_1$ in $K_2$ reduces to that of determining emptiness of $K_1 \cap K_2^c$. Since $K_1 \cap K_2^c = L_m(G_1 \| (\overline{G_2})^c)$, this can be checked by checking whether $L_m(G_1 \| (\overline{G_2})^c) = \emptyset$.

## *Minimization*

Suppose $K \subseteq \Sigma^*$ is regular. Let $G := (X, \Sigma, \alpha, x_0, X_m)$ be a trim DFSM such that $L_m(G) = K$. We are interested in obtaining a minimal DFSM with marked language $K$ by combining some of the "language equivalent" states of $G$. Note that if $K = \Sigma^*$, then it can be accepted by a minimal DFSM having a single state. Hence we assume without loss of generality that $K \neq \Sigma^*$.

DFSM $G := (X, \Sigma, \alpha, x_0, X_m)$ induces an equivalence relation on $X$ defined as:

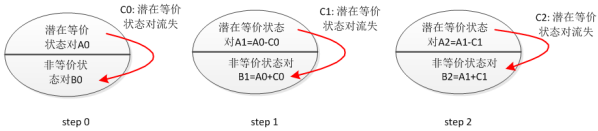
$$\forall x, x' \in X : x \cong x' \Leftrightarrow [\forall s \in \Sigma^* : \alpha(x, s) \in X_m \Leftrightarrow \alpha(x', s) \in X_m].$$

Using this equivalence relation we define a language equivalent DFSM $\hat{G} := (\hat{X}, \Sigma, \hat{\alpha}, \hat{x}_0, \hat{X}_m)$, where $\hat{X} := \{[x] \mid x \in X\}$, $\hat{x}_0 := [x_0]$, $\hat{X}_m := \{[x] \mid x \in X_m\}$, and

$$\forall\, [x] \in \hat{X}, \sigma \in \Sigma : \hat{\alpha}([x], \sigma) := \begin{cases} [\alpha(x, \sigma)] & \text{if } \alpha(x, \sigma) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Using the fact that $G$ is trim, it is readily verified that $\hat{G}$ is well defined, and $(L_m(\hat{G}), L(\hat{G})) = (L_m(G), L(G)) = (K, pr(K))$. Moreover, it follows from Theorem 1.6 that $\hat{G}$ is a minimal state machine with marked language $K$.

[1] Given two real valued functions $f(\cdot), g(\cdot)$; $f(\cdot)$ is said to be of *order* $g(\cdot)$, denoted $O(g(\cdot))$, if there exists a constant $c \in \mathcal{R}$ such that $f(\cdot) \leq cg(\cdot)$.






如果K=Sigma*，最小FA只有一个状态。

An algorithm for efficiently identifying the equivalence classes $\{[x] \mid x \in X\}$ is presented next. Note that each state pair $(x, x') \in (X_m \times X_m) \cup [(X - X_m) \times (X - X_m)]$ is a possible pair of equivalent states. First consider $\overline{G} := (\overline{X}, \Sigma, \overline{\alpha}, x_0, X_m)$, the completion of $G$. Then since $K \neq \Sigma^*$, $X_m \neq \overline{X} = X \cup \{x_D\}$, where $x_D$ the "dump" state. This implies $\overline{X} - X_m \neq \emptyset$.

A: {潜在等价状态对}; B: {非等价状态对}; C: {本次迭代流失的潜等价状态对}
A0：(final状态对)并(非final状态对)，是潜在的等价状态对
B0：(全体状态对) - A0= {(final状态，非final状态)}，非等价状态对(即是非A0状态对)
C0：A0中两大类中的(状态对)存在字母sigma进入B0类，潜在状态对-->非等价状态对

**Algorithm 1.2**

1. Initiation step:

$$A_0 := [X_m \times X_m] \cup [(\overline{X} - X_m) \times (\overline{X} - X_m)]; \quad B_0 := [\overline{X} \times \overline{X}] - A_0; \quad k := 0.$$

2. Iteration step:

$$\begin{aligned} C_k &:= \{(x, x') \in A_k \mid \exists \sigma \in \Sigma \text{ s.t. } (\alpha(x, \sigma), \alpha(x', \sigma)) \in B_k\} \\ A_{k+1} &:= A_k - C_k \\ B_{k+1} &:= B_k \cup C_k = [\overline{X} \times \overline{X}] - A_{k+1} \quad \{\text{非 } A_{k+1} \text{ 状态对}\} \end{aligned}$$

3. Termination step:
   If $A_{k+1} = A_k$, then stop; else, $k := k + 1$, and go to step 2.

It can be verified that after termination, each state pair in $A_k$ is an equivalent pair of states. Finally, the minimal state machine $\hat{G}$ is obtained by combining each state pair in $A_k$ as described above, and removing the equivalence class of the dump state. Algorithm 1.2 terminates in $O(m^2)$ steps, where $m$ is the number of states in $G$.

**Example 1.12** Consider the complete DFSM of Figure 1.5. Then $X_m = \{1, 2, 3, 4\}$ and $X = \{1, 2, 3, 4, x_D\}$. Algorithm 1.2 can be applied to minimize the DFSM as follows:

$$A_0 = [X_m \times X_m] \cup \{(x_D, x_D)\}; \quad B_0 = [X \times X] - A_0.$$

Then it can be seen that

$$C_0 = \{(1,2), (2,1), (1,3), (3,1), (2,4), (4,2), (3,4), (4,3)\}.$$

Hence

$$\begin{aligned} A_1 &= \{(1,4), (4,1), (2,3), (3,2), (1,1), (2,2), (3,3), (4,4), (x_D, x_D)\} \\ B_1 &= [X \times X] - A_1. \end{aligned}$$

Since $A_1 \neq A_0$, the algorithm does not terminate, and computes $C_1$:

$$C_1 = \{(1,4),(4,1),(2,3),(3,2)\}.$$

This implies that

$$\begin{aligned} A_2 &= \{(1,1),(2,2),(3,3),(4,4),(x_D,x_D)\} \\ B_2 &= [X \times X] - A_2. \end{aligned}$$

Clearly, all the state pairs in $A_2$ are equivalent pairs, i.e., $C_2 = \emptyset$; hence the algorithm terminates. Thus the minimal DFSM is the DFSM of Figure 1.6(b) which does not contain the dump state. ■

## 1.5 NON-REGULAR LANGUAGES

State machine based representation of languages results in a very coarse partitioning of languages: regular and non-regular. Although a regular language, in general, is infinite, it can only describe the behavior of a DES with finitely many states. Thus regular languages are quite limited in their *descriptive power*. Hence other concise ways of representing non-regular languages are needed. This leads us to the introduction of grammar-based representation of languages, which can concisely represent any "computable" language. However, this increase in descriptive power comes at an expense of loss in decision power.

**Definition 1.4** A grammar, denoted $\mathcal{G}$, is a quadruple

$$\mathcal{G} := (V, T, P, S),$$

where $V$ is the finite set of *variables* of $\mathcal{G}$, $T$ is the finite set of *terminals* of $\mathcal{G}$, $P$ is the finite set of *production rules* of the form $LHS \rightarrow RHS$ with $LHS, RHS \in (V \cup T)^*$, and $S \in V$ is the *start variable* of $\mathcal{G}$.

We can draw an analogy between a grammar and a state machine by viewing variables as states, terminals as events, production rules as state transitions, and the start variable as the initial state. Execution of a production rule replaces any $LHS$ string with the corresponding $RHS$ string. Initially, a production rule that replaces a start variable with a string of variables and terminals is executed. Strings consisting of only terminals are not replaced any more; they belong to the marked language of the grammar, denoted $L_m(\mathcal{G}) \subseteq T^*$.

**Example 1.13** Consider a grammar $\mathcal{G} := (V, T, P, S)$ with $V = \{S\}$, $T = \{a, b\}$, and $P = \{S \rightarrow aSb, S \rightarrow ab, S \rightarrow \epsilon\}$. Then it is easily seen that $L_m(\mathcal{G}) = \{a^i b^i \mid i \geq 0\}$. ■

### 1.5.1 Chomsky Hierarchy of Languages

Grammars have different descriptive power depending on constraints on their production rules. Noam Chomsky gave the following classification of languages, called *types* of languages, by defining different kinds of constraints on production rules.

**Type 0:** This is the most general form of a grammar in which no constraint on production rules is imposed. Languages of such grammars are equivalent to languages of *Turing machines* and are also known as *recursively enumerable* languages.

**Type 1:** This is a grammar with the constraint that the length of $RHS$ is no smaller than the length of $LHS$ in each production rule. Languages of such grammars are called *context-sensitive* languages and are equivalent to languages of *linear bounded automata.*

**Type 2:** This is a grammar with the constraint that $LHS \in V$ in each production rule. Thus it is possible to replace each variable of $LHS$ by its corresponding $RHS$ independent of its context. Languages of such grammars are called *context-free* languages and are equivalent to languages of *push down automata.*

**Type 3:** This a grammar with the constraint that $LHS \in V$ and $RHS \in T^*V \cup T^*$ in each production rule, i.e., each $RHS$ contains at most one variable at its end. These grammars are also called *right-linear* grammars. Languages of such grammars are regular languages and thus are equivalent to languages of DFSMs.

The grammar of Example 1.13 is a Type 2 grammar. It follows from Examples 1.13 and 1.10 that the class of regular languages is strictly contained in the class of Type 2 languages.

### 1.5.2 Turing Machine

The most general class of languages in Chomsky hierarchy is the Type 0 languages. Type 0 languages are of paramount interest as it is hypothesized, based on strong reasons, that they can describe the behavior of *any* computing device such as a modern day computer. This hypothesis is known as *Church's Hypothesis.* There exist many state transition based systems that accept Type 0 languages; the Turing machine (TM) is among the simplest ones.

Informally, a Turing machine consists of a finite state machine, a semi-infinite input tape divided into cells, and a tape head as shown in Figure 1.10. Initially

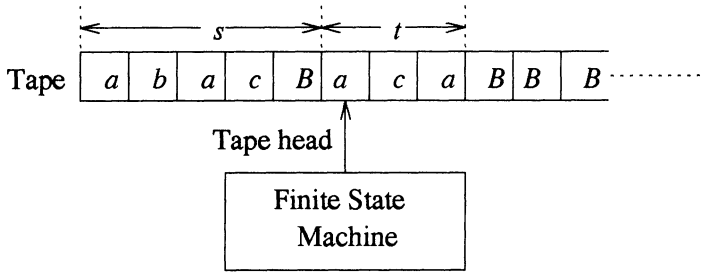



**Figure 1.10** Diagram illustrating structure of a Turing machine

each tape cell contains one of the tape symbols with the constraint that there is a tape cell to the right of which all tape cells contain the "blank" tape symbol. The tape head scans a tape cell symbol at any given point of time; initially the tape head scans the leftmost tape cell symbol. The Turing machine, depending upon the tape symbol being scanned and the state of the finite state machine, (i) changes the state of the finite state machine, (ii) writes a new tape symbol on the cell being scanned, and (iii) shifts the tape head by one cell either left or right. Formally, a (deterministic) Turing machine, denoted $T$, is a 7-tuple:

$$T = (X, \Gamma, B, \Sigma, \alpha, x_0, X_m),$$

where $X$ is the finite set of *states*, $\Gamma$ is finite set of *tape symbols*, $B \in \Gamma$ is the *blank symbol*, $\Sigma \subseteq \Gamma$ is the set of *input symbols* not containing the blank symbol, $\alpha : X \times \Gamma \rightarrow X \times \Gamma \times \{L, R\}$ is the transition function (where $L$ represents a left shift of tape head, and $R$ represents a right shift of the tape head), $x_0 \in X$ is the *initial state*, and $X_m \subseteq X$ is the set of *marked* or *accepting* states.

An *instantaneous description (ID)* of $T$ is a string $sxt \in \Gamma^* X \Gamma^*$, where $s \in \Gamma^*$ is the tape string to the left of the tape head, $x \in X$ is the state of the finite state machine, and $t \in \Gamma^*$ is the tape string to the right of the tape head (up to the rightmost non-blank infinite sequence of symbols), including the tape

symbol scanned by the tape head (refer to Figure 1.10). Initially, $s = \epsilon$, i.e., the tape head scans the left most cell, and $x = x_0$. At any point, $T$ changes its ID according to its transition function, and *halts* when $x \in X_m$. The language marked or accepted by $T$, denoted $L_m(T) \subseteq \Sigma^*$, is the set of all tape strings $t \in \Sigma^*$ such that the execution starting with the initial ID $\epsilon x_0 t$ results in halting of $T$. Note that $T$ only halts on those tape strings that belong to its marked language. It is possible that $T$ does not halt, i.e., it does not reach a marked state, on a certain initial tape string in $\Sigma^*$. Refer to Example 1.16 for an illustration of a TM.

The class of languages accepted by Turing Machines is called recursively enumerable, and can be shown to coincide with the Type 0 languages. These languages are called recursively enumerable because there exists a recursive procedure which can enumerate all strings of the language accepted by a given Turing Machine.

**Definition 1.5** Given a language $K \subseteq \Sigma^*$, it is called to be *recursively enumerable (r.e.)* if there exists a Turing machine $T$ such that $L_m(T) = K$; it is said to be *recursive* or *decidable* if $K$ and $K^c$ are both r.e.; it is said to be *undecidable* if $K$ is not decidable.

The following example illustrates that the class of r.e. languages is strictly contained in the class of all languages.

**Example 1.14** Note that a TM has a finite description, which implies that the set of all TMs is a countable set. On the other hand, the set of all languages is an uncountable set. Hence there exist languages which are not r.e. We give an example of such a language.

Let $\{T_i\}_{i \in \mathcal{N}}$ be an enumeration of all TMs, and $\{s_i\}_{i \in \mathcal{N}}$ be an enumeration of strings in $\Sigma^*$. (Note that $\Sigma^*$ is a countable set.) Define the "diagonal language":

$$L_d := \{s_i \in \Sigma^* \mid s_i \notin L_m(T_i)\}.$$

We show that there exists no $j \in \mathcal{N}$ such that $L_m(T_j) = L_d$. If $s_j \in L_m(T_j)$, then $s_j \notin L_d$. On the other hand, if $s_j \notin L_m(T_j)$, then $s_j \in L_d$. Thus $L_m(T_j) \neq L_d$ for any $j \in \mathcal{N}$, which proves that $L_d$ is not r.e. ■

The following example illustrates that the class of recursive languages is strictly contained in the class of r.e. languages.

**Example 1.15** Define the "universal language":

$$L_u := (L_d)^c = \{s_i \in \Sigma^* \mid s_i \in L_m(T_i)\},$$

where $\{T_i\}_{i \in \mathcal{N}}$ and $\{s_i\}_{i \in \mathcal{N}}$ are enumerations of all TMs and all strings in $\Sigma^*$ respectively. Construct the "universal TM" $T_u$ such that $T_u$ emulates TM $T_i$ on input $s_i$. Then clearly $L_m(T_u) = L_u$, which proves that $L_u$ is r.e. We show that $L_u$ is not recursive. Suppose for contradiction that it is recursive. Then it follows from the definition of recursive languages that $(L_u)^c = L_d$ is r.e., which is a contradiction (refer to the previous example). ■

Finally, the following example illustrates that the class of regular languages is strictly contained in the class of recursive languages.

**Example 1.16** We proved in Example 1.10 that the language $K := \{a^i b^i \mid i \geq 0\}$ defined over the event set $\Sigma = \{a, b\}$ is non-regular. Define a TM $T := (X, \Gamma, B, \Sigma, \alpha, x_0, X_m)$, where $X = \{x_0, x_1, x_2, x_3, x_4\}$, $\Gamma = \Sigma \cup \{\phi, \psi\} \cup \{B\}$, $X_m = \{x_4\}$, and the transition function is defined in Table 1.3.

| | $a$ | $b$ | $\phi$ | $\psi$ | $B$ |
|---|---|---|---|---|---|
| $x_0$ | $(x_1, \phi, R)$ | – | – | $(x_3, \psi, R)$ | $(x_4, B, R)$ |
| $x_1$ | $(x_1, a, R)$ | $(x_2, \psi, L)$ | – | $(x_1, \psi, R)$ | – |
| $x_2$ | $(x_2, a, L)$ | – | $(x_0, \phi, R)$ | $(x_2, \psi, L)$ | – |
| $x_3$ | – | – | – | $(x_3, \psi, R)$ | $(x_4, B, R)$ |
| $x_4$ | – | – | – | – | – |

**Table 1.3** Transition function of Turing Machine accepting language $\{a^i b^i \mid i \geq 0\}$

We show that $T$ halts on a string $s \in \Sigma^*$ if and only if $s \in K$, proving that $K$ is r.e. Suppose initially the tape contains the string $a^i b^j$ followed by an infinite number of blank symbols. Then initial ID of $T$ is $\epsilon x_0 a^i b^j$. If $i = j = 0$, then $T$ changes the state to $x_4$ and halts. If $i = 0$, $j \neq 0$, then no transition occurs in $T$, and $T$ does not halt. Consider next the case when $i \geq 1$, then $T$ replaces the leftmost $a$ by $\phi$, shifts right, and changes the state to $x_1$. While in state $x_1$, $T$ sequentially shifts right to the leftmost $b$ or blank symbol $B$, which ever is leftmost (without altering state and tape symbols). If $b$ is scanned, then $T$ replaces it by $\psi$, shifts left, and changes the state to $x_2$; else, if $B$ is scanned,

then $j = 0$, and no transition occurs in $T$, and $T$ does not halt. While in state $x_2$, $T$ sequentially shifts left to the rightmost $\phi$ (without altering state and tape symbols); where it shifts right and changes the state back to $x_0$ without altering the tape symbols. At this point if the tape symbol scanned is either $a$ or $b$, then a transition occurs as described above. However, if the tape symbol scanned is $\psi$, then $T$ changes the state to $x_3$ and shifts right without altering the tape symbol. While in state $x_3$, $T$ sequentially shifts right to the leftmost $b$ or $B$, which ever is leftmost. If $b$ is scanned, then no transition occurs in $T$, and $T$ does not halt; otherwise $T$ changes the state to $x_4$ and halts. Clearly, $T$ halts if and only if $i = j$.

| | $a$ | $b$ | $\phi$ | $\psi$ | $B$ |
|---|---|---|---|---|---|
| $x_0$ | $(x_1, \phi, R)$ | $(x_4, B, R)$ | – | $(x_3, \psi, R)$ | – |
| $x_1$ | $(x_1, a, R)$ | $(x_2, \psi, L)$ | – | $(x_1, \psi, R)$ | $(x_4, B, R)$ |
| $x_2$ | $(x_2, a, L)$ | – | $(x_0, \phi, R)$ | $(x_2, \psi, L)$ | – |
| $x_3$ | – | – | – | $(x_3, \psi, R)$ | – |
| $x_4$ | – | – | – | – | – |

**Table 1.4** Transition function of Turing Machine accepting language $\{a, b\}^* - \{a^i b^i \mid i \geq 0\}$

Next define a TM $T' := (X, \Gamma, B, \Sigma, \alpha', x_0, X_m)$, where the new transition function is defined in Table 1.4. It can be argued as above that $T'$ halts on a string $s \in \Sigma^*$ if and only if $s \in K^c$, proving that $K^c$ is also r.e. ■

# 1.6 EXERCISES

1. *Interleaving* of two strings $s, t \in \Sigma^*$, denoted $s[\!]t$, is defined inductively as:

$$\begin{aligned} \forall s \in \Sigma^* : s[\!]\epsilon = \epsilon[\!]s &:= s \\ \forall s, s' \in \Sigma^*, \sigma, \sigma' \in \Sigma : \sigma s[\!]\sigma' s' &:= \sigma(s[\!]\sigma' s') + \sigma'(\sigma s[\!]s'). \end{aligned}$$

Given $K_1, K_2 \subseteq \Sigma^*$, the *interleaving* between $K_1$ and $K_2$, denoted $K_1[\!]K_2$, is defined as

$$K_1[\!]K_2 := \{s \in \Sigma^* \mid \exists t_1 \in K_1, t_2 \in K_2 \text{ s.t. } s \in t_1[\!]t_2\}.$$

Given a language $K \subseteq \Sigma^*$, the *interleaving* closure of $K$, denoted $K^\circ$, is defined as

$$K^\circ := \bigcup_{n \geq 0} K^{(n)},$$

where $K^{(0)} := \{\epsilon\}$; and for each $n \geq 0$, $K^{(n+1)} := K^{(n)} \| K$. Prove that

(a) $(K_1 + K_2) \| K_3 = (K_1 \| K_3) + (K_2 \| K_3)$, where $K_3 \subseteq \Sigma^*$ is a language.

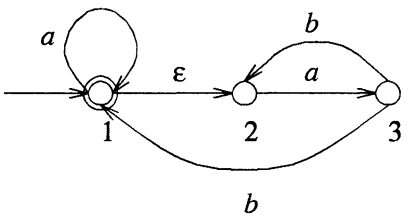
(b) $(K_1 + K_2)^\circ = K_1^\circ \| K_2^\circ$.

(c) $(K^*)^\circ = K^\circ$.

(d) $(K^\circ)^\circ = K^\circ$.

(The interleaving operation of languages is used to model *concurrent* operation of the corresponding DESs.)

2. Construct a language equivalent DFSM for the $\epsilon$-NFSM of Figure 2.




3. Prove or disprove for the regular expressions $r, s$:

(a) $(rs + r)^* r = r(sr + r)^*$

(b) $(r + s)^* = (r^* s^*)^*$

(c) $(r^* + s)^* = (r + s)^*$

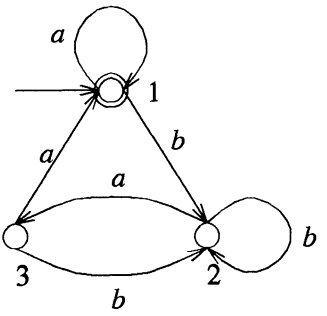
4. Using the inductive definition of $r_{ij}^k$ in the proof of Theorem 1.3 compute the marked language of the DFSM shown in Figure 4.

5. Construct DFSMs with marked languages given by the following regular expressions:

(a) $ab + (b + aa)b^*a$

(b) $ba[\{(ab)^* + aaa\}^* + b]^*a$

6. Given a DFSM $G$, prove that if $G$ is accessible, then it is co-accessible if and only if $pr(L_m(G)) = L(G)$.




7. Using the Myhill-Nerode characterization prove that the language $K = \{a^i b^i \mid i \geq 0\}$ defined over the event set $\{a, b\}$ is not regular.
**Hint:** Show that for each $i, k \geq 0$, $a^i \not\equiv a^{i+k} (R_K)$.

8. Show that reversal, projection on a given event set, and interleaving operations preserve regularity.

9. Given a language $K \subseteq \Sigma^*$, consider the equivalence relation on $\Sigma^*$ induced by $K$, denoted $\simeq$, defined as:

$$\forall s, t \in \Sigma^* : [s \simeq t] \Leftrightarrow [K/\{s\} = K/\{t\}].$$

Prove that $\simeq$ is indeed an equivalence relation, and it has finite index if and only if $K$ is regular.
**Hint:** Use the fact that $K$ is regular if and only if $K^R$ is regular.

10. Given a language $K \subseteq \Sigma^*$, define $s, t \in \Sigma^*$ to be equivalent, denoted $s \equiv t$, if

$$\forall u, v \in \Sigma^* : usv \in K \Leftrightarrow utv \in K.$$

Show that $\equiv$ is indeed an equivalence relation, and it has finite index if and only if $K$ is regular.

11. Given regular languages $K_1, K_2 \subseteq \Sigma^*$, prove that $K_1/K_2$ is regular. Given two deterministic finite state machines $G_1$ and $G_2$, give an *algorithm* to construct a finite state machine for $L(G_1)/L(G_2)$.

12. Consider languages $K_1 = (a+b)^* c^*$ and $K_2 = (a^* + b^*) d^*$ defined over the event set $\{a, b, c, d\}$.

    (a) Draw DFSMs marking $K_1$ and $K_2$.

    (b) Draw DFSMs marking $K_1 + K_2$, $K_1 \cap K_2$, $K_1 - K_2$, and $K_1 \| K_2$.

13. Let $f : \Sigma \rightarrow 2^{\Sigma^*}$ be such that for each $\sigma \in \Sigma$, $f(\sigma)$ is regular. Prove that given any regular $L \subseteq \Sigma^*$, $f(L)$ is also regular, where $f(L) := \bigcup_{s \in L} f(s)$ and for each $s \in \Sigma^*$, $f(s)$ is defined inductively as:

$$f(\epsilon) := \{\epsilon\}; \quad \forall s \in \Sigma^*, \sigma \in \Sigma : f(s\sigma) := f(s)f(\sigma).$$

14. Prove that the set of regular languages is not closed under arbitrary union.

15. Which of the following languages over $\Sigma = \{a, b\}$ are regular? Prove your answers.

    (a) $\{a^i \mid i \text{ is prime }\}$.

    (b) $\{s.s^R \mid s \in \Sigma^*\}$.

    (c) Set of all strings which do not have two consecutive a's.

    (d) Set of all strings so that the difference in the number of $a$'s and $b$'s in each string is an even number.

16. Prove the assertion of Theorem 1.6.

17. Draw a NFSM for the regular expression: $(11 + 0)^*.(00 + 1)^*$. Draw a DFSM equivalent to it. Obtain the corresponding minimal DFSM.

18. Prove that the set of recursive (respectively, r.e.) languages is closed (respectively, not closed) under boolean operations, i.e., the operations of union, intersection, and complement.

19. Construct a TM that accepts the language $\{ss^R \mid s \in \{0, 1\}^*\}$.

## 1.7 BIBLIOGRAPHIC REMARKS

All the material covered in this chapter can be found in standard books on automata theory and formal languages such as Hopcroft-Ullman [HU79].

# 2

# INTRODUCTION TO LATTICE THEORY

In this chapter we introduce the notion of lattices and obtain techniques for obtaining extremal solutions of inequations involving operations over lattices. Our motivation for studying these concepts and techniques stems from our interest in solving a system of inequations involving operations over the lattice of languages. Results presented in this chapter have a "primal" and a "dual" version. We only prove the primal version, as the dual version can be proved analogously.

## 2.1 PARTIAL ORDER AND LATTICE

A *relation* $R$ over $X$ is a subset of $X \times X$. A relation is *reflexive* if for each $x \in X$, $(x, x) \in R$; it is *symmetric* if $(x, y) \in R$ implies $(y, x) \in R$; it is *anti-symmetric* if $(x, y) \in R$ and $(y, x) \in R$ implies $x = y$; it is *transitive* if $(x, y) \in R$ and $(y, z) \in R$ implies that $(x, z) \in R$. Given a set $X$, a *partial order relation* , denoted $\leq$, over $X$ is a reflexive, anti-symmetric and transitive relation. For $x, y \in X$, if $x \leq y$, then $x$ is said to be *smaller* than $y$, and $y$ is said to be *greater* than $x$. Some examples of posets are the set of natural numbers and the set of real numbers under their usual orderings. Both of these sets also satisfy the additional property of totality. A partial-order relation is *total* if for each $x, y \in X$, either $x \leq y$ or $y \leq x$. An example of a poset which is not a total order is the set of natural numbers with the following relation: $x \leq y$ if and only if $x$ divides $y$. The containment relation defined on a power set is another example of a partial order which is not a total order.

**Definition 2.1** The pair $(X, \leq)$, where $X$ is a set and $\leq$ is a partial order over $X$, is called a *partially ordered set* or a *poset*. A totally ordered subset of $X$ is called a *chain*.

Finite posets are often depicted graphically using *Hasse* diagrams. A Hasse diagram of a poset is a directed acyclic graph with the property that there is a directed path from vertex $x$ to vertex $y$ if and only if $x \leq y$. Consider, for example, the following poset:

$$X := \{p, q, r, s\}; \quad \leq := \{(p, q), (q, r), (p, r), (p, s)\}.$$

Its Hasse diagram is shown in Figure 2.1.

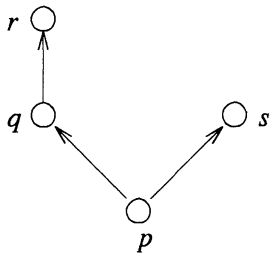



**Figure 2.1** Hasse Diagram

Given $Y \subseteq X$, $x \in X$ is said to be the *supremal* of $Y$ if

- (upper bound): $\forall y \in Y : y \leq x$, and
- (least upper bound): $\forall z \in X : [\forall y \in Y : y \leq z] \Rightarrow [x \leq z]$.

It is easy to check that the supremal of $Y$ is unique whenever it exists. The notation $supY$ is used to denote the supremal of $Y$. We also use $\sqcup Y$ to denote $supY$. In particular, given $x, y \in X$, $x \sqcup y$ is used to denote $sup\{x, y\}$.

Similarly, $x \in X$ is called the *infimal* of $Y \subseteq X$ if

- (lower bound): $\forall y \in Y : x \leq y$, and
- (greatest lower bound): $\forall z \in X : [\forall y \in Y : z \leq y] \Rightarrow [z \leq x]$.

It is easy to check that the infimal of $Y$ is unique whenever it exists. The notation $infY$ is used to denote the infimal of $Y$. We also use $\sqcap Y$ to denote

$infY$. In particular, given $x, y \in X$, $x \sqcap y$ is used to denote $inf\{x, y\}$. For example, in the set of natural numbers ordered by the divides relation, $x \sqcap y$ corresponds to finding the greatest common divisor and $x \sqcup y$ corresponds to finding the least common multiple of $x$ and $y$. For an example of the infimal and the supremal of an infinite set, consider the interval $[0, 2) := \{x \in \mathcal{R} \mid 0 \leq x < 2\}$. Then $sup[0, 2) = 2$ and $inf[0, 2) = 0$. Note that it follows from the above definitions that $sup\emptyset = infX$ (whenever it exists), and $inf\emptyset = supX$ (whenever it exists).

**Definition 2.2** A poset $(X, \leq)$ is said to be a *lattice* if $supY, infY \in X$ for any finite $Y \subseteq X$. If $supY, infY \in X$ for arbitrary $Y \subseteq X$, then $(X, \leq)$ is called a *complete lattice.*

A poset may be a lattice, but it may have a set $Y$ of infinite size for which $infY$ or $supY$ may not exist. A simple example is the set of natural numbers. There is no supremal element of set of even numbers even though the infimal and the supremal exists for any finite set. Another example is the set of rational numbers. For a finite subset of rational numbers, the supremal element corresponds to computing the maximum. However, consider the set $Y := \{x \mid x \text{ rational and } x \leq \sqrt{2}\}$. The supremal element of $Y$ is $\sqrt{2}$ which is not a rational number.

**Example 2.1** Given a set $X$, $(2^X, \subseteq)$—the set of all subsets of $X$ together with the containment partial order—is called the *power set lattice* of $X$. A power set lattice is an example of a complete lattice. ■

As described in the previous chapter, at the qualitative or *logical* level of abstraction, the behavior of a DES is described using the set of all possible sequences of events that it can execute. In the next two chapters we study extremal solutions of inequations defined over the power set lattice $(2^{\Sigma^*}, \subseteq)$—the set of all languages together with the containment partial order.

**Remark 2.1** It is sometimes useful to consider a weaker requirement than that of a complete lattice. A poset $(X, \leq)$ is called a *complete partial order (cpo)* if

- $infX$ exists, and
- the least upper bound exists for any increasing chain of $X$.

Note that a complete lattice is always a cpo. The set of natural numbers with the natural ordering is an example of a lattice which is not a cpo. ■

## 2.2 EXTREMAL FIXED POINTS

Given a set $X$ and a function $f : X \to X$, $x \in X$ is called a *fixed* point of $f(\cdot)$ if $f(x) = x$. Here we present sufficient conditions under which such a function possesses fixed points, and obtain techniques for computing some of them. Fixed point calculations are useful in *analysis* of DESs, as often it is required that the behavior of a DES satisfy a certain *invariance* property. Fixed point calculations can be used to compute the portion of the behavior which satisfies the required invariance property. Fixed point techniques are also used for *specifying* the behavior of a DES—instead of explicitly specifying the behavior, it is implicitly specified as a fixed point of a function defined over languages.

We begin by identifying several useful properties of functions defined over lattices. Given a poset $(X, \leq)$, a function $f : X \to X$ is said to be *idempotent* if

$$\forall x \in X : f(x) = f(f(x));$$

it is said to be *monotone* if

$$\forall x, y \in X : [x \leq y] \Rightarrow [f(x) \leq f(y)].$$

Given a complete lattice $(X, \leq)$, a function $f : X \to X$ is said to be *disjunctive* if

$$\forall Y \subseteq X : f(\sqcup_{y \in Y} Y) = \sqcup_{y \in Y} f(y);$$

it is said to be *conjunctive* if

$$\forall Y \subseteq X : f(\sqcap_{y \in Y} Y) = \sqcap_{y \in Y} f(y);$$

it is said to be *sup-continuous* if

$$\forall \text{ increasing chain } \{y_i, i \geq 0\} \subseteq X : f(\sqcup_{i \geq 0} y_i) = \sqcup_{i \geq 0} f(y_i);$$

it is said to be *inf-continuous* if

$$\forall \text{ decreasing chain } \{y_i, i \geq 0\} \subseteq X : f(\sqcap_{i \geq 0} y_i) = \sqcap_{i \geq 0} f(y_i).$$

It is readily verified that disjunctive, conjunctive, sup-continuous, and inf-continuous functions are also monotone. Note that since $sup\emptyset = infX$ and

$inf\emptyset = supX$, by setting $Y = \emptyset$ in the last two definitions we obtain for a disjunctive function that $f(infX) = infX$, and for a conjunctive function that $f(supX) = supX$.

**Example 2.2** Consider the power set lattice of languages defined over the event set $\Sigma$, and the prefix and extension closure operations defined over it. Then it is easily verified that for a language $K \subseteq \Sigma^*$, $pr(K) = K/\Sigma^*$ and $ext(K) = K\Sigma^*$. Thus prefix closure operation is a special case of the quotient operation and extension closure operation is a special case of of concatenation operation. It can be checked that both concatenation and quotient operations are disjunctive and thus monotone; however, none of them are conjunctive. The prefix and extension closure operations are also both idempotent.

Now consider the operation $f(K) := K \cup L$, where $L$ is any fixed non-empty language. $f$ is conjunctive since $f(\cap_i K_i) = (\cap_i K_i) \cup L = \cap_i (K_i \cup L) = \cap_i f(K_i)$. It is not disjunctive because $f(infX) = f(\emptyset) = L$ which is different from $infX$.

The function defined as $f(K) := L - K$ is not even monotone. ■

The following fixed point theorem is due to Knaster and Tarski:

**Theorem 2.1** Let $(X, \leq)$ be a complete lattice and $f : X \rightarrow X$ be a monotone function. Let $Y := \{x \in X \mid f(x) = x\}$ be the set of fixed points of $f$. Then

1. $infY \in Y$, and $infY = inf\{x \in X \mid f(x) \leq x\}$.
2. $supY \in Y$, and $supY = sup\{x \in X \mid x \leq f(x)\}$.

**Proof:** For notational simplicity, define $Z := \{x \in X \mid f(x) \leq x\}$. Then it suffices to show that $infY = infZ$ and $infZ \in Y$. Since $X$ is a complete lattice, $supX \in X$. By the definition of $supX$, $f(supX) \leq supX$, i.e., $supX \in Z$, which implies that $Z \neq \emptyset$. Also, since $X$ is a complete lattice, it follows that $infZ \in X$. By definition, we have $Y \subseteq Z$. Hence $infZ \leq infY$. It remains to show that $infY \leq infZ$. Since $infY$ is the infimal fixed point of $f$, it suffices to show that $infZ$ is a fixed point of $f$, i.e., $f(infZ) = Z$.

We first show that $f(infZ) \leq infZ$. By definition, we have $infZ \leq z$ for each $z \in Z$. Monotonicity of $f$ implies that $f(infZ) \leq f(z)$ for each $z \in Z$. Since for each $z \in Z$, $f(z) \leq z$, this implies that $f(infZ) \leq z$ for each $z \in Z$.

Hence $f(infZ) \leq infZ$. Next we show that $infZ \leq f(infZ)$. From above we have $f(infZ) \leq infZ$. Monotonicity of $f$ implies that $f(f(infZ)) \leq f(infZ)$. Hence it follows from the definition of $Z$ that $f(infZ) \in Z$, which implies that $infZ \leq f(infZ)$. ■

It follows from Theorem 2.1 that a monotone function defined over a complete lattice always has an infimal and a supremal fixed point. The following theorem provides a technique for computing such fixed points under stronger conditions. We first define the notion of disjunctive and conjunctive closure. Given a complete lattice $(X, \leq)$, and a function $f : X \rightarrow X$, the *disjunctive closure* of $f$, denoted $f^*$, is the map $f^* : X \rightarrow X$ defined as:

$$\forall x \in X : f^*(x) := \sqcup_{i \geq 0} f^i(x);$$

and the *conjunctive closure* of $f$, denoted $f_*$, is the map $f_* : X \rightarrow X$ defined as:

$$\forall x \in X : f_*(x) := \sqcap_{i \geq 0} f^i(x),$$

where $f^0$ is defined to be the identity function, and for each $i \geq 0$, $f^{i+1} := ff^i$. It is easy to see that the disjunctive as well conjunctive closures of $f$ are idempotent.

**Theorem 2.2** Let $(X, \leq)$ be a complete lattice and $f : X \rightarrow X$ be a function. Let $Y := \{x \in X \mid f(x) = x\}$ be the set of fixed points of $f$.

1. If $f$ is sup-continuous, then $infY = f^*(infX)$.
2. If $f$ is inf-continuous, then $supY = f_*(supX)$.

**Proof:** Since $f$ is sup-continuous, it is also monotone. Hence it follows from Theorem 2.1 that $infY \in Y$. We first show that $\{f^i(infX), i \geq 0\}$ is an increasing chain. By definition we have

$$f^0(infX) = infX \leq f(infX) = f^1(infX).$$

Hence monotonicity of $f$ implies that for each $i \geq 0$, $f^i(infX) \leq f^{i+1}(infX)$.

Next we prove that $f^*(infX)$ is a fixed point. This follows from equalities:

$$\begin{aligned} f(f^*(infX)) &= f[\sqcup_{i \geq 0} f^i(infX)] \\ &= \sqcup_{i \geq 0} f^{i+1}(infX) \\ &= \sqcup_{i \geq 1} f^i(infX) \\ &= \sqcup_{i \geq 0} f^i(infX) \\ &= f^*(infX), \end{aligned}$$

where the second equality follows from sup-continuity of $f$, and the fourth equality follows from the fact that $f^0(infX) = infX \leq f^i(infX)$ for all $i \geq 1$.

It remains to show that $f^*(infX) \leq y$ for any $y \in Y$. Fix $y \in Y$. By definition of $infX$ we have $infX \leq y$. Since $f$ is sup-continuous, it is also monotone. Hence $infX \leq y$ implies

$$\forall i \geq 0 : f^i(infX) \leq f^i(y) = y,$$

where the last equality follows from the fact that $y$ is a fixed point. Thus we conclude that $f^*(infX) = \sqcup_{i \geq 0} f^i(infX) \leq y$, as desired. ■

**Remark 2.2** In the first part of Theorem 2.2, since $\{f^i(infX), i \geq 0\}$ is an increasing chain, $f^*(infX)$ exists under the weaker condition that $(X, \leq)$ is a cpo. Thus it is possible to compute the infimal fixed point of a sup-continuous function whenever it is defined over a cpo. ■

## 2.3 DUAL, CO-DUAL, INVERSE, AND CONVERSE OPERATIONS

In this section we develop the notion of dual, co-dual, inverse, and converse operations and study some of their properties. These concepts are used in the next section for obtaining extremal solutions of a system of inequations. We begin by providing conditions for the existence of extremal solutions of simple inequations.

**Lemma 2.1** Consider a complete lattice $(X, \leq)$ and functions $f, g : X \to X$.

1. If $f$ is disjunctive, then the supremal solution of the inequation $f(x) \leq y$, in the variable $x$, exists for each $y \in X$.
2. If $g$ is conjunctive, then the infimal solution of the inequation $y \leq g(x)$, in the variable $x$, exists for each $y \in X$.

**Proof:** Since $(X, \leq)$ is complete, $infX \in X$, and by definition $infX \leq y$. Using the disjunctivity of $f$ we obtain $f(infX) = infX \leq y$. Thus the set of solutions of the inequation $f(x) \leq y$ is nonempty. Let $I$ be an indexing set

such that for each $i \in I$, $x_i \in X$ is a solution of the inequation $f(x) \leq y$. Then it suffices to show that $\sqcup_{i \in I} x_i$ is also a solution of the inequation. Since $(X, \leq)$ is complete, it follows that $\sqcup_{i \in I} x_i \in X$. Also, $f(\sqcup_{i \in I} x_i) = \sqcup_{i \in I} f(x_i) \leq y$, where the equality follows from the fact that $f$ is disjunctive and the inequality follows from the fact that $f(x_i) \leq y$ for each $i \in I$. ■

Lemma 2.1 can be used to define the notion of dual of a disjunctive function and co-dual of a conjunctive function.

**Definition 2.3** Consider a complete lattice $(X, \leq)$ and functions $f, g : X \to X$. If $f$ is disjunctive, then its *dual*, denoted $f^{\perp}(\cdot)$, is defined to be the supremal solution of the inequation $f(x) \leq (\cdot)$. If $g$ is conjunctive, then its *co-dual*, denoted $g^{\top}(\cdot)$, is defined to be the infimal solution of the inequation $(\cdot) \leq g(x)$. ($x \in X$ is the variable of the inequation.)

**Example 2.3** Consider the power set lattice of languages defined over the event set $\Sigma$ and the prefix and extension closure operations. Since these operations are disjunctive, their duals exist. It follows from the definition of duality that for $K \subseteq \Sigma^*$, $pr^{\perp}(K)$ is the supremal language whose prefix closure is contained in $K$. Thus $pr^{\perp}(K)$ is the *supremal prefix closed sublanguage* of $K$, which we denote as $supP(K)$. Similarly, $ext^{\perp}(K)$ is the *supremal extension closed* sublanguage of $K$, which we denote as $supE(K)$.

As another example of duality, consider $f(K) := K \cap L$, where $L$ is any nonempty fixed language. Since intersection with a constant set is a disjunctive operation, its dual exists. It can be verified that $f^{\perp}(K) = K \cup L^c$. ■

The following proposition provides an alternative definition of duality as well as of co-duality.

**Theorem 2.3** Consider a complete lattice $(X, \leq)$, a disjunctive function $f : X \to X$, and a conjunctive function $g : X \to X$. Then the following are equivalent.

1. $f^{\perp} = g$.
2. $\forall x, y \in X : [f(x) \leq y] \Leftrightarrow [x \leq g(y)]$.
3. $g^{\top} = f$.

**Proof:** We only prove the equivalence of the first and the second assertion; the equivalence of the second and the third assertion can be proved analogously. Since $f$ is disjunctive, $f^{\perp}$ is defined. Suppose the first assertion is true. In order to see the forward implication of the second assertion, suppose $f(x) \leq y$, which implies $x$ is a solution of the inequation. Since $f^{\perp}(y) = g(y)$ is the supremal solution of the inequation, it follows that $x \leq g(y)$. Next in order to see the backward implication, suppose $x \leq g(y)$. So from monotonicity of $f$ we obtain that $f(x) \leq f(g(y))$. Since $g(y) = f^{\perp}(y)$ is a solution of the inequation, we have $f(g(y)) \leq y$. So $f(x) \leq f(g(y)) \leq y$, as desired.

Next suppose the second assertion holds. By setting $x = g(y)$ in the second assertion, we obtain that for all $y \in X$, $f(g(y)) \leq y$. This shows that $g(y)$ is solution of the inequation. Finally using the forward implication of the second assertion we conclude that if $x$ is a solution of the inequation, then $x \leq g(y)$. This shows that $g(y)$ is the supremal solution of the inequation. So $g(y) = f^{\perp}(y)$ for all $y \in X$. ■

Note that the equivalence of the first two assertions in Theorem 2.3 does not require $g$ to be conjunctive. Hence if we replace $g$ by $f^{\perp}$, then the first assertion is identically true; consequently, the second assertion is also identically true. Similarly it can be argued that the second assertion is identically true with $f$ replaced by $g^{\top}$. This is stated in the following corollary.

**Corollary 2.1** Consider a complete lattice $(X, \leq)$, and functions $f, g : X \rightarrow X$.

1. If $f$ is disjunctive, then $\forall x, y \in X : [f(x) \leq y] \Leftrightarrow [x \leq f^{\perp}(y)]$.
2. If $g$ is conjunctive, then $\forall x, y \in X : [g^{\top}(x) \leq y] \Leftrightarrow [x \leq g(y)]$.

Corollary 2.1 can be used to obtain several interesting properties of the dual and co-dual operations. We first show that dual of a disjunctive function is conjunctive, and co-dual of a conjunctive function is disjunctive.

**Lemma 2.2** Consider a complete lattice $(X, \leq)$ and functions $f, g : X \rightarrow X$.

1. If $f$ is disjunctive, then $f^{\perp}$ is conjunctive.
2. If $g$ is conjunctive, then $g^{\top}$ is disjunctive.

**Proof:** Pick $Y \subseteq X$. We need to show that $f^{\perp}(\sqcap_{y \in Y} y) = \sqcap_{y \in Y} f^{\perp}(y)$. The forward inequality can be shown as follows:

$$\begin{aligned}
[f^{\perp}(\sqcap_{y \in Y} y) \leq f^{\perp}(\sqcap_{y \in Y} y)] &\Leftrightarrow [f(f^{\perp}(\sqcap_{y \in Y} y)) \leq \sqcap_{y \in Y} y] \\
&\Leftrightarrow [\forall y \in Y : f(f^{\perp}(\sqcap_{y \in Y} y)) \leq y] \\
&\Leftrightarrow [\forall y \in Y : f^{\perp}(\sqcap_{y \in Y} y) \leq f^{\perp}(y)] \\
&\Leftrightarrow [f^{\perp}(\sqcap_{y \in Y} y) \leq \sqcap_{y \in Y} f^{\perp}(y)],
\end{aligned}$$

where the first and the third equivalence follow from Corollary 2.1.

Next the reverse inequality can be obtained as follows:

$$\begin{aligned}
[\forall y \in Y : \sqcap_{y \in Y} f^{\perp}(y) \leq f^{\perp}(y)] &\Leftrightarrow [\forall y \in Y : f(\sqcap_{y \in Y} f^{\perp}(y)) \leq y] \\
&\Leftrightarrow [f(\sqcap_{y \in Y} f^{\perp}(y) \leq \sqcap_{y \in Y} y] \\
&\Leftrightarrow [\sqcap_{y \in Y} f^{\perp}(y) \leq f^{\perp}(\sqcap_{y \in Y} y)],
\end{aligned}$$

where the first and the final equivalence follow from Corollary 2.1. ■

It follows from Lemma 2.2 that it is possible to define co-dual of the dual of a disjunctive function and dual of the co-dual of a conjunctive function. The following proposition describes some other properties of dual and co-dual operations.

**Theorem 2.4** Consider a complete lattice $(X, \leq)$, functions $f, f_1, f_2 : X \to X$ that are disjunctive, and functions $g, g_1, g_2 : X \to X$ that are conjunctive.

| | | |
|---|---|---|
| 1. ($^{\perp}$, $^{\top}$ inverses) | $(f^{\perp})^{\top} = f$ | $(g^{\top})^{\perp} = g$ |
| 2. (composition) | $(f_1 f_2)^{\perp} = f_2^{\perp}(f_1^{\perp})$ | $(g_1 g_2)^{\top} = g_2^{\top}(g_1^{\top})$ |
| 3. (idempotence) | $[ff = f] \Leftrightarrow [f^{\perp} f^{\perp} = f^{\perp}]$ | $[gg = g] \Leftrightarrow [g^{\top} g^{\top} = g^{\top}]$ |

**Proof:** 1. Since $f$ is disjunctive, it follows from Lemma 2.2 that $f^{\perp}$ is conjunctive, so $(f^{\perp})^{\top}$ is defined. By replacing $g$ with $f^{\perp}$ in Theorem 2.3 we obtain from its third assertion that $(f^{\perp})^{\top} = f$, as desired.

2. Since disjunctivity is preserved under composition of functions, $f_1 f_2$ is disjunctive, so that its dual is defined. Fix $x, y \in X$. Then the repeated application of Corollary 2.1 yields the following series of equivalences:

$$\begin{aligned}
[f_1 f_2(x) \leq y] &\Leftrightarrow [f_2(x) \leq f_1^{\perp}(y)] \\
&\Leftrightarrow [x \leq f_2^{\perp} f_1^{\perp}(y)].
\end{aligned}$$

Since $f_2^\perp f_1^\perp$ is conjunctive (follows from Lemma 2.2, and the fact that conjunctivity is preserved under composition of functions), if we replace $f$ by $f_1 f_2$ and $g$ by $f_2^\perp f_1^\perp$ in Theorem 2.3, we obtain $(f_1 f_2)^\perp = f_2^\perp f_1^\perp$, as desired.

3. The forward implication can be shown as follows: $f^\perp = (ff)^\perp = f^\perp f^\perp$, where the first equality follows from the hypothesis, and the second from part 2. The backward implication can be obtained as follows: $f = (f^\perp)^\top = (f^\perp f^\perp)^\top = ff$, where the first equality follows from part 1, the second from the hypothesis, and the final from parts 2 and 1. ■

**Example 2.4** Consider the power set lattice of languages defined over the event set $\Sigma$. We showed in Example 2.3 that $pr^\perp = supP$ and $ext^\perp = supE$. Then it follows from Lemma 2.2 that $supP$ as well as $supE$ are conjunctive. Moreover, Theorem 2.4 implies that $(supP)^\top = (pr^\perp)^\top = pr$ and $(supE)^\top = (ext^\top)^\perp = ext$. Finally, since $pr$ and $ext$ are idempotent, it follows from Theorem 2.4 that $pr^\perp = supP$ and $ext^\top = supE$ are idempotent. ■

### 2.3.1 Conjugate Operation

The notions of duality and co-duality can be defined for functions defined over complete lattices. However, if the lattice is also a Boolean lattice, so that each lattice element can be *uniquely complemented*, then the notion of the *conjugate* of a function can also be defined. This is then used to define the *inverse* of a disjunctive function and the *converse* of a conjunctive function.

**Definition 2.4** A lattice $(X, \leq)$ is said to be a *Boolean* lattice, if

- (Bounded): $infX, supX \in X$, and
- (Distributive): $\forall x, y, z \in X : x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$, and
- (Complement): $\forall x \in X : \exists$ unique $x^c \in X$ s.t. $x \sqcap x^c = infX$; $x \sqcup x^c = supX$.

For a pair $x, y$ of elements of a Boolean lattice $(X, \leq)$, the notation $x - y$ is used to denote $x \sqcap y^c$. A power set lattice is an example of a Boolean lattice that is also complete. The following hold for a Boolean lattice:

**Lemma 2.3** Let $(X, \leq)$ be a Boolean lattice. Then

1. $(infX)^c = supX$ and $(supX)^c = infX$.
2. $\forall x \in X : (x^c)^c = x$.
3. (de Morgan's Law): $\forall x, y \in X : (x \sqcap y)^c = x^c \sqcup y^c; \quad (x \sqcup y)^c = x^c \sqcap y^c$.
4. $\forall x, y \in X : [x \leq y] \Leftrightarrow [x \sqcap y^c = infX]$.

**Definition 2.5** Given a complete Boolean lattice (CBL) $(X, \leq)$ and a function $f : X \rightarrow X$, the *conjugate* of $f$, denoted $\overline{f}$, is defined as:

$$\forall x \in X : \overline{f}(x) := (f(x^c))^c.$$

If $f$ is disjunctive, then its *inverse*, denoted $f^{-1}$, is defined to be the function $\overline{f}^{\perp}$; and if $f$ is conjunctive, then its *converse*, denoted $f^{\natural}$, is defined to be the function $(\overline{f})^{\perp}$.

Note that if $f$ is disjunctive (respectively, conjunctive), then it follows from de Morgan's law that $\overline{f}$ is conjunctive (respectively, disjunctive). Hence the inverse (respectively, converse) of a disjunctive (respectively, conjunctive) function is well defined. Moreover, it follows from Lemma 2.2 that the inverse of a disjunctive function is disjunctive, and converse of a conjunctive function is conjunctive. Appendix 2.5 provides a justification for the choice of the name inverse for the operation conjugate of dual.

**Example 2.5** Consider the power set lattice of languages defined over the event set $\Sigma$ and the prefix and extension closure operations. It follows from the definition of conjugate that for a language $K \subseteq \Sigma^*$, $\overline{pr}(K) = \Sigma^* - pr(\Sigma^* - K) = supE(K)$, the supremal extension closed sublanguage of $K$. Similarly, $\overline{ext}(K) = \Sigma^* - ext(\Sigma^* - K) = supP(K)$, the supremal prefix closed sublanguage of $K$. These relations between prefix and extension closure operations are not coincidental, rather they can be derived as we show later in this chapter.

It can also be easily seen that if $f(K) := K \cap L$, then $\overline{f}(K) = K \cup L^c$. Thus it follows from Example 2.3 that the conjugate and the dual of an intersection operator are the same. ■

The following lemma lists a few properties of the conjugate operation.

**Lemma 2.4** Consider a CBL $(X, \leq)$ and functions $f, g : X \rightarrow X$.

1. (self-inverse) $\overline{\overline{f}} = f$
2. (composition) $\overline{fg} = \overline{f}\,\overline{g}$
3. (idempotence) $[ff = f] \Leftrightarrow [\overline{f}\,\overline{f} = \overline{f}]$

**Proof:** The first assertion is obvious. In order to see the second assertion, pick $x \in X$. Then we have $\overline{fg}(x) = (fg(x^c))^c = (f(\overline{g}(x))^c)^c = \overline{f}\,\overline{g}(x)$. The forward implication of the third assertion is obtained as follows: $\overline{f}\,\overline{f} = \overline{ff} = \overline{f}$, where the first equality follows from part 2, and the second from the hypothesis. The backward implication of the third assertion is obtained as follows: $f = \overline{\overline{f}} = \overline{\overline{f}\,\overline{f}} = ff$, where the first equality follows from part 1, the second from the hypothesis, and the third from parts 2 and 1. ■

Next we provide a few properties of the inverse and the converse operations. The following proposition provides an alternative definition of inverse as well as converse.

**Theorem 2.5** Consider a CBL $(X, \leq)$, and functions $f, g, h : X \to X$.

1. If $f$ is disjunctive, then $f^{-1} = h$ if and only if

$$\forall x, y \in X : [f(x) \sqcap y = inf X] \Leftrightarrow [x \sqcap h(y) = inf X]. \tag{2.1}$$

2. If $g$ is conjunctive, then $g^{\sharp} = h$ if and only if

$$\forall x, y \in X : [g(x) \sqcup y = sup X] \Leftrightarrow [x \sqcup h(y) = sup X].$$

**Proof:** Since $f^{-1} := \overline{(f^{\perp})}$, it follows from the first part of Lemma 2.4 that $f^{-1} = h$ if and only if $f^{\perp} = \overline{h}$. Thus it suffices to show that $f^{\perp} = \overline{h}$ is equivalent to (2.1). From Theorem 2.3, $f^{\perp} = \overline{h}$ is equivalent to

$$\forall x, y \in X : [f(x) \leq y] \Leftrightarrow [x \leq \overline{h}(y)]. \tag{2.2}$$

Hence it suffices to show the equivalence of (2.2) and (2.1). Replacing $y$ by $y^c$ in (2.2) we obtain

$$\forall x, y \in X : [f(x) \leq y^c] \Leftrightarrow [x \leq (h(y))^c].$$

Thus the desired equivalence follows from the part 4 of Lemma 2.3. ■

The following proposition provides additional properties of inverse and converse operations.

**Theorem 2.6** Consider a CBL $(X, \leq)$, disjunctive functions $f, f_1, f_2 : X \to X$, and conjunctive functions $g, g_1, g_2 : X \to X$.

| | | |
|---|---|---|
| 1. (commutation) | $\overline{f^{\perp}} = \overline{f}^{\top}$ | $\overline{g^{\top}} = \overline{g}^{\perp}$ |
| 2. (self-inverse) | $(f^{-1})^{-1} = f$ | $(g^{\sharp})^{\sharp} = g$ |
| 3. (composition) | $(f_1 f_2)^{-1} = f_2^{-1} f_1^{-1}$ | $(g_1 g_2)^{\sharp} = g_2^{\sharp} g_1^{\sharp}$ |
| 4. (idempotence) | $[ff = f] \Leftrightarrow [f^{-1} f^{-1} = f^{-1}]$ | $[gg = g] \Leftrightarrow [g^{\sharp} g^{\sharp} = g^{\sharp}]$ |

**Proof:** 1. Since $\overline{f^{\perp}} = f^{-1}$, it follows from the first part of Theorem 2.5 that it suffices to show that (2.1) holds with $h$ replaced by $\overline{f}^{\top}$. This can be shown as follows:

$$\begin{aligned}
[f(x) \sqcap y = inf X] &\Leftrightarrow [f(x) \leq y^c] \\
&\Leftrightarrow [y \leq (f(x))^c] \\
&\Leftrightarrow [y \leq \overline{f}(x^c)] \\
&\Leftrightarrow [\overline{f}^{\top}(y) \leq x^c] \\
&\Leftrightarrow [\overline{f}^{\top}(y) \sqcap x = inf X],
\end{aligned}$$

where the fourth equivalence follows from the second part of Corollary 2.1, and the other equivalences follow from Lemma 2.3.

2. From part 1, $f^{-1} = \overline{f}^{\top}$. Hence from Theorem 2.3, $(f^{-1})^{\perp} = \overline{f}$. By applying the conjugate operation on both sides of this identity, we obtain $(f^{-1})^{-1} = f$.

3. We have $(f_1 f_2)^{-1} = \overline{((f_1 f_2)^{\perp})} = \overline{f_2^{\perp} f_1^{\perp}} = \overline{f_2^{\perp}}\,\overline{f_1^{\perp}} = f_2^{-1} f_1^{-1}$, as desired.

4. The forward implication can be obtained as follows: $f^{-1} = (ff)^{-1} = f^{-1} f^{-1}$, where the first equality comes from the hypothesis, and the second from part 3. The backward implication can be obtained as follows: $f = (f^{-1})^{-1} = (f^{-1} f^{-1})^{-1} = ff$, where the first equality comes from part 2, the second from the hypothesis, and the final from parts 3 and 2. ■

**Remark 2.3** The commutative diagram of Figure 2.2 summarizes the relationship among the various operations that we have obtained above. Note that given any disjunctive function $f$, by applying $(\cdot)^{\perp}, (\cdot)^{\top}$ and $\overline{(\cdot)}$ any number of times, we get four unique functions: $(f, f^{-1}, f^{\perp}, \overline{f})$, the first two of which are disjunctive and the last two are conjunctive.

The corresponding commutative diagram for the operations of $pr$, $ext$, $supP$, and $supE$ is shown in Figure 2.3. ■

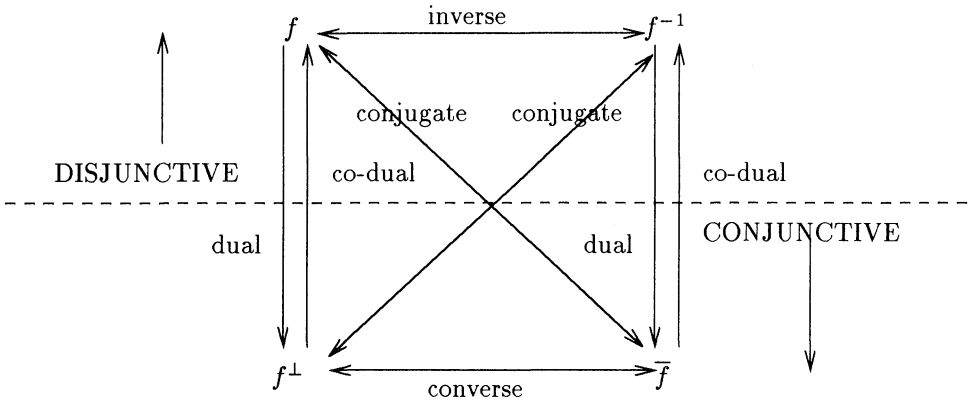



**Figure 2.2** Commutative diagram for dual, co-dual, conjugate, inverse, and converse

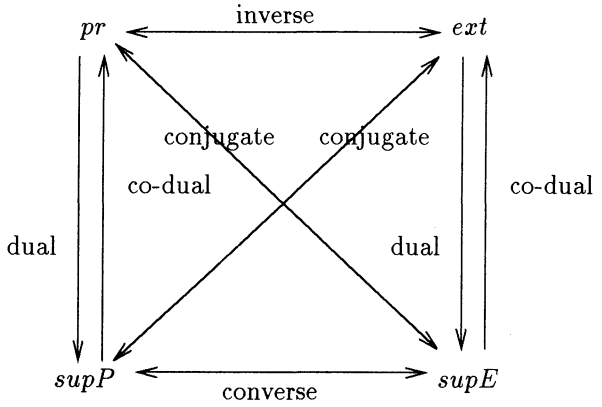



**Figure 2.3** Commutative diagram for *pr*, *ext*, *supP*, and *supE* operations

## 2.4 EXTREMAL SOLUTIONS OF INEQUATIONS

Given a complete lattice $(X, \leq)$ and a finite family of functions $\{f_i, g_i : X \to X\}_{i \leq n}$, where $n \in \mathcal{N}$, we next consider computation of extremal solutions of the system of inequations:

$$[\forall i \leq n : f_i(x) \leq g_i(x)],$$

where $x \in X$ is the variable of the system of inequations. Note that this also allows us to obtain extremal solutions of a system of *equations*, as each equation can equivalently be written as a pair of *inequations*. We show that the computation of extremal solutions of the above system of inequations can be reduced to extremal fixed point computations of certain induced functions. We need the result of the following lemma:

**Lemma 2.5** Consider the system of inequations $\{f_i(x) \leq g_i(x)\}_{i \leq n}$ over a complete lattice $(X, \leq)$. Define functions $h_1, h_2 : X \to X$ as:

$$\forall y \in X : h_1(y) := \sqcap_{i \leq n} f_i^{\perp}(g_i(y)); \quad \forall y \in X : h_2 := \sqcup_{i \leq n} g_i^{\top}(f_i(y)). \tag{2.3}$$

1. If $f_i$ is disjunctive and $g_i$ is monotone for each $i \leq n$, then $h_1$ is monotone, and $\forall y, z \in X : [y \leq h_1(z)] \Leftrightarrow [\forall i \leq n : f_i(y) \leq g_i(z)]$.
2. If $f_i$ is monotone and $g_i$ is conjunctive for each $i \leq n$, then $h_2$ is monotone, and $\forall y, z \in X : [h_2(y) \leq z] \Leftrightarrow [\forall i \leq n : f_i(y) \leq g_i(z)]$.

**Proof:** In order to show the monotonicity of $h_1$, it suffices to show that for each $i \leq n$, $f_i^{\perp} g_i$ is monotone. This follows from the facts that $g_i$ is given to be monotone, $f_i^{\perp}$ is conjunctive (refer to Lemma 2.2), so that it is also monotone, and monotonicity is preserved under composition of functions.

In order to see the second claim, fix $y, z \in X$. Then we have the following series of equivalences:

$$\begin{aligned} [y \leq h_1(z)] &\Leftrightarrow [y \leq \sqcap_{i \leq n} f_i^{\perp}(g_i(z))] \\ &\Leftrightarrow [\forall i \leq n : y \leq f_i^{\perp}(g_i(z))] \\ &\Leftrightarrow [\forall i \leq n : f_i(y) \leq g_i(z)], \end{aligned}$$

where the final equivalence follows from the first part of Corollary 2.1. ■

**Theorem 2.7** Consider the system of inequations $\{f_i(x) \leq g_i(x)\}_{i \leq n}$ over a complete lattice $(X, \leq)$. Let

$$Y := \{y \in X \mid \forall i \leq n : f_i(y) \leq g_i(y)\} \tag{2.4}$$

be the set of all solutions of the system of inequations; and

$$Y_1 := \{y \in X \mid h_1(y) = y\}, \quad Y_2 := \{y \in X \mid h_2(y) = y\} \tag{2.5}$$

be the sets of all fixed points of $h_1$ and $h_2$, respectively, where $h_1$ and $h_2$ are defined by (2.3).

1. If $f_i$ is disjunctive and $g_i$ is monotone, then $supY \in Y$, $supY_1 \in Y_1$ and $supY = supY_1$.
2. If $f_i$ is monotone and $g_i$ is conjunctive, then $infY \in Y$, $infY_2 \in Y_2$ and $infY = infY_2$.

**Proof:** It follows from the first part of Lemma 2.5 that $h_1$ is monotone. Hence it follows from the second part of Theorem 2.1 that $supY_1 \in Y_1$, and

$$supY_1 = supY_1', \text{ where } Y_1' := \{y \in X \mid y \leq h_1(y)\}. \tag{2.6}$$

It remains to show that $supY \in Y$ and $supY = supY_1$. In view of (2.6), it suffices to show that $Y = Y_1'$, i.e, $y \in X$ is a solution of the system of inequations if and only if $y \leq h_1(y)$. This follows from the first part of Lemma 2.5 by setting $z = y$. ■

Theorem 2.7 provides a condition for the existence of extremal solutions of a system of inequations. The following theorem provides techniques for the computation of such extremal solutions.

**Theorem 2.8** Consider the system of inequations $\{f_i(x) \leq g_i(x)\}_{i \leq n}$ over a complete lattice $(X, \leq)$; and the set $Y$ of all solutions of the system of inequations as defined by (2.4).

1. Let $f_i$ be disjunctive and $g_i$ be monotone. Consider the following iterative computation:

- $y_0 := supX$,
- $\forall k \geq 0 : y_{k+1} := h_1(y_k)$,

where $h_1$ is defined by (2.3). Suppose $m \in \mathcal{N}$ is such that $y_{m+1} = y_m$; then $y_m = supY$.

2. Let $f_i$ be monotone and $g_i$ be conjunctive. Consider the following iterative computation:

   - $y_0 := infX$,
   - $\forall k \geq 0 : y_{k+1} := h_2(y_k)$,

   where $h_2$ is defined by (2.3). Suppose $m \in \mathcal{N}$ is such that $y_{m+1} = y_m$; then $y_m = infY$.

**Proof:** It follows from Theorem 2.7 that the supremal solution of the system of inequations, $supY$, exists. We first show that $y_m \in Y$, i.e., it is a solution of the system of inequations. Since $y_{m+1} = y_m$, we have $h_1(y_m) = y_m$, which implies that $y_m \leq h_1(y_m)$. Thus by setting $y = z = y_m$ in the equivalence of the first part of Lemma 2.5, we obtain that $y_m \in Y$.

Next we show that if $z \in Y$ is another solution of the system of inequations, then $z \leq y_m$. We use induction to show that for each $k \geq 0$, $z \leq y_k$. If $k = 0$, then $y_k = supX$, so that $z \leq y_k = supX$. Thus the base step trivially holds. Suppose for the induction hypothesis that $z \leq y_k$ for some $k \geq 0$. From the first part of Lemma 2.5 we have that $h_1$ is monotone. This together with the induction hypothesis implies that $h_1(z) \leq h_1(y_k) = y_{k+1}$. Thus it suffices to show that $z \leq h_1(z)$. Since $z \in Y$, $f_i(z) \leq g_i(z)$ for each $i \leq n$. Thus by setting $y = z$ in the first part of Lemma 2.5, we obtain that $z \leq h_1(z)$. ■

**Example 2.6** Suppose that we are interested in the supremal solution of the inequation $X \cap A \subseteq X \cap B$ defined over the power set lattice of languages. Then, the iteration of Theorem 2.8 yields the following:

$$y_0 = \Sigma^*$$
$$y_1 = (\Sigma^* \cap B) \cup A^c = B \cup A^c$$
$$y_2 = ((B \cup A^c) \cap B) \cup A^c = B \cup A^c.$$

Since $y_2 = y_1$, it follows from Theorem 2.8 that $y_1$ is the supremal solution of the inequation. ■

### 2.4.1 Some Special Cases of Extremal Solutions of Inequations

In many applications we are interested in finding the supremal solution smaller than a given element $w \in X$, and/or the infimal solution greater than the given element $w$, of a system of inequations of the form:

$$\{f_i(x) \sqcap v_i \leq g_i(x)\}_{i \leq n},$$

where $\{v_i \in X\}_{i \leq n}$ is a given family of fixed elements. Note that if $w = v_i = supX$, then this problem reduces to the problem analyzed in Theorems 2.7 and 2.8. Conversely, we show that the problem just described can be analyzed using techniques developed in Theorems 2.7 and 2.8 provided the lattice is also Boolean, so that lattice elements can be uniquely complemented.

First note that the constraint that the supremal solution be smaller than $w$ can be captured by adjoining the following additional inequation:

$$f_a(x) \leq g_a(x),$$

where $f_a$ is the identity function, i.e., $f_a(x) := x$, and $g_a$ is the constant function $g_a(x) := w$. Similarly the constraint that the infimal solution should be greater than $w$ can be captured by adjoining the following additional inequation:

$$f_b(x) \leq g_b(x),$$

where $f_b$ is the constant function $f_b(x) := w$, and $g_b(x)$ is the identity function. Next note that

$$[f_i(x) \sqcap v_i \leq g_i(x)] \Leftrightarrow [f_i(x) \leq g_i(x) \sqcup v_i^c].$$

Thus if we define $g_i'(x) := g_i(x) \sqcup v_i^c$ and $f_i'(x) := f_i(x) \sqcap v_i$, then we obtain

$$[f_i(x) \sqcap v_i \leq g_i(x)] \Leftrightarrow [f_i(x) \leq g_i'(x)] \Leftrightarrow [f_i'(x) \leq g_i(x)].$$

It can be checked that (i) $g_i'$ is monotone if and only if $g_i$ is monotone; $f_i'$ is monotone if and only if $f_i$ is monotone, (ii) the identity function is disjunctive as well as conjunctive, (iii) a constant function is monotone, (iv) $f_a^{\top}(g_a(x)) = g_b^{\top}(f_b(x)) = w$, and (v) $(g_i'(x))^c = (g_i(x) \sqcup v_i^c)^c = v_i \sqcap (g_i(x))^c = v_i - g_i(x)$. We also have

$$\begin{aligned}
[\sqcap_{i \leq n} f_i^{\perp}(g_i'(x))] \sqcap f_a^{\perp}(g_a(x)) &= \sqcap_{i \leq n} [f_i^{-1}[v_i - g_i(x)]]^c \sqcap w \\
&= w \sqcap [\sqcup_{i \leq n} f_i^{-1}[v_i - g_i(x)]]^c \\
&= w - \sqcup_{i \leq n} f_i^{-1}[v_i - g_i(y)].
\end{aligned}$$

Thus the following result can be obtained as a corollary of Theorems 2.7 and 2.8.

**Corollary 2.2** Given a CBL $(X, \leq)$, a fixed $w \in X$, and a family of lattice elements $\{v_i \in X\}_{i \leq n}$, consider the system of inequations $\{f_i(x) \sqcap v_i \leq g_i(x)\}_{i \leq n}$ over the given CBL. Define functions $h'_1, h'_2 : X \to X$ as:

$$\forall y \in X : h'_1(y) := w - [\sqcup_{i \leq n} f_i^{-1}(v_i - g_i(y))]; \quad h'_2(y) := w \sqcup [\sqcup_{i \leq n} g_i^{\top}(v_i \sqcap f_i(y))]. \tag{2.7}$$

1. Suppose $f_i$ is disjunctive and $g_i$ is monotone for each $i \leq n$. Then
   (a) $h'_1$ is monotone; and the supremal solution smaller than $w$ of the given system of inequations exists and equals the supremal fixed point of the function $h'_1$.
   (b) Consider the following iterative computation:
   - $y_0 := w$,
   - $\forall k \geq 0 : y_{k+1} := h'_1(y_k)$.

   Suppose $m \in \mathcal{N}$ is such that $y_{m+1} = y_m$; then $y_m$ equals the supremal solution smaller than $w$ of the given system of inequations.
2. Suppose $f_i$ is monotone and $g_i$ is conjunctive for each $i \leq n$. Then
   (a) $h'_2$ is monotone; and the infimal solution greater than $w$ of the given system of inequations exists and equals the infimal fixed point of the function $h'_2$.
   (b) Consider the following iterative computation:
   - $y_0 := w$,
   - $\forall k \geq 0 : y_{k+1} := h'_2(y_k)$.

   Suppose $m \in \mathcal{N}$ is such that $y_{m+1} = y_m$; then $y_m$ equals the infimal solution greater than $w$ of the given system of inequations.

We next apply Theorem 2.8 and Corollary 2.2 to compute the extremal solution of a single inequation.

**Theorem 2.9** Consider a CBL $(X, \leq)$, a disjunctive and idempotent function $f : X \to X$, and fixed lattice elements $w, v \in X$.

1. The supremal solution smaller than $w$ of $f(x) \sqcap v \leq x$ equals $w - f^{-1}(v - w)$.
2. The infimal solution greater than $w$ of $f(x) \sqcap v \leq x$ equals $w \sqcup [f(w) \sqcap v]$.

**Proof:** Since $f$ is disjunctive, and identity function is monotone, it follows from the first part of Corollary 2.2 that the supremal solution smaller than $w$ of $f(x) \sqcap v \leq x$ exists. Also, the function $h_1'$ defined by (2.7) simplifies to:

$$\forall y \in X : h_1'(y) = w - f^{-1}(v - y).$$

We apply the iterative computation of Corollary 2.2 for computing the desired supremal solution. Then $y_0 = w$. This implies

$$y_1 = h_1'(y_0) = h_1'(w) = w - f^{-1}(v - w) \tag{2.8}$$
$$y_2 = h_1'(y_1) = w - f^{-1}(v - y_1) \tag{2.9}$$

We show that $y_1 = y_2$. First note that $y_1 = w - f^{-1}(v - w) \leq w = y_0$. Hence monotonicity of $h_1'$ (refer to part 1(a) of Corollary 2.2) implies that $y_2 = h_1'(y_1) \leq h_1'(y_0) = y_1$. Thus it suffices to show that $y_2 \geq y_1$. Using (2.8) we obtain

$$\begin{aligned} f^{-1}(v - y_1) &= f^{-1}(v - [w - f^{-1}(v - w)]) \\ &= f^{-1}(v \sqcap [w \sqcap [f^{-1}(v - w)]^c]^c) \\ &= f^{-1}(v \sqcap [w^c \sqcup f^{-1}(v - w)]) \\ &= f^{-1}[(v \sqcap w^c) \sqcup (v \sqcap f^{-1}(v - w)] \\ &= f^{-1}(v - w) \sqcup f^{-1}(v \sqcap f^{-1}(v - w)), \end{aligned}$$

where the last equality follows from the fact that $f^{-1}$ is disjunctive. Hence from (2.9) we obtain

$$\begin{aligned} y_2 &= [w - f^{-1}(v - w)] - f^{-1}(v \sqcap f^{-1}(v - w)) \\ &\geq [w - f^{-1}(v - w)] - f^{-1}(f^{-1}(v - w)) \\ &= w - f^{-1}(v - w) \\ &= y_1, \end{aligned}$$

where the inequality follows from the fact that $f^{-1}$ is disjunctive, so that it is also monotone, and the second equality follows from the fact that $f^{-1}$ is idempotent. Thus it follows from part 1(b) of Corollary 2.2 that $y_1 = y_2 = w - f^{-1}(v - w)$ is the supremal solution smaller than $w$ of $f(x) \sqcap v \leq x$. ■

**Example 2.7** We apply Theorem 2.9 to obtain the supremal solution of the inequation $pr(X) \cap L \subseteq X$ contained in $K$. Since the inverse of $pr$ is $ext$ it follows that the required solution is $K - ext(L - K) = K - (L - K)\Sigma^*$. ■

In Theorem 2.9, it is required that the function $f$ be idempotent. However, if this is not the case, then we can replace $f$ by its disjunctive closure $f^*$. We need the result of the following lemma:

**Lemma 2.6** Consider a CBL $(X, \leq)$, a disjunctive function $f : X \to X$, and a fixed $v \in X$. Then

$$\forall x \in X : [f(x) \sqcap v \leq x] \Leftrightarrow [f^*(x) \sqcap v \leq x].$$

**Proof:** It suffices to show the forward inequality, as the reverse inequality is obvious. Again, it suffices to show that for each $i \in \mathcal{N}$,

$$[f(x) \sqcap v \leq x] \Rightarrow [f^i(x) \sqcap v \leq x].$$

We show this using induction on $i$. If $i = 0$, then $f^i(x) \sqcap v = x \sqcap v \leq x$. Thus the base step trivially holds. Suppose for induction hypothesis that $f^i(x) \sqcap v \leq x$, i.e., $f^i(x) \leq x \sqcup v^c$. Then by applying $f$ on both sides of the last inequation and using monotonicity of $f$ we obtain:

$$f^{i+1}(x) \leq f(x \sqcup v^c) = f(x) \sqcup f(v^c) \leq (x \sqcap v^c) \sqcup (v^c \sqcup v^c) = x \sqcup v^c,$$

where the first equality follows from disjunctivity of $f$, and the second inequality follows from the hypothesis that for each $x \in X$, $f(x) \sqcap v \leq x$, i.e., $f(x) \leq x \sqcup v^c$. This establishes the induction step and completes the proof. ■

Since the disjunctive closure of any function is idempotent, and preserves disjunctivity, the following result can be obtained as a corollary of Theorem 2.9 and Lemma 2.6.

**Corollary 2.3** Consider a CBL $(X, \leq)$, a disjunctive function $f : X \to X$, and fixed lattice elements $w, v \in X$.

1. The supremal solution smaller than $w$ of $f(x) \sqcap v \leq x$ equals $w - (f^*)^{-1}(v - w)$.
2. The infimal solution greater than $w$ of $f(x) \sqcap v \leq x$ equals $w \sqcup [f^*(w) \sqcap v]$.

It is evident from Theorem 2.9 and Corollary 2.3 that the extremal solution of a single inequation of the type $f(x) \sqcap v \leq x$ can be easily obtained. We show in the following theorem that under certain conditions this can be used for computing

the extremal solutions of a system of inequations in a *modular* fashion, i.e., it is possible to first compute the extremal solution of inequations of the type $f(x) \sqcap v \leq x$ as in Theorem 2.9 and Corollary 2.3, and then compute the extremal solutions of the remaining inequations. Thus the iterative computation scheme can be considerably simplified. For simplicity of illustration we only consider a pair of inequations.

**Theorem 2.10** Given a CBL $(X, \leq)$, a fixed $w \in X$, and a pair of lattice elements $\{v_i \in X\}_{i \leq 2}$, consider a pair of inequations

$$\{f_i(x) \sqcap v_i \leq g_i(x)\}_{i \leq 2} \tag{2.10}$$

over the given CBL, where $f_1$ is disjunctive and idempotent and $g_1$ is the identity function.

1. Let $f_2$ be disjunctive and $g_2$ be monotone, and $f_1^{\perp} f_2^{\perp} = f_2^{\perp}$ (or equivalently, $f_1^{-1} f_2^{-1} = f_2^{-1}$). Consider the following iterative computation:

   - $y_0 := w - f_1^{-1}(v_1 - w)$
   - $\forall k \geq 0 : y_{k+1} := y_k - f_2^{-1}(v_2 - g_2(y_k))$

   Suppose $m \in \mathcal{N}$ is such that $y_{m+1} = y_m$, then $y_m$ is the supremal solution smaller than $w$ of $\{f_i(x) \sqcap v_i \leq g_i(x)\}_{i \leq 2}$.

2. Let $f_2$ be monotone and $g_2$ be conjunctive, and $f_1 g_2^{\top} = g_2^{\top}$. Consider the following iterative computation:

   - $y_0 := w \sqcup [f_1(w) \sqcap v_1]$
   - $\forall k \geq 0 : y_{k+1} := y_k \sqcup g_2^{\top}(v_2 \sqcap f_2(y_k))$

   Suppose $m \in \mathcal{N}$ is such that $y_{m+1} = y_m$, then $y_m$ is the infimal solution greater than $w$ of $\{f_i(x) \sqcap v_i \leq g_i(x)\}_{i \leq 2}$.

**Proof:** It follows from part 1(a) of Corollary 2.2 that the required supremal solution exists. Also, it follows from the first part of Theorem 2.9 that the supremal solution smaller than $w$ of $f_1(x) \sqcap v_1 \leq g_1(x) = x$ is $w - f^{-1}(v_1 - w)$. Hence we obtain from part 1(b) of Corollary 2.2 that $y_m \leq w$ equals the supremal solution of $f_2(x) \sqcap v_2 \leq g_2(x)$ smaller than the supremal solution of $f_1(x) \sqcap v_1 \leq g_1(x) = x$. Hence if $z \leq w$ denotes the supremal solution of (2.10), then $z \leq y_m$. Since $z \leq w$ is the supremal solution of (2.10), in order to show that $y_m \leq z$, it suffices to show that $y_m \leq w$ is a solution of (2.10).

As noted above $y_m \leq w$ is a solution of $f_2(x) \sqcap v_2 \leq g_2(x)$. We show using induction that for each $k \geq 0$, $y_k$ is a solution of $f_1(x) \sqcap v_1 \leq g_1(x) = x$, i.e.,

$$\begin{aligned}[f_1(y_k) \sqcap v_1 \leq y_k] &\Leftrightarrow [f_1(y_k) \sqcap (v_1 - y_k) = inf X] \\ &\Leftrightarrow [y_k \sqcap f_1^{-1}(v_1 - y_k) = inf X] \\ &\Leftrightarrow [f_1^{-1}(v_1 - y_k) \leq y_k^c],\end{aligned}$$

where the second equivalence follows from (2.1). Since $y_0 \leq w$ is the supremal solution of $f_1(x) \sqcap v_1 \leq x$ (refer to the first part of Theorem 2.9), it follows that the base step holds. Assume for the induction hypothesis that $f_1^{-1}(v_1 - y_k) \leq y_k^c$ for some $k \geq 0$. Then the induction step can be established as follows:

$$\begin{aligned}f_1^{-1}(v_1 - y_{k+1}) &= f_1^{-1}(v_1 - [y_k - f_2^{-1}(v_2 - g_2(y_k))]) \\ &= f_1^{-1}[(v_1 - y_k) \sqcup (v_1 \sqcap f_2^{-1}(v_2 - g_2(y_k)))] \\ &= f_1^{-1}(v_1 - y_k) \sqcup f_1^{-1}[v_1 \sqcap f_2^{-1}(v_2 - g_2(y_k))] \\ &\leq y_k^c \sqcup f_1^{-1}(f_2^{-1}(v_2 - g_2(y_k))) \\ &= y_k^c \sqcup f_2^{-1}(v_2 - g_2(y_k)) \\ &= y_{k+1}^c,\end{aligned}$$

where the first equality follows from the definition of $y_k$, the third equality follows from the fact that $f_1^{-1}$ is disjunctive, the inequality follows from induction hypothesis and the fact that $f_1^{-1}$ is monotone (as it is disjunctive), the fourth equality follows from the fact that $f_1^{\perp} f_2^{\perp} = f_2^{\perp}$, which is equivalent to $f_1^{-1} f_2^{-1} = f_2^{-1}$, and the final equality follows from the definition of $y_{k+1}$. ■

## 2.5 REMARK ON INVERSE OPERATION

We conclude this chapter by presenting a justification for using the terminology *inverse* for the operation of conjugate of dual. We present an intuitive definition of the inverse of a function defined over a power set lattice, say $(2^X, \subseteq)$, and show that the this definition coincides with one given earlier. Elements of the set $X$ are called the *atoms* of the lattice. A power set lattice possesses the additional feature that its lattice elements can be described using the atoms of the lattice, i.e.,

$$\forall Y \subseteq X : Y = \{x \in X \mid \{x\} \cap Y \neq \emptyset\}.$$

Given a function $f : 2^X \rightarrow 2^X$, we have the following intuitive definition of its inverse:

$$\forall Y \subseteq X : f^{-1}(Y) := \{x \in X \mid f(\{x\}) \cap Y \neq \emptyset\}.$$

Then we have the following proposition:

**Theorem 2.11** Consider a power set lattice $(2^X, \subseteq)$ and a function $f : 2^X \rightarrow 2^X$. If $f$ is disjunctive, then

$$\forall Y, Z \subseteq X : [f(Y) \cap Z = \emptyset] \Leftrightarrow [Y \cap f^{-1}(Z) = \emptyset]. \tag{2.11}$$

**Proof:** We begin by proving the contrapositive of (2.11). First suppose $f(Y) \cap Z \neq \emptyset$. Since $f$ is disjunctive, $f(Y) = \bigcup_{y \in Y} f(\{y\})$. Hence $f(Y) \cap Z \neq \emptyset$ implies there exists $y \in Y$ such that $f(\{y\}) \cap Z \neq \emptyset$. Consequently, $y \in f^{-1}(Z)$, i.e., $Y \cap f^{-1}(Z) \neq \emptyset$. Next suppose $Y \cap f^{-1}(Z) \neq \emptyset$. Then there exists $y \in Y$ such that $y \in f^{-1}(Z)$. This implies that $f(\{y\}) \cap Z \neq \emptyset$. Since $f$ is disjunctive, it is also monotone. Hence $f(Y) \cap Z \neq \emptyset$. ■

It follows from the first part of Theorem 2.5 that (2.11) uniquely defines the inverse of a disjunctive function. Thus the two definitions of inverse coincide. Note that the first definition of inverse is only defined for a disjunctive function over a CBL, whereas the second definition of inverse is defined for any function over a power set lattice. Therefore, the equivalence of (2.11) only holds when the function is also disjunctive.

**Example 2.8** Consider the prefix closure operation defined over the power set lattice of languages. Then it follows from above that

$$\forall H \subseteq \Sigma^* : pr^{-1}(H) = \{s \in \Sigma^* \mid pr(\{s\}) \cap H \neq \emptyset\} = ext(H),$$

as expected. ■

## 2.6 EXERCISES

1. Give an example of a binary relation which is symmetric and transitive but not reflexive.

2. Let $S$ be the set of all infinite sequences of extended reals. Thus, $x \in S$ implies that $x = x_0, x_1, ...$, where $x_i \in \mathcal{R} \cup \{-\infty, +\infty\}$. For $x, y \in S$, let $x \leq y$ if and only if $\forall i : x_i \leq y_i$. Show that $(S, \leq)$ is a complete lattice.

3. Let $(X, \leq)$ be a complete lattice and $f : X \to X$ be a function. Prove that disjunctivity, conjunctivity, sup-continuity, and inf-continuity of $f$ each imply monotonicity of $f$.

4. Consider the complete lattice $(X, \leq)$, where

$$X = \{a, b, c, d\}; \quad \leq = \{(a, b), (a, c), (b, d), (c, d), (a, d)\}.$$

Let $f : X \to X$ be defined as: $f(a) = f(b) = f(c) = d$ and $f(d) = d$.

   (a) Draw the Hasse diagram for the poset $(X, \leq)$.

   (b) Determine whether $f$ is disjunctive or conjunctive.

5. Let $(X, \leq)$ be a complete lattice and $f : X \to X$ be a function. $f$ is said to be *increasing* if $x \leq f(x), \forall x \in X$; it is said to be *decreasing* if $f(x) \leq x, \forall x \in X$.

   (a) Prove that $f^*$ is increasing and idempotent; and $f_*$ is decreasing, idempotent.

   (b) If $f$ is monotone (respectively, disjunctive), then show that $f^*$ is monotone (respectively, disjunctive).

   (c) If $f$ is monotone (respectively, conjunctive), then show that $f_*$ is monotone (respectively, conjunctive).

6. Let $(X, \leq)$ be a complete lattice and $f : X \to X$ be a monotone function. Show that the set of fixed points of $f$ is also a complete lattice.

7. Let $(X, \leq)$ be a complete lattice and $f : X \to X$ be a monotone function. Prove that $sup\{x \in X \mid f(x) = x\} = sup\{x \in X \mid x \leq f(x)\}$.

8. Let $(X, \leq)$ be a complete lattice and $f : X \to X$ be an inf-continuous function. Prove that $f_*(supX)$ is the supremal fixed point of $f$.

9. Consider the power set lattice $(2^{\Sigma^*}, \subseteq)$ of all languages defined over the event set $\Sigma$. Let $K_1, K_2 \subseteq \Sigma^*$ be two fixed languages. Define a function $f : 2^{\Sigma^*} \to 2^{\Sigma^*}$ as follows:

$$\forall H \subseteq \Sigma^* : f(H) := K_1 H + K_2.$$

Show that $f$ commutes with the supremal operation taken over a countable set. Using Theorem 2.2 compute the infimal fixed point of $f$.

10. Prove the second part of Lemma 2.1.

11. Prove the equivalence of the second and third assertions in Theorem 2.3.

12. Prove the second part of Lemma 2.2.

13. Prove Lemma 2.3.

14. Consider the power set lattice of languages defined over the event set $\Sigma$. Define the *suffix-closure* operation $suf : 2^{\Sigma^*} \rightarrow 2^{\Sigma^*}$ as:

$$\forall K \subseteq \Sigma^* : suf(K) := \{s \in \Sigma^* \mid \exists t \in \Sigma^* \text{ s.t. } ts \in K\}.$$

    (a) Show that $suf(K) = K \backslash \Sigma^*$.

    (b) Prove that $suf$ is disjunctive.

    (c) Determine the functions $suf^{\perp}$, $suf^{-1}$, and $\overline{suf}$.

15. Prove the second parts of Lemma 2.5 and Theorem 2.7.

16. Prove the second parts of Theorems 2.8 and 2.9.

17. Prove the second part of Theorem 2.10.

18. Consider the power set lattice of a set $X$, and a function $f : 2^X \rightarrow 2^X$ such that it is increasing, monotone, and idempotent. Consider the inequation $f(Y) \subseteq Y$, where $Y \subseteq X$ is the variable of the inequation.

    (a) Give an example of $X$ and $f$ to illustrate that a supremal solution of the inequation need not exist.

    (b) Prove that the infimal solution of the inequation exists.

    (c) Obtain a formula for the infimal solution larger than a given set $Z \subseteq X$.

## 2.7 BIBLIOGRAPHIC REMARKS

For a general introduction to the subject of lattice theory we refer the readers to Davey-Priestley [DP90]. One of the early results on existence of fixed points of a monotone function is due to Knaster-Tarski [Tar55]. Lassez-Nguyen-Sonenberg [LNS82] provide a nice historical account of this and other fixed point theorems. Our treatment of inequations is influenced by Dijkstra-Scholten [DS90]; in particular, the terms *conjugate* and *converse* of a function have been borrowed from there. The existence and computation of extremal solutions of a system of inequations over lattices is reported in Kumar-Garg [KG94b].

# 3

# CONTROL UNDER COMPLETE OBSERVATION

In this chapter, we present the supervisory control theory of discrete event systems under complete observation of events. The theory is applicable in any general setting of languages; however, for the termination of algorithms it is required that the languages be regular.

## 3.1 INTRODUCTION

Let $\Sigma$ denote a finite set of *events* that occur in a DES to be controlled, also called a *plant*. Let $(K_m, K)$ be the language model of the plant with $K_m, K \subseteq \Sigma^*$ and $K_m \subseteq K = pr(K) \neq \emptyset$. Using the Myhill-Nerode construction it can equivalently be represented as a deterministic state machine $G := (X, \Sigma, \alpha, x_0, X_m)$, where symbols have their usual meaning, and $(L_m(G), L(G)) = (K_m, K)$.

The notion of control in the logical setting of DES is introduced by partitioning the event set as $\Sigma = \Sigma_u \cup (\Sigma - \Sigma_u)$, the sets of *uncontrollable* and *controllable* events. A controller, also called a supervisor, restricts the behavior of the plant by dynamically disabling some of the controllable events upon execution of each event. Given a language model $(K_m, K)$ of a plant, a map $f : K \rightarrow 2^{\Sigma - \Sigma_u}$ from the generated plant behavior to the set of all subsets of controllable events defines a control law based supervisor . For each sequence $s \in K$, $f(s) \subseteq (\Sigma - \Sigma_u)$ is the set of events that are disabled by such a supervisor after execution of $s$. The interaction between a plant and a supervisor is depicted in Figure 3.1.

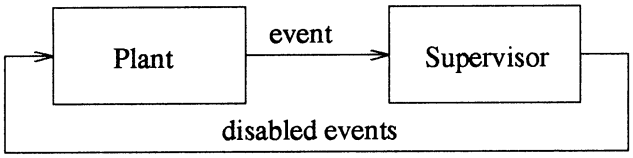



**Figure 3.1** A plant and a supervisor in closed loop control

**Definition 3.1** The generated language, denoted $K^f \subseteq K$, and the marked language, denoted $K_m^f \subseteq K^f$, of the controlled system under such a supervision is defined as:

- $\epsilon \in K^f$; $\forall s \in \Sigma^*, \sigma \in \Sigma : s \in K^f, s\sigma \in K, \sigma \notin f(s) \Rightarrow s\sigma \in K^f$,
- $K_m^f := K^f \cap K_m$.

In other words, if a string $s$ is in the controlled generated behavior (i.e., $s \in K^f$), an event $\sigma$ can occur in the plant after $s$ (i.e., $s\sigma \in K$), and $\sigma$ is not disabled by the supervisor after $s$ (i.e., $\sigma \notin f(s)$), then $s\sigma$ is in the controlled generated behavior (i.e., $s\sigma \in K^f$). The controlled marked behavior equals the set of marked strings of the plant that survive under control (i.e., $K_m^f = K^f \cap K_m$). It is clear that $(K_m^f, K^f)$ is also a language model, i.e., $K_m^f \subseteq K^f = pr(K^f) \neq \emptyset$. This implies that $pr(K_m^f) \subseteq K^f$. However, the reverse containment may not hold in general. Thus it is possible for the controlled system to execute a sequence of events (so that this sequence belongs to its generated behavior) which cannot be extended to a sequence belonging to its marked behavior. Thus the controlled system may get "blocked". A supervisor $f(\cdot)$ is said to be *non-blocking supervisor* if $pr(K_m^f) = K^f$.

It can be easily seen that a supervisor of the type described above can equivalently be represented as a state machine operating in *synchronous composition* with a state machine representation of the plant (refer to exercise problem 1). Let $S := (Y, \Sigma, \beta, y_0, Y_m)$ denote a state machine representing a supervisor, with the usual interpretation of symbols. We denote its synchronous composition with the plant state machine as $G||S := (Z, \Sigma, \gamma, z_0, Z_m)$. Since $L(G||S) = L(G) \cap L(S)$ and $L_m(G||S) = L_m(G) \cap L_m(S)$, synchronous composition restricts the behavior of the plant.

In order for the control achieved by a synchronous composition based supervisor to be equivalent to the one achieved by a control law based supervisor of the type $f : L(G) \rightarrow 2^{\Sigma - \Sigma_u}$, the following two conditions must hold.

1. An uncontrollable event should never be disabled, and
2. $L_m(G||S) = L(G||S) \cap L_m(G)$.

The first requirement follows from the fact that a control law of the type $f : L(G) \rightarrow 2^{\Sigma - \Sigma_u}$ never disables an uncontrollable event, and the second requirement follows from the definition of the marked behavior under such a control law (refer to Definition 3.1), which states that the controlled marked behavior equals the set of marked strings of the plant that survive under control. A synchronous composition based supervisor meets the first requirement if

$$\forall s \in \Sigma^*, \sigma \in \Sigma_u : s \in L(G||S), s\sigma \in L(G) \Rightarrow s\sigma \in L(G||S). \tag{3.1}$$

In other words, if the controlled system has executed a string $s$ (i.e., $s \in L(G||S)$) and an uncontrollable event $\sigma$ can occur after execution of $s$ (i.e., $s\sigma \in L(G)$), then it should also be enabled after $s$ (i.e., $s\sigma \in L(G||S)$). Moreover, since $L(G||S) \cap L_m(G) = [L(G) \cap L(S)] \cap L_m(G) = L_m(G) \cap L(S)$, a synchronous composition based supervisor meets the second requirement if

$$L_m(G) \cap L(S) = L_m(G||S). \tag{3.2}$$

**Definition 3.2** A synchronous composition based supervisor $S$ is said to be $\Sigma_u$-*enabling* if condition of (3.1) holds; it is said to be *non-marking* if condition of (3.2) holds; and it is said to be *non-blocking* if $pr(L_m(G||S)) = L(G||S)$.

Note that a sufficient condition for $S$ to be non-marking is that $L(S) = L_m(S)$.

**Remark 3.1** The control achieved by a $\Sigma_u$-enabling and non-marking supervisor $S$ is equivalent to that achieved by the control law based supervisor $f : L(G) \rightarrow 2^{\Sigma - \Sigma_u}$ defined as

$$\forall s \in L(G) : f(s) := \begin{cases} \{\sigma \in \Sigma \mid \gamma(z_0, s\sigma) \text{ not defined}\} & \text{if } \gamma(z_0, s) \text{ defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

If a control law based supervisor $f : L(G) \rightarrow 2^{\Sigma - \Sigma_u}$ exercises the same control action following any two strings which leads to the same state in the plant, then it is called a (control law based) state feedback supervisor ; formally,

$$\forall s, t \in L(G) : \alpha(x_0, s) = \alpha(x_0, t) \Rightarrow f(s) = f(t).$$

Clearly, a control law based state feedback supervisor $f : L(G) \rightarrow 2^{\Sigma - \Sigma_u}$ can be viewed as a map of the type $\bar{f} : X \rightarrow 2^{\Sigma - \Sigma_u}$, where

$$\forall x \in X : \bar{f}(x) := f(s_x),$$

where $s_x \in L(G)$ is such that $\alpha(x_0, s_x) = x$. It is easily seen that the control exercised by such a supervisor is equivalent to a synchronous composition based $\Sigma_u$-enabling and non-marking supervisor $S$, if $S$ is a sub-state machine [1] of $G$ (refer to exercise problem 2). ■

## 3.2 CENTRALIZED CONTROL

In this section we present conditions under which a certain desired behavior—given as a sublanguage of the plant language—can be achieved using a synchronous composition based supervisor, and present algorithms for verifying such conditions as well as for synthesizing corresponding supervisors. We first present the definitions of controllability and relative closure that are quite useful in this context.

**Definition 3.3** Given a plant $G$, a language $K \subseteq \Sigma^*$ is said to be *controllable* (with respect to $G$ and $\Sigma_u$) if

$$pr(K)\Sigma_u \cap L(G) \subseteq pr(K).$$

The controllability condition is equivalent to the following condition:

$$\forall s \in \Sigma^*, \sigma \in \Sigma_u : s \in pr(K), s\sigma \in L(G) \Rightarrow s\sigma \in pr(K).$$

Hence it follows from condition of (3.1) that a supervisor $S$ is $\Sigma_u$-enabling if and only if $L(G||S)$ is controllable. Note that $K$ is controllable if and only if $pr(K)$ is controllable. Figure 3.2 illustrates the controllability of $K$ with respect to $G$. It depicts that execution of an uncontrollable event $\sigma$ following a certain trace in $pr(K)$ must either result in a trace of $pr(K)$ or a trace of $\Sigma^* - L(G)$ (shown using solid edges), but not a trace of $L(G) - pr(K)$ (shown using a dashed edge).

[1] $S := (Y, \Sigma, \beta, y_0, Y_m)$ is said to be a sub-state machine of $G := (X, \Sigma, \alpha, x_0, X_m)$ if there exists an injective map $h : Y \rightarrow X$ such that

$$\forall y \in Y, \sigma \in \Sigma : \beta(y, \sigma) \text{ defined } \Rightarrow h(\beta(y, \sigma)) = \alpha(h(y), \sigma).$$

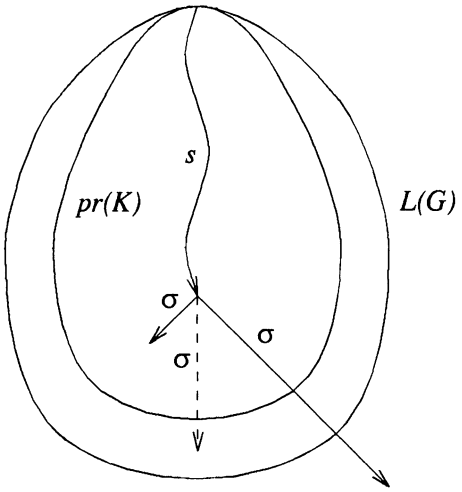



**Figure 3.2** Diagram illustrating controllability of $K$ with respect to $G$

**Definition 3.4** Given a plant $G$, a language $K \subseteq \Sigma^*$ is said to be *relative closed* (closed relative to $L_m(G)$) if

$$pr(K) \cap L_m(G) = K \cap L_m(G).$$

Note that when $K \subseteq L_m(G)$, then condition of relative closure reduces to $pr(K) \cap L_m(G) = K$.

### 3.2.1 Existence of Supervisors

The following theorem provides a condition for the existence of a supervisor so that the controlled generated behavior of a given plant equals a given desired language.

**Theorem 3.1** Let $G$ be a plant and $K \subseteq \Sigma^*$ be a desired generated behavior. There exists a $\Sigma_u$-enabling and non-marking supervisor $S$ such that $L(G||S) = K$ if and only if $\emptyset \neq K = pr(K) \subseteq L(G)$, and $K$ is controllable. In this case, $S$ can be chosen to be any DSM with $L_m(S) = L(S) = K$.

**Proof:** Consider first the necessity part. Since $L(G||S) = K$, we have $K = L(G) \cap L(S) \subseteq L(G)$. By definition, $L(G||S) \neq \emptyset$, hence $K \neq \emptyset$. Since $S$

is $\Sigma_u$-enabling, $L(G||S)$ is controllable. Also, since $L(G||S)$ is a generated language, it is prefix closed. Thus controllability and prefix closure of $K$ follows from the fact that $K = L(G||S)$.

Next consider the sufficiency part. Since $K$ is prefix closed and non-empty, using the Myhill-Nerode construction we can obtain a DSM $S$ such that $L_m(S) = L(S) = K$ (so that $S$ is non-marking). Since $K \subseteq L(G)$, this implies $L(S) \subseteq L(G)$. Hence $L(G||S) = L(G) \cap L(S) = L(S) = K$. Since $K$ is controllable, this implies $L(G||S)$ is controllable, i.e., $S$ is $\Sigma_u$-enabling. ■

**Example 3.1** Consider the plant $G$ shown in Figure 3.3(a) with $\Sigma = \{a, b\}$ and $\Sigma_u = \{b\}$. Then $L(G) = pr(a^*ba^*)$ and $L_m(G) = a^*ba^*$. Suppose the

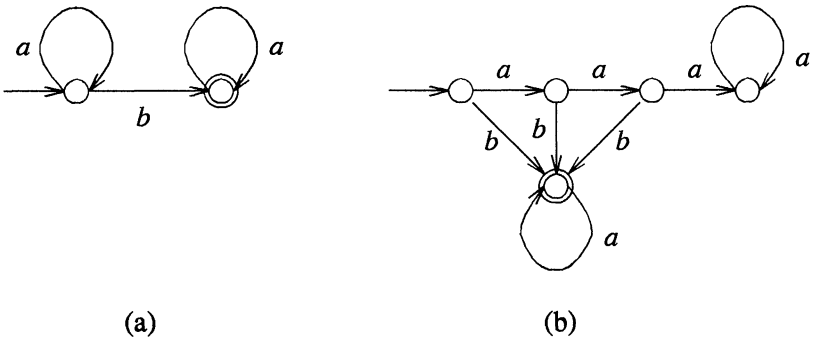



**Figure 3.3** Diagram illustrating a plant

desired generated language is given by $K = pr[\{a^kba^k, k \geq 0\}] \subseteq L(G)$, i.e., it is required that the controlled plant generate prefixes of strings with an equal number of $a$'s preceding and following a single $b$. Consider a string $s \in pr(K) = K$. If $s \in a^*$, then $sb \in K$; on the other hand, if $s \notin a^*$, then $sb \notin L(G)$. Thus there does not exist a string $s \in K$ such that $sb \in L(G) - K$, i.e., $K$ is controllable. Hence it follows from Theorem 3.1 that any DSM $S$ with $L_m(S) = L(S) = K$ is $\Sigma_u$-enabling and non-marking and yields $K$ as the generated language of the controlled plant. This supervisor simply disables the event $a$ following any string of the type $a^kba^k$.

On the other hand, suppose the desired generated language is modified to $K^n = \{a^k, k > n\} \cup pr[\{a^kba^*, k \leq n\}]$, i.e., it is required that the controlled plant block the event $b$ after at most $n$ initial occurrences of the event $a$. A DSM generator for $K^n$ is depicted in Figure 3.3(b) for the case of $n = 2$. Then it is easy to see that $K^n$ is not controllable for any $n \geq 0$ (for example $a^{n+1} \in K^n$,

and $a^{n+1}b \in L(G) - K^n$). Hence it follows from Theorem 3.1 that there does not exist any $\Sigma_u$-enabling and non-marking supervisor for which the generated behavior of the controlled plant equals $K^n$. ■

Theorem 3.1 provides a necessary and sufficient condition for the existence of a $\Sigma_u$-enabling and non-marking supervisor. However, a supervisor thus synthesized need not be non-blocking. An additional condition of $K = pr(L_m(G) \cap K)$ is needed if non-blockingness of supervisor is also desired (refer to exercise problem 3). The next theorem provides a condition for the existence of a supervisor so that the controlled marked behavior of a given plant equals a given desired language.

**Theorem 3.2** Let $G$ be a plant and $K \subseteq \Sigma^*$ be a desired marked behavior. There exists a $\Sigma_u$-enabling, non-marking and non-blocking supervisor $S$ such that $L_m(G||S) = K$ if and only if $\emptyset \neq K = pr(K) \cap L_m(G)$, and $K$ is controllable. In this case, $S$ can be chosen to be any DSM with $L_m(S) = L(S) = pr(K)$.

**Proof:** First consider the necessity part. Since $S$ is non-blocking, $L(G||S) = pr(L_m(G||S))$, which implies that $L(G||S) = pr(K)$. Since $S$ is $\Sigma_u$-enabling, this implies $pr(K)$ is controllable, i.e., $K$ is controllable. Also, since $L(G||S) \neq \emptyset$, $pr(K) \neq \emptyset$, which implies $K \neq \emptyset$. It remains to show that $pr(K) \cap L_m(G) = K$. Since $pr(K) = L(G||S)$, we have

$$pr(K) \cap L_m(G) = L(G||S) \cap L_m(G) = [L(G) \cap L(S)] \cap L_m(G) = L(S) \cap L_m(G), \tag{3.3}$$

where the last equality follows from the fact that $L_m(G) \subseteq L(G)$. Since $S$ is non-marking, $L(S) \cap L_m(S) = L_m(G||S))$. Hence it follows from (3.3) that

$$pr(K) \cap L_m(G) = L(S) \cap L_m(G) = L_m(G||S) = K,$$

where the final equality follows from the hypothesis.

Next consider the sufficiency part. Since $K \subseteq L_m(G) \subseteq L(G)$, $pr(K) \subseteq L(G)$. Also, since $pr(K)$ is non-empty (since $K$ is non-empty), prefix closed, and controllable (since $K$ is controllable), it follows from Theorem 3.1 that any DSM with $L_m(S) = L(S) = pr(K)$ is $\Sigma_u$-enabling, non-marking, and $L(G||S) = pr(K)$. Since $S$ is non-marking, $L_m(G||S) = L_m(G) \cap L(G||S) = L_m(G) \cap pr(K) = K$, where the last equality follows from the hypothesis. Since $L_m(G||S) = K$ and $L(G||S) = pr(K)$, $S$ is obviously non-blocking. ■

**Example 3.2** Consider the plant $G$ of Example 3.1, and let the desired marked language be given by $K = \{a^k b a^k, k \geq 0\} \subseteq L_m(G)$, i.e., it is required that the controlled plant only recognize those strings which have an equal number of $a$'s preceding and following $b$. Since $pr(K)$ is controllable (refer Example 3.1), it follows that $K$ is also controllable. However, it is easy to see that $K$ is not relative-closed (for example the string $ab \in pr(K) \cap L_m(G) - K$). Hence it follows from Theorem 3.2 that there does not exist any $\Sigma_u$-enabling, non-marking, and non-blocking supervisor for which the marked behavior of the controlled plant equals $K$.

On the other hand, consider a relaxed desired marked language specification given by $K' = \{a^k b a^l, k \geq l \geq 0\} \subseteq L_m(G)$, i.e., it is required that the controlled plant only recognize those strings in which the number of $a$'s preceding $b$ is not less than the number of $a$'s following $b$. Then $pr(K') = pr(K)$; so $K'$ is controllable. Consider $s \in pr(K')$. If $s \in a^*$, then $s \notin L_m(G)$. On the other hand, if $s \notin a^*$, then $s \in K'$. Thus there does not exist any string $s \in pr(K')$ such that $s \in L_m(G) - K'$, i.e., $K'$ is relative-closed. Hence it follows from Theorem 3.2 that any DSM $S$ with $L_m(S) = L(S) = pr(K')$ is $\Sigma_u$-enabling, non-marking, non-blocking, and yields $K'$ as the marked behavior of the controlled plant. ■

The following corollary combines the results of Theorems 3.1 and 3.2, the proof of which is left as an exercise (refer to exercise problem 4).

**Corollary 3.1** Let $G$ be a plant and $K_1, K_2 \subseteq \Sigma^*$ be desired marked and generated behaviors, respectively. There exists a $\Sigma_u$-enabling, non-marking and non-blocking supervisor $S$ such that $L_m(G||S) = K_1, L(G||S) = K_2$ if and only if $\emptyset \neq K_1 = pr(K_1) \cap L_m(G)$, $pr(K_1) = K_2$, and $K_2$ is controllable. In this case $S$ can be chosen to be any DSM with $L_m(S) = L(S) = K_2$.

### 3.2.2 Synthesis of Supervisors

As shown above, existence of a supervisor requires that one or more of the properties of prefix closure, relative closure, or controllability be satisfied by the desired behavior. In this section we consider the synthesis of supervisors when such conditions fail to hold. The basic idea is to consider a *maximal* sublanguage or a *minimal* superlanguage of the desired behavior that satisfies the required properties, and synthesize a supervisor, called the *minimally restrictive* or *maximally permissive* supervisor, that achieves such a behavior. Next

we show that given any language, there exists a unique maximal sublanguage of it, called the *supremal sublanguage*, and/or a unique minimal superlanguage of it, called the *infimal superlanguage*, satisfying one or more of the properties of prefix closure, relative closure, and controllability.

Consider the definition of prefix closure for a language $H \subseteq \Sigma^*$. $H$ is prefix-closed if it satisfies the inequation $pr(H) \subseteq H$ on the lattice of languages. This equation is of the form:

$$f(H) \subseteq g(H),$$

where $f$ is the prefix closure operation which is disjunctive and idempotent, and $g$ is the identity function which is conjunctive and idempotent. Hence it follows from Corollary 2.2 that the *supremal prefix closed sublanguage* as well as *infimal prefix closed superlanguage* of a given language $K \subseteq \Sigma^*$, denoted $supP(K)$ and $inf\overline{P}(K)$, respectively, exist. Furthermore it follows from Theorem 2.9 that

$$\begin{aligned} supP(K) &= K - pr^{-1}(\Sigma^* - K) = K - (\Sigma^* - K)\Sigma^*; \\ inf\overline{P}(K) &= K \cup pr(K) = pr(K). \end{aligned}$$

Next consider the definition of relative closure of a language $H \subseteq L_m(G)$. This is of the form:

$$f(H) \cap L_m(G) \subseteq g(H),$$

where $f$ is the prefix closure operation which is disjunctive and idempotent, and $g$ is the identity function which is conjunctive and idempotent. Hence it follows from Corollary 2.2 that the *supremal relative closed sublanguage* as well as *infimal relative closed superlanguage* of a given language $K \subseteq L_m(G)$, denoted $supR(K)$ and $inf\overline{R}(K)$, respectively, exist. Furthermore it follows from the first part of Theorem 2.9 that

$$supR(K) = K - pr^{-1}(L_m(G) - K) = K - (L_m(G) - K)\Sigma^*.$$

On the other hand, it follows from the second part of Theorem 2.9 that

$$inf\overline{R}(K) = K \cup (pr(K) \cap L_m(G)) = pr(K) \cap L_m(G),$$

where the second equality follows from the fact that $K \subseteq L_m(G)$.

**Example 3.3** Consider the plant $G$ and the desired marked behavior $K$ of Example 3.2, which we know is controllable but not relative closed. Then we have $L_m(G) - K = \{a^k b a^l, k \neq l\}$. Hence $(L_m(G) - K)\Sigma^* \supset \{a^k b a^l, k = l\} = K$. Consequently, it follows from above that $supR(K) = \emptyset$.

On the other hand, $pr(K) \cap L_m(G) = pr[\{a^k ba^k, k \geq 0\}] \cap \{a^* ba^*\} = \{a^k ba^l, k \geq l \geq 0\}$. Thus $inf\overline{R}(K) = K'$, where $K'$ is given in Example 3.2. ■

Next consider the definition of controllability of a language $H \subseteq L(G)$. This is of the form:

$$f(H) \cap L(G) \subseteq g(H),$$

where $f$ is the composition of the prefix closure operation and the operation of concatenation with the event set $\Sigma_u$, and $g$ is the prefix closure operation, which is monotone but not conjunctive. Since the prefix closure and concatenation operations are disjunctive, and disjunctivity is preserved under composition of functions, it follows that $f$ is disjunctive. Hence it follows from Corollary 2.2 that the supremal controllable sublanguage of a language $K \subseteq L(G)$, denoted $supC(K)$, exists. However, since $g$ is not conjunctive, existence of the infimal controllable superlanguage of $K$ cannot be concluded using the lattice theoretic arguments of Chapter 2. The following example illustrates that such a language does not exist in general.

**Example 3.4** Consider the plant $G$ of Example 3.1 with $L(G) = pr(a^* ba^*)$. Consider languages $K_1 = \{a, b, ab\}, K_2 = \{a, ba, ab\} \subseteq L(G)$. Then it is easy to see that both $K_1$ and $K_2$ are controllable. However, $K := K_1 \cap K_2 = \{a, ab\}$ is not controllable, since $\epsilon \in pr(K)$, but $b \in L(G) - pr(K)$. Thus, in general, controllability is not preserved under intersection; consequently, the infimal controllable superlanguage of a given language need not exist. ■

Since $f(\cdot) = pr(\cdot)\Sigma_u$, it follows from Figure 2.3 and Theorem 2.6 that $f^{-1}(\cdot) = [(\cdot)/\Sigma_u]\Sigma^*$. Hence the following iterative scheme of Corollary 2.2 can be used to compute $supC(K)$:

- $K_0 := K$,
- $K_{i+1} := K_i - [(L(G) - K_i)/\Sigma_u]\Sigma^*$.

If there exists $m \in \mathcal{N}$ such that $K_{m+1} = K_m$, then $K_m = supC(K)$. It can be shown that such an $m$ exists whenever $K$ and $L(G)$ are regular languages (refer to exercise problem 8).

**Example 3.5** Consider the plant $G$ and the desired behavior $K^n \subseteq L(G)$ of Example 3.1, which we know is not controllable. We apply the above iterative

scheme for computing $supC(K^n)$. We have $K_0 = K^n$. In order to compute $K_1$ we first note that $L(G) - K_0 = L(G) - K^n = \{a^k ba^*, k > n\}$. Hence $(L(G) - K_0)/\Sigma_u = \{a^k ba^*, k > n\}/\{b\} = \{a^k b, k > n\}/\{b\} = \{a^k, k > n\}$. So

$$K_1 = K_0 - [(L(G) - K_0)/\Sigma_u]\Sigma^* = K^n - \{a^k, k > n\}\Sigma^* = pr[\{a^k ba^*, k \leq n\}].$$

Next in order to compute $K_2$ we note that $L(G) - K_1 = \{a^k ba^*, k > n\} \cup \{a^k, k > n\}$. So $(L(G) - K_1)/\Sigma_u = \{a^k, k > n\}$. Consequently, $K_2 = K_1$, which implies that $supC(K^n) = K_1$. ■

If we require that the extremal language be controllable as well as prefix closed, then we must consider the extremal solution of the following two inequations:

$$pr(H)\Sigma_u \cap L(G) \subseteq pr(H); \quad pr(H) \subseteq H, \tag{3.4}$$

where $H \subseteq L(G)$ is the variable of inequation. Using the fact that $H \subseteq L(G)$, we show that the two inequations of (3.4) are equivalent to the following single inequation:

$$pr(H)\Sigma_u \cap L(G) \subseteq H. \tag{3.5}$$

It is clear that (3.4) implies (3.5). Also, since $H \subseteq pr(H)$, the first inequation of (3.4) follows from (3.5). It remains to show that (3.5) implies $pr(H) \subseteq H$. First note that if $f(\cdot) = pr(\cdot)\Sigma_u$, then $f^*(\cdot) = pr(\cdot)\Sigma_u^*$. Hence it follows from Lemma 2.6 that (3.5) is equivalent to

$$pr(H)\Sigma_u^* \cap L(G) \subseteq H. \tag{3.6}$$

Since $pr(H) \subseteq pr(H)\Sigma_u^*$ and $pr(H) \subseteq L(G)$ (as $H \subseteq L(G)$), it follows from (3.6) that $pr(H) \subseteq pr(H)\Sigma_u^* \cap L(G) \subseteq H$, as desired.

It follows from the above discussions that (3.5), or equivalently (3.6), can be used to compute the extremal prefix closed and controllable language. We consider (3.6), which is of the form

$$f(H) \cap L(G) \subseteq g(H),$$

where $f(\cdot) = pr(\cdot)\Sigma_u^*$, which is disjunctive as well as idempotent, and $g$ is the identity function. Consequently, it follows from Corollary 2.2 that the *supremal prefix closed and controllable sublanguage* and *infimal prefix closed and controllable superlanguage* of $K \subseteq L(G)$, denoted $supPC(K)$ and $infPC(K)$, respectively, exist. Furthermore, it follows from Theorem 2.9 that

$$\begin{aligned} supPC(K) &= K - f^{-1}(L(G) - K) = K - [(L(G) - K)/\Sigma_u^*]\Sigma^* \\ infPC(K) &= K \cup (f(K) \cap L(G)) = K \cup (pr(K)\Sigma_u^* \cap L(G)) \\ &= pr(K)\Sigma_u^* \cap L(G). \end{aligned}$$

**Example 3.6** Consider the plant $G$ and the desired behavior $K^n$ of Example 3.1. Then $L(G) - K^n = \{a^k ba^*, k > n\}$. Hence $(L(G) - K^n)/\Sigma_u^* = \{a^k ba^*, k > n\}/b^* = \{a^k ba^*, k > n\} \cup \{a^k, k > n\}$. This implies

$$supPC(K^n) = K^n - [\{a^k ba^*, k > n\}\Sigma^* \cup \{a^k, k > n\}\Sigma^*] = pr[\{a^k ba^*, k \leq n\}].$$

Also, since $pr(K^n)\Sigma_u^* = pr[\{a^k ba^*, k \leq n\}]b^* \cup \{a^k, k > n\}b^*$, it follows that

$$inf\overline{PC}(K^n) = pr(K^n)\Sigma_u^* \cap L(G) = pr[\{a^k ba^*, k \leq n\}] \cup pr[\{a^k b, k > n\}].$$

■

We can also compute extremal languages that are relative closed (rather than prefix closed) and controllable. In this case we must consider the extremal solutions of the following two inequations:

$$pr(H) \cap L_m(G) \subseteq H; \quad pr(H)\Sigma_u \cap L(G) \subseteq pr(H),$$

where $H \subseteq L(G)$ is the variable of inequations. It can be argued as above that the *supremal relative closed and controllable sublanguage* of $K \subseteq L_m(G)$, denoted $supRC(K)$, exists; however, the infimal relative closed and controllable sublanguage need not exist. Moreover, if we define

$$f_1(\cdot) := pr(\cdot); \quad f_2(\cdot) := pr(\cdot)\Sigma_u,$$

then

$$f_1^{-1}(f_2^{-1}(\cdot)) = f_1^{-1}[((\cdot)/\Sigma_u)\Sigma^*] = [((\cdot)/\Sigma_u)\Sigma^*]\Sigma^* = ((\cdot)/\Sigma_u)\Sigma^* = f_2^{-1}(\cdot).$$

It follows from the first part of Theorem 2.10 that it is possible to modularly compute $supRC(K)$, i.e.,

$$supRC(K) = supC(supR(K)).$$

(Similar arguments can be used to show that $supPC(K) = supC(supP(K))$.) The following iterative computation of Theorem 2.10 can be used for computing $supRC(K)$:

- $K_0 := K - (L_m(G) - K)\Sigma^*$,
- $K_{i+1} := K_i - [(L(G) - K_i)/\Sigma_u]\Sigma^*$.

If $m \in \mathcal{N}$ is such that $K_{m+1} = K_m$, then $K_m = supRC(K)$. It can be shown that such an $m$ exists whenever $K$ and $L(G)$ are regular (refer to exercise problem 9).

### 3.2.3 Computation of Supervisors

It follows from the above discussion that to determine whether there exists a supervisor for achieving a desired behavior constraint, one must verify properties such as prefix closure, controllability and relative closure. Further, synthesis of minimally restrictive supervisors requires computation of supremal languages satisfying one or more of these properties. In this section we present some algorithms for verifying the above properties as well as for computing supremal languages satisfying the properties. In doing so, we will assume that the languages involved are regular so that they can be generated and marked by finite state machines. This is a reasonable assumption since most discrete event systems have finitely many states. Moreover, as is shown below, algorithms do not exist in a general setting. This follows from a theorem due to Rice, which deals with non-trivial properties of recursively enumerable (r.e.) languages, i.e., languages marked by Turing machines. (A property of r.e. languages is said to be *trivial* if and only if it holds for either all such languages or no such language. A property that is not trivial is called *non-trivial.*)

**Theorem 3.3** Any non-trivial property of r.e. languages is undecidable.

Clearly, prefix closure, controllability, relative closure, etc., are non-trivial properties. Consequently, from Theorem 3.3, there exists no algorithm for deciding any of these properties for r.e. languages.

As above, let $G := (X, \Sigma, \alpha, x_0, X_m)$ denote a *finite* state machine representing a plant; so that $L_m(G)$ and $L(G)$ are both regular. Without loss of generality, we assume that $G$ is accessible. Let $K \subseteq \Sigma^*$ denote a desired behavior; it is assumed that $K$ is regular. We use $S := (Y, \Sigma, \beta, y_0, Y_m)$ to denote a finite state machine recognizer of $K$, i.e., $L_m(S) = K$. Without loss of generality, we assume that $S$ is *trim*; consequently, $L(S) = pr(L_m(S)) = pr(K)$. We use $m, n \in \mathcal{N}$ to denote the number of states in $G$ and $S$ respectively, and ignore the dependence of algorithmic complexity on any parameter other than $m$ and $n$.

It can be easily seen that for any $K \subseteq \Sigma^*$,

$$K = pr(K) \Leftrightarrow Re_S(y_0) = Y_m. \tag{3.7}$$

Since the reachability of any state of $S$ can be determined in $O(n)$ time, so can be the prefix closure of $K$.

In order to test whether $K \subseteq L_m(G)$ is relative closed, one must verify whether $pr(K) \cap L_m(G) \subseteq K$. For this we consider the finite state machine $L(G||S) := (Z, \Sigma, \gamma, z_0, Z_m)$. Then

$$pr(K) \cap L_m(G) \subseteq K \Leftrightarrow X_m \times Y \subseteq X_m \times Y_m. \tag{3.8}$$

Since number of states in $G||S$ is $mn$, relative closure of $K$ can be determined in $O(mn)$ time.

In order to verify whether $K \subseteq L(G)$ is controllable, we must verify whether $pr(K)\Sigma_u \cap L(G) \subseteq K$. For this we consider the finite state machine $G||\bar{S} := (\bar{Z}, \Sigma, \bar{\gamma}, z_0, Z_m)$ where $\bar{S}$ is the completion of $S$ with a dump state. Then

$$\begin{aligned} &pr(K)\Sigma_u \cap L(G) \subseteq pr(K) \Leftrightarrow \\ &\forall (x,y) \in Z, \sigma \in \Sigma_u : \bar{\gamma}((x,y),\sigma) \text{ defined } \Rightarrow \bar{\gamma}((x,y),\sigma) \notin X \times \{y_D\} \end{aligned} \tag{3.9}$$

Thus controllability of $K$ can also be determined in $O(mn)$ time. The proof of the above assertions is left as an exercise (refer to exercise problem 10).

**Example 3.7** Consider the plant $G$ and desired generated language $K^n$ of Example 3.1 with $n = 2$. Then a generator $S$ for this case is shown in Figure 3.3(b). The DFSM $G||\bar{S}$ obtained by synchronous composition of $G$ with the completion of $S$ is shown in Figure 3.4. The only state with second coordinate

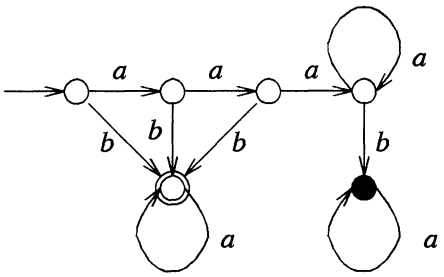



**Figure 3.4** Diagram illustrating test for controllability

$y_D$ is painted dark, which is the only state in $\bar{Z} - Z$. The "non-dark" states are in $Z$. Since there exists a transition on the uncontrollable event $b$ from a non-dark state to a dark state, we conclude that $K^n$ is not controllable. ■

Next we present an algorithm for computing the supremal controllable sublanguage $supC(K)$ of any regular language $K \subseteq L(G)$. Recall from (3.6) that

$K \subseteq L(G)$ is controllable if and only if for each $s \in pr(K)$ and $u \in \Sigma_u^*$ such that $su \in L(G)$, we have $su \in pr(K)$. Thus the supremal controllable sublanguage of $K$ can be obtained by removing those strings from $K$ (and any of their extensions in $K$) for which there exists a prefix $s$ and $u \in \Sigma_u^*$ such that $su \in L(G) - pr(K)$. Such a string $s \in pr(K)$ is called an *uncontrollable prefix* of $K$. The following lemma shows that if $s \in pr(K)$ is an uncontrollable prefix of $K$, then any other string, execution of which leads to the same state as that reached by execution of $s$ in $G||\bar{S}$, is also an uncontrollable prefix of $K$.

**Lemma 3.1** $s \in pr(K)$ is an uncontrollable prefix of $K$ if and only if all strings in $[s](R_{G||\bar{S}})$ are uncontrollable prefixes of $K$.

**Proof:** We only need to prove the necessity part. Suppose $s \in pr(K)$ is an uncontrollable prefix of $K$. Then there exists $u \in \Sigma_u^*$ such that $su \in L(G) - pr(K)$, i.e, $\bar{\gamma}(\bar{\gamma}(z_0, s), u) \in X \times \{y_D\}$. Pick $t \in [s](R_{G||\bar{S}})$. Then we have $\bar{\gamma}(z_0, t) = \bar{\gamma}(z_0, s) \in Z$, i.e., $t \in pr(K)$. Also, the last equality implies that $\bar{\gamma}(\bar{\gamma}(z_0, t), u) \in X \times \{y_D\}$, i.e., $tu \in L(G) - pr(K)$, which proves that $t$ is an uncontrollable prefix of $K$. ■

Lemma 3.1 implies that in order to remove the set of strings with uncontrollable prefixes (and any of their extensions) from $K$, it suffices to remove the states reached by their execution in $G||\bar{S}$. Thus we have the following algorithm, in which the notation $(G||\bar{S})_l$ is used to denote the state machine obtained by removing the "bad" states (and transitions leading into them) at the $l$th iteration:

**Algorithm 3.1**

1. Initiation step:
$$Z_0 := \{(x, y_D) \mid x \in X\}; \quad k := 0.$$
2. Iteration step:
   (a) $Z_0' := Z_k; \quad l := 0.$
   (b) $Z_{l+1}' := Z_l' \cup \{z \in Z - Z_l' \mid \exists \sigma \in \Sigma_u : \bar{\gamma}(z, \sigma) \in Z_l'\}.$
   (c) If $Z_{l+1}' = Z_l'$, then go to step (d); else, set $l := l + 1$, and go to step (b).
   (d) $Z_{k+1} := Z_l' \cup \{z \in Z - Z_l' \mid z \notin Re_{(G||\bar{S})_l^R}(Z_m - Z_l')\}.$

3. Termination step:
   If $Z_{k+1} = Z_k$, the stop; else, set $k := k + 1$, and go to step 2.

Note that in the initiation step, states in $Z_0$ correspond to the strings in $L(G) - pr(K)$, i.e., $z \in Z_0$ if and only if there exists $s \in L(G) - pr(K)$ such that $\bar{\gamma}(z_0, s) = z$. In the iteration step, steps (a)-(c) identify those states in $z \in Z$ (note that these states correspond to strings in $pr(K)$), for which there exists $u \in \Sigma_u^*$ such that $\bar{\gamma}(z, u) \in Z_0$, i.e., steps (a)-(c) identify all the uncontrollable prefixes of $K$. Step (d) of the iteration step computes all the states that are no longer co-accessible. The algorithm terminates if the set of such states is empty; otherwise, it repeats the iteration step. It can be verified that Algorithm 3.1 terminates in $O(m^2n^2)$ steps. Moreover, if $K$ is prefix closed, then it terminates in a single iteration, in which case it takes $O(mn)$ steps to terminate. (Refer to exercise problem 11.)

**Example 3.8** Consider the plant $G$ and the desired generated language $K^n$ with $n = 2$ of Example 3.1, which we know is not controllable. Algorithm 3.1 can be applied to compute $supC(K^n)$ as follows. First the DFSM $G||\bar{S}$ is constructed which is depicted in Figure 3.5(a). The initiation step of the algorithm identifies the states in $Z_0$ (the states with second coordinate $y_D$). The only such state is painted dark in Figure 3.5(a). Next the states in $Z_1' - Z_0'$ are computed; in Figure 3.5 these are the non-dark states from where a dark state is reachable by execution of an uncontrollable event. Once such states are identified, the states in $Z_1'$ are pained dark as shown in Figure 3.5(b). Since $Z_1' \neq Z_0'$, $Z_2' - Z_1'$ is computed. In Figure 3.5(b) these are the non-dark states from where a dark state is reachable by execution of an uncontrollable event. Clearly, $Z_2' - Z_1'$ is empty, which implies $Z_2' = Z_1'$. Next $Z_1 - Z_1'$ is computed; in Figure 3.5(b) these are the non-dark states that are not co-accessible. Since each non-dark state in Figure 3.5(b) is co-accessible, $Z_1 - Z_1'$ is empty, which implies $Z_1 = Z_1'$. This completes one iteration of the algorithm. Since $Z_1 \neq Z_0$, $Z_2$ is computed in the next iteration in a similar manner. It is readily verified that $Z_2 = Z_1$, so the algorithm terminates, yielding the DFSM of Figure 3.5(c) as the generator for $supC(K^n)$. ■

The following proposition illustrates that when $K$ is prefix closed, then Algorithm 3.1 is optimal in its order of computation.

**Theorem 3.4** Given a minimal state machine $G$ with $m$ states, and a minimal state machine $S$ with $n$ states generating $K$, i.e., $L(S) = K$, any algorithm for computing $supC(K) = supPC(K)$ takes $O(mn)$ time.

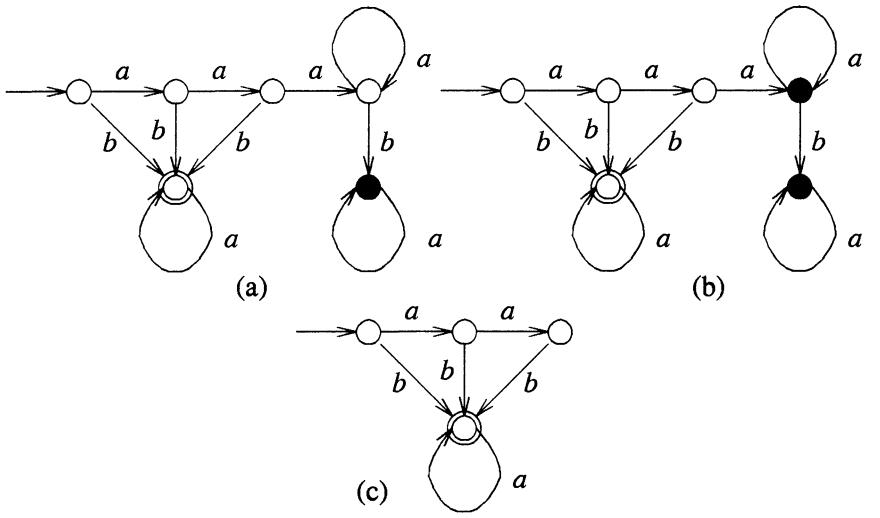



**Figure 3.5** Diagram illustrating supremal controllable language computation

**Proof:** We provide an example of languages $L(G)$ and $K$ with $m$ and $n$ states respectively, in their minimal finite state machine generators with the property that the generator for $supC(K) = supPC(K)$ has $O(mn)$ states; which shows that any algorithm for computing the generator for $supC(K)$ must take at least $O(mn)$ time.

Let $\Sigma = \{a, b, c, d, e\}$ and $\Sigma_u = \{e\}$. Let $L(G)$ and $K$ be as given below:

$$\begin{aligned} L(G) &= \{s \in \Sigma^* \mid 0 \leq [\#(a,s) - \#(b,s)] \leq (m-1); \\ &\quad \forall te \leq s : 0 \leq [\#(a,t) - \#(b,t)] \leq (m-2)\}, \\ K &= \{s \in \Sigma^* \mid 0 \leq [\#(c,s) - \#(d,s)] \leq (n-1); \\ &\quad \forall te \leq s : 0 \leq [\#(c,t) - \#(d,t)] \leq (n-2)\}; \end{aligned}$$

where the notation $\#(\sigma, s)$ denotes the number of events $\sigma$ in the string $s$. $K$ is not controllable with respect to $G$, for the string $c^{n-1} \in K \cap (L(G) - K)/\{e\}$. The following language is the supremal controllable sublanguage of $K$:

$$\begin{aligned} supPC(K) &= \{s \in \Sigma^* \mid 0 \leq [\#(a,s) - \#(b,s)] \leq (m-2); \\ &\quad 0 \leq [\#(c,s) - \#(d,s)] \leq (n-2)\} \\ &\quad \cup \{s \in (\Sigma - \{e\})^* \mid 0 \leq [\#(a,s) - \#(b,s)] = (m-1); \\ &\quad 0 \leq [\#(c,s) - \#(d,s) \leq](n-1)\}. \end{aligned}$$

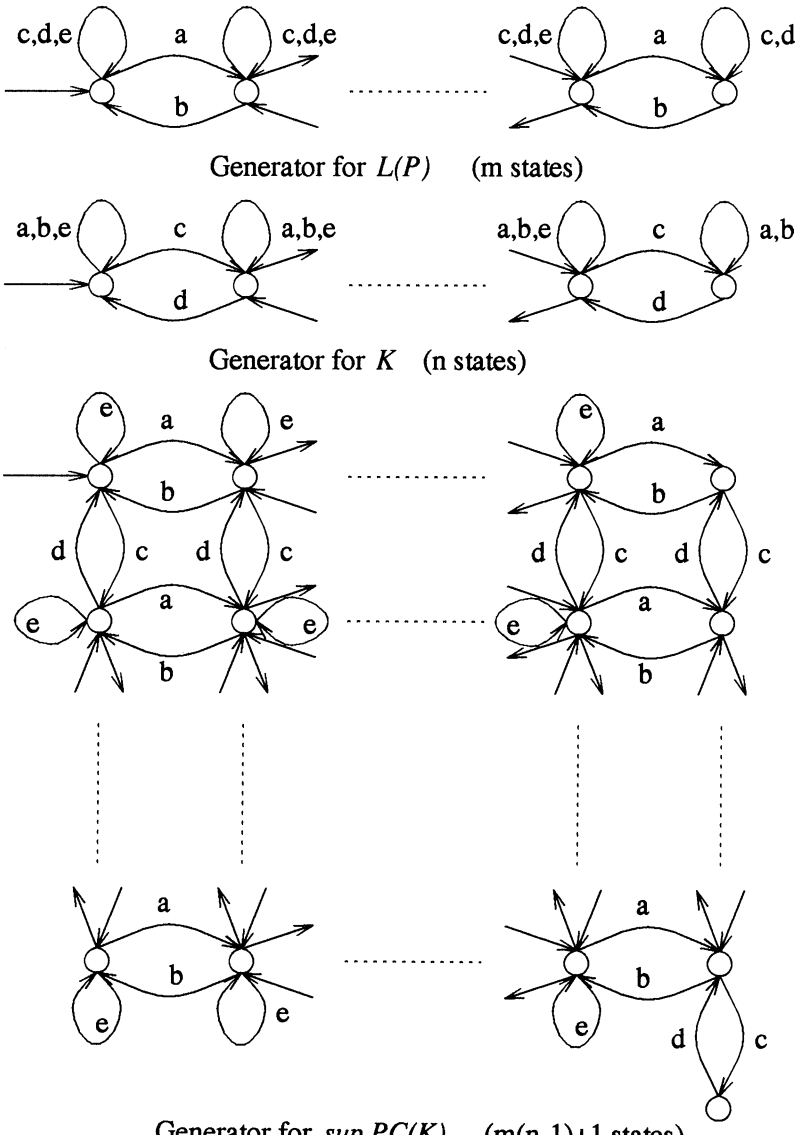



**Figure 3.6** Diagram illustrating optimality of the algorithm for $supPC(K)$

The generators of languages $L(G)$, $K$ and $supPC(K)$ are shown in Figure 3.6.

In the above example $L(G)$ and $K$ have $m$ and $n$ states respectively, in their minimal finite state machine generators, whereas $supC(K)$ has $m(n-1)+1$ states, i.e., $O(mn)$ states, in its minimal finite state machine generator; thus any algorithm that computes $supC(K)$ must take $O(mn)$ time. ■

## 3.3 MODULAR CONTROL

In the above discussion, the desired behavior was expressed as a single sublanguage of the plant language. Often, the desired behavior is expressed as intersection of two or more languages. In other words, a constraint is often described as: "The system should satisfy a property of one kind, *as well as* a property of another kind". One can either construct a "central" supervisor that achieves the intersection of the given languages as the desired behavior; or one can construct separate "modular" supervisors, and combine these supervisors in such a way that intersection of languages is achieved as the controlled system behavior. The second approach, whenever feasible, results in a computationally more efficient design.

The approach of modular synthesis utilizes the property that controllability of modular languages is preserved under intersection.

**Definition 3.5** $K_1, K_2 \subseteq \Sigma^*$ are said to be *modular* if $pr(K_1 \cap K_2) = pr(K_1) \cap pr(K_2)$.

Note that the modularity of $K_1$ and $K_2$ is equivalent to $pr(K_1) \cap pr(K_2) \subseteq pr(K_1 \cap K_2)$, as the reverse containment always holds. Thus modularity of two given languages requires that whenever they share a prefix, then they also share a string containing that prefix. The proof of the following lemma is left as an exercise (refer to exercise problem 13).

**Lemma 3.2** Let $K_1, K_2 \subseteq \Sigma^*$ be modular and controllable. Then $K_1 \cap K_2$ is controllable.

Since a pair of closed languages is always modular, Lemma 3.2 implies that controllability of prefix closed language is preserved under intersection.

The next theorem states that modular synthesis is always possible for prefix closed languages.

**Theorem 3.5** Let $G$ be a plant and $K_1, K_2 \subseteq \Sigma^*$. Suppose $\Sigma_u$-enabling and non-marking supervisors $S_1, S_2$ are such that $L(G||S_1) = K_1$ and $L(G||S_2) = K_2$. Then $S := S_1||S_2$ is $\Sigma_u$-enabling and non-marking; and $L(G||S) = K_1 \cap K_2$.

**Proof:** Since $S_1$ and $S_2$ are non-marking, we have $L_m(G||S_1) = L(G) \cap L_m(S_1)$ and $L_m(G||S_2) = L(G) \cap L(S_2)$. Hence $L_m(G||S) = L_m(G||S_1) \cap L_m(G||S_2) = [L(G) \cap L_m(S_1)] \cap [L(G) \cap L_m(S_2)] = L(G) \cap L_m(S)$, which shows that $S$ is non-marking. In order to prove that $S$ is $\Sigma_u$-enabling, we show that

$$L(G||S) = L(G) \cap L(S_1) \cap L(S_2) = L(G||S_1) \cap L(G||S_2) \tag{3.10}$$

is controllable. This follows from the facts that $L(G||S_1)$ and $L(G||S_2)$ are prefix closed and controllable (since $S_1$ and $S_2$ are $\Sigma_u$-enabling), and controllability of prefix closed languages is preserved under intersection (Lemma 3.2). Also, it follows from (3.10) that $L(G||S) = K_1 \cap K_2$. ■

**Example 3.9** Consider a system consisting of two machines $M_1$ and $M_2$. A DFSM model for machine $M_i$ $(i = 1, 2)$ is shown in Figure 3.7(a). Machine $M_i$ is initially in the idle state, where it inputs a part (event $a_i$) and goes to the working state. In this state, it either completes processing the part (event $b_i$) and returns to idle state, or breaks down (event $c_i$) and goes to the broken state. In the broken state machine may be repaired (event $d_i$) returning it to the working state. Machines $M_1$ and $M_2$ operate in tandem, and there is buffer $B$ of unit capacity between them as shown in Figure 3.7(b). So when $M_1$ completes processing the part, it puts it in the buffer, which is then processed by $M_2$. The uncontrolled plant is given by $G = M_1||M_2$.

It is required that (1) the buffer must never overflow or underflow; and (2) $M_1$ has priority of repair over $M_2$. Then it is easy to check that the desired behavior specifications $K_1$ and $K_2$ corresponding to constraints (1) and (2), respectively, are prefix closed and hence modular. Thus, if $K_1$ and $K_2$ are individually controllable, then the desired specification $K_1 \cap K_2$ is controllable. The remaining details of the problem are left as an exercise (refer to exercise problem 15). ■

In Theorem 3.5, $S := S_1||S_2$ serves as the modular supervisor. Note that Theorem 3.5 requires that $K_1$ and $K_2$ be prefix closed and controllable. Otherwise,

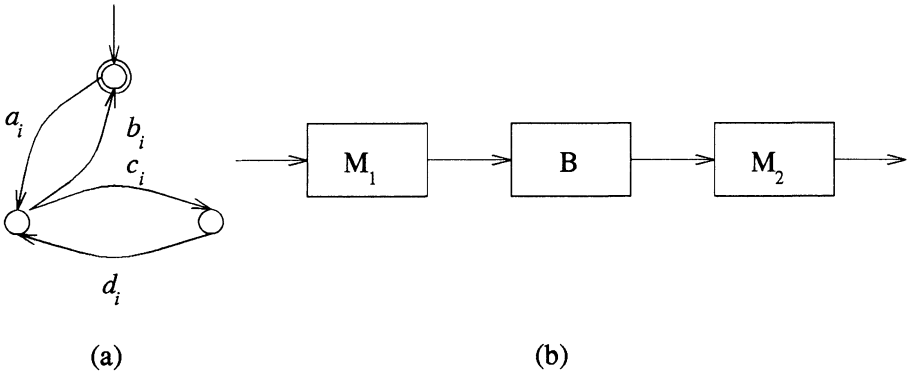



**Figure 3.7** Diagram illustrating modular supervision

one must consider $supPC(K_1)$ and $supPC(K_2)$. The following proposition states that such a design does not result in any loss of performance.

**Theorem 3.6** Let $K_1, K_2 \subseteq L(G)$. Then $supPC(K_1 \cap K_2) = supPC(K_1) \cap supPC(K_2)$.

**Proof:** For notational simplicity, let $K := K_1 \cap K_2$. Since $K \subseteq K_1$, we have $supPC(K) \subseteq supPC(K_1)$; similarly, $supPC(K) \subseteq supPC(K_2)$. Consequently, $supPC(K) \subseteq supPC(K_1) \cap supPC(K_2)$. We need to show the reverse containment. By definition, $supPC(K_1) \subseteq K_1$ and $supPC(K_2) \subseteq K_2$; hence $supPC(K_1) \cap supPC(K_2) \subseteq K$. This together with Lemma 3.2 implies that $supPC(K_1) \cap supPC(K_2)$ is prefix closed and controllable sublanguage of $K$, which implies that $supPC(K_1) \cap supPC(K_2) \subseteq supPC(K)$. ■

The following theorem states that modular synthesis is possible for non-prefix closed languages under the condition of modularity.

**Theorem 3.7** Let $G$ be a plant and $K_1, K_2 \subseteq \Sigma^*$. Suppose $\Sigma_u$-enabling, non-marking, non-blocking supervisors $S_1, S_2$ are such that $L_m(G||S_1) = K_1$ and $L_m(G||S_2) = K_2$. Then $S := S_1||S_2$ is $\Sigma_u$-enabling and non-marking; and $L_m(G||S) = K_1 \cap K_2$. Furthermore, $S$ is non-blocking if and only if $K_1$ and $K_2$ are modular.

**Proof:** Since $S_1$ and $S_2$ are non-blocking, $L(G||S_1) = pr(K_1)$ and $L(G||S_2) = pr(K_2)$. Hence $L(G||S_1) = pr(K_1) \cap pr(K_2)$. On the other hand, $L_m(G||S) =$

$L_m(G||S_1) \cap L_m(G||S_2) = K_1 \cap K_2$. These imply that $S$ is non-blocking, i.e., $pr(L_m(G||S)) = L(G||S)$, if and only if $K_1$ and $K_2$ are modular. That $S$ is $\Sigma_u$-enabling and non-marking follows from Theorem 3.5. ■

In Theorem 3.7, the existence of $\Sigma_u$-enabling, non-marking and non-blocking supervisors $S_1, S_2$ such that $L(G||S_1) = K_1$ and $L(G||S_2) = K_2$ requires that $K_1$ and $K_2$ be relative closed and controllable. Otherwise, one must consider $supRC(K_1)$ and $supRC(K_2)$. The following proposition states that such a design does not result in any loss of performance if $supRC(K_1)$ and $supRC(K_2)$ are modular, the proof of which is left as an exercise (refer to exercise problem 13).

**Theorem 3.8** Let $K_1, K_2 \subseteq L_m(G)$ be such that $supRC(K_1), supRC(K_2)$ are modular. Then

$$supRC(K_1 \cap K_2) = supRC(K_1) \cap supRC(K_2).$$

## 3.4 EXERCISES

1. Given a control law $f : L(G) \rightarrow 2^{\Sigma - \Sigma_u}$, define the following equivalence relation on $L(G)$:

   $$\forall s, t \in L(G) : s \cong t \Leftrightarrow [f(s) = f(t)] \wedge [\forall u \in \Sigma^* : su \in L(G) \Leftrightarrow tu \in L(G)].$$

   Let $[s]$ denote the equivalence class containing the string $s$. Define a state machine $S := (Y, \Sigma, \beta, y_0, Y_m)$, where $Y = Y_m := \{[s] \mid s \in L(G)\}$, $y_0 := [\epsilon]$, and

   $$\forall [s] \in Y, \sigma \in \Sigma : \beta([s], \sigma) := \begin{cases} [s\sigma] & \text{if } s\sigma \in L(G) \text{ and } \sigma \notin f(s) \\ \text{undefined} & \text{otherwise} \end{cases}$$

   Show that $(L(G))^f = L(G||S)$ and $(L_m(G))^f = L_m(G||S)$. Also, show that $S$ is $\Sigma_u$-enabling and non-marking.

2. Let $G$ be a plant.

   (a) Let $f : L(G) \rightarrow 2^{\Sigma - \Sigma_u}$ be a control law based state feedback supervisor. Show that there exists a $\Sigma_u$-enabling and non-marking supervisor $S$ such that $L(G||S) = (L(G))^f$, $L_m(G||S) = ((L_m(G))^f$ and $S$ is a sub-automaton of $G$.

(b) Let $S$ be a $\Sigma_u$-enabling, non-marking synchronous composition based supervisor such that $S$ is a sub-automaton of $G$. Show that there exists a control law based state feedback supervisor $f : L(G) \rightarrow 2^{\Sigma - \Sigma_u}$ such that $(L(G))^f = L(G||S)$ and $(L_m(G))^f = L_m(G||S)$.

3. Given $K \subseteq \Sigma^*$, prove that there exists a $\Sigma_u$-enabling, non-marking and non-blocking supervisor $S$ such that $L(G||S) = K$ if and only if $\emptyset \neq K = pr(K) = pr(L_m(G) \cap K)$ and $K$ is controllable.

4. Let $G$ be a plant and $K_1, K_2 \subseteq \Sigma^*$ be the desired marked and generated behavior respectively. Prove that there exists a $\Sigma_u$-enabling, non-marking, non-blocking supervisor such that $L_m(G||S) = K_1, L(G||S) = K_2$ if and only if $pr(K_1) = K_2 \neq \emptyset$, $K_1$ is relative closed, and $K_2$ is controllable.

5. Prove that if $H \subseteq K = pr(K) \subseteq L = pr(L) \subseteq \Sigma^*$ are such that $H$ is controllable with respect to $K$ and $\Sigma_u$ and $K$ is controllable with respect to $L$ and $\Sigma_u$, then $H$ is controllable with respect to $L$ and $\Sigma_u$.

6. Prove that $PC(K)$ is a complete lattice where $PC(K)$ is the set of all prefix-closed and controllable sublanguages of $K$.

7. Given a language $K \subseteq L(G)$, prove that $supPC(K) = supPC(supP(K))$; if $K = pr(K)$, then $supC(K) = supPC(K)$.

8. Given $K \subseteq L(G)$, define the following sequence of languages:

   - $K_0 := K$
   - $\forall i \geq 0 : K_{i+1} := K_i - [(L(G) - K_i))/\Sigma_u]\Sigma^*$

   Suppose $K$ and $L(G)$ are regular. Show that there exists $m \geq 0$ such that $K_m = K_{m+1}$.

9. Given $K \subseteq L_m(G) \subseteq L(G)$, define the following sequence of languages:

   - $K_0 := K - (L_m(G) - K)\Sigma^*$
   - $\forall i \geq 0 : K_{i+1} := K_i - [(L(G) - K_i))/\Sigma_u]\Sigma^*$

   Suppose $K, L_m(G)$ and $L(G)$ are regular. Show that there exists $m \geq 0$ such that $K_m = K_{m+1}$.

10. Given $K \subseteq \Sigma^*$, prove assertions (3.7)-(3.8).

11. Consider Algorithm 3.1 that computes $supC(K)$ for a language $K \subseteq L(G)$.

    (a) Prove the correctness of Algorithm 3.1.

    (b) Prove that it terminates in $O(m^2n^2)$ steps, where $m, n$ are number of states in $G$ and the DFSM with marked language $K$.

(c) Prove that when $K$ is prefixed closed, then it terminates in $O(mn)$ steps.

12. A cat and a mouse are placed in the maze of Figure 3.8. Each door in the

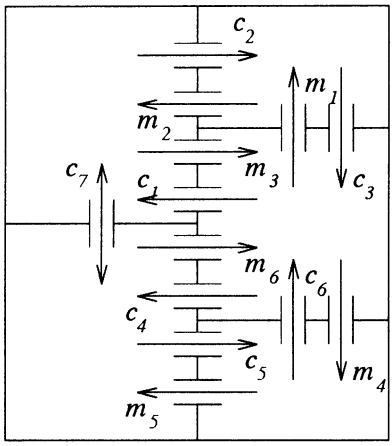



**Figure 3.8** Maze for cat and mouse

maze is exclusively for either the cat or the mouse and can be used only in the direction indicated. Initially the cat is in room 2 and the mouse in room 4.

(a) Draw DFSMs $C$ and $M$ describing the behavior of the cat and the mouse, respectively. Obtain the plant $G = C||M$.

(b) Suppose each door, with the exception of $c_7$, can be opened or closed. It is required that (1) the cat and the mouse never share a room, and (2) the cat and the mouse can return to their initial rooms. Draw a DFSM representing this desired behavior $K$.

(c) Obtain a DFSM accepting the supremal controllable sublanguage of $K$.

13. Consider $K_1, K_2 \subseteq L_m(G)$.

(a) Suppose $K_1, K_2$ are modular. Prove that relative closure and controllability of $K_1$ and $K_2$ imply relative closure and controllability of $K_1 \cap K_2$.

(b) Suppose $supRC(K_1), supRC(K_2)$ are modular. Prove that

$$supRC(K_1 \cap K_2) = supRC(K_1) \cap supRC(K_2).$$

14. Given a pair of regular languages $K_1$ and $K_2$, provide an algorithm for testing whether they are modular. State the complexity of your algorithm.

15. Consider the problem described in Example 3.9.

    (a) Draw the DFSM $G = M_1 || M_2$.

    (b) Draw DFSMs representing the desired behavior constraints $K_1$ and $K_2$.

    (c) Suppose $b_i$ and $c_i$ $(i = 1, 2)$ are uncontrollable events. Obtain DFSM generators for $supC(K_1)$ and $supC(K_2)$ using Algorithm 3.1.

    (d) Prove that $supC(K_1)$ and $supC(K_2)$ are modular.

## 3.5 BIBLIOGRAPHIC REMARKS

Controllability in the context of DESs was introduced by Ramadge-Wonham [RW87b]. Refer to Ramadge-Wonham [RW89] and Thistle [Thi94] for survey articles on supervisory control of DESs. Synchronous composition [Hoa85] based supervisors were first studied by Heymann [Hey90], Smedinga [Sme87], and Kumar-Garg-Marcus [KGM91]. An algorithm to compute the supremal controllable sublanguage was first given by Ramadge-Wonham [RW87a]. A more efficient algorithm for the case when $K$ is prefix-closed was first given by Kumar-Garg-Marcus [KGM91]. A closed-form formula for the supremal closed sublanguage was first reported in [BGK+90]. Our proof based on lattice theory is taken from [KG94b]. The blocking issues have been studied by Chen-Lafortune [CL91]. The issue of modular control was first investigated by Ramadge-Wonham [WR88]. Exercise problems 12 and 15 are taken from Ramadge-Wonham [RW89].

# 4

# CONTROL UNDER PARTIAL OBSERVATION

In this chapter, we extend the supervisory control theory of discrete event systems to the setting of partial observation of events—such partial observation naturally arises when there is an insufficient number of sensors. Both centralized and decentralized control techniques are studied.

## 4.1 INTRODUCTION

It was assumed in the previous chapter that a supervisor is capable of observing the occurrence of all events that plant executes. In many situations, it is difficult, if not impossible, for a supervisor to observe all events due to lack of sensors, or due to the presence of faulty sensors. Thus, a particular event may be completely unobservable to a supervisor, or two events may be indistinguishable to a supervisor. The presence of such partial observation can be captured by defining an observation *mask* from the set of events to the set of "observed events": $M : \Sigma \rightarrow \Delta \cup \{\epsilon\}$. If an event is mapped to $\epsilon$, then the corresponding event is unobservable to a supervisor. Similarly, if two events are mapped to the same symbol, then the corresponding events are indistinguishable to a supervisor. Since a supervisor takes its control actions based on observed sequence of events, it must take identical control action following all sequences of events that have identical mask values. Such a supervisor is called observation-compatible, or simply $M$-compatible.

**Definition 4.1** Given a plant $G$ and an observation mask $M : \Sigma \rightarrow \Delta \cup \{\epsilon\}$, a supervisor $S$ is said to be *M-compatible* if

$$\forall s, t \in L(G||S), \sigma \in \Sigma : M(s) = M(t), s\sigma \in L(G||S), t\sigma \in L(G) \Rightarrow t\sigma \in L(G||S).$$

In other words, if $\sigma$ is enabled after $s$ (i.e., $s\sigma \in L(G||S)$), and if it can occur in plant after $t$ (i.e., $t\sigma \in L(G)$), and if $s$ and $t$ look alike under the mask (i.e., $M(s) = M(t)$), then $\sigma$ should also be enabled after $t$ (i.e., $t\sigma \in L(G||S)$). An important property of a synchronous composition based $M$-compatible supervisor $S$ is that it can equivalently be represented as a control law based supervisor $f : M(L(G)) \rightarrow 2^{\Sigma - \Sigma_u}$, which takes the control actions based on masked event sequences executed by the plant and is defined as:

$$\forall s \in M(L(G)) : f(s) := \begin{cases} \{\sigma \in \Sigma \mid \gamma(z_0, \hat{s}\sigma) \text{ not defined}\} & \text{if } \gamma(z_0, \hat{s}) \text{ defined} \\ \text{undefined} & \text{otherwise,} \end{cases}$$

where $\hat{s} \in L(G||S)$ is *any* string such that $M(\hat{s}) = s$. Note that $M$-compatibility of $S$ implies that there is no ambiguity in the definition of $f(\cdot)$.

## 4.2 CENTRALIZED CONTROL

Next we present the definition of observability and normality which are useful in the context of designing supervisors under partial observation.

**Definition 4.2** Given a plant $G$ and an observation mask $M$, $K \subseteq \Sigma^*$ is said to be *observable* (with respect to $G$ and mask $M$) if

$$\forall s, t \in pr(K), \sigma \in \Sigma : M(s) = M(t), s\sigma \in pr(K), t\sigma \in L(G) \Rightarrow t\sigma \in pr(K).$$

It is said to be *normal* (with respect to $G$ and $M$) if

$$\forall s, t \in L(G) : s \in pr(K), t \in L(G), M(s) = M(t) \Rightarrow t \in pr(K).$$

It follows from the definitions of $M$-compatibility and observability that a supervisor $S$ is $M$-compatible if and only if $L(G||S)$ is observable. Note that $K$ is observable (respectively, normal) if and only if $pr(K)$ is observable (respectively, normal). It is easy to see that the condition of normality implies the condition of observability.

Figure 4.1(a) illustrates the observability of $K$ with respect to $G$. If the execution of an event $\sigma$ following a trace $s \in pr(K)$ results in a trace of $pr(K)$, then execution of $\sigma$ following a trace $t \in pr(K)$, which is indistinguishable from $s$, must either result in a trace of $pr(K)$ or a trace of $\Sigma^* - L(G)$ (shown using solid edges), but not a trace of $L(G) - pr(K)$ (shown using a dashed edge).

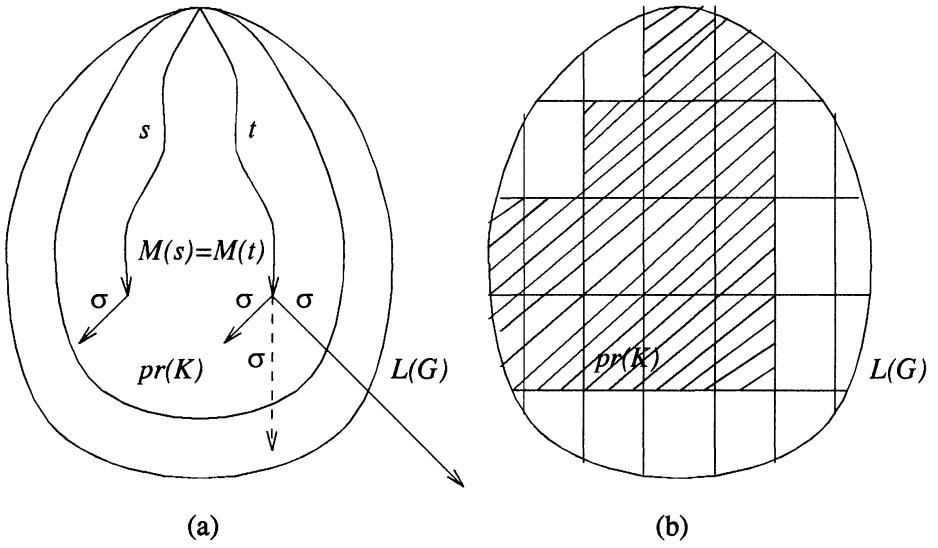



**Figure 4.1** Diagram illustrating observability and normality

Figure 4.1(b) illustrates the normality of $K$ with respect to $G$. Each region in this figure consists of those traces in $L(G)$ that are indistinguishable from each other. Then for $K$ to be normal, $pr(K)$ must be a union of some such regions. This observation leads to the following concise definition of normality:

$$M^{-1}M(pr(K)) \cap L(G) \subseteq pr(K). \tag{4.1}$$

A concise definition of observability is given next.

We begin by defining a modification of the mask function, denoted $\widetilde{M} : \Sigma^* \to \Delta^*\Sigma \cup \{\epsilon\}$, as follows:

$$\forall s \in \Sigma^* : \widetilde{M}(s) := \begin{cases} \epsilon & \text{if } s = \epsilon \\ M(\bar{s})\sigma & \text{if } s = \bar{s}\sigma, \end{cases}$$

where $\bar{s} \in \Sigma^*$ and $\sigma \in \Sigma$. In other words, the effect of applying $\widetilde{M}$ to a string $s \in \Sigma^*$ is to mask all but the last event in $s$.

We are mainly interested in the properties of the function $\widetilde{M}^{-1}\widetilde{M} : \Sigma^* \to 2^{\Sigma^*}$. It is clear from above definition that

$$\forall s \in \Sigma^* : \widetilde{M}^{-1}\widetilde{M}(s) = \begin{cases} \{\epsilon\} & \text{if } s = \epsilon \\ \{\bar{t}\sigma \in \Sigma^* \mid M(\bar{t}) = M(\bar{s})\} & \text{if } s = \bar{s}\sigma, \end{cases}$$

where $\bar{s} \in \Sigma^*$ and $\sigma \in \Sigma$. For notational simplicity, we use $\mathcal{M}$ for $M^{-1}M$ and $\widetilde{\mathcal{M}}$ for $\widetilde{M^{-1}M}$.

In the following Lemma we present an identity, a lower and an upper bound, and a monotonicity result for the operator $\widetilde{\mathcal{M}}$.

**Lemma 4.1** Let $H, \hat{H} \subseteq \Sigma^*$. Then

1. $\widetilde{\mathcal{M}}(H) - \{\epsilon\} = \bigcup_{\sigma \in \Sigma}[\mathcal{M}(H/\sigma).\sigma]$
2. $H \subseteq \widetilde{\mathcal{M}}(H) \subseteq \mathcal{M}(H)$
3. $H \subseteq \hat{H} \Rightarrow \widetilde{\mathcal{M}}(H) \subseteq \widetilde{\mathcal{M}}(\hat{H})$

**Proof:** For notational simplicity, let $H' := \bigcup_{\sigma \in \Sigma}[\mathcal{M}(H/\sigma).\sigma]$.

1. We first show that $\widetilde{\mathcal{M}}(H) - \{\epsilon\} \subseteq H'$. Consider $s \in \widetilde{\mathcal{M}}(H) - \{\epsilon\}$. Then $s = \bar{s}\sigma$ with $\bar{s} \in \Sigma^*$ and $\sigma \in \Sigma$. It follows from the definition of $\widetilde{\mathcal{M}}(H)$ that there exists a string $t = \bar{t}\sigma \in H$ such that $\bar{s} \in \mathcal{M}(\bar{t})$. This implies that $s \in \mathcal{M}(t/\sigma).\sigma \subseteq H'$. This proves that $s \in H'$.

In order to prove the converse, consider $s \in H'$. Then, $s = \bar{s}\sigma$ with $\bar{s} \in \Sigma^*$ and $\sigma \in \Sigma$, and $\bar{s} \in \mathcal{M}(H/\sigma)$. Thus, there exists $t = \bar{t}\sigma \in H$ such that $\bar{s} \in \mathcal{M}(\bar{t})$. This shows that $s \in \widetilde{\mathcal{M}}(H)$. Since $s \neq \epsilon$, we obtain that $s \in \widetilde{\mathcal{M}}(H) - \{\epsilon\}$. This completes the proof of the first part.

2. The first inequality is easily verified from the fact that for all $s$, $s \in \widetilde{\mathcal{M}}(s)$. Next we show that $\widetilde{\mathcal{M}}(H) \subseteq \mathcal{M}(H)$. If $\epsilon \in \widetilde{\mathcal{M}}(H)$, then $\epsilon \in H$, which implies that $\epsilon \in \mathcal{M}(H)$. Thus it suffices to show that $\widetilde{\mathcal{M}}(H) - \{\epsilon\} \subseteq \mathcal{M}(H)$. From the first part we obtain that

$$\begin{aligned}
\widetilde{\mathcal{M}}(H) - \{\epsilon\} &= \bigcup_{\sigma \in \Sigma}[\mathcal{M}(H/\sigma).\sigma] \\
&\subseteq \bigcup_{\sigma \in \Sigma}[\mathcal{M}(H/\sigma).\mathcal{M}(\sigma)] \\
&= \bigcup_{\sigma \in \Sigma}[\mathcal{M}((H/\sigma).\sigma)] \\
&= \mathcal{M}(H).
\end{aligned}$$

3. Straightforward. ∎

Now we are able to give a concise definition of observability, similar in form to that of normality in (4.1).

**Theorem 4.1** Let $K \subseteq L(G)$. Then $K$ is observable if and only if

$$supP[\widetilde{\mathcal{M}}(pr(K))] \cap L(G) \subseteq pr(K) \tag{4.2}$$

**Proof:** Note that if $K = \emptyset$, then $K$ is observable, and also (4.2) trivially holds. Hence we assume that $K \neq \emptyset$.

First consider the proof for necessity. We prove, using induction on the length of strings, that if $s \in supP[\widetilde{\mathcal{M}}(pr(K))] \cap L(G)$, then $s \in pr(K)$. If $s = \epsilon$, then $s \in pr(K)$. This establishes the base step. In order to prove the induction step, consider $s = \bar{s}\sigma$ with $s \in \Sigma^*$ and $\sigma \in \Sigma$. Then $s = \bar{s}\sigma \in supP[\widetilde{\mathcal{M}}(pr(K))]$ and $s = \bar{s}\sigma \in L(G)$. Since prefix closure is preserved under intersection, $supP[\widetilde{\mathcal{M}}(pr(K))] \cap L(G)$ is prefix closed; hence $\bar{s} \in supP[\widetilde{\mathcal{M}}(pr(K))] \cap L(G)$. Thus it follows from the induction hypothesis that $\bar{s} \in pr(K)$. Since $s = \bar{s}\sigma \in supP[\widetilde{\mathcal{M}}(pr(K))]$, we have $s = \bar{s}\sigma \in \widetilde{\mathcal{M}}(pr(K))$. Hence it follows from first part of Lemma 4.1 that there exists a string $t = \bar{t}\sigma \in pr(K)$ such that $M(\bar{t}) = M(\bar{s})$. Since $K$ is observable, we obtain that $s = \bar{s}\sigma \in pr(K)$. This establishes the induction step.

Next consider the proof for sufficiency. Pick strings $\bar{s}, \bar{t} \in pr(K)$ with $M(\bar{s}) = M(\bar{t})$, and an event $\sigma \in \Sigma$ such that $s = \bar{s}\sigma \in pr(K)$ and $t = \bar{t}\sigma \in L(G)$. In order to establish observability of $K$, we need to show that $t \in pr(K)$. Since the containment of (4.2) holds and since $t \in L(G)$, in order to prove that $t \in pr(K)$, it suffices to show that $t \in supP[\widetilde{\mathcal{M}}(pr(K))]$, i.e., each prefix of $t = \bar{t}\sigma$ belongs to $\widetilde{\mathcal{M}}(pr(K))$. Since $\bar{t} \in pr(K)$, it follows that each prefix of $\bar{t}$ is in $pr(K)$. Thus it follows from the second part of Lemma 4.1 that each prefix of $\bar{t}$ is in $\widetilde{\mathcal{M}}(pr(K))$. Hence it remains to show that $\bar{t}\sigma \in \widetilde{\mathcal{M}}(pr(K))$. Since $s = \bar{s}\sigma \in pr(K)$ and $M(\bar{s}) = M(\bar{t})$, it is clear that $t = \bar{t}\sigma \in \widetilde{\mathcal{M}}(pr(K))$. This completes the proof. ■

### 4.2.1 Existence of Supervisors

The following theorem provides a condition for the existence of a supervisor under partial observation, the proof of which is similar to those of Theorems 3.1 and 3.2 and is omitted.

**Theorem 4.2** Let $G$ be a plant, $K \subseteq \Sigma^*$, and $M$ be an observation mask.

1. There exists a $\Sigma_u$-enabling, non-marking and $M$-compatible supervisor $S$ such that $L(G||S) = K$ if and only if $\emptyset \neq K = pr(K) \subseteq L(G)$ and $K$ is controllable and observable with respect to $M$. In this case $S$ can be chosen to be any DSM with $L_m(S) = L(S) = K$.

2. There exists a $\Sigma_u$-enabling, non-marking, non-blocking and $M$-compatible supervisor $S$ such that $L_m(G||S) = K$ if and only if $\emptyset \neq K = pr(K) \cap L_m(G)$ and $K$ is controllable and observable with respect to $M$. In this case $S$ can be chosen to be any DSM with $L_m(S) = L(S) = pr(K)$.

**Example 4.1** Consider the plant $G$ of Example 3.1. Suppose $K = pr(a^*b) \subseteq L(G)$ is the desired generated language. Consider a string $s \in K$. If $s \in a^*$, then $sb \in K$; on the other hand, if $s \notin a^*$, then $sb \notin L(G)$. Thus $K$ is controllable. Suppose $a$ is the only unobservable event, i.e., $M(a) = \epsilon$ and $M(b) = b$. Consider $s, t \in K$ such that $M(s) = M(t)$. Then either $s, t \in a^*$, or $s, t \in a^*b$. If $s, t \in a^*$, then $a$ as well as $b$ are enabled after both $s$ and $t$; on the other hand, if $s, t \in a^*b$, then no event is enabled after both $s$ and $t$. Thus $K$ is observable. Hence it follows from Theorem 4.2 that any DSM $S$ with $L_m(S) = L(S) = K$ is $\Sigma_u$-enabling, non-marking, and $M$-compatible, and yields $K$ as the generated language of the controlled plant.

Next consider a different mask $M'$ such that events $a$ and $b$ are indistinguishable, i.e., $M'(a) = M'(b) \neq \epsilon$. Then it is easy to see that $K$ is not observable with respect to this mask. For example $a, b \in K$ with $M'(a) = M'(b)$, $aa \in K$, but $ba \in L(G) - K$. Hence it follows from Theorem 4.2 that there exists no $\Sigma_u$-enabling, non-marking, and $M'$-compatible supervisor that yields $K$ as the generated language of the controlled plant. ■

**Remark 4.1** In many situations a supervisor is able to observe only a subset of events, called the local set of events, and control only those controllable events that it observes. Such supervisors are called *local*. Thus in this setting, if $\Sigma_{loc} \subseteq \Sigma$ represents the set of events that a local supervisor observes, then the event set $\Sigma_{u,loc} := \Sigma_u \cup (\Sigma - \Sigma_{loc})$ is the set of uncontrollable events for such a local supervisor. It follows from Theorem 4.2 that given a desired behavior $K \subseteq \Sigma^*$, there exists a $\Sigma_u$-enabling, non-marking and local supervisor $S$ such that $L(G||S) = K$ if and only if $\emptyset \neq K = pr(K) \subseteq L(G)$, $K$ is controllable with respect to $\Sigma_{u,loc}$ and observable with respect to $M_{loc} : \Sigma \to \Sigma_{loc} \cup \{\epsilon\}$ defined

as

$$\forall \sigma \in \Sigma : M_{loc}(\sigma) := \begin{cases} \sigma & \text{if } \sigma \in \Sigma_{loc} \\ \epsilon & \text{otherwise} \end{cases}$$

Conditions for the existence of a $\Sigma_u$-enabling, non-marking, non-blocking and local supervisor that achieves a desired marked behavior can be obtained in a similar way. ■

The following theorem establishes the significance of the condition of normality in designing local supervisors.

**Theorem 4.3** Let $\emptyset \neq K \subseteq \Sigma^*$. Then the following are equivalent:

1. $K$ is controllable with respect to $\Sigma_{u,loc}$ and observable with respect to $M_{loc}$.
2. $K$ is controllable with respect to $\Sigma_u$ and normal with respect to $M_{loc}$.

**Proof:** Suppose $K$ is controllable with respect to $\Sigma_{u,loc}$ and observable with respect to $M_{loc}$. Since $\Sigma_u \subseteq \Sigma_{u,loc}$, this implies that

$$pr(K)\Sigma_u \cap L(G) \subseteq pr(K)\Sigma_{u,loc} \cap L(G) \subseteq pr(K),$$

i.e., $K$ is controllable with respect to $\Sigma_u$. Suppose for contradiction that $K$ is not normal with respect to $M_{loc}$. Let $s \in \Sigma^*$ be the smallest string such that $s \in [M_{loc}^{-1} M_{loc}(pr(K)) \cap L(G)] - pr(K)$. Since $K \neq \emptyset$, $s \neq \epsilon$. Hence there exists $\bar{s} \in \Sigma^*, \sigma \in \Sigma$ such that $s = \bar{s}\sigma$ and $\bar{s} \in pr(K)$. We have two different cases to consider: (i) Suppose $\sigma \notin \Sigma_{loc}$; then $\sigma \in \Sigma_{u,loc}$. However, this contradicts the controllability of $K$ with respect to $\Sigma_{u,loc}$ (since $\bar{s} \in pr(K)$ and $s = \bar{s}\sigma \in L(G) - pr(K)$). (ii) Suppose $\sigma \in \Sigma_{loc}$; then since $s \in M_{loc}^{-1} M_{loc}(pr(K)) \cap L(G)$, there exists $t \in pr(K)$ such that $M_{loc}(t) = M_{loc}(s)$, which implies that $t = \bar{t}\sigma$ (since $\sigma \in \Sigma_{loc}$). However, this contradicts the observability of $K$ with respect to $M_{loc}$ (since $t = \bar{t}\sigma \in pr(K), M_{loc}(\bar{t}) = M_{loc}(\bar{s}), s = \bar{s}\sigma \in L(G)$ and $s = \bar{s}\sigma \notin pr(K)$).

Next suppose that $K$ is controllable with respect to $\Sigma_u$ and normal with respect to $M_{loc}$. Then since normality implies observability (with respect to the same observation mask), $K$ is observable with respect to $M_{loc}$. Suppose for contradiction that $K$ is not controllable with respect to $\Sigma_{u,loc}$. Then there exists a string $s \in pr(K)$ and an event $\sigma \in \Sigma_{u,loc}$ such that $s\sigma \in L(G) - pr(K)$. Since $K$ is controllable with respect to $\Sigma_u$, $\sigma \in (\Sigma_{u,loc} - \Sigma_u) \subseteq (\Sigma - \Sigma_{loc})$. Hence $M_{loc}(s\sigma) = M_{loc}(s)$. However, this contradicts the normality of $K$ with respect to $M_{loc}$ (since $s \in pr(K), s\sigma \in L(G) - pr(K)$ and $M_{loc}(s) = M_{loc}(s\sigma)$). ■

**Remark 4.2** It follows from Theorem 4.3 that when $\Sigma - \Sigma_u \subseteq \Sigma_{loc}$ (i.e., when every controllable event is also observable) then in the presence of controllability, observability and normality are equivalent. ■

Theorem 4.3 motivates the following definition of a local supervisor.

**Definition 4.3** Given a plant $G$, a supervisor $S$ is said to be *local* (with respect to the event set $\Sigma_{loc} \subseteq \Sigma$) if

$$\forall s, t \in L(G||S) : M_{loc}(s) = M_{loc}(t), t \in L(G) \Rightarrow t \in L(G||S).$$

In other words, $S$ is local with respect to the event set $\Sigma_{loc}$ if and only if $L(G||S)$ is normal with respect to $M_{loc}$—the projection operation induced by $\Sigma_{loc}$. The following theorem follows from Theorems 4.2 and 4.3:

**Theorem 4.4** Let $G$ be a plant, $K \subseteq \Sigma^*$, and $\Sigma_{loc} \subseteq \Sigma$ be a local event set.

1. There exists a $\Sigma_u$-enabling, non-marking and local supervisor $S$ such that $L(G||S) = K$ if and only if $\emptyset \neq K = pr(K) \subseteq L(G)$ and $K$ is controllable and normal with respect to $M_{loc}$.
2. There exists a $\Sigma_u$-enabling, non-marking, non-blocking and local supervisor $S$ such that $L_m(G||S) = K$ if and only if $\emptyset \neq K = pr(K) \cap L_m(G)$ and $K$ is controllable and normal with respect to $M_{loc}$.

### 4.2.2 Synthesis of Supervisors

As discussed above, existence of a supervisor under partial observation—either an $M$-compatible supervisor or a local supervisor—requires that properties of observability or normality be satisfied by the desired behavior. Next we study the synthesis of minimally restrictive supervisors when such properties fail to hold.

First consider the definition of observability given by Theorem 4.1 for a language $H \subseteq L(G)$. This is of the form:

$$f(H) \cap L(G) \subseteq g(H),$$

where $f(\cdot) = supP[\widetilde{\mathcal{M}}(pr(\cdot))]$ which is monotone but not disjunctive (recall that $supP$ is conjunctive), and $g$ is the prefix closure operation which is monotone but not conjunctive. Hence existence of the supremal observable sublanguage and the infimal observable superlanguage of $H$ cannot be concluded using the lattice theoretic arguments of Chapter 2. The following example illustrates that such languages do not exist in general.

**Example 4.2** Consider the plant $G$ of Example 3.1 with $L(G) = pr(a^*ba^*)$, and an observation mask $M$ with $M(a) = M(b) \neq \epsilon$. Consider languages $K_1 = \{b\}, K_2 = \{aa\} \subseteq L(G)$. Then it is easy to see that both $K_1$ and $K_2$ are observable. However, $K := K_1 \cup K_2 = \{b, aa\}$ is not observable, since $a, b \in pr(K)$ with $M(a) = M(b)$, $aa \in pr(K)$, but $ba \in L(G) - pr(K)$. Thus, in general, observability is not preserved under union; consequently, the supremal observable sublanguage of a given language need not exist.

Next consider $K_3 = \{b, aa, baa, aaa\}, K_4 = \{b, aa, ba\} \subseteq L(G)$. Then one can check that both $K_3$ and $K_4$ are observable. However, $K_3 \cap K_4 = K$, which we know from above is not observable. Thus, in general, observability is not preserved under intersection; consequently, the infimal observable superlanguage of a given language need not exist. ■

If we are interested in an extremal observable language which is also prefix closed, then we must consider the extremal solution of following two inequations:

$$supP[\widetilde{\mathcal{M}}(pr(H))] \cap L(G) \subseteq pr(H); \quad pr(H) \subseteq H, \tag{4.3}$$

where $H \subseteq L(G)$ is the variable of inequation. Using this fact it is easily seen that (4.3) is equivalent to the following single inequation:

$$supP[\widetilde{\mathcal{M}}(pr(H))] \cap L(G) \subseteq H. \tag{4.4}$$

It is clear that (4.3) implies (4.4). Also, since $H \subseteq pr(H)$, the first inequation of (4.3) follows from (4.4). It remains to show that $pr(H) \subseteq H$. Since $H \subseteq L(G)$, we have that $pr(H) \subseteq L(G)$. Moreover, from the second part of Lemma 4.1, $pr(H) \subseteq \widetilde{\mathcal{M}}(pr(H))$, which implies $pr(H) = supP(pr(H)) \subseteq supP[\widetilde{\mathcal{M}}(pr(H))]$. Thus $pr(H) \subseteq supP[\widetilde{\mathcal{M}}(pr(H))] \cap L(G)$. Hence it follows from (4.4) that $pr(H) \subseteq H$, as desired.

(4.4) is of the form:

$$f(H) \cap L(G) \subseteq g(H),$$

where $f$ is monotone and idempotent, and $g$ is the identity function which is conjunctive. Consequently, it follows from Corollary 2.2 that the *infimal prefix closed and observable superlanguage* of a given language $K \subseteq L(G)$, denoted $inf\overline{PO}(K)$, exists. Furthermore, it follows from Theorem 2.9 that

$$inf\overline{PO}(K) = K \cup [supP[\widetilde{\mathcal{M}}(pr(K))] \cap L(G)] = supP[\widetilde{\mathcal{M}}(pr(K))] \cap L(G). \quad (4.5)$$

An algorithm of polynomial time complexity for constructing a nondeterministic generator for $inf\overline{PO}(K)$ when $K$ is regular is given in the next subsection.

Next consider the definition of normality. This is of the form:

$$f(H) \cap L(G) \subseteq g(H),$$

where $f(\cdot) = \mathcal{M}(pr(\cdot))$ which is disjunctive as well as idempotent (as disjunctivity and idempotency are preserved under composition of functions), and $g = pr$ which is monotone but not conjunctive. So the *supremal normal sublanguage* of a language $K \subseteq L(G)$, denoted $supN(K)$ exists; however, the existence of infimal normal superlanguage cannot be concluded using the lattice theoretic arguments of Chapter 2. The following example illustrates that such a language does not exist in general.

**Example 4.3** Consider the plant $G$ and mask $M$ of Example 4.2. Consider $K_1 = \{a, b\}, K_2 = \{a, ba, aa, ab\} \subseteq L(G)$. Then it is easy to see that both $K_1$ and $K_2$ are normal. However, $K := K_1 \cap K_2 = \{a\}$ is not normal, since $a \in pr(K)$, and $M(b) = M(a)$, but $b \in L(G) - pr(K)$. Thus, in general, normality is not preserved under intersection, which implies that infimal normal superlanguage of a given language need not exist. ■

Since $(\mathcal{M})^{-1} = \mathcal{M}$, the following iterative computation for computing $supN(K)$ results from Corollary 2.2:

- $K_0 := K$,
- $K_{i+1} := K_i - [\mathcal{M}(L(G) - K_i)]\Sigma^*$.

If there exists $m \in \mathcal{N}$ such that $K_{m+1} = K_m$, then $K_m = supN(K)$. That such an $m$ exists can be shown whenever $K$ and $L(G)$ are regular languages.

It can be argued as in the previous chapter that the *supremal prefix closed and normal sublanguage* and *infimal prefix closed and normal superlanguage* of a

language $K \subseteq L(G)$, denoted $supPN(K)$ and $inf\overline{PN}(K)$, respectively, exist. Moreover,

$$supPN(K) = K - [\mathcal{M}(L(G) - K)]\Sigma^*; \quad inf\overline{PN}(K) = \mathcal{M}(pr(K)) \cap L(G). \quad (4.6)$$

An algorithm of polynomial time complexity for constructing a nondeterministic generator for $inf\overline{PN}(K)$ when $K$ is regular is given in the next subsection. Similar ideas can be used for constructing a nondeterministic generator for $supPN(K)$ when $K$ is regular.

Similarly, one can argue that the supremal relative closed and normal sublanguage of $K \subseteq L_m(G)$, denoted $supRN(K)$, exists. However, the infimal relative closed and normal superlanguage need not exist. It can be verified that the hypothesis of Theorem 2.10 holds so that $supRN(K) = supN(supR(K))$, i.e., a "modular" computation is possible.

**Remark 4.3** In order to synthesize a minimally restrictive local supervisor for achieving the desired behavior $K$, we need to compute either $supPCN(K)$, the supremal prefix closed, controllable and normal sublanguage of $K$, or $supRCN(K)$, the supremal relative closed, controllable and normal sublanguage of $K$. Suppose that the observation mask has the following property: whenever an uncontrollable and a controllable event are observable (i.e., have non-epsilon mask value), then they are also distinguishable (i.e., have non-identical mask value). In this case $supPCN(K) = supN(supPC(K))$, and $supRCN(K) = supN(supRC(K))$ (refer to exercise problem 7). For example, this property is satisfied by an observation mask which is a projection operator. Hence these formulas are applicable in synthesis of local supervisors. In general, however, these formulas do not hold, and in order to compute $supPCN(K)$ (respectively, $supRCN(K)$) one could alternately perform the computations of supremal prefix (respectively, relative) closed and controllable sublanguage, and supremal normal sublanguage until a fixed point is reached. Such a scheme utilizes the fact that relative closure is preserved under the computations of supremal controllable and supremal normal sublanguages. (Refer to exercise problem 5.) ■

Next we present a formula for a prefix closed and observable sublanguage—similar in form to (4.6)—that is larger than the supremal prefix closed and normal sublanguage, and it preserves controllability. This formula can be used for synthesizing supervisors under partial observation. Given a prefix closed language $K \subseteq L(G)$, define

$$K_{PO} := K - [\widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma)]\Sigma^*. \quad (4.7)$$

We prove that $K_{PO}$ is a closed and observable sublanguage of $K$ larger than the supremal closed and normal sublanguage of $K$.

**Theorem 4.5** Let $K \subseteq L(G)$ be prefix closed. Then

1. $supPN(K) \subseteq K_{PO} \subseteq K$, and
2. $K_{PO}$ is prefix closed and observable.

**Proof:** For simplicity of notation, let $B := [\widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma)]\Sigma^*$.

1. It follows from (4.7) that $K_{PO} \subseteq K$. In order to prove that $supPN(K) \subseteq K_{PO}$, consider the second part of Lemma 4.1, and use the substitution $H = L(G) - K$ to obtain that $\widetilde{\mathcal{M}}(L(G) - K) \subseteq \mathcal{M}(L(G) - K)$. Since the operator $\widetilde{\mathcal{M}}$ is monotone (third part of Lemma 4.1), it follows that $\widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma) \subseteq \widetilde{\mathcal{M}}(L(G) - K) \subseteq \mathcal{M}(L(G) - K)$. Hence $[\widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma)]\Sigma^* \subseteq [\mathcal{M}(L(G) - K)]\Sigma^*$. We conclude that $supPN(K) \subseteq K_{PO}$. This completes the proof of the first part.

2. $K_{PO}$ is prefix closed follows from the fact that $K_{PO} = K - B$, where $K$ is prefix closed and $B$ is extension closed (so their set difference is prefix closed).

Since $K_{PO}$ is closed, in order to prove that $K_{PO}$ is observable, it suffices to consider strings $s, t \in K_{PO}$ and an event $\sigma \in \Sigma$ such that $M(s) = M(t)$, $s\sigma \in K_{PO}$ and $t\sigma \in L(G)$. We need to show that $t\sigma \in K_{PO}$, i.e., $t\sigma \in K$ and $t\sigma \notin B$. Since $s\sigma \in K_{PO}$, it follows that $s\sigma \notin B$. Also, since $B$ is extension closed, this implies that $s\sigma \notin \widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma)$. In other words, there does not exist a string $s'\sigma \in (L(G) - K) \cap K\Sigma$ such that $M(s') = M(s)$. Thus, if there exists a string $s'\sigma \in L(G) \cap K\Sigma$ such that $M(s') = M(s)$, then $s'\sigma \in K$. Since $t\sigma$ is such a string, we obtain that $t\sigma \in K$.

It remains to show that $t\sigma \notin B$, i.e., no prefix of $t\sigma$ belongs to the set $\widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma)$. By assumption $t \in K_{PO}$. Hence $t \notin B$, i.e., no prefix of $t$ belongs to the set $\widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma)$. Thus it suffices to show that $t\sigma \notin \widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma)$, i.e. there does not exist a string $t'\sigma \in (L(G) - K) \cap K\Sigma$ such that $M(t') = M(t)$. The last assertion follows from the fact that $s\sigma \notin \widetilde{\mathcal{M}}((L(G) - K)) \cap K\Sigma)$. This completes the proof of the second part. ■

Next we illustrate by an example that, in general, $K_{PO}$ is strictly larger than $supPN(K)$, and that it need not be a *maximal* prefix closed and observable sublanguage of $K$.

**Example 4.4** Let $\Sigma = \{a, b\}$, and the mask $M$ be such that $M(a) = M(b) \neq \epsilon$. Let $L(G) = pr(aa + aba + baa + bba)$, and $K = pr(aba + ba + bba)$. It is clear that $K \subseteq L(G) \subseteq \{a, b\}^*$, and $K$ is prefix closed. It can also be verified that $K$ is not observable, as $a, b \in K$, $M(a) = M(b)$, $ba \in K$, and $aa \in L(G) - K$. Consequently, $K$ is not normal either. It can be verified that $supPN(K) = pr(a + b)$, and $K_{PO} = pr(ab + bb)$. Thus $supPN(K) \subset K_{PO}$. It can further be verified that the language $pr(aba + bba) \subseteq K$ is also prefix closed and observable. In fact, it is a maximal observable sublanguage of $K$ (an other maximal observable sublanguage is $pr(ba) \subseteq K$). It is clear that $K_{PO} \subset pr(aba + bba)$. This proves that $K_{PO}$ is not necessarily a maximal observable sublanguage. ■

Next we prove a quite useful property of the language $K_{PO}$.

**Theorem 4.6** Let $K \subseteq L(G)$ be prefix closed. If $K$ is controllable, then $K_{PO}$ is also controllable.

**Proof:** Since $K_{PO}$ is closed, in order to show that $K_{PO}$ is controllable it suffices to consider a string $s \in K_{PO}$ and an uncontrollable event $\sigma \in \Sigma_u$ such that $s\sigma \in L(G)$. Then we need to show that $s\sigma \in K_{PO}$. Since $K_{PO} \subseteq K$, we have $s \in K$. It then follows from the controllability of $K$ that $s\sigma \in K$. Thus it suffices to show that $s\sigma \notin [\widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma)]\Sigma^*$. In other words, we need to show that no prefix of $s\sigma$ belongs to $\widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma)$. Since $s \in K_{PO}$, we have that $s \notin [\widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma)]\Sigma^*$, i.e., no prefix of $s$ belongs to $\widetilde{\mathcal{M}}((L(G)-K)\cap K\Sigma)$. Therefore, we must show that $s\sigma \notin \widetilde{\mathcal{M}}((L(G)-K)\cap K\Sigma)$.

Assume for contradiction that $s\sigma \in \widetilde{\mathcal{M}}((L(G) - K) \cap K\Sigma)$. It then follows that there exists a string $t\sigma \in (L(G)-K)\cap K\Sigma$ such that $M(t) = M(s)$. Thus $t \in K$ and $t\sigma \in L(G) - K$. This is contradictory to the fact that $K$ is controllable. ■

**Remark 4.4** It follows from Theorem 4.6 that $(supPC(K))_{PO}$ is a prefix closed, observable and controllable sublanguage of $K$. Thus if $K$ represents the desired behavior, then it follows from Theorem 4.2 that it is possible to synthesize a supervisor under partial observation such that the behavior of the controlled system is $(supPC(K))_{PO}$. Some of the advantages of using such a supervisor are:

- This supervisor is less restrictive than the one that achieves $supPCN(K)$.

- Computation of $(supPC(K))_{PO}$ does not require an iterative computation, whereas the computation of $supPCN(K)$ requires an iterative computation involving alternate computations of supremal prefix closed and controllable sublanguage and supremal normal sublanguage. ■

Finally, we show that the above computation is also useful in synthesis of minimally restrictive local supervisors. Recall from Theorem 4.4 that given a desired prefix closed language $K \subseteq L(G)$ and a local event set $\Sigma_{loc} \subseteq \Sigma$, synthesis of such a supervisor requires computation of $supPCN(K)$—the supremal prefix closed, controllable (with respect to $\Sigma_u$) and normal (with respect to $M_{loc}$) sublanguage of $K$ . We indicate these dependencies explicitly in the notation by writing $supPC_{\Sigma_u}N_{M_{loc}}(K)$ instead of $supPCN(K)$. The following corollary follows from Theorem 4.3:

**Corollary 4.1** Let $G$ be a plant, $K \subseteq \Sigma^*$, and $\Sigma_{loc} \subseteq \Sigma$ be a local event set. Then $K$ is controllable with respect to $\Sigma_u$ and normal with respect to $M_{loc}$ if and only if $K$ is controllable with respect to $\Sigma_{u,loc}$ and normal with respect to $M_{loc}$.

**Proof:** Since $\Sigma_u \subseteq \Sigma_{u,loc}$, the sufficiency part is trivially true. The necessity part follows from Theorem 4.3. ■

Corollary 4.1 implies that the following equality holds:

$$supPC_{\Sigma_u}N_{M_{loc}}(K) = supPC_{\Sigma_{u,loc}}N_{M_{loc}}(K). \tag{4.8}$$

This can be used to obtain the following theorem, where the operation $(\cdot)_{PO_{M_{loc}}}$ computes the prefix closed and observable sublanguage (with respect to mask $M_{loc}$) as defined in (4.7).

**Theorem 4.7** Given a plant $G$, $K \subseteq \Sigma^*$, and a local event set $\Sigma_{loc} \subseteq \Sigma$,

$$supPC_{\Sigma_u}N_{M_{loc}}(K) = (supPC_{\Sigma_{u,loc}}(K))_{PO_{M_{loc}}}.$$

**Proof:** For notational simplicity define

$$\begin{aligned} H_1 &:= supPC_{\Sigma_u}N_{M_{loc}}(K); \\ H_2 &:= (supPC_{\Sigma_{u,loc}}(K))_{PO_{M_{loc}}}; \\ H_3 &:= supPC_{\Sigma_{u,loc}}N_{M_{loc}}(K). \end{aligned}$$

We need to show that $H_1 = H_2$. From (4.8) we have $H_1 = H_3$. We first show that $H_1 \subseteq H_2$ by showing that $H_3 \subseteq H_2$. Since $M_{loc}$ is a projection operator, we have that

$$H_3 = supN_{M_{loc}}(supPC_{\Sigma_{u,loc}}(K)).$$

(Refer to Remark 4.3.) Hence it follows from the first part of Theorem 4.5 that $H_3 \subseteq H_1$ (replace $K$ by $supPC_{\Sigma_{u,loc}}(K)$ in the first part of Theorem 4.5).

Next we show that $H_2 \subseteq H_1$. It suffices to show that $H_2$ is a prefix closed, controllable (with respect to $\Sigma_u$), and normal (with respect to $M_{loc}$) sublanguage of $K$, as $H_1$ is the supremal such language. It follows from Theorem 4.6 that $H_2$ is controllable (with respect to $\Sigma_{u,loc}$). By definition it is a prefix closed and observable (with respect to $M_{loc}$) sublanguage of $K$. Hence it follows from Theorem 4.3 that $H_2$ is a prefix closed, controllable (with respect to $\Sigma_u$), and normal (with respect to $M_{loc}$) sublanguage of $K$, as desired. ■

## 4.2.3 Computation of Supervisors

In this section we present algorithms for determining whether a supervisor under partial observation exists for achieving a desired behavior. Also, we present techniques for computing such supervisors. As before, in doing so, we will assume that the languages involved are regular, so that they can be generated and marked by finite state machines. Thus we assume that $G := (X, \Sigma, \alpha, x_0, X_m)$ is an accessible finite state machine, and $S := (Y, \Sigma, \beta, y_0, Y_m)$ is a trim finite state machine with marked behavior $K$. Let $m, n \in \mathcal{N}$ denote the number of states in $G, S$ respectively. As before, we ignore the dependence of algorithmic complexity on any parameter other than $m$ and $n$.

We begin by presenting an algorithm for testing observability of $K$ (with respect to $L(G)$ and an observation mask $M$). For convenience, given any prefix closed language $H \supseteq K$, we say that $K$ is $H$-observable to imply that it is observable with respect to $H$ (and mask $M$). Let $K^M$ denote the infimal prefix closed and $\Sigma^*$-observable superlanguage of $K$. Then it follows from (4.5), the formula of $inf\widetilde{PO}(K)$ (the infimal prefix closed and $L(G)$-observable superlanguage of $K$), that $K^M = supP[\widetilde{\mathcal{M}}(pr(K))]$ (just replace $L(G)$ by $\Sigma^*$ in the formula). Furthermore, it follows from Theorem 4.1 that $K \subseteq L(G)$ is $L(G)$-observable if and only if $K^M \cap L(G) \subseteq pr(K)$, which is equivalent to

$$K^M \cap L(G) \cap (pr(K))^c = \emptyset. \tag{4.9}$$

This can be used for testing $L(G)$-observability. We need to construct a generator for $K^M$, which requires a new characterization of $\Sigma^*$-observability. Recall

from Chapter 1 that the notation $s \cong t(R_H)$ is used to denote the fact that strings $s$ and $t$ are equivalent under the Myhill-Nerode equivalence induced by a language $H$.

**Theorem 4.8** Consider a language $H \subseteq \Sigma^*$ and an observation mask $M$. Then the following are equivalent:

1. $H$ is $\Sigma^*$-observable.
2. $\forall s, t \in pr(H) : [M(s) = M(t)] \Rightarrow [s \cong t(R_{pr(H)})]$.
3. $\forall s \in pr(H); \sigma, \sigma' \in \Sigma \cup \{\epsilon\} : [M(\sigma) = M(\sigma'); s\sigma, s\sigma' \in pr(H)] \Rightarrow [s\sigma \cong s\sigma'(R_{pr(H)})]$.

Theorem 4.8 can be proved by first showing the equivalence of 1 and 2 (the forward implication requires induction on length of strings in $\Sigma^*$, whereas the backward implication is trivial), and next showing the equivalence of 2 and 3 (the forward implication is trivial, whereas the backward implication requires induction on $|s| + |t|$). The detailed proof is left as an exercise. (Refer to exercise problem 8.)

It follows from Theorem 4.8 that given the trim acceptor $S$ for $K$, the following transitions can be added in $S$ to get a generator for $K^M$:

1. If $(y_1, \sigma, y_2)$ and $(y_1, \sigma', y_3)$ are transitions in $S$ such that $M(\sigma) = M(\sigma')$, then add the transitions $(y_1, \sigma, y_3)$ and $(y_1, \sigma', y_2)$.
2. If $(y_1, \sigma, y_2)$ is a transition in $S$ such that $M(\sigma) = \epsilon$, then add the transitions $(y_1, \epsilon, y_2)$ and $(y_1, \sigma, y_1)$.

Step 1 ensures that a pair of strings of the type $s\sigma$ and $s\sigma'$ are Nerode equivalent whenever $\sigma$ and $\sigma'$ are indistinguishable events, whereas the second step ensures that $s\sigma$ is Nerode equivalent to $s$ whenever $\sigma$ is an unobservable event. Let $S_{OBS}$ be the resulting $\epsilon$-NFSM. Then it follows from (4.9) that $L(G)$-observability of $K$ can be tested by testing the emptiness of the language accepted by $S_{OBS}||G||(\bar{S})^c$, which is an $O(mn^2)$ test. Note that the $\epsilon$-NFSM $S_{OBS}||G$ generates the language $inf\overline{PO}(K)$.

**Example 4.5** Consider the plant $G$ of Example 3.1, the desired generated language $K = pr(a^*b)$ and the mask $M$ with $M(a) = \epsilon$ and $M(b) = b$ of

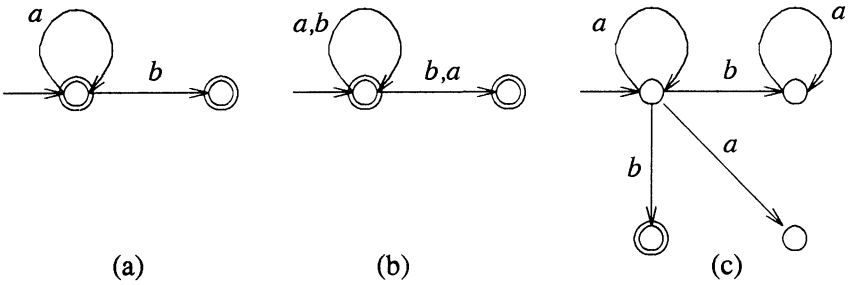



**Figure 4.2** Diagram illustrating test for observability

Example 4.1. The generator $S$ for $K$ is shown in Figure 4.2(a). Since there already exists a self-loop on the unobservable event $a$ at the state where it is executable, it follows that in this case $S_{OBS} = S$, which implies $L(S_{OBS}||G) = L(S||G) = L(S) \cap L(G) = K \cap L(G) = K$. Thus $K$ is observable with respect to the given mask $M$.

Next consider the mask $M'$ with $M'(a) = M'(b) \neq \epsilon$ of Example 4.1. Then the nondeterministic generator $S_{OBS}$ for $K^{M'}$ is as shown in Figure 4.2(b). Figure 4.2(c) depicts the NFSM $S_{OBS}||G$. From this figure it is clear that the generated language of $S_{OBS}||G$ is $pr(a^*ba^*) = L(G) \supset K$. Hence $K$ is not observable with respect to the mask $M'$. Also, it follows that $inf\overline{PO}(K) = pr(a^*ba^*)$. ■

Next we give a test for normality of $K$ (with respect to $L(G)$ and mask $M$). As above, for convenience, given a prefix closed language $H \supseteq K$, we say $K$ is $H$-normal to imply that it is normal with respect to $H$ (and mask $M$). $L(G)$-normality of $K$ can be tested by checking whether

$$\mathcal{M}(pr(K)) \cap L(G) \subseteq pr(K). \tag{4.10}$$

Given the trim acceptor $S$ for $K$, the following transitions can be added in $S$ to get a generator for $\mathcal{M}(pr(K))$:

1. If $(y_1, \sigma, y_2)$ is a transition in $S$, then add the transition $(y_1, \sigma', y_2)$ such that $M(\sigma') = M(\sigma)$.
2. If $M(\sigma) = \epsilon$, then add the transition $(y, \sigma, y)$ at every state $y$ of $S$.

Let $S_{NRM}$ be the resulting NFSM. Then it follows from the above construction that whenever a string is generated by $S$, all strings that are indistinguishable

to it are generated by $S_{NRM}$, i.e., $S_{NRM}$ generates $\mathcal{M}(pr(K))$. Hence it follows from (4.10) that $L(G)$-normality of $K$ can be tested by testing the emptiness of the language accepted by $S_{NRM}||G||(\bar{S})^c$, which again is an $O(mn^2)$ test. Note that the $\epsilon$-NFSM $S_{NRM}||G$ generates the language $inf\overline{PN}(K)$.

**Example 4.6** Consider the plant $G$, desired generated behavior $K$ and mask $M$ as in Example 4.5. Then using the generator $S$ for $K$ shown in Figure 4.2(a), the generator $S_{NRM}$ is obtained; it has the same structure as that of $G$ shown in Figure 3.3(a). So it follows that $L(S_{NRM}||G) = L(G) \neq K$, which implies that $K$ is not normal with respect to the given mask. Also, since $L(S_{NRM}||G) = L(G) = pr(a^*ba^*)$, it follows that $inf\overline{PN}(K) = pr(a^*ba^*)$. ■

Next we present a technique that can be used for determining a supervisor that achieves $(supPC(K))_{PO}$ as the controlled generated behavior, which is a prefix closed, controllable and observable sublanguage of $K$ larger than the supremal prefix closed, controllable and normal sublanguage of $K$. We will assume for simplicity that $K$ is prefix closed. However, this does not result in a loss of generality, as for a non-prefix closed $K$, $supPC(K) = supPC(supP(K))$.

Let $S$ be a synchronous composition based supervisor with $L(S) = supPC(K)$. Then the control achieved by $S$ is equivalent to the control law based supervisor $f : L(G) \rightarrow 2^{\Sigma - \Sigma_u}$ defined as

$$\forall s \in L(G) : f(s) := \{\sigma \in \Sigma \mid s\sigma \in L(G) - supPC(K)\}. \tag{4.11}$$

Since $supPC(K)$ is controllable, it is clear that $f(s) \subseteq (\Sigma - \Sigma_u)$ for each $s \in L(G)$. Note that such a supervisor requires complete observation of events executed by the plant. However, it can be used to obtain another control law based supervisor $f_M : supPC(K) \rightarrow 2^{\Sigma - \Sigma_u}$ which can be used under partial observation:

$$\forall s \in supPC(K) : f_M(s) := \left[ \bigcup_{t \in supPC(K) : M(t) = M(s)} f(t) \right]. \tag{4.12}$$

It follows from the definition in (4.12) that (i) for each $s \in supPC(K)$, $f_M(s) \subseteq \Sigma - \Sigma_u$, and (ii) if $s, t \in supPC(K)$ are such that $M(s) = M(t)$, then $f_M(s) = f_M(t)$. Hence it can be used as a supervisor under partial observation. We show that the closed-loop behavior under its supervision equals $(supPC(K))_{PO}$.

**Theorem 4.9** Let $G$ be a plant, and $K \subseteq L(G)$. If $(supPC(K))_{PO} \neq \emptyset$, then $(L(G))^{f_M} = (supPC(K))_{PO}$, where $f_M : supPC(K) \rightarrow 2^{\Sigma - \Sigma_u}$ is defined by (4.12).

**Proof:** We first show using induction on the length of strings that $(L(G))^{f_M} \subseteq (supPC(K))_{PO}$. Since $(supPC(K))_{PO}$ is prefix closed and nonempty, we have $\epsilon \in (supPC(K))_{PO}$. This establishes the base step. In order to prove the induction step, consider $s = \bar{s}\sigma$ with $\bar{s} \in \Sigma^*$ and $\sigma \in \Sigma$ such that $s = \bar{s}\sigma \in (L(G))^{f_M}$. Since $(L(G))^{f_M}$ is prefix closed, it follows that $\bar{s} \in (L(G))^{f_M}$. Hence from induction hypothesis, $\bar{s} \in (supPC(K))_{PO}$. It then follows from definition of $(\cdot)_{PO}$ given in (4.7) that (i) $\bar{s} \in supPC(K)$, and (ii) none of the prefixes of $\bar{s}$ belong to the language $\widetilde{\mathcal{M}}[(L(G) - supPC(K)) \cap (supPC(K))\Sigma]$. Thus in order to show that $s = \bar{s}\sigma \in (supPC(K))_{PO}$ it suffices to show that (i) $s = \bar{s}\sigma \in supPC(K)$, and (ii) $s \notin \widetilde{\mathcal{M}}[(L(G) - supPC(K)) \cap (supPC(K))\Sigma]$. Since $s = \bar{s}\sigma \in (L(G))^{f_M}$, it follows from the definition of the closed-loop behavior that $\sigma \notin f_M(s)$. Thus from the definition of $f_M$ given in (4.12), we have that if $\bar{t} \in L(G)$ is such that $M(\bar{t}) = M(\bar{s})$ and $t := \bar{t}\sigma \in L(G)$, then $t = \bar{t}\sigma \in supPC(K)$. This implies that (i) $s = \bar{s}\sigma \in supPC(K)$, and (ii) $s = \bar{s}\sigma \notin \widetilde{\mathcal{M}}[(L(G) - supPC(K)) \cap (supPC(K))\Sigma]$. This proves the forward containment.

We prove the reverse containment, also by using induction on the length of strings. By definition of $(L(G))^{f_M}$ we have that $\epsilon \in (L(G))^{f_M}$, which proves the base step. In order to prove the induction step consider $s = \bar{s}\sigma$ with $\bar{s} \in \Sigma^*$ and $\sigma \in \Sigma$ such that $s = \bar{s}\sigma \in (supPC(K))_{PO}$. Since $(supPC(K))_{PO}$ is prefix closed, this implies that $\bar{s} \in (supPC(K))_{PO}$. Hence it follows from the induction hypothesis that $\bar{s} \in (L(G))^{f_M}$. Thus in order to show that $s = \bar{s}\sigma \in (L(G))^{f_M}$, it suffices to show $\sigma \notin f_M(s)$, i.e., if $\bar{t} \in L(G)$ is such that $M(\bar{t}) = M(\bar{s})$ and $t := \bar{t}\sigma \in L(G)$, then $t = \bar{t}\sigma \in supPC(K)$. Since $s = \bar{s}\sigma \in (supPC(K))_{PO}$, we have (i) $\bar{s} \in supPC(K)$, which implies $s = \bar{s}\sigma \in (supPC(K))\Sigma$, and (ii) $s = \bar{s}\sigma \notin \widetilde{\mathcal{M}}[(L(G) - supPC(K)) \cap (supPC(K))\Sigma]$. These two together imply that if $t = \bar{t}\sigma \in L(G)$ is such that $M(\bar{t}) = M(\bar{s})$, then $t = \bar{t}\sigma \in supPC(K)$. This completes the proof. ■

The control exercised by the $M$-compatible supervisor $f_M : supPC(K) \rightarrow 2^{\Sigma - \Sigma_u}$ defined in (4.12) can be computed in an on-line manner such that each iteration has a computational complexity of $O(mn)$ time. (Refer to exercise problem 10.)

## 4.3 MODULAR CONTROL

In the previous chapter we saw that under complete observation, modular synthesis of prefix closed languages is always possible, whereas modular syn-

thesis of non-prefix closed languages requires the condition of modularity. We show in this section that the same holds for modular synthesis under partial observation. The following lemma is straightforward:

**Lemma 4.2** Let $K_1, K_2 \subseteq \Sigma^*$ be modular. If $K_1$ and $K_2$ are observable (respectively, normal), then $K_1 \cap K_2$ is observable (respectively, normal).

Using this result, the next theorem can be obtained in a manner analogous to the corresponding result for the case of complete observation.

**Theorem 4.10** Let $G$ be a plant, $K_1, K_2 \subseteq \Sigma^*$, $M$ be an observation mask, and $\Sigma_{loc} \subseteq \Sigma$ be a local event set.

1. Suppose there exist $\Sigma_u$-enabling, non-marking, and $M$-compatible supervisors $S_1, S_2$ such that $L(G||S_1) = K_1$ and $L(G||S_2) = K_2$. Then $S := S_1||S_2$ is $\Sigma_u$-enabling, non-marking, and $M$-compatible; and $L(G||S) = K_1 \cap K_2$.

2. Suppose there exist $\Sigma_u$-enabling, non-marking, $M$-compatible, and non-blocking supervisors $S_1, S_2$ such that $L_m(G||S_1) = K_1$ and $L_m(G||S_2) = K_2$. Then $S := S_1||S_2$ is $\Sigma_u$-enabling, non-marking, and $M$-compatible; and $L_m(G||S) = K_1 \cap K_2$. Furthermore, $S$ is non-blocking if and only if $K_1$ and $K_2$ are modular.

3. Suppose there exist $\Sigma_u$-enabling, non-marking, and local supervisors $S_1, S_2$ such that $L(G||S_1) = K_1$ and $L(G||S_2) = K_2$. Then $S := S_1||S_2$ is $\Sigma_u$-enabling, non-marking, and local; and $L(G||S) = K_1 \cap K_2$.

4. Suppose there exist $\Sigma_u$-enabling, non-marking, local, and non-blocking supervisors $S_1, S_2$ such that $L_m(G||S_1) = K_1$ and $L_m(G||S_2) = K_2$. Then $S := S_1||S_2$ is $\Sigma_u$-enabling, non-marking, and local; and $L_m(G||S) = K_1 \cap K_2$. Furthermore, $S$ is non-blocking if and only if $K_1$ and $K_2$ are modular.

**Remark 4.5** The first part of Theorem 4.10 requires that $K_i$ $(i = 1, 2)$ both be prefix closed, controllable and observable. Otherwise $(supPC(K_i))_{PO}$ can be considered instead of $K_i$, since $(supPC(K_1)_{PO} \cap (supPC(K_2)_{PO})$ is prefix closed, controllable and observable. Similar strategies can be derived for other parts of Theorem 4.10. ■

## 4.4 DECENTRALIZED CONTROL

As mentioned in the previous chapter, when the desired behavior is specified as the intersection of two or more languages, then it is preferable to design a modular supervisor, as it results in a computationally efficient design. Modular supervision is also desired when, due to the physically distributed nature of the plant, it is impractical to design a centralized supervisor, even when the desired behavior is specified as a *single* language. Modular control in such situations is referred as decentralized control. Thus, in this setting we are interested in synthesizing two or more supervisors, each compatible with its own observation mask and each having its own set of controllable events, such that a desired controlled behavior is achieved. In other words, given a plant $G$, event sets $\Sigma_{u1}, \Sigma_{u2} \subseteq \Sigma$ and masks $M_1, M_2$, we are interested in designing $\Sigma_{ui}$-enabling and $M_i$-compatible supervisors $S_i$ $(i = 1, 2)$ such that the controlled behavior under the control of $S := S_1||S_2$ equals some desired behavior.

For notational simplicity define $\Sigma_u := \Sigma_{u1} \cap \Sigma_{u2}$. These events represent the uncontrollable events for the composed supervisor $S$, as none of the supervisors can prevent occurrence of events in the set $\Sigma_u$. Hence it is expected for the existence of decentralized supervision that the desired behavior be controllable (with respect to $\Sigma_u$). On the other hand, the events in the set $\Sigma - \Sigma_u$ can be controlled by at least one of the supervisors. However, their enablement and disablement must satisfy the restriction imposed by the partial observability of the supervisors. This leads to the introduction of the notion of co-observability:

**Definition 4.4** Given a plant $G$, the uncontrollable event sets $\Sigma_{u1}$ and $\Sigma_{u2}$ of the two supervisors and their observation masks $M_1$ and $M_2$, respectively, a language $K \subseteq \Sigma^*$ is said to be *co-observable* (with respect to $\Sigma_{u1}, \Sigma_{u2}, M_1, M_2$ and $G$) if

$\forall s_1, s_2, t \in pr(K), \sigma \in \Sigma - \Sigma_u$:

1. $\sigma \in \Sigma_{u2} - \Sigma_{u1}, M_1(s_1) = M_1(t), s_1\sigma \in pr(K), t\sigma \in L(G) \Rightarrow t\sigma \in pr(K)$
2. $\sigma \in \Sigma_{u1} - \Sigma_{u2}, M_2(s_2) = M_2(t), s_2\sigma \in pr(K), t\sigma \in L(G) \Rightarrow t\sigma \in pr(K)$
3. $\sigma \in (\Sigma_{u1} \cup \Sigma_{u2})^c, M_1(s_1) = M_1(t), M_2(s_2) = M_2(t), s_1\sigma, s_2\sigma \in pr(K)$, $t\sigma \in L(G) \Rightarrow t\sigma \in pr(K)$.

Thus if an event is controllable by only one of the supervisors and it is enabled following a trace, then it must be enabled following any other trace that is

indistinguishable to that supervisor (provided it can occur in the plant). On the other hand, if the event is controllable by both supervisors, and it can occur in the plant following a trace which is indistinguishable from a certain trace to the first supervisor and from another trace to the second supervisor, and the event is enabled following this latter pair of traces, then the event must also be enabled following the former trace. It is clear that $K$ is co-observable if and only if $pr(K)$ is co-observable. Also, as is the case with observability, co-observability of prefix-closed languages is preserved under intersection; consequently, the infimal prefix-closed and co-observable superlanguage of a given language exists.

We show below that controllability together with co-observability is required for decentralized supervision. It is clear that observability with respect to each of the masks implies co-observability. Thus a weaker condition than observability with respect to each of the masks is needed for decentralized supervision; this is because the events that are controllable by both supervisors can be disabled by either of them. However, if the common controllable event set is empty, then under the condition of controllability, co-observability is equivalent to observability with respect to each of the masks.

Next we establish a relationship between controllability, observability, and co-observability.

**Lemma 4.3** Consider a plant $G$, event sets $\Sigma_{u1}, \Sigma_{u2}$ with $\Sigma_u := \Sigma_{u1} \cap \Sigma_{u2}$, and masks $M_1, M_2$. If $K_i \subseteq \Sigma^*$ $(i = 1, 2)$ are prefix closed, $\Sigma_{ui}$-controllable and $M_i$-observable, then $K := K_1 \cap K_2$ is prefix closed, $\Sigma_u$-controllable and co-observable.

**Proof:** Prefix closure of $K$ follows from prefix closure of $K_i$ $(i = 1, 2)$. Since $K_i$ is $\Sigma_{ui}$-controllable and $\Sigma_u \subseteq \Sigma_{ui}$, it follows that $K_i$ is $\Sigma_u$-controllable. Since $K_i$ is prefix closed, and controllability of prefix closed languages is preserved under intersection, it follows that $K$ is $\Sigma_u$-controllable.

In order to show co-observability, pick $s_1, s_2, t \in K$ (note that $K$ is prefix closed, so $pr(K) = K$) and $\sigma \in \Sigma - \Sigma_u$. Then we must consider the three different cases of the definition of co-observability. First suppose $\sigma \in \Sigma_{u2} - \Sigma_{u1}$, $M_1(s_1) = M_1(t)$, $s_1\sigma \in K$ and $t\sigma \in L(G)$; we need to show that $t\sigma \in K$. Since $t \in K_1$ (as $t \in K$), it follows from the $M_1$-observability of $K_1$ that $t\sigma \in K_1$. On the other hand, since $t \in K_2$ (as $t \in K$) and $\sigma \in \Sigma_{u2} - \Sigma_{u1} \subseteq \Sigma_{u2}$, it follows from the $\Sigma_{u2}$-controllability of $K_2$ that $t\sigma \in K_2$. The second case of co-observability can be obtained symmetrically, and the final case follows from the $M_i$-observability of $K_i$. ■

The result of Lemma 4.3 can be used to obtained a stronger result which is represented in the next theorem. First we define the following classes of languages: Given $K \subseteq L(G)$, event sets $\Sigma_{u1}, \Sigma_{u2}$ and observation masks $M_1, M_2$, define for $i = 1, 2$:

$$\begin{aligned} \overline{PC_{\Sigma_{ui}, O_{M_i}}}(K) &:= \{H \subseteq L(G) \mid K \subseteq H = pr(H), \\ &\quad H \text{ is } \Sigma_{ui}\text{-controllable and } M_i\text{-observable}\} \\ \overline{PCCo}(K) &:= \{H \subseteq L(G) \mid K \subseteq H = pr(H), \\ &\quad H \text{ is } \Sigma_u\text{-controllable and co-observable}\} \end{aligned}$$

It is easily seen that $\overline{PC_{\Sigma_{ui}, O_{M_i}}}(K)$ and $\overline{PCCo}(K)$ are closed under arbitrary intersection; consequently $K^i := inf\overline{PC_{\Sigma_{ui}, O_{M_i}}}(K)$ and $K^{12} := inf\overline{PCCo}(K)$ exist and are unique (refer to exercise problem 9). The next theorem states that $K^{12} = K^1 \cap K^2$.

**Theorem 4.11** Consider a plant $G$ and $K \subseteq L(G)$. Let $\Sigma_{ui}, M_i, K^i, K^{12}$ $(i = 1, 2)$ be as defined above. Then $K^{12} = K^1 \cap K^2$.

**Proof:** Clearly, the required equality holds when $K = \emptyset$. So we assume that $K \neq \emptyset$.

Since $K \subseteq K^i$, it follows that $K \subseteq K^1 \cap K^2$. Also, it follows from Lemma 4.3 that $K^1 \cap K^2$ is prefix closed, $\Sigma_u$-controllable, and co-observable. So $K^1 \cap K^2$ is a prefix-closed, $\Sigma_u$-controllable and co-observable superlanguage of $K$. Hence we have that $K^{12} \subseteq K^1 \cap K^2$. In order to see the reverse containment, it suffices to show that non-zero length strings of $K^1 \cap K^2$ are also in $K^{12}$, as the zero length string $\epsilon$ does belong to $K^{12}$. Thus we need to show that for any string $t \in K^{12}$ and an event $\sigma$ such that $t\sigma \in K^1 \cap K^2$, $t\sigma \in K^{12}$. If $\sigma \in \Sigma_u$, then it follows from the prefix-closure and $\Sigma_u$-controllability of $K^{12}$ that $t\sigma \in K^{12}$. (Note that by definition $K^1, K^2 \subseteq L(G)$, which implies $t\sigma \in L(G)$.)

On the other hand, if $\sigma \in \Sigma - \Sigma_u$, then we show that the following holds:

1. $[\sigma \in \Sigma_{u2} - \Sigma_{u1}] \Rightarrow [\exists s_1 : M_1(s_1) = M_1(t), s_1\sigma \in pr(K)]$
2. $[\sigma \in \Sigma_{u1} - \Sigma_{u2}] \Rightarrow [\exists s_2 : M_2(s_2) = M_2(t), s_2\sigma \in pr(K)]$
3. $[\sigma \in (\Sigma_{u1} \cup \Sigma_{u2})^c] \Rightarrow [\exists s_1, s_2 : M_1(s_1) = M_1(t), M_2(s_2) = M_2(t), s_1\sigma, s_2\sigma \in pr(K)]$,

as this together with co-observability of $K^{12}$ clearly implies that $t\sigma \in K^{12}$. We only prove that case (1) holds, as the proof for the other two cases is similar. Since $t\sigma \in K^1 \cap K^2 \subseteq K^1$, $t\sigma \in K^1$. Also, since $\sigma \in \Sigma_{u2} - \Sigma_{u1}$, $\sigma \notin \Sigma_{u1}$. Hence it follows from the definition of $K^1$ that there exists $s_1$ such that $s_1\sigma \in pr(K)$ and $M_1(s_1) = M_1(t)$. ■

With this result in hand, we are now ready to present a necessary and sufficient condition for decentralized supervision.

**Theorem 4.12** Consider a plant $G$, $K \subseteq \Sigma^*$, event sets $\Sigma_{u1}, \Sigma_{u2}$ with $\Sigma_u := \Sigma_{u1} \cap \Sigma_{u2}$, and masks $M_1, M_2$ as defined above.

1. Then for $i$=1, 2, there exist $\Sigma_{ui}$-enabling, non-marking, and $M_i$-compatible supervisors $S_i$ such that $L(G||S) = K$, where $S := S_1||S_2$, if and only if $\emptyset \neq K = pr(K) \subseteq L(G)$, and $K$ is $\Sigma_u$-controllable and co-observable.

2. Then for $i$=1, 2, there exist $\Sigma_{ui}$-enabling, non-marking, and $M_i$-compatible supervisors $S_i$ such that $S := S_1||S_2$ is non-blocking and $L(G||S) = K$, if and only if $\emptyset \neq K = pr(K) \cap L_m(G)$, and $K$ is $\Sigma_u$-controllable and co-observable.

In both the above cases $S_i$ can be chosen to be any DSM with $L_m(S_i) = L(S_i) = K^i$, the infimal prefix closed, $\Sigma_{ui}$-controllable and $M_i$-observable superlanguage of $K$.

**Proof:** We only prove the first part, as the second part can be proved similarly. We begin by proving the necessity. It follows from the hypothesis that $L(G||S_i)$ is prefix closed, $\Sigma_{ui}$-enabling, and $M_i$-observable. Hence we obtain from Lemma 4.3 that $L(G||S_1) \cap L(G||S_2) = L(G||S) = K$ is prefix closed, $\Sigma_u$-controllable, and co-observable.

In order to show the sufficiency part select $S_i$ such that $L(S_i) = L_m(S_i) = K^i$. Then clearly $S_i$ is non-marking. Also, since $K^i \subseteq L(G)$, $L(G||S_i) = K^i$. Hence it follows from the definition of $K^i$ that $S_i$ is $\Sigma_{ui}$-controllable and $M_i$-observable. Also, it follows from Theorem 4.11 that $L(G||S) = L(G||S_1) \cap L(G||S_2) = K^1 \cap K^2 = K^{12}$. On the other hand, since $K$ is prefix closed, $\Sigma_u$-controllable and co-observable, it follows that $K^{12} = K$. Thus $L(G||S) = K$, as desired. ■

**Remark 4.6** It follows from Theorem 4.12 that existence of a decentralized supervisor requires that the desired language $K \subseteq L(G)$ be prefix closed, controllable and co-observable. In case the desired behavior fails to satisfy any of these properties, then a minimally restrictive decentralized supervisor cannot be synthesized since co-observability is not preserved under union. However, the following formula similar to that of (4.7) can be used for computing a prefix closed and co-observable sublanguage of a prefix closed language $K$:

$$\begin{aligned} K_{PCo} := K - [&[\widetilde{\mathcal{M}}_1(L(G) - K) \cap K(\Sigma_{u2} - \Sigma_{u1})] \\ &\cup [\widetilde{\mathcal{M}}_2(L(G) - K) \cap K(\Sigma_{u1} - \Sigma_{u2})] \\ &\cup [\widetilde{\mathcal{M}}_{12}(L(G) - K) \cap K(\Sigma_{u1} \cup \Sigma_{u2})^c]\,]\Sigma^*, \end{aligned} \tag{4.13}$$

where for $H \subseteq \Sigma^*$, $\widetilde{\mathcal{M}}_i := \widetilde{M}_i^{-1}\widetilde{M}_i$ for $i = 1, 2$, and

$$\widetilde{\mathcal{M}}_{12}(H) := \{t \in \Sigma^* \mid \exists s_1, s_2 \in H \text{ s.t. } \widetilde{\mathcal{M}}_1(s_1) = \widetilde{\mathcal{M}}_1(t), \widetilde{\mathcal{M}}_2(s_2) = \widetilde{\mathcal{M}}_2(t)\}.$$

If $K$ is not prefix closed, then it can be replaced by $supP(K)$. Also it can be shown as in Theorem 4.6 that this formula preserves controllability; i.e., if $K$ is controllable, then so is $K_{PCo}$ (refer to exercise problem 13). Thus it is possible to design a decentralized supervisor that achieves $(supPC(K))_{PCo}$ as the controlled generated behavior. The corresponding decentralized supervisor can be computed in an on-line manner so that the computational complexity of each iteration is $O(mn)$. ■

## 4.5 EXERCISES

1. Prove that normality implies observability, and that the converse need not be true.

2. Prove that if $H \subseteq K = pr(K) \subseteq L = pr(L) \subseteq \Sigma^*$ are such that $H$ is observable (respectively, normal) with respect to $K$ and mask $M$, and $K$ is observable (respectively, normal) with respect to $L$ and mask $M$, then $H$ is observable (respectively, normal) with respect to $L$ and mask $M$.

3. Given $K \subseteq L(G)$ and an observation mask $M$, define the following sequence of decreasing languages:

   - $K_0 := K$
   - $\forall n \geq 0 : K_{n+1} := K_n - \mathcal{M}(L(G) - pr(K_n))\Sigma^*$

   Suppose $\bar{n} \geq 0$ is such that $K_{\bar{n}} = K_{\bar{n}+1}$. Show that $K_{\bar{n}} = supN(K)$. Also prove that such an $\bar{n}$ exists when $K$ is regular and $G$ is a DFSM.

4. Given a language $K \subseteq L(G)$, prove that $supPN(K) = supPN(supP(K))$ and that if $K = pr(K)$, then $supN(K) = supPN(K)$.

5. Given a language $K \subseteq L_m(G)$, prove that $supRN(K) = supN(supR(K))$.

6. Two vehicles share a single one-way track consisting of four sections, with stoplights (*) and detectors (!) installed at various junctions as shown in Figure 4.3. Initially the first vehicle is on section 1, whereas the second vehicle is on section 3 of the track.

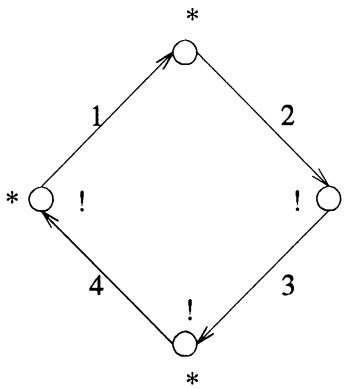



**Figure 4.3** Track with stoplights (*) and detectors (!)

   (a) Let $x_{ij}$ be the state denoting that vehicle $i$ is in track $j$, and $\sigma_{ij}$ be the event denoting that vehicle $i$ leaves section $j$ and enters section $(j+1)$ mod 4. Draw the DFSMs $V_1$ and $V_2$ for the two vehicles, and obtain the plant $G := V_1 || V_2$.

   (b) It is required that the two vehicles never share a section of the track. Draw a DFSM representing this desired behavior $K$.

   (c) Obtain a DFSM generating the language $K_1 := supPC(K)$.

   (d) Obtain a DFSM generating the language $(K_1)_{PO}$ and $supN(K_1)$.

7. Consider $K \subseteq L(G)$ and an observation mask $M$, and define the following decreasing sequence of languages:

   - $K_0 := K$
   - $K_{n+1} := supN(supC(K))$

   Suppose there exists $\bar{n} \geq 0$ such that $K_{\bar{n}+1} = K\bar{n}$.

   (a) Show that $K_{\bar{n}} = supCN(K)$.

(b) If the observation mask is such that $M^{-1}[M(\Sigma_u) - \{\epsilon\}] \subseteq \Sigma_u$, then show that $\bar{n} = 1$.

8. Prove the result of Theorem 4.8.

9. Given a finite state plant $G$ and a regular $K \subseteq L(G)$, provide algorithms for computing $inf\overline{PO}(K)$ and $inf\overline{PN}(K)$.

10. Give an on-line computation of the supervisor $f_M : L(G) \to 2^{\Sigma - \Sigma_u}$ defined in (4.12) so that the computational complexity of each iteration is $O(mn)$.

11. Let $G$ be a plant, and $M_1, M_2$ be two observation masks. Given $K \subseteq L(G)$, define $\overline{PCo}(K) := \{H \subseteq L(G) \mid K \subseteq H = pr(H), H \text{ is co-observable}\}$. Prove that

    (a) $inf\overline{PCo}(K)$ exists (and is unique).

    (b) Show that

$$\begin{aligned} inf\overline{PCo}(K) = supP[\,[\widetilde{\mathcal{M}}_1(pr(K)) \cap \Sigma^*(\Sigma_{u2} - \Sigma_{u1})] \\ \cup [\widetilde{\mathcal{M}}_2(pr(K)) \cap \Sigma^*(\Sigma_{u1} - \Sigma_{u2})] \\ \cup [\widetilde{\mathcal{M}}_{12}(pr(K)) \cap \Sigma^*(\Sigma_{u1} \cup \Sigma_{u2})^c]\,] \cap L(G). \end{aligned}$$

    (c) Show that this can be used to obtain an alternative definition of co-observability.

12. Provide a polynomial time test for co-observability of a regular language with respect to a finite state plant.

13. Given a plant $G$, a prefix closed language $K \subseteq L(G)$, event sets $\Sigma_{u1}, \Sigma_{u2}$, and observation masks $M_1, M_2$, prove that

    (a) $K_{PCo}$ defined by (4.13) is prefix closed and co-observable,

    (b) If $K$ is controllable, then $K_{PCo}$ is also controllable.

## 4.6 BIBLIOGRAPHIC REMARKS

The notion of observability was first introduced by Lin-Wonham [LW88] and by Cieslak et. al. [CDFV88]. Normality was also introduced by Lin-Wonham [LW88]. Formulas for observability and normality were reported in Kumar-Garg-Marcus [BGK+90, KGM91], Rudie-Wonham [RW90], Kumar [Kum93] and Fa-Yang-Zheng [FYZ93]. Their proof based on lattice theory is taken from

Kumar-Garg [KG94b]. A polynomial test for observability was first given by Tsitsiklis [Tsi89]. On-line computation of a supervisor under partial observation with linear complexity was given by Heymann-Lin [HL93]. Decentralized control was first studied by Cieslak et. al. [CDFV88] and Rudie-Wonham [RW92]. Exercise problem 6 is a modification of that studied in Lin-Wonham [LW88].

# 5

# CONTROL OF NON-TERMINATING BEHAVIOR

So far we have restricted our attention to behaviors consisting of finite length strings. Such behaviors are useful in describing *safety* properties of a system, i.e., those properties which state that some conditions must *never* occur. In this chapter we will study supervisory control of behaviors consisting of infinite length strings. Such behaviors are useful in describing *progress* properties of a system, i.e., those properties which state that some conditions must occur *eventually*. In general it is not possible to express such progress constraints as constraints on finite length strings. Consider for example a communication system in which every transmitted message must be received eventually. Since each finite length trace of the communication system can be extended in a manner such that the above constraint is satisfied, this constraint does not impose any restriction on finite length traces of the communication system. However, it is clear that it imposes restriction on traces of infinite length. Thus there is a need to separately study issues related to behaviors consisting of infinite length strings, or *non-terminating* behaviors. Such behaviors are represented using *$\omega$-languages*. In order to avoid any confusion, we use the term *$*$-language* to denote a set of finite length strings.

## 5.1 INTRODUCTION

Given an event set $\Sigma$ over which a DES evolves, and a $*$-language $K \subseteq \Sigma^*$, the notation $K^\omega$ is used to denote the set of all infinite length sequences obtained by concatenating non-zero length strings from $K$:

$$K^\omega := \{e = s_1 s_2 \ldots \mid \forall n \in \mathcal{N} : s_n \in K - \{\epsilon\}\}.$$

Thus $\Sigma^\omega$ denotes the set of all infinite length sequences of events from $\Sigma$. For $e \in \Sigma^\omega$. and $n \in \mathcal{N}$, the notation $e^n \in \Sigma^*$ is used to denote the prefix of length $n$ of $e$, i.e., $e^n := e(1)e(2)\ldots e(n)$.

A subset of $\Sigma^\omega$ is called an *$\omega$-language*. Symbols $A, B$, etc., are used for denoting $\omega$-languages. The prefix operation is extended in a natural way to the set of $\omega$-languages. Thus for $B \subseteq \Sigma^\omega$, $pr(B) \subseteq \Sigma^*$ is the set:

$$pr(B) := \{s \in \Sigma^* \mid \exists e \in B \text{ s.t. } s < e\}.$$

For example, $pr(a^*b^\omega) = a^*b^*$. Thus the prefix operation maps an $\omega$-language to a $*$-language. Conversely, the *limit* operation, also called the Eilenberg's limit, is used to define a map from $*$-languages to $\omega$-languages. Given $K \subseteq \Sigma^*$, $lim(K) \subseteq \Sigma^\omega$ is defined as follows:

$$lim(K) := \{e \in \Sigma^\omega \mid \exists \text{ infinitely many } n \in \mathcal{N} \text{ s.t. } e^n \in K\},$$

i.e., an infinite length string belongs to the limit of a $*$-language $K$ only if infinitely many prefixes of it belong to $K$. Thus if $K$ is prefix closed, then an infinite length string belongs to limit of $K$ if and only if all its prefixes belong to $K$. As an example, $lim(a^*b^*) = a^\omega + a^*b^\omega$.

It can be verified that for $B \subseteq \Sigma^\omega$ and $K \subseteq \Sigma^*$:

$$B \subseteq lim(pr(B)); \quad pr(lim(K)) \subseteq pr(K).$$

(Refer to exercise problem 1.) However, in general, the corresponding reverse containments do not hold. We first examine a condition under which the first reverse containment holds. This requires the definition of a metric topology on $\Sigma^\omega$.

### 5.1.1 A Metric Topology on $\omega$-languages

Given a set $X$, $\mathcal{T} \subseteq 2^X$—a collection of subsets of $X$—is called a *topology* if

- $\emptyset, X \in \mathcal{T}$, and
- $\mathcal{T}$ is closed under arbitrary union and finite intersection;

in which case the pair $(X, \mathcal{T})$ is called a *topological space*. Sets in $\mathcal{T}$ are called *open*. A set $Y \subseteq X$ is said to be *closed* if $X - Y$ is open. Since the set of open sets is closed under arbitrary union, it follows that the set of closed sets

is closed under arbitrary intersection. The *closure* of a set $Y \subseteq X$, denoted $clo(Y) \subseteq X$, is defined to be the infimal closed set larger than $Y$; thus $Y$ is closed if and only if $Y = clo(Y)$.

An often used method to define a topology is a metric. A positive real-valued function $d : X \times X \rightarrow \mathcal{R}_+$ is said to be *metric* over $X$, if

- $\forall x \in X : d(x, x) = 0$, and
- $\forall x, y \in X : d(x, y) = d(y, x)$, and
- $\forall x, y, z \in X : d(x, y) + d(y, z) \geq d(x, z)$.

For a metric $d(\cdot, \cdot)$, an *open ball* of radius $\delta > 0$ and center $x \in X$, denoted $\mathcal{B}(x, \delta) \subseteq X$, is defined to be

$$\mathcal{B}(x, \delta) := \{y \in X \mid d(y, x) < \delta\}.$$

The topology defined by a metric is the smallest collection of sets containing all open balls $\{\mathcal{B}(x, \delta) \mid x \in X, \delta > 0\}$ such that it is closed under arbitrary union and finite intersection. It is easy to check that a set $Y \subseteq X$ is closed in such a topology if and only if

$$\forall x \in X : [\forall \delta > 0 : \mathcal{B}(x, \delta) \cap Y \neq \emptyset \Rightarrow x \in Y].$$

We next define a metric over $\Sigma^\omega$. This metric provides a meaning to distance between sequences.

**Definition 5.1** Define $d : \Sigma^\omega \times \Sigma^\omega \rightarrow \mathcal{R}$ as:

$$\forall e_1, e_2 \in \Sigma^\omega : d(e_1, e_2) := \begin{cases} \frac{1}{n} & \text{if exists } n \in \mathcal{N} \text{ s.t. } e_1^{n-1} = e_2^{n-1} \text{ and } e_1^n \neq e_2^n \\ 0 & \text{otherwise} \end{cases}$$

It can be verified is $d(\cdot, \cdot)$ is a metric over $\Sigma^\omega$. It can be further verified that for $B \subseteq \Sigma^\omega$, $B = lim(pr(B))$ if and only if $B$ is closed (in the topology defined by the metric of Definition 5.1). (Refer to exercise problem 4.) For this reason, the operation $lim(pr(\cdot))$ defined on the set of $\omega$-languages is also called the *$\omega$-closure* operation, and is denoted as $clo(\cdot)$.

## 5.1.2 Deadlock-free Languages

In this subsection we examine a condition under which a given $*$-language $K \subseteq \Sigma^*$ satisfies the equality $pr(lim(K)) = pr(K)$. It is clear from the definition of

the limit operation that $lim(K)$ does not depend on those strings in $K$ which do not have extensions of unbounded lengths in $K$. Hence we define the notion of deadlock-free languages.

**Definition 5.2** $K \subseteq \Sigma^*$ is said to be *deadlock-free* if for each $s \in pr(K)$, there exists $\sigma \in \Sigma$ such that $s\sigma \in pr(K)$. Similarly, a state machine $G := (X, \Sigma, \alpha, x_0, X_m)$ is said to be deadlock-free if for each $x \in X$ there exists $\sigma \in \Sigma$ such that $\alpha(x, \sigma) \neq \emptyset$.

Clearly, deadlock-freeness of $K$ is equivalent to deadlock-freeness of $pr(K)$. Also, $K$ is deadlock-free if and only if any deterministic trim state machine with marked language $K$ is deadlock-free. Furthermore, $K$ is deadlock-free if and only if for each $s \in K$ there exists $t \in K$ such that $s < t$. Finally, the supremal deadlock-free sublanguage of $K \subseteq \Sigma^*$, denoted $supD(K)$, exists (refer to exercise problem 5).

**Theorem 5.1** Given a language $K \subseteq \Sigma^*$, $pr(lim(K)) = pr(K)$ if and only if $K$ is deadlock-free.

**Proof:** First suppose $K$ is deadlock-free. Since $pr(lim(K)) \subseteq pr(K)$ is always true, it suffices to show that the reverse containment also holds. Pick $s \in pr(K)$. Then there exists $t \in K$ such that $s \leq t$. Since $K$ is deadlock-free, for each $n \in \mathcal{N}$ there exists $t_n \in K$ such that $t < t_1 < \ldots < t_n < \ldots$. In other words, there exists $e \in lim(K)$ such that $s \leq t < e$. This implies that $s \in pr(lim(K))$, as desired.

Next suppose $pr(lim(K)) = pr(K)$. Pick $s \in K$, so $s \in pr(K)$. Hence it follows from the hypothesis that $s \in pr(lim(K))$; i.e., there exists $e \in lim(K)$ such that $s < e$. This implies that there exist infinitely many $n \in \mathcal{N}$ such that $e^n \in K$. Hence there exists $n_0 \in \mathcal{N}$ such that $s < e^{n_0}$. Since $e^{n_0} \in K$, this implies that $K$ is deadlock-free. ■

Note that since for any $B \subseteq \Sigma^\omega$, $pr(B)$ is deadlock-free, it follows from Theorem 5.1 that $pr(clo(B)) = pr(lim(pr(B))) = pr(pr(B)) = pr(B)$.

### 5.1.3 $\omega$-Language Model

Recall from the first chapter that a "$*$-language model" of a DES is a $*$-language pair $(K_m, K)$ such that $K_m \subseteq K = pr(K) \neq \emptyset$. $K \subseteq \Sigma^*$ is the set of all finite

length traces that the DES can execute, and $K_m \subseteq K$ is the set of those traces whose execution implies completion of a certain task. A $*$-language model $(K_m, K)$ can equivalently be represented using a DSM. Moreover, an equivalent DFSM representation exists if and only if both $K_m$ and $K$ are regular. Here we extend the definition of a $*$-language model to also include the non-terminating behavior of a DES. Note that each infinite length trace in the non-terminating behavior of a DES must be such that infinitely many of its prefixes correspond to completion of a certain task. Thus we have the following definition of a $\omega$-language model of a DES:

**Definition 5.3** A triple $(B, K_m, K)$, where $B \subseteq \Sigma^\omega$ and $K_m, K \subseteq \Sigma^*$ is called a *$\omega$-language model* if $(K_m, K)$ is a $*$-language model and $B \subseteq lim(K_m)$.

## 5.2 BUCHI MACHINE AS ACCEPTOR FOR $\omega$-LANGUAGES

This section studies the class of $\omega$-languages representable using NSMs. We first define the $\omega$-language accepted by a NSM, $G := (X, \Sigma, \alpha, x_0, X_m)$. This requires the notion of a *path function*, which maps each sequence $e \in lim(L(G))$ to those sequences of states that may be visited in $G$ when $e$ is executed. Letting $\pi : lim(L(G)) \rightarrow 2^{X^\omega}$ denote the path function, for each $e \in lim(L(G))$, $\chi \in \pi(e)$ if and only if

$$\chi(1) = x_0; \quad \forall n \in \mathcal{N} : \chi(n+1) \in \alpha(\chi(n), e(n)).$$

Then the $\omega$-language *accepted* or *recognized* by $G$, denoted $L_\omega(G) \subseteq lim(L(G))$, is defined as:

$$L_\omega(G) := \{e \in lim(L(G)) \mid \exists \chi \in \pi(e), \text{ and infinitely many } n \text{ s.t. } \chi(n) \in X_m\}.$$

That is, a sequence $e$ is in the $\omega$-language accepted by $G$ if there exists a sequence of states $(\chi)$ in the path function for $e$ which visits a marked state infinitely often. The above definition of the $\omega$-language accepted by a NSM is due to Buchi, and a NSM when treated as an acceptor of an $\omega$-language as defined above is known as a *Buchi machine* (BM). A Buchi machine is said to be a finite Buchi machine (FBM) if it has finitely many states. Many other SM based acceptors for $\omega$-languages such as Muller machines, Rabin machines and Streett machines have appeared in literature. The class of $\omega$-languages accepted by these acceptors is the same as that accepted by BMs. (Refer to exercise problem 9 for definition of a Muller machine.)

**Example 5.1** Consider the FBM $G$ shown in Figure 5.1, where $\Sigma = \{a, b\}$. It

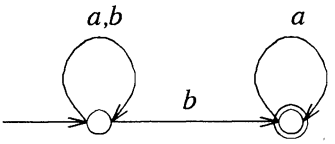



**Figure 5.1** A Buchi Machine

is easily seen that $L_\omega(G) = (a+b)^* b a^\omega$. ■

It is easy to see that for a BM $G$, $L_\omega(G) \subseteq lim(L_m(G))$. Thus the triple $(L_\omega(G), L_m(G), L(G))$ is a $\omega$-language model. In the following proposition we provide a condition under which $L_\omega(G) = lim(L_m(G))$.

**Theorem 5.2** If a given BM $G$ is deterministic, then $L_\omega(G) = lim(L_m(G))$.

**Proof:** Since $L_\omega(G) \subseteq lim(L_m(G))$ is always true, it suffices to prove the reverse containment. Pick $e \in lim(L_m(G))$. This implies that there exist infinitely many $n \in \mathcal{N}$ such that $e^n \in L_m(G)$. Also, since $G$ is deterministic, there exists a unique path $\chi \in X^\omega$ such that $\pi(e) = \chi$. Hence there exist infinitely many $n \in \mathcal{N}$ such that $\chi(n) \in X_m$, which proves that $e \in L_\omega(G)$, as desired. ■

The following example illustrates that if $G$ is not deterministic, then the equality of Theorem 5.2 may not hold.

**Example 5.2** Consider the FBM $G$ shown in Figure 5.1. Then $L_\omega(G) = (a+b)^* b a^\omega$. However, $L_m(G) = (a+b)^* b a^*$, which implies that $lim(L_m(G)) = (a+b)^\omega$. Thus $L_\omega(G) \neq lim(L_m(G))$. ■

## 5.2.1 $\omega$-Regular and Limit $\omega$-Languages

An $\omega$-language that can be recognized using a FBM is called *$\omega$-regular*; it is said to be *deterministic $\omega$-regular* if it can be accepted using a deterministic FBM (DFBM); it is said to be a *limit $\omega$-language* if it can be written as the limit of a $*$-language. The notion of limit $\omega$-languages is useful as many of the

operations on a limit $\omega$-language can be performed by first performing similar operations on the associated $*$-language, and then applying the limit operation.

It follows from Theorem 5.2 that an $\omega$-language is deterministic $\omega$-regular if and only if it can be written as the limit of a regular $*$-language. Thus the deterministic $\omega$-regular class is a *proper* subclass of the class of limit $\omega$-languages. In the following example we show that the deterministic $\omega$-regular class is also a proper subclass of the $\omega$-regular class.

**Example 5.3** Consider the $\omega$-language $B = (a+b)^*ba^\omega$ defined over the event set $\Sigma = \{a, b\}$. A FBM acceptor for $B$ is shown in Figure 5.1. Thus $B$ is $\omega$-regular. We claim that $B$ is not deterministic $\omega$-regular. It suffices to show that $B$ is not a limit $\omega$-language, i.e., there does not exist $K \subseteq \Sigma^*$ such that $lim(K) = B$. Suppose for contradiction that there exists such a $K$. Since $ba^\omega \in B = lim(K)$, there exists $i_0 \in \mathcal{N}$ such that $s_0 := ba^{i_0} \in K$. Similarly, since $s_0ba^\omega \in B = lim(K)$, there exists $i_1 \in \mathcal{N}$ such that $s_1 := s_0ba^{i_1} \in K$. Note that $s_0 < s_1$. Thus we can inductively define a sequence of strings $\{s_n \in K\}_{n \geq 0}$ as:

- $s_0 := ba^{i_0}$, where $i_0 \in \mathcal{N}$ such that $ba^{i_0} \in K$
- $\forall n \in \mathcal{N} : s_{n+1} := s_nba^{i_{n+1}}$, where $i_{n+1} \in \mathcal{N}$ s.t. $s_nba^{i_{n+1}} \in K$

with the property that for each $n \geq 0$, $s_n < s_{n+1}$. Hence there exists $e \in lim(K)$ such that $e^m = s_n$ for each $n \in \mathcal{N}$ and infinitely many $m \in \mathcal{N}$. It is clear from the construction of the sequence $\{s_n\}$ that $e$ has infinitely many occurrences of $b$. Since $lim(K) = B$, $e \in B$. Thus we arrive at an contradiction, as each sequence in $B$ has only finitely many occurrences of $b$. ■

## 5.3 $\omega$-CONTROLLABILITY

In this section we present conditions under which a given discrete event plant can be controlled to achieve a desired non-terminating behavior specified as a certain sublanguage of the $\omega$-language accepted by the plant. We also provide algorithms for verifying these conditions as well as for synthesizing a supervisor whenever one exists.

### 5.3.1 Existence of Supervisors

Let $(B, K_m, K)$ be the $\omega$-language model of a plant that evolves over $\Sigma$, i.e., $B \subseteq \Sigma^\omega; K_m, K \subseteq \Sigma^*$; $K_m \subseteq K = pr(K) \neq \emptyset$; and $B \subseteq lim(K_m)$. Let $f : K \to 2^{\Sigma - \Sigma_u}$ be a control law based supervisor. Then the $\omega$-language model of the controlled plant is denoted as $(B^f, K_m^f, K^f)$, where $(K_m^f, K^f)$ is the $*$-language model of the controlled plant as defined earlier, and

$$B^f := lim(K^f) \cap B. \tag{5.1}$$

Since $K^f$ is prefix closed, an infinite length string belongs to $lim(K^f)$ if and only if all its prefixes belong to $K^f$. Thus (5.1) states that an infinite length trace belongs to the non-terminating behavior of the *controlled* plant if and only if it belongs to the non-terminating behavior of the *uncontrolled* plant and all of its prefixes can occur in the controlled plant. This is a natural definition as a supervisor can prevent the generation of infinite length traces only by preventing their finite length prefixes. We first show that $(B^f, K_m^f, K^f)$ is a $\omega$-language model. Since $(K_m^f, K^f)$ is a $*$-language model, it suffices to show that $B^f \subseteq lim(K_m^f)$. We need the result of the following lemma:

**Lemma 5.1** Suppose $K_1, K_2 \subseteq \Sigma^*$ are such that $K_1$ is prefix closed. Then $lim(K_1 \cap K_2) = lim(K_1) \cap lim(K_2)$.

**Proof:** Since $K_1 \cap K_2 \subseteq K_1$, it follows that $lim(K_1 \cap K_2) \subseteq lim(K_1)$. Similarly, $lim(K_1 \cap K_2) \subseteq lim(K_2)$. Hence we have $lim(K_1 \cap K_2) \subseteq lim(K_1) \cap lim(K_2)$. Thus we only need to prove the reverse containment. Pick $e \in lim(K_1) \cap lim(K_2)$. Then there exist infinitely many $n, m \in \mathcal{N}$ such that $e^n \in K_1$ and $e^m \in K_2$. Thus for each $m$, there exists $n_m \in \mathcal{N}$ such that $e^m \leq e^{n_m}$. Since $e^{n_m} \in K_1$ and $K_1$ is prefix closed, this implies that $e^m \in K_1$. Hence there exist infinitely many $m$ such that $e^m \in K_1 \cap K_2$, i.e., $e \in lim(K_1 \cap K_2)$, as desired.■

Using Lemma 5.1 we can conclude that $B^f \subseteq lim(K_m^f)$ as follows:

$$B^f = lim(K^f) \cap B \subseteq lim(K^f) \cap lim(K_m) = lim(K^f \cap K_m) = lim(K_m^f),$$

where the first containment follows from the fact that $B \subseteq lim(K_m)$, the middle equality follows from the fact that $K^f$ is prefix closed, and the final equality follows from the definition of $K_m^f$. Since $B^f \subseteq lim(K_m^f)$, it follows that

$$pr(B^f) \subseteq pr(lim(K_m^f)) \subseteq pr(K_m^f) \subseteq K^f.$$

Recall that a control law based supervisor is defined to be non-blocking if $pr(K_m^f) = K^f$; we define it to be *$\omega$-non-blocking* if $pr(B^f) = K^f$. Thus the

notion of $\omega$–non-blocking is a stronger notion, and requires that each finite length trace of the controlled plant be a prefix of some infinite length trace of the controlled plant.

Here we are interested in supervisory control using a synchronous composition based supervisor, which requires that the plant be represented as a NSM and supervisor be represented as a DSM. (The control exercised by a control law based supervisor can always be achieved by a deterministic synchronous composition based supervisor.) If the DSM $S := (Y, \Sigma, \beta, y_0, Y_m)$ represents a synchronous composition based supervisor as defined in earlier, then the $\omega$-language model of the controlled plant is given by $(L_\omega(G||S), L_m(G||S), L(G||S))$, where the definition of synchronous composition has been extended to NSMs in an obvious way. As before, we have $L_m(G||S) = L_m(G) \cap L_m(S)$ and $L(G||S) = L(G) \cap L(S)$. Moreover, it can be verified that $L_\omega(G||S) = L_\omega(G) \cap L_\omega(S)$. Since $S$ is deterministic, Theorem 5.2 implies that $L_\omega(S) = lim(L_m(S))$; so $L_\omega(G||S) = L_\omega(G) \cap lim(L_m(S))$. $S$ is said to be $\omega$–non-blocking if $pr(L_\omega(G||S)) = L(G||S)$.

Recall that in order for the control achieved by a synchronous composition based supervisor $S$ to be equivalent to the one achieved by a control law based supervisor, $S$ must be $\Sigma_u$-enabling and non-marking, which ensures that (i) an uncontrollable event is never disabled, and (ii) $L_m(G||S) = L(G||S) \cap L_m(G)$. Since we are interested in also controlling the non-terminating behavior of $G$, (5.1) implies that the following additional constraint must be satisfied: $L_\omega(G||S) = lim(L(G||S)) \cap L_\omega(G)$. Clearly, $L_\omega(G||S) \subseteq lim(L(G||S)) \cap L_\omega(G)$. We show in the following lemma that the reverse containment holds whenever $S$ is non-marking.

**Lemma 5.2** Consider a plant $G$ and a non-marking deterministic supervisor $S$. Then $L_\omega(G||S) = lim(L(G||S)) \cap L_\omega(G)$.

**Proof:** Since the forward containment always holds, it suffices to show that the reverse containment holds. We show this using the facts that (i) $S$ is non-marking, so that $L_m(G||S) = L(G||S) \cap L_m(G)$, and (ii) $S$ is deterministic, so that $L_\omega(S) = lim(L_m(S))$.

$$\begin{aligned} lim(L(G||S)) \cap L_\omega(G) &= lim(L(G||S)) \cap [lim(L_m(G)) \cap L_\omega(G)] \\ &= lim[L(G||S) \cap L_m(G)] \cap L_\omega(G) \\ &= lim(L_m(G||S)) \cap L_\omega(G) \\ &\subseteq lim(L_m(S)) \cap L_\omega(G) \\ &= L_\omega(S) \cap L_\omega(G) \end{aligned}$$

$$= L_\omega(G||S),$$

where the first equality follows from the fact that $L_\omega(G) \subseteq lim(L_m(G))$, the second equality follows from Lemma 5.1 as $L(G||S)$ is prefix closed, and the third equality follows from the fact that $S$ is non-marking. ■

Next we provide a necessary and sufficient condition under which a desired non-terminating behavior can be achieved by a $\Sigma_u$-enabling, non-marking and $\omega$–non-blocking supervisor . Given a plant $G$, $B \subseteq \Sigma^\omega$ is said to be *relative $\omega$-closed* (with respect to $G$), if $clo(B) \cap L_\omega(G) = B \cap L_\omega(G)$. Thus if $B \subseteq L_\omega(G)$, then relative $\omega$-closure of $B$ is equivalent to $clo(B) \cap L_\omega(G) = B$.

**Theorem 5.3** Let $G$ be a plant and $B \subseteq \Sigma^\omega$ be the desired non-terminating behavior. There exists a $\Sigma_u$-enabling, non-marking and $\omega$–non-blocking supervisor $S$ such that $L_\omega(G||S) = B$ if and only if $clo(B) \cap L_\omega(G) = B \neq \emptyset$ and $pr(B)$ is controllable.

**Proof:** Consider first the necessity part. Since $L_\omega(G||S) = B$ and $S$ is $\omega$–non-blocking, $pr(B) = pr(L_\omega(G||S)) = L(G||S)$. Thus, since $S$ is $\Sigma_u$-enabling, $L(G||S) = pr(B)$ is controllable. Also, since $L(G||S) \neq \emptyset$, $pr(B) \neq \emptyset$, i.e., $B \neq \emptyset$. It remains to show that $clo(B) \cap L_\omega(G) = B$, which follows from the fact that $S$ is $\omega$–non-blocking and non-marking as follows:

$$clo(B) \cap L_\omega(G) = lim(pr(B)) \cap L_\omega(G) = lim(L(G||S)) \cap L_\omega(G) = L_\omega(G||S) = B,$$

where the second equality follows from the fact that $S$ is $\omega$–non-blocking, and the third equality follows from the fact that $S$ is non-marking (refer to Lemma 5.2).

Next consider the sufficiency part. Since $pr(B)$ is controllable and non-empty (as $B$ is non-empty), there exists a non-marking and $\Sigma_u$-enabling supervisor $S$ such that $L(G||S) = pr(B)$. Hence

$$L_\omega(G||S) = lim(L(G||S)) \cap L_\omega(G) = lim(pr(B)) \cap L_\omega(G) = clo(B) \cap L_\omega(G) = B,$$

where the first equality follows from the fact that $S$ is non-marking (refer to Lemma 5.2) and the last equality follows from the hypothesis. Finally, since $pr(L_\omega(G||S)) = pr(B) = L(G||S)$, it follows that $S$ is $\omega$–non-blocking, as desired. ■

**Example 5.4** Consider the plant $G$ shown in Figure 5.2 with $\Sigma = \{a, b\}$ and $\Sigma_u = \{b\}$. We have $L_m(G) = L(G) = pr(a^*ba^*)$ and $L_\omega(G) = lim(L_m(G)) =$

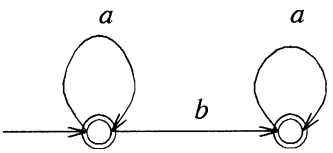



**Figure 5.2** Diagram illustrating a plant

$a^\omega + a^*ba^\omega$. Let $B = a^*ba^\omega \subseteq L_\omega(G)$ be the desired non-terminating behavior, i.e., the controlled plant must generate only those infinite traces in which the event $b$ must occur eventually.

We have $pr(B) = pr(a^*ba^*) = L(G)$, which implies that $pr(B)$ is controllable. However, $clo(B) = L_\omega(G)$, so that $clo(B) \cap L_\omega(G) = L_\omega(G) \supset B$. This implies that $B$ is not relative $\omega$-closed. As a result a $\Sigma_u$-enabling, non-marking, and $\omega$–non-blocking supervisor for achieving $B$ does not exist. This is expected because any supervisor that generates all finite length traces consisting of only the event $a$ (note that $a^* \subseteq pr(B)$), will also generate the infinite length trace $a^\omega$. Recall that a supervisor's decision to enable or disable events is based on its observation of finite length prefixes.

Consider a different desired behavior given by $B_n = \{a^kba^\omega, 0 \leq k \leq n\} \subseteq L_\omega(G)$, where $n \in \mathcal{N}$. Then it can be verified that $B_n$ is relative $\omega$-closed and has a controllable prefix, so that it can be achieved as a controlled plant behavior. The supervisor simply disables the event $a$ following the initial $n$ occurrences of $a$. ■

Next we introduce the notion of $\omega$-controllability and show that it is possible to satisfactorily control the plant when the desired non-terminating behavior is $\omega$-controllable.

**Definition 5.4** Given a plant $G$, $B \subseteq \Sigma^\omega$ is said to be *$\omega$-controllable* if for each $s \in pr(B)$, there exists $B_s \subseteq B\backslash\{s\}$ such that

1. $B_s \neq \emptyset$
2. $B_s$ is relative $\omega$-closed with respect to $G\backslash\{s\}$, i.e., $clo(B_s) \cap L_\omega(G)\backslash\{s\} = B_s \cap L_\omega(G)\backslash\{s\}$
3. $pr(B_s)$ is controllable with respect to $G\backslash\{s\}$, i.e., $pr(B_s)\Sigma_u \cap L(G)\backslash\{s\} \subseteq pr(B_s)$.

In other words, $B$ is $\omega$-controllable if after execution of any trace $s \in pr(B)$, it is possible to control the plant so that the controlled non-terminating behavior is contained in $B$. It is easy to check that if $B$ is relative $\omega$-closed and has a controllable prefix, then $B$ is also $\omega$-controllable. (For any $s \in pr(B)$, choose $B_s := B\backslash\{s\}$, where $B_s$ is as in Definition 5.4.) The converse, however, is not true:

**Example 5.5** Consider the plant $G$ and the $\omega$-language $B$ as given in Example 5.4. Then $B$ is $\omega$-controllable, since for each $s \in pr(B)$, we may define $B_s := ba^\omega$ if $s \subseteq \{a\}^*$ and $B_s := B\backslash s$ otherwise. However, we know from Example 5.4 that $B$ is not relative $\omega$-closed. ■

Thus $\omega$-controllability is a weaker notion. Nevertheless, as shown in the next theorem, it is necessary and sufficient for the existence of a quite satisfactory control policy. We need the result of the following lemma:

**Lemma 5.3** Given a plant $G$, let $B \subseteq L_\omega(G)$ be $\omega$-controllable. Then $pr(B)$ is controllable.

**Proof:** Note that if $B = \emptyset$, then $pr(B) = \emptyset$, which is controllable. So suppose $B \neq \emptyset$, and pick $s \in pr(B)$, $\sigma \in \Sigma_u$ such that $s\sigma \in L(G)$. It follows from $\omega$-controllability of $B$ that there exists $B_s \subseteq B\backslash\{s\}$ such that $B_s \neq \emptyset$ and $pr(B_s)\Sigma_u \cap L(G)\backslash\{s\} \subseteq pr(B_s)$. Non-emptiness of $B_s$ implies that $\epsilon \in pr(B_s)$; hence controllability of $pr(B_s)$ with respect to $G\backslash\{s\}$ implies that $\sigma_u \in pr(B_s)$, as $\sigma_u \in L(G)\backslash\{s\}$. Thus $s\sigma \in \{s\}pr(B_s) \subseteq pr(B)$, as desired. ■

**Theorem 5.4** Let $G$ be a plant and $B \subseteq L_\omega(G)$ be a desired non-terminating behavior. Then for any $\delta > 0$, there exists a $\Sigma_u$-enabling, non-marking and $\omega$–non-blocking supervisor $S$ such that $L_\omega(G||S) \subseteq B$ and for each $e \in B$, there exists $f \in L_\omega(G||S)$ with $d(f, e) < \delta$ if and only if $B$ is non-empty and $\omega$-controllable.

**Proof:** In order to show the necessity part, pick $s \in pr(B)$ and define $\delta := \frac{1}{|s|}$. Then from the hypothesis there exists a $\Sigma_u$-enabling, non-marking and $\omega$–non-blocking supervisor $S$ such that $L_\omega(G||S) \subseteq B$ and for each $e \in B$, there exists $f \in L_\omega(G||S)$ with $d(f, e) < \delta$. For notational simplicity, let $B_\delta := L_\omega(G||S)$. Then it follows from the necessity part of Theorem 5.3 that $B_\delta$ is non-empty

and relative $\omega$-closed and has a controllable prefix. Since $B_\delta \subseteq B$, it follows from the non-emptiness of $B_\delta$ that $B$ is non-empty.

Next we show that $B$ is $\omega$-controllable. Since $s \in pr(B)$ is arbitrary, it suffices to show that there exists $B_s \subseteq B \backslash \{s\}$ such that $B_s$ is non-empty and relative $\omega$-closed with respect to $G\backslash\{s\}$ and $pr(B_s)$ is controllable with respect to $G\backslash\{s\}$. Define $B_s := B_\delta \backslash \{s\}$. Since $B_\delta$ is relative $\omega$-closed and $pr(B_\delta)$ is controllable, it is easily verified that $B_s$ is relative $\omega$-closed with respect to $G\backslash\{s\}$ and $pr(B_s)$ is controllable with respect to $G\backslash\{s\}$. It remains to show that $B_s \neq \emptyset$. Since $B_\delta \neq \emptyset$, it suffices to show that $s \in pr(B_\delta)$. Since $s \in pr(B)$, there exists $e \in B$ such that $s < e$. Hence it follows from the hypothesis that there exists $f \in B_\delta$ such that $d(f, e) < \delta = \frac{1}{|s|}$. This implies that $f^{|s|} = e^{|s|} = s$. Thus $s < f$, which implies that $s \in pr(B_\delta)$.

In order to show the sufficiency part, pick any $\delta > 0$ and define the following sublanguage of $B$:

$$B_\delta := \left[ \bigcup_{s \in pr(B): |s| > \frac{1}{\delta}} \{s\} B_s \right],$$

where for each $s \in pr(B)$ with $|s| > \frac{1}{\delta}$, $B_s \subseteq B\backslash\{s\}$ is chosen such that it is non-empty, it is relative $\omega$-closed with respect to $G\backslash\{s\}$, and $pr(B_s)$ is controllable with respect to $G\backslash\{s\}$. We show that $B_\delta$ is non-empty and relative $\omega$-closed sublanguage of $B$, and $pr(B_\delta)$ is controllable. From the hypothesis $B \neq \emptyset$, hence the set $\{s \in pr(B) : |s| > \frac{1}{\delta}\} \neq \emptyset$. This together with the fact that $B$ is $\omega$-controllable implies that $B_\delta \neq \emptyset$. Since $B_s \subseteq B\backslash\{s\}$, it follows that $\{s\}B_s \subseteq B$, which implies $B_\delta \subseteq B$. Relative $\omega$-closure of $B_\delta$ follows from the fact that each $\omega$-language in the collection $\{\{s\}B_s \mid s \in pr(B), |s| > \frac{1}{\delta}\}$ is relative $\omega$-closed, and the collection itself is pair-wise disjoint (so that any limit point of $B_\delta$ is also a limit point of one of the $\omega$-languages in the collection). Finally, to show that $pr(B_\delta)$ is controllable, pick $t \in pr(B_\delta)$, $\sigma \in \Sigma_u$ such that $t\sigma \in L(G)$. Since $B_\delta \subseteq B$, $t \in pr(B)$. Hence it follows from $\omega$-controllability of $B$ and Lemma 5.3 that $t\sigma \in pr(B)$. From the definition of $B_\delta$ we have

$$\{s \in pr(B) : |s| \leq \frac{1}{\delta} + 1\} = \{s \in pr(B_\delta) : |s| \leq \frac{1}{\delta} + 1\},$$

which implies that if $|t| \leq \frac{1}{\delta}$, so that $|t\sigma| \leq \frac{1}{\delta} + 1$, then $t\sigma \in pr(B_\delta)$. On the other hand, if $|t| > \frac{1}{\delta}$, then there exists $s_t \leq t$ such that $|s_t| > \frac{1}{\delta}$ and $pr(B_{s_t})$ is controllable with respect to $G\backslash\{s\}$, where $B_{s_t} \subseteq B\backslash\{s\}$ is as in definition of $B_\delta$. This implies that $t\sigma \in pr(\{s_t\}B_{s_t}) \subseteq pr(B_\delta)$, as desired.

Finally, since $B_\delta \subseteq B$ is non-empty and relative $\omega$-closed, and $pr(B_\delta)$ is controllable, it follows from the sufficiency part of Theorem 5.3 that there exists a $\Sigma_u$-enabling, non-marking and $\omega$–non-blocking supervisor $S$ such that $L_\omega(G||S) = B_\delta$. It can be readily verified that for each $e \in B$, there exists $f \in B_\delta$ with $d(f, e) < \delta$. This completes the proof. ■

**Remark 5.1** The supervisor in the proof of the sufficiency part of Theorem 5.4 exercises the following control policy: Given a trace $s \in L(G)$, if $|s| \leq \frac{1}{\delta}$, then it disables events in the set $\Sigma_s := [L(G)\backslash\{s\} - pr(B)\backslash\{s\}] \cap \Sigma$ (note that $\Sigma_s \subseteq \Sigma - \Sigma_u$, as $B$ is $\omega$-controllable, which implies $pr(B)$ is controllable); else it disables events in the set $\Sigma_s := [L(G)\backslash\{s\} - pr(B_s)] \cap \Sigma$ (again note that $\Sigma_s \subseteq \Sigma - \Sigma_u$, as $B_s$ is chosen such that $pr(B_s)$ is controllable with respect to $G\backslash\{s\}$). ■

**Example 5.6** Consider the plant $G$ and the $\omega$-language $B$ as given in Example 5.4. Then we know from Example 5.5 that it is $\omega$-controllable. It follows from Theorem 5.4 that there exists a supervisor that achieves the desired behavior $B$ with any given accuracy. If the measure of accuracy is given by $\delta$ as in Theorem 5.4, then we can design a supervisor that achieves the $\omega$-language $B_{n_\delta}$ defined in Example 5.4 as the controlled plant behavior, where $n_\delta$ is any number such that $\frac{1}{n_\delta} < \delta$. Recall from Example 5.4 that it is possible to do so. ■

The following corollary follows from Theorems 5.3 and 5.4 and Lemma 5.3:

**Corollary 5.1** Let $G$ be a plant and $B \subseteq L^\omega(G)$. If $B$ is relative $\omega$-closed and $pr(B)$ is controllable, then $B$ is $\omega$-controllable. The converse is true whenever $B$ is relative $\omega$-closed.

### 5.3.2 Synthesis of Supervisors

It follows from Theorems 5.3 and 5.4 that the existence of a supervisor to achieve a desired non-terminating behavior requires that one or more of the conditions of relative $\omega$-closure, controllable prefix and $\omega$-controllability be satisfied. In this subsection we study synthesis of supervisors when such conditions are not satisfied, and examine the possibility of a minimally restrictive supervision.

Given $B \subseteq L_\omega(G)$, we use the following notation for denoting the set of relative $\omega$-closed and the set of $\omega$-controllable sublanguages of $B$:

$$\begin{aligned} R^\omega(B) &:= \{A \subseteq B \mid A \text{ is relative } \omega\text{-closed}\}; \\ C^\omega(B) &:= \{A \subseteq B \mid A \text{ is } \omega\text{-controllable}\}. \end{aligned}$$

Since a topological space is not closed under arbitrary union of its closed sets, it is expected that a supremal relative $\omega$-closed sublanguage of $B$ does not exist. This is illustrated by the following example.

**Example 5.7** Consider the plant $G$ and $\omega$-languages $B$, $\{B_n\}_{n\geq 0}$ of Example 5.4. Then we know that each $B_n$ is relative $\omega$-closed. However, $B = \bigcup_{n\geq 0} B_n = a^*ba^\omega$ is not relative $\omega$-closed. ■

Therefore it is not possible to synthesize a minimally restrictive supervisor of the kind considered in Theorem 5.3. However, it is easy to verify that $\omega$-controllability is preserved under union (refer to exercise problem 10), so that the supremal $\omega$-controllable sublanguage of $B$, denoted $supC^\omega(B)$, exists. Consequently, it is possible to synthesize a minimally restrictive supervisor of the kind considered in Theorem 5.4. In the following proposition we show that $supC^\omega(B)$ is relative $\omega$-closed whenever $B$ is relative $\omega$-closed; consequently, under the condition of relative $\omega$-closure of $B$, it is possible to design a minimally restrictive supervisor of the kind considered in Theorem 5.3. We need the result of the following straightforward lemma:

**Lemma 5.4** Suppose $B_1, B_2 \subseteq \Sigma^\omega$ are such that $B_1 \subseteq B_2$. Then $pr(clo(B_1) \cap B_2) = pr(B_1)$.

**Proof:** Since $B_1 \subseteq B_2$, it is clear that $B_1 \subseteq clo(B_1) \cap B_2$, which implies that $pr(B_1) \subseteq pr(clo(B_1) \cap B_2)$. On the other hand,

$$pr(clo(B_1) \cap B_2) \subseteq pr(clo(B_1)) \cap pr(B_2) = pr(B_1) \cap pr(B_2) = pr(B_1),$$

where the middle equality follows from the fact that $pr(clo(\cdot)) = pr(\cdot)$, and the final equality follows from the fact that $B_1 \subseteq B_2$. ■

**Theorem 5.5** Given a plant $G$, let $B \subseteq L_\omega(G)$ be such that $B$ is relative $\omega$-closed; then $supC^\omega(B)$ equals the supremal sublanguage of $B$ which is relative $\omega$-closed and has a controllable prefix.

**Proof:** For notational simplicity define $A := supC^{\omega}(B)$. Since $\emptyset$ is relative $\omega$-closed and has a controllable prefix, the set of those sublanguages of $B$ which are relative $\omega$-closed and have a controllable prefix is non-empty. Let $I$ be an index set such that for each $i \in I$, $B_i \subseteq B$ is relative $\omega$-closed and has a controllable prefix. Then it follows from Corollary 5.1 that for each $i \in I$, $B_i$ is $\omega$-controllable. Since $\omega$-controllability is preserved under union, this implies that $\bigcup_{i \in I} B_i$ is also $\omega$-controllable; consequently $\bigcup_{i \in I} \subseteq A$. In order to show the reverse containment, it suffices to show that there exists $i \in I$ such that $A = B_i$, or equivalently, $A$ is relative $\omega$-closed and $pr(A)$ is controllable. Since $A$ is $\omega$-controllable, it follows from Lemma 5.3 that $pr(A)$ is controllable. Thus it remains to show that $A$ is relative $\omega$-closed.

Since $A \subseteq B \subseteq L_{\omega}(G)$, in order to show that $A$ is relative $\omega$-closed, it suffices to show $clo(A) \cap L_{\omega}(G) \subseteq A$. Since $A$ is the supremal $\omega$-controllable sublanguage of $B$, it is enough to show that $clo(A) \cap L_{\omega}(G)$ is an $\omega$-controllable sublanguage of $B$. First note that since $A \subseteq B$, and $B$ is relative $\omega$-closed,

$$\begin{aligned} clo(A) \cap L_{\omega}(G) &= [clo(A) \cap clo(B)] \cap L_{\omega}(G) \\ &= clo(A) \cap [clo(B) \cap L_{\omega}(G)] \\ &= clo(A) \cap B; \end{aligned}$$

which is a sublanguage of $B$. We show that $clo(A) \cap B$ is $\omega$-controllable by first showing that $clo(A) \cap B$ is relative $\omega$-closed and has a controllable prefix, and next applying the result of Corollary 5.1. Since $clo(A) \cap B$ is an intersection of two relative $\omega$-closed $\omega$-languages, $clo(A) \cap B$ is also relative $\omega$-closed (as a topological space is closed under intersection of its closed sets). Finally, since $A \subseteq B$, Lemma 5.4 implies $pr(clo(A) \cap B) = pr(A)$. Since $pr(A)$ is controllable, we obtain that $pr(clo(A) \cap B)$ is controllable, as desired. ■

In the next theorem we present a formula for computing $supC^{\omega}(B)$ when $B$ is given to be a limit $\omega$-language. (An iterative algorithm for computing $supC^{\omega}(B)$ for the general case is given as exercise problem 12.) We need the result of the following lemma:

**Lemma 5.5** Consider a plant $G$ and a $*$-language $K \subseteq \Sigma^*$. Let $B \subseteq lim(K)$ be such that $pr(B)$ is controllable. Then $pr(B) \cap K$ is a controllable and deadlock-free sublanguage of $K$.

**Proof:** Clearly, $pr(B) \cap K$ is a sublanguage of $K$. In order to see that $pr(B) \cap K$ is controllable, pick $s \in pr(pr(B) \cap K)$, $\sigma \in \Sigma_u$ such that $s\sigma \in L(G)$. Since $pr(pr(B) \cap K) \subseteq pr(pr(B)) = pr(B)$, and $pr(B)$ is controllable, it follows

that $s\sigma \in pr(B)$; consequently, there exists $e \in B$ such that $s\sigma < e$. Since $e \in B \subseteq lim(K)$, there exists infinitely many $n \in \mathcal{N}$ such that $e^n \in K$. Hence there exists $n_0 \in \mathcal{N}$ such that $s\sigma < e^{n_0} \in K$. Also, since $e^{n_0} < e$, $e^{n_0} \in pr(B)$. Thus $e^{n_0} \in pr(B) \cap K$, which implies that $s\sigma \in pr(pr(B) \cap K)$, as desired.

In order to see that $pr(B) \cap K$ is deadlock-free, pick $s \in pr(pr(B) \cap K) \subseteq pr(B)$. Then since $pr(B)$ is deadlock-free (as it is the prefix of an $\omega$-language), there exists $\sigma \in \Sigma$ such that $s\sigma \in pr(B)$. Then it can be concluded as above that $s\sigma \in pr(pr(B) \cap K)$, proving that $pr(B) \cap K$ is deadlock-free. ■

In the following theorem the notation $supCD(K)$ is used to denote the supremal controllable and deadlock-free sublanguage of $K \subseteq \Sigma^*$.

**Theorem 5.6** Given a plant $G$ and $B \subseteq L_\omega(G)$, let $K \subseteq \Sigma^*$ be such that $B = lim(K)$. Then $supC^\omega(B) = lim(supCD(K))$.

**Proof:** For simplicity of notation, define $A := supC^\omega(B)$ and $H := supCD(K)$. Then we need to show that $A = lim(H)$. Note that by definition, $A$ is $\omega$-controllable, and $H$ is controllable as well as deadlock-free.

We first prove the forward containment. Since $A$ is $\omega$-controllable, it follows from Lemma 5.3 that $pr(A)$ is controllable. Also, since $A \subseteq B = lim(K)$, it follows from Lemma 5.5 that $pr(A) \cap K$ is a controllable and deadlock-free sublanguage of $K$. Since $H$ is the supremal such language, this implies that $pr(A) \cap K \subseteq H$; consequently, $lim(pr(A) \cap K) \subseteq lim(H)$. From Lemma 5.1 we have

$$lim(pr(A) \cap K) = lim(pr(A)) \cap lim(K) = clo(A) \cap B.$$

Thus we obtain that $clo(A) \cap B \subseteq lim(H)$. Since $A \subseteq clo(A) \cap B$, this implies that $A \subseteq lim(H)$.

Next we show the reverse containment. Since $A$ is the supremal $\omega$-controllable sublanguage of $B$, it suffices to show that $lim(H)$ is an $\omega$-controllable sublanguage of $B$. Since $H \subseteq K$, clearly $lim(H) \subseteq lim(K) = B$, i.e., $lim(H)$ is a sublanguage of $B$. In order to see that $lim(H)$ is $\omega$-controllable, pick $s \in pr(lim(H)) = pr(H)$, where the last equality follows from Theorem 5.1 as $H$ is deadlock-free. We need to show that there exists $B_s \subseteq lim(H)\backslash\{s\}$ which is non-empty and relative $\omega$-closed with respect to $G\backslash\{s\}$ and such that $pr(B_s)$ is controllable with respect to $G\backslash\{s\}$. Define $B_s := lim(pr(H)\backslash\{s\}) = lim(H)\backslash\{s\}$. Since $s \in pr(H)$, $pr(H)\backslash\{s\} \neq \emptyset$. Also, since $H$ is deadlock-free, $pr(H)\backslash\{s\}$ is deadlock-free; hence $B_s = lim(pr(H)\backslash\{s\}) \neq \emptyset$. Next, since

$H$ is controllable, $pr(H)$ is also controllable. Hence $pr(lim(pr(H)\backslash\{s\})) = pr(H)\backslash\{s\}$ is controllable with respect to $G\backslash\{s\}$. Finally, to show that $B_s$ is relative $\omega$-closed with respect to $G\backslash\{s\}$, we show that $B_s$ is $\omega$-closed as follows:

$$\begin{aligned} clo(B_s) &= lim(pr(lim(pr(H)\backslash\{s\}))) \\ &= lim(pr(pr(H)\backslash\{s\})) \\ &= lim(pr(H)\backslash\{s\}) \\ &= B_s, \end{aligned}$$

where the second equality follows from Theorem 5.1 as $pr(H)\backslash\{s\}$ is deadlock-free, and the third equality follows from the fact that $pr(H)\backslash\{s\}$ is prefix closed. This completes the proof. ■

The following corollary is a consequence of Theorem 5.6, and provides a way of testing $\omega$-controllability of a limit $\omega$-language. (Refer to exercise problem 13.)

**Corollary 5.2** Given a plant $G$ and $B \subseteq L_\omega(G)$, let $K \subseteq \Sigma^*$ be such that $lim(K) = B$. Then $B$ is $\omega$-controllable if and only if $supD(K)$ is controllable.

### 5.3.3 Computation of Supervisors

In this subsection we give an algorithm for computing $supCD(K)$. Using the result of Theorem 5.6, this can be used for computing $supC^\omega(B)$, where $B = lim(K)$. Note that it is easy to obtain a iterative technique for computing $supCD(K)$ using lattice theoretic arguments (refer to exercise problem 5).

To ensure that our algorithm terminates in a finite number of steps, we require that $K$ be regular, which implies that $B = lim(K)$ is deterministic $\omega$-regular. Let $S := (Y, \Sigma, \beta, y_0, Y_m)$ be a trim DFSM $L_m(S) = K$, so that $L(S) = pr(L_m(S)) = K$ and $L_\omega(S) = lim(K) = B$. We also assume that the discrete event plant $G$ can also be represented as a DFSM $G := (X, \Sigma, \alpha, x_0, X_m)$. Then we have $L_\omega(G) = lim(L_m(G))$. Since $lim(L_m(S)) = lim(K) = B \subseteq L_\omega(G) = lim(L_m(G))$, we assume without loss of generality that $L_m(S) \subseteq L_m(G)$ (otherwise use $supD(K)$ instead of $K$ since $lim(supD(K)) = lim(K)$ and $lim(supD(K)) \subseteq lim(L_m(G))$ implies $supD(K) \subseteq L_m(G)$). We present an algorithm for constructing a DFSM with marked *-language $supCD(K)$. Since this state machine is deterministic, it follows that its non-terminating behavior equals $lim(supCD(K)) = supC^\omega(B)$, as desired.

The algorithm is obtained by modifying the algorithm for computing $supC(K)$ given earlier. First construct the DFSM $G||\bar{S} := (Z, \Sigma, \gamma, z_0, Z_m)$, where $\bar{S}$ denotes the completion of $S$, $Z := X \times (Y \cup \{y_D\})$ ($y_D \notin Y$ is the dump state), $z_0 := (x_0, y_0)$, and $Z_m := X_m \times Y_m$. It follows that $L_m(G||\bar{S}) = K$ and $L(G||\bar{S}) = L(G)$ (thus $L_\omega(G||S) = lim(L_m(G||S)) = lim(K) = B$).

The algorithm works as follows. It first identifies the set of initial "bad" states. These are the states in $Z$ which correspond to strings in $L(G) - pr(K)$ as well those strings in $K$ where "deadlock" occurs (step 1). A "good" state is declared to be bad whenever either it is possible to reach a bad state from it on an uncontrollable transition, or all transitions from it lead to bad states (step 2(b)). A good state is also declared to be bad if it does not belong to the "trim-component" of the state machine obtained by removing the bad states (step 2(d)). The algorithm terminates when it does not find any more bad states. The notation $(G||\bar{S})_l$ is used to denote the state machine obtained by removing the "bad" states (and transitions leading into them) at the $l$th iteration.

**Algorithm 5.1**

1. Initiation step:

   $Z_0 := \{(x, y_D) \mid x \in X\} \cup \{z \in X \times Y \mid \forall \sigma \in \Sigma : \gamma(z, \sigma) \text{ undefined}\}; k := 0.$

2. Iteration step:

   (a) $Z'_0 := Z_k; \quad l := 0.$

   (b) $Z'_{l+1} := Z'_l \cup \{z \in Z - Z'_l \mid [\exists \sigma \in \Sigma_u : \gamma(z, \sigma) \in Z'_l] \vee [\forall \sigma \in \Sigma : \gamma(z, \sigma) \in Z'_l]\}.$

   (c) If $Z'_{l+1} = Z'_l$, then go to step (d); else, set $l := l + 1$, and go to step (b).

   (d) $Z_{k+1} := Z'_l \cup \{z \in Z - Z'_l \mid z \notin Re_{(G||\bar{S})^R_l}(Z_m - Z'_l)\}.$

3. Termination step:
   If $Z_{k+1} = Z_k$, then stop; else, set $k := k + 1$, and go to step 2.

The DFSM with marked language $supCD(K)$ is obtained by removing all states in $Z_k$ (and all transitions leading into them) from $G||\bar{S}$. If the number of states in $G, S$ are $m, n$ respectively, then it can be checked that Algorithm 5.1 terminates in $O(m^2n^2)$ steps. Furthermore, if $K$ is prefix closed, then Algorithm 5.1 terminates in a single iteration, i.e., it takes $O(mn)$ steps to terminate (refer to exercise problem 14).

## 5.4 EXERCISES

1. Given $B \subseteq \Sigma^\omega, K \subseteq \Sigma^*$, prove that $B \subseteq lim(pr(B))$ and $pr(lim(K)) \subseteq pr(K)$.

2. For collections $\{B_i \subseteq \Sigma^\omega\}$ and $\{K_i \subseteq \Sigma^*\}$ prove that

   (a) $pr(\cup_i B_i) = \cup_i pr(B_i)$, whereas $pr(\cap_i B_i) \subseteq \cap_i pr(B_i)$.

   (b) $lim(\cup_i K_i) \supseteq \cup_i lim(K_i)$, with equality for finite union; whereas $lim(\cap_i K_i) \subseteq \cap_i lim(K_i)$.

3. Given $K \subseteq \Sigma^*$, show that

   (a) $K^\omega \subseteq lim(K^*)$; and the reverse containment does not necessarily hold.

   (b) If $lim(K) = \emptyset$, then $lim(K^*) \subseteq K^\omega$.

4. Consider the function function $d(\cdot, \cdot)$ as defined in Definition 5.1. Prove that

   (a) $d(\cdot, \cdot)$ is a metric over $\Sigma^\omega$.

   (b) Given $B \subseteq \Sigma^\omega$, it is closed (in the topology defined by the metric of Definition 5.1) if and only if $B = lim(pr(B))$.

5. Let $K \subseteq \Sigma^*$. Prove that

   (a) $K$ is deadlock-free if and only if any deterministic trim SM with marked language $K$ is deadlock-free.

   (b) $K$ is deadlock-free if and only if for each $s \in K$, there exists $t \in K$ such that $s < t$.

   (c) $K$ is deadlock-free if and only if $pr(K) \subseteq pr(K)/\Sigma$.

   (d) Using lattice theoretic arguments, and the result of the previous item show that $supD(K)$ and $supCD(K)$ exist. Provide an iterative technique for computing them.

6. Let $H, K \subseteq \Sigma^*$. Prove that

   (a) $lim(K) = lim(supD(K))$.

   (b) $lim(K) = lim(H)$ if and only if $supD(K) = supD(H)$.

7. Give BMs for accepting following $\omega$-languages over the event set $\Sigma = \{a, b\}$.

   (a) In each sequence accepted by the BM, an even number of $b$ occurs between any two occurrences of $a$.

(b) In each sequence accepted by the BM, either both $a$ and $b$ occur infinitely often or neither of them occur infinitely often.

(c) In each sequence accepted by the BM, after any occurrence of $a$ there is an occurrence of $b$.

8. Given an $\omega$-language model $(B, K_m, K)$, prove that there exists a NFSM $G$ such that
$$(L_\omega(G), L_m(G), L(G)) = (B, K_m, K)$$
if and only if $B$ is $\omega$-regular and $K_m$ as well as $K$ are regular.

9. A *Muller machine* is a "modified" DSM $G := (X, \Sigma, \alpha, x_0, \mathcal{X}_m)$, where $X$ is the finite state set, $\alpha : X \times \Sigma \to X$ is the deterministic partial transition function, $x_0 \in X$ is the initial state, and $\mathcal{X}_m \subseteq 2^X$ is a collection of subsets of $X$. Given $e \in lim(L(G))$, let the *recurrent states* of $e$, denoted $\Re(e) \subseteq X$, be defined as
$$\Re(e) := \{x \in X \mid \exists \text{ infinitely many } n \in \mathcal{N} \text{ s.t. } (\pi(e))(n) = x\}.$$
Then the $\omega$-language accepted by the Muller machine $G$, denoted $L_\omega(G) \subseteq lim(L(G))$ is defined to be:
$$L_\omega(G) := \{e \in lim(L(G)) \mid \exists X_m \in \mathcal{X}_m \text{ s.t. } \Re(e) = X_m\}.$$
Show that the class of $\omega$-languages accepted by Muller machine coincides with the $\omega$-regular class.

10. Given a plant $G$ and $B \subseteq L_\omega(G)$, prove that the supremal $\omega$-controllable sublanguage of $B$ exists.

11. Suppose $A \subseteq \Sigma^\omega$ is a limit $\omega$-language, and $B \subseteq A$ is relative $\omega$-closed with respect to $A$, i.e., $clo(B) \cap A = B$. Prove that $B$ is also a limit $\omega$-language.

12. Given a plant $G$, let $B \subseteq L_\omega(G)$ be a desired non-terminating behavior. Define the following decreasing sequence of $\omega$-languages:
    - $B_0 := B$
    - $\forall n \geq 0 : B_{n+1} := lim(supCD(pr(B_n))) \cap L_\omega(G)$

    Suppose $\bar{n} \geq 0$ is such that $B_{\bar{n}+1} = B_{\bar{n}}$; then show that $B_{\bar{n}} = supC^\omega(B)$.

13. Given a plant $G$, $B \subseteq L_\omega(G)$, and $K \subseteq \Sigma^*$ such that $lim(K) = B$, show that $B$ is $\omega$-controllable if and only if $supD(K)$ is controllable.

14. Consider Algorithm 5.1 that computes $supCD(K)$.

    (a) Prove the correctness of Algorithm 5.1.

(b) Prove that it terminates in $O(m^2n^2)$ steps where $m$ is the number of states in the plant and $n$ is the number of states in the DFSM with marked language $K$.

(c) Prove that when $K$ is prefix closed, then it terminates in $O(mn)$ steps.

15. Let $M : \Sigma \rightarrow \Delta \cup \{\epsilon\}$ be an observation mask. Given a plant $G$ and a desired non-terminating behavior $B \subseteq \Sigma^\omega$, give a necessary and sufficient condition for the existence of a $\Sigma_u$-enabling, non-marking, $\omega$-non-blocking and $M$-compatible supervisor $S$ such that $L_\omega(G||S) = B$.

## 5.5 BIBLIOGRAPHIC REMARKS

For a general treatment of $\omega$-language theory readers are referred to Eilenberg [Eil74] and Thomas [Tho90]. Supervisory control of non-terminating behavior was first studied by Ramadge [Ram89]. The notion of $\omega$-controllability is due to Thistle-Wonham [TW94b, TW94a]. Young-Spanjol-Garg [YSG92] also presented the equivalent notion of *finite stabilizability*. The notion of deadlock-free languages and its relation to $\omega$-languages was reported by Kumar-Garg-Marcus [KGM92]. The formula for the supremal $\omega$-controllable sublanguage is due to Kumar-Garg [KG94a].

# REFERENCES


[BGK+90] R. D. Brandt, V. K. Garg, R. Kumar, F. Lin, S. I. Marcus, and W. M. Wonham. Formulas for calculating supremal controllable and normal sublanguages. *Systems and Control Letters*, 15(8):111–117, 1990.

[CDFV88] R. Cieslak, C. Desclaux, A. Fawaz, and P. Varaiya. Supervisory control of discrete event processes with partial observation. *IEEE Transactions on Automatic Control*, 33(3):249–260, 1988.

[CL91] E. Chen and S. Lafortune. Dealing with blocking in supervisory control of discrete event systems. *IEEE Transactions on Automatic Control*, 36(6):724–735, 1991.

[DP90] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, UK, 1990.

[DS90] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics.* Springer-Verlag, New York, 1990.

[Eil74] S. Eilenberg. *Automata, Languages, and Machines: Volume A.* Academic Press, New York, NY, 1974.

[FYZ93] Jinghuai Fa, Xiaojun Yang, and Yingping Zheng. Formulas for a class of controllable and observable sublanguages larger than the supremal controllable and normal sublanguage. *Systems and Control Letters*, 20:11–18, 1993.

[Hey90] M. Heymann. Concurrency and discrete event control. *IEEE Control Systems Magazine*, 10(4):103–112, 1990.

[HL93] M. Heymann and F. Lin. On-line control of partially observed discrete event systems. Technical Report CIS-9310, Department of Computer Science, Technion-Israel Institute of Technology, Hafia, Israel, 1993.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1985.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, MA, 1979.

[KG94a] R. Kumar and V. K. Garg. Computation and formula for supremal $\omega$-controllable languages. *IEEE Transactions on Automatic Control*, 1994. Submitted.

[KG94b] R. Kumar and V. K. Garg. Extremal solutions of inequations over lattices with applications to supervisory control. In *Proceedings of 1994 IEEE Conference on Decision and Control*, Orlando, FL, December 1994. To Appear.

[KGM91] R. Kumar, V. K. Garg, and S. I. Marcus. On controllability and normality of discrete event dynamical systems. *Systems and Control Letters*, 17(3):157–168, 1991.

[KGM92] R. Kumar, V. K. Garg, and S. I. Marcus. On supervisory control of sequential behaviors. *IEEE Transactions on Automatic Control*, 37(12):1978–1985, December 1992.

[Kum93] R. Kumar. Formulas for observability of discrete event dynamical systems. In *Proceedings of 1993 Conference on Information Sciences and Systems*, pages 581–586, Johns Hopkins University, Baltimore, MD, March 1993.

[LNS82] J. L. Lassez, V. L. Nguyen, and E. A. Sonenberg. Fixed point theorems and semantics: A folk tale. *Information Processing Letters*, 14(3):112–116, 1982.

[LW88] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, 1988.

[Ram89] P. J. Ramadge. Some tractable supervisory control problems for discrete event systems modeled by buchi automata. *IEEE Transactions on Automatic Control*, 34(1):10–19, 1989.

[RW87a] P. J. Ramadge and W. M. Wonham. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, 1987.

[RW87b] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1):206–230, 1987.

[RW89] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of IEEE: Special Issue on Discrete Event Systems*, 77:81–98, 1989.

[RW90] K. Rudie and W. M. Wonham. The infimal prefix closed and observable superlanguage of a given language. *Systems and Control Letters*, 15(5):361–371, 1990.

[RW92] K. Rudie and W. M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, November 1992.

[Sme87] R. Smedinga. Using trace theory to model discrete events. In P. Varaiya and A. B. Kurzhanski, editors, *Discrete Event Systems: Models and Applications*, pages 81–99. Springer-Verlag, 1987.

[Tar55] A. Tarski. A lattice-theoretical fixed point theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[Thi94] J. G. Thistle. Logical aspects of control of discrete event systems: a survey of tools and techniques. In Guy Cohen and Jean-Pierre Quadrat, editors, *Lecture Notes in Control and Information Sciences 199*, pages 3–15. Springer-Verlag, New York, 1994.

[Tho90] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 135–187. Elsevier Science Publishers, 1990.

[Tsi89] J. N. Tsitsiklis. On the control of discrete event dynamical systems. *Mathematics of Control Signals and Systems*, 2(2):95–107, 1989.

[TW94a] J. G. Thistle and W. M. Wonham. Control of infinite behavior of finite automata. *SIAM Journal of Control and Optimization*, 1994. To Appear.

[TW94b] J. G. Thistle and W. M. Wonham. Supervision of infinite behavior of discrete event systems. *SIAM Journal of Control and Optimization*, 1994. To Appear.

[WR88] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete event systems. *Mathematics of Control Signals and Systems*, 1(1):13–30, 1988.

[YSG92] S. Young, D. Spanjol, and V. K. Garg. Control of discrete event systems modeled with deterministic Buchi automata. In *Proceedings of 1992 American Control Conference*, pages 2809–2813, Chicago, IL, 1992.

# INDEX