IOWA STATE UNIVERSITY
Digital Repository

Retrospective Theses and Dissertations

Iowa State University Capstones, Theses and Dissertations

2005

Decentralized/distributed failure diagnosis and supervisory control of discrete event systems

Wenbin Qiu
Iowa State University

Follow this and additional works at: https://lib.dr.iastate.edu/rtd

 Part of the Electrical and Electronics Commons

Recommended Citation
Qiu, Wenbin, "Decentralized/distributed failure diagnosis and supervisory control of discrete event systems " (2005). *Retrospective Theses and Dissertations*. 1850.
https://lib.dr.iastate.edu/rtd/1850

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact digirep@iastate.edu.

www.manaraa.com

Decentralized/distributed failure diagnosis and supervisory control of discrete event systems

by

Wenbin Qiu

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Electrical Engineering

Program of Study Committee:
Ratnesh Kumar, Major Professor
Nicola Elia
Andrew Miner
Simanta Mitra
Murti Salapaka

Iowa State University

Ames, Iowa

2005

Copyright © Wenbin Qiu, 2005. All rights reserved.

UMI Number: 3184605

INFORMATION TO USERS

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

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

UMI®

UMI Microform 3184605

Copyright 2005 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

Wenbin Qiu

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

DEDICATION

TO

my wife Jun Xu, and my parents Zhenshan Qiu & Faying Qian

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Discrete event systems (DESs) are event-driven systems, which change their discrete states upon asynchronous occurrence of certain events. Examples of DESs include telecommunication networks, robotic and manufacturing systems, computer networks, reactive programs, etc.. This dissertation addresses decentralized/distributed failure diagnosis and supervisory control of DESs.

In a decentralized diagnosis architecture, a local diagnoser performs failure diagnosis completely based on its own observations without communicating with others. A notion of codiagnosability is introduced to capture the property that a system should satisfy such that its failure behaviors are diagnosable by one of the local diagnosers within a bounded delay of their occurrences. Algorithms with polynomial complexity in the size of system/specification models are presented for verifying codiagnosability, computing diagnosis delay bound, synthesizing local diagnosers, and online diagnosis using them. Further diagnosis properties are investigated through the introduction of strong-(co)diagnosability and safe-codiagnosability.

In a distributed diagnosis architecture, local diagnosers exchange their individual observation with each other to perform failure diagnosis collaboratively. The communication can introduce bounded or unbounded delay. Finite automata models are constructed to capture communication delays, and the system/specification/sensing models are augmented with respect to the communication delay models. Via those augmented models, a distributed diagnosis problem (with communication) is converted to a decentralized diagnosis problem (without communication). This allows distributed diagnosis analysis to be performed in same as decentralized diagnosis analysis. Also, in the unbounded delay case decidability of the problem is established, in contrast to a prior work which conjectured it to be an undecidable problem.

For the decentralized supervisory control of DESs, prioritized synchronous composition (PSC) based decentralized control and nondeterministic decentralized control are introduced. A PSC based decision fusion rule is more general than the conventional conjunctive/disjunctive decision fusion rule since it has control-authority besides control-capability. Algorithms are presented for existence and synthesis of PSC based supervisors. Computational complexity of the former is polynomial in the size of both system and specification models, while complexity of the latter is polynomial in the size of systems model, and exponential in the size of specification model. By using nondeterministic supervisors, a weaker condition than the condition of controllability together with co-observability, which serves as a necessary and sufficient condition for the existence of deterministic decentralized supervisors, is obtained for decentralized control. Algorithms of polynomial complexity are presented for both existence and synthesis of nondeterministic supervisors in target control and range control problems.

CHAPTER 1. INTRODUCTION

1.1 Discrete Event Systems

Discrete event systems (DESs) are event-driven systems, which change their discrete states upon asynchronous occurrence of certain events. States in DESs are represented by some symbolic variables. For example, a machine in a manufacturing system may have three discrete states: idle, working, and broken states. Events in DESs are some discrete qualitative changes. Examples of events include the arrival of a message in a communication channel, completion of operations in manufacturing systems, failure of sensors/actuators in control systems, termination of computer programs, etc.. Thus, many man-made systems are examples of DESs such as telecommunication networks, robotic and manufacturing systems, computer networks, and reactive programs. Also, continuous behaviors in a system can be modeled at a certain level of abstraction in the framework of DESs.

In general, there are two types of DESs: untimed DESs and timed DESs. Untimed DESs focus on logic behaviors of a system without considering timing properties of the system. Thus dynamics of untimed DESs are determined by the order/sequence of states or events, not by their timing properties. In timed DESs, system behaviors are affected by timing properties, which are captured by extra "timing" events or states with clock ticks. In this dissertation, we are focused on untimed DESs based on Ramadge-Wonham's modeling framework [53].

Since state transitions in a DES are generally irregular with respect to real time, we cannot use differential or difference equations to describe behaviors of DESs. Instead, formal languages and finite automata or state machines are used as fundamental models for analyzing logic behaviors of DESs. Due to its interdisciplinary nature, research on DESs brings together ideas and techniques from computer science, control theory, and operations research. In this

dissertation, we study two important problems of DESs in decentralized or distributed settings, the failure diagnosis and supervisory control problems.

1.2 Failure Diagnosis of DESs

Failure diagnosis is critical to achieve fault-tolerance and high-performance of large complex systems. Due to its importance, the problem of failure diagnosis has received considerable attention in the literature. Various approaches have been proposed including fault-trees, expert systems, neural networks, fuzzy logic, Bayesian networks, and analytical redundancy [50]. These are broadly categorized into non-model based (where observed behavior is matched to known failures), and model based (where observed behavior is compared against model predictions for any abnormality). For discrete event systems a certain model based approach for failure diagnosis is proposed in [60], and extended in [59, 22, 25, 24, 13, 77]. The application of DESs failure diagnosis includes heating, ventilation, and air conditioning systems [61], transportation systems [41, 15], communication networks [3, 1, 42], manufacturing systems [10, 45], digital circuits [37, 70], and power systems [16].

Failure diagnosis in DESs requires that once a failure occurred, it be detected and diagnosed within bounded "delay" (bounded number of transitions). This is captured by the notion of failure diagnosability introduced in [60]. Polynomial tests for diagnosability are given in [21, 76]. In [59], the notion of active failure diagnosis was introduced where control is exercised to meet given specifications while satisfying diagnosability. In [10, 45], a template based approach was developed for failure diagnosis in timed discrete event system. [58] also studied failure diagnosis in timed DESs.

The above approaches can be thought to be "event-based" as failure is modeled as execution of certain "faulty events". An equivalent "state-based" approach was considered in [37, 77], where the occurrence of a failure is modeled as reaching of certain "faulty states". A theory for failure diagnosis of repeatedly-occurring/intermittent failures was introduced in [25]. The notion of diagnosability was extended to $[1, \infty]$-diagnosability to allow diagnosis of a failure each time it occurred. Polynomial complexity algorithms for testing $[1, \infty]$-diagnosability as

well as for off-line diagnoser synthesis were presented in [25]. Algorithms of complexity that are an order lower have been reported in [74]. To facilitate generalization of failure specifications, linear-time temporal logic (LTL) based specification and diagnosis of its failure was proposed in [22]. LTL can be used to specify violations of safety as well as liveness properties, allowing diagnosis of failures that have already occurred (safety violations) as well as prognosis of failures that are inevitable in future (liveness failures). [24] extended the use of LTL based specifications for representing and diagnosing repeatedly-occurring/intermittent failures.

The above mentioned work dealt with *centralized* failure diagnosis, where a central diagnoser is responsible for failure detection and diagnosis in the system. Many large complex systems, however, are physically distributed which introduces variable communication delays and communication errors when diagnosis information collected at physically distributed sites are sent to a centralized site for analysis. Consequently, although all diagnosis information can be gathered centrally, owing to the delayed/corrupted nature of the data, a centralized failure diagnosis approach may not always be appropriate for physically distributed systems, and instead diagnosis may need to be performed decentrally at sites where diagnosis information is collected.

[13, 55, 62, 2, 65] studied distributed diagnosis in which diagnosis is performed by either diagnosers communicating with each other directly or through a coordinator and thereby pooling together the observations. [13] addressed the problem of distributed failure diagnosis based on a "coordinated decentralized architecture", where local diagnosers do not communicate with each other directly, but send local information to a coordinator. Then the coordinator makes the final diagnosis decision. Later, they extended their work by considering one-step out-of-order transmission caused by communication delays [12]. [62] discussed the distributed diagnosis problem, where communication directly exists between local diagnosers, and is assumed to be lossless, and in order. Notion of "decentralized diagnosis" was formulated, which was proved to be undecidable. We will show in this dissertation that decentralized diagnosis is not an adequate property to capture distributed diagnosability under unbounded delay communication, and a stronger notion is required, which is further shown to be decidable. [55] studied the

diagnosis problem of a timed discrete event system based on the asynchronous communication between diagnosers. The decentralized diagnosis problem with asymmetric communication was discussed in [2], where communication is one-way and without delays. Aiming to increase the scalability and robustness of diagnosers, an automaton-based architecture was proposed for distributed diagnosis in [65].

The problem of *decentralized* diagnosis was first considered as one special case of *distributed* diagnosis in [13]. In that paper, "lack of fully ambiguous traces" was stated as a sufficient condition for decentralized diagnosis to be equivalent to that of centralized one, and an algorithm was presented for verifying the "lack of fully ambiguous traces". The algorithm was based upon structural properties of global (centralized) and local (decentralized) diagnosers, which has an exponential complexity in the size of the system owing to the exponential size of the diagnosers.

1.2.1 Our Study on Decentralized/Distributed Failure Diagnosis

For decentralized diagnosis, instead of capturing a condition under which decentralized diagnosis is equivalent to centralized diagnosis as in [13], this dissertation studies the condition a system should satisfy such that its failure behaviors can be diagnosed in a decentralized setting. We make the notion of decentralized diagnosis involving no communication among diagnosers precise by introducing the notion of codiagnosability that requires that the occurrence of any failure be diagnosed within bounded delay by at least one local diagnoser using its own observations of the system execution. The property of codiagnosability is stronger than that of diagnosability (under the aggregated observations). In other words, it is possible that a system is centrally diagnosable under the aggregated observations, but not decentrally diagnosable. However, when a centralized diagnosis is not possible (due to the physically distributed nature of the underlying system), the system must satisfy the stronger property of codiagnosability to allow for the detection/diagnosis of each failure by some local diagnosers.

In this dissertation we represent a failure as violation of a specification represented as a language, and also as the execution of certain failure events. We study codiagnosability first

in specification language framework and later specialize it to the failure event framework. The specification language framework can be viewed as a generalized state/event-based framework (in those frameworks the model generating the specification language is a subautomaton of the system model). We present algorithms of complexity polynomial in the size of the system and the non-fault specification for (i) testing codiagnosability, (ii) computing the delay bound of diagnosis, (iii) off-line synthesis of diagnosers, and (iv) on-line diagnosis using them.

The computation of the delay bound is important for the following reason. After a failure is detected/diagnosed, a failure recovery procedure ought to be initiated. There may be requirements on how late such recovery procedures may be initiated from the time the failure occurred. If delay of diagnosis is longer than the delay by which the recovery procedures are to be initiated, satisfaction of diagnosability is of little use. For centralized diagnosis, [60] presented a method to compute the delay bound. The method is based on the construction of a diagnoser and has exponential complexity. In [76], the authors presented a method of polynomial complexity for determining the delay bound for centralized diagnosis. In this paper, a notion of delay associated with decentralized diagnosis is introduced, and a polynomial algorithm for computing it is provided.

The notion of codiagnosability guarantees that occurrence of any failure is detected within finite delay by one of the diagnosers, but there is no guarantee that the non-occurrence of failure is unambiguously known. To capture the capability of being certain about the failure as well as non-failure conditions in a system within bounded delay, we introduce the notion of strong-codiagnosability. The corresponding notion of strong-diagnosability can also be defined for the centralized setting. We illustrate that a diagnosable (resp., codiagnosable) system need not be strongly-diagnosable (resp., strongly-codiagnosable).

In order to react to a failure in a timely fashion, while it is necessary that the failure be detected within a bounded delay, such a property alone is not sufficient. It is also needed that the detection occur before the system behavior becomes "unsafe". To capture this additional requirement for failure detection, the notion of *safe-diagnosability* was introduced in [46]. We extend this notion to the decentralized setting by formulating the notion of *safe-*

codiagnosability. The safe behavior includes all of non-faulty behavior and some of post-fault behavior where system performance may be degraded but still tolerable. Safe-codiagnosability requires that when the system executes a trace that is faulty, then exists at least one diagnoser that can detect this within bounded delay and also before the safety specification is violated. We give an algorithms of polynomial complexity for verifying safe-codiagnosability. (The verification algorithm presented in [46] was based upon the structural property of a deterministic diagnoser, and had an exponential complexity owing to the exponential size of the diagnoser.)

To capture the effect of communication delays in distributed diagnosis, a notion of *joint$_k$-diagnosability* is introduced so that any failure can be diagnosed within a bounded delay of its occurrence by one of the local sites using its own observations and the information received from other local sites. Then we discuss the distributed diagnosis problem under an immediate observation passing protocol, and convert it to a decentralized diagnosis problem. Results for analyzing codiagnosability are applied for verifying joint$_k$-diagnosability under that protocol, and synthesizing local diagnosers.

For distributed diagnosis under unbounded communication delay, we show that it is a decidable problem by converting it to an instance of decentralized diagnosis problem. This is in contrast to the result in [62], where the authors claimed that the problem is undecidable. Our study shows that the decentralized diagnosability property introduced in [62] is not an adequate to analyze this problem. Instead, we capture the essence of the problem by joint$_\infty$-diagnosability, and show that this property is equivalent to codiagnosability, which is known to be decidable. We also demonstrate that joint$_\infty$-diagnosability is strictly stronger than decentralized-diagnosability.

1.3 Supervisory Control of DESs

Supervisory control of DES is a model-based control problem. The *feasible* behaviors of system to be controlled, called *plant*, are captured by plant models, which usually are finite automata or formal languages. The *desired* behaviors of the controlled system are captured by specification models. Specification models can also be in the format of finite automata or

formal languages. To facilitate the modeling of system requirements, some other formats of models are used as well, such as rule-based models [7, 8] and temporal logics [66, 36, 43, 39, 40]. In this dissertation, finite automata and formal language are used for modeling DESs.

The problem of supervisory control is to design a controller such that the controlled system does not violate the specification. In the setting of formal language, that means that the generated language of the controlled system is a subset of the specification language. The controller designed is *maximal permissive*. I.e., it only disables certain transitions/events, which otherwise would lead the system to states violating the specification, and does not determine specific events to be executed at a certain state. Thus, this control problem is called *supervisory* control, and the controller designed is called a *supervisor*. The goal of supervisory control is to allow plant behaviors the maximum freedom without violating the specification, and disable as least events as possible.

Supervisory control of DESs under complete observation was first proposed by Ramadge and Wonham [54]. Later Cieslak *et al.* [9] and Lin and Wonham [38] studied supervisory control under partial observation and introduced the notion of *observability*. When the plant is physically distributed, multiple supervisors are used for control, called decentralized control. In this setting the condition of *coobservability* was formalized by Cieslak *et al.* [9] and Rudie and Wonham [57].

1.3.1 Decentralized Decision Fusion Architecture

An important issue in decentralized control is how to fuse together control decisions from local supervisors. Conventionally, the *conjunctive* fusion rule is used, where an event is globally enabled if and only if it is enabled by all local supervisors. A non-conjunctive fusion rule based decentralized control was first considered by Prosser *et al.* [51]. Yoo and Lafortune [75] developed a more comprehensive theory for decentralized control based on the *conjunctive+disjunctive* fusion rules. In this generalized architecture, controllable events are partitioned into two disjoint sets: one where fusion occurs using conjunctive fusion rule, and another where fusion occurs using the disjunctive fusion rule.

Control of DESs and interaction among local supervisors can be achieved through composition operations taken over their automata representations [19, 47]. The most common mode of composition is taken to be *strict synchronous composition* (SSC) [54, 26, 9, 71, 28]. Since feasible uncontrollable events cannot be disabled by a supervisor, the supervisor is required to synchronously execute all the uncontrollable events that the plant can execute.

To relax such a synchronization requirement on interacting systems (such as plant and supervisor), Heymann [17, 18] introduced the notion of *prioritized synchronous composition* (PSC). In PSC, a priority set is associated with each system. For an event to be enabled in the interconnected system, it must be enabled in all systems whose priority sets contain that event. In the setting of supervisory control, the priority set of a supervisor is chosen to be the set of controllable events, thereby a supervisor is not required to participate in the occurrence of uncontrollable events. Supervisory control of DESs via PSC has been studied in [63, 32, 33, 30, 23].

Decentralized control of discrete event systems via PSC was first studied in [33], where it was required that the priority sets of the supervisors exhaust the entire controllable event set. In another words, each controllable event needs the participation of the supervisors having the priority over it. However, as is shown in this dissertation, such a requirement is restrictive, and also may not even be necessary. For example, in a manufacturing system, a machine under control of several supervisors may start to work on command from *any* supervisor. In that case, the control action of that machine is outside the priority sets of all supervisors.

In this dissertation, we study PSC-based decentralized control without the restriction imposed in [33]. As shown in Chapter 5, without that restriction the controller can achieve a strictly larger class of behavior. We introduce the notion of *PSC-coobservability*, which together with controllability captures the property of specifications achievable using PSC-based decentralized control. A polynomial algorithm is presented for verifying PSC-coobservability. Under certain restrictions on the priority sets, PSC-coobservability is reduced to C&P $\vee$ D&A-coobservability [75], and when there is flexibility in choosing the priority and conjunction/disjunction sets, the class of PSC-coobservable and C&P $\vee$ D&A-coobservable languages

coincide. Otherwise, PSC-based decentralized control should be considered a generalization of conjunction+disjunction-based decentralized control since it has the mechanism to support *control-authority* besides *control-capability.*

We also present an algorithm to construct automata realizations of local PSC-based supervisors. It is well known that there exist such realizations of supervisors for centralized control, and for conjunction-based decentralized control when plant and specification both have finite automata representations [34]. For example, supervisors can be chosen to be generators of infimal prefix-closed controllable and observable superlanguages with respect to local controllable events and local observation masks. This ceases to work in the non-conjunctive case as we illustrate through an example in Chapter 5. In [75], authors constructed the supervisors based on the construction of observers, and the control action at each observer state is computed manually. Further the state-space of the observer is exponential in both the plant and specification states, whereas the complexity of our algorithm is polynomial in plant states and exponential only in specification states. (A lower complexity algorithm is unlikely owing to the NP-completeness result of supervisor synthesis under partial observation [69].) A nice property of our synthesis method is that it is also applicable to centralized control, conjunction-based decentralized control, and non-conjunction-based decentralized control.

1.3.2 Supervisory Control Using Nondeterministic Supervisors

Most prior work on control of qualitative/logical behavior of DESs is based on deterministic controllers/supervisors. Inan [20] first advocated the use of nondeterministic supervisors for control under partial observation. Recently, Kumar *et al.* [31] proposed the notion of achievability to capture properties of more general nondeterministic supervisors, where "change in control action without any new observation", and non-projection type observation masks are allowed. Further, the form in which the condition of achievability was defined, it allowed the separation of limitations caused by "partial-control" and partial-observation. This paper extends the work in [31] from the centralized setting to the decentralized setting with the rule for decision fusion being based on conjunction (an event is enabled if and only if all supervisors

having control over the event enable it).

A deterministic (resp., nondeterministic) control policy can be represented a (Σ_u, M)-compatible deterministic (resp. nondeterministic) state machine (DSM (reps., NSM)), where Σ_u is the uncontrollable event set and M is the observation mask. (Σ_u, M)-compatibility is equivalent to the Σ_u-compatibility and M-compatibility combined. Σ_u-compatibility requires that no uncontrollable event can be disabled at any state of the supervisor, and M-compatbility requires that any pair of indistinguishable events defined at a certain state must have the the same set of successor states. The notion of (Σ_u, M)-achievability was introduced in [31] to characterize the class of languages that can be enforced as the controlled behavior by (Σ_u, M)-compatibile nondeterministic supervisors. It was shown that achievability is a weaker condition than controllability and observability combined, which serves as a necessary and sufficient condition for the existence of deterministic control [38, 9].

Allowing supervisors to be nondeterministic offers several advantages. For example, although the property of controllability and observability can be tested polynomially [53, 73, 28, 69], an off-line computation of a deterministic supervisor for control under partial observation has an exponential complexity [69]. In contrast, in the nondeterministic setting, both the existence test and synthesis of nondeterministic supervisors can be performed polynomially. Further, achievability is preserved under union and intersection over prefix-closed languages, implying the existence of the supremal achievable sublanguages and the infimal prefix-closed and achievable superlanguages, respectively. In contrast, observability is only preserved under set intersection over prefix-closed languages [38].

In decentralized control, when the local supervisors are required to be deterministic, the condition of co-observability together with controllability serves as a necessary and sufficient condition for the existence of decentralized supervisors. A weaker condition of existence, which we introduce as co-achievability, is needed when supervisors are allowed to be nondeterministic. Alike co-observability, co-achievability is also preserved under intersection over prefix-closed languages implying the the existence of infimal co-achievable superlanguages. On the other hand, co-achievability is not preserved under union. This is in contrast to achievability (which

has been shown to be preserved under union in [31]), implying that the decentralized control problems are inherently more difficult than the centralized ones. This difficulty is also witnessed in the papers reporting undecidability of the decentralized estimation and control problems [35, 67].

Based on the algorithms for centralized nondeterministic control, we develop algorithms for testing co-achievability and synthesis of decentralized supervisors for the "target control", and show they can be polynomially performed. Similarly, the existence test and synthesis of decentralized nondeterministic supervisors for the "range control" (specified as a lower bound and upper bound languages) is of polynomial complexity. This is in contrast to decentralized deterministic control, where both the existence and synthesis is of exponential complexity. Thus there are three main advantages to allowing nondeterministic supervisors for decentralized control: (i) weaker condition for existence, (ii) For target control problem synthesis is polynomial, (iii) For range control problem both existence and synthesis are polynomial.

1.4 Organization of Dissertation

This dissertation is organized as follows:

In Chapter 2, we present necessary notation and preliminaries for this dissertation. First, basic DES theories based on finite automata and formal languages are introduced. Then previous results on failure diagnosis and supervisory control of DESs relevant to the development of our work will be presented. Various notions, such as diagnosability, controllability, observability, and coobservability, will be introduced.

In Chapter 3, we study decentralized failure diagnosis problem, where no communication is involved among local diagnosers and each diagnoser performs diagnosis completely based on its own observation. We first formalize this problem by introducing a notion of codiagnosability. Then algorithms with polynomial complexity in the size of system and specification models are presented for verifying codiagnosability, synthesizing local diagnosers, and online diagnosis using them. Also, properties of strong-diagnosability and strong-codiagnosability are introduced to capture the capability of a diagnosis system being certain about its fault/non-fault

status. For safety-critical systems, an additional property of safe-codiagnosability is presented to guarantee that safety specification is not violated when reacting to system failures, though tolerable performance degradation may be expected.

In Chapter 4, we study distributed failure diagnosis under bounded/unbounded communication delay, where multiple diagnosers communicate with each other by sharing their local observations or other diagnosis information to achieve diagnosis purpose collaboratively. We define a notion of joint$_k$-diagnosability so that any failure can be diagnosed within a bounded delay of its occurrence by one of the local sites using its own observations and the communicated information from other local sites. For an immediate observation passing protocol, models are constructed to capture communication delays, and to extend system/specification models. Then distributed diagnosis problem based on such protocol is converted to a decentralized diagnosis problem, and is solved by extending algorithms introduced for decentralized diagnosis in Chapter 3. Also, we show the decidability of distributed failure diagnosis under unbounded communication delay.

In Chapter 5, we discuss decentralized supervisory control using prioritized synchronization based decision fusion rules. A necessary and sufficient condition is introduced for the existence of decentralized supervisors in this fusion architecture. Properties of the novel decision fusion architecture and comparison with conjunction/disjunction decision fusion architecture are presented. And an algorithm for synthesizing supervisors is presented with complexity polynomial in plant and exponential in specification size.

In Chapter 6, we extend nondeterministic supervisory control to the decentralized setting. We first introduce a weaker condition, termed as co-achievability, for the existence of nondeterministic supervisors. Then polynomial algorithms are presented for verifying such condition, and synthesize local supervisors for target and range control problems.

In Chapter 7, we summarize this dissertation, and discuss directions of future research.

CHAPTER 2. NOTATION AND PRELIMINARIES

In this chapter, we first present some notation and preliminaries in formal languages and finite automata/finite state machine theories, which will be used in modelinng DESs. Then we introduce some preliminary theories on failure diagnosis and supervisory control of DESs. For more details on DESs theories, readers are referred to [54, 26, 5].

2.1 Formal Languages, and Finite Automata/Finite State Machines

Given an event set Σ, Σ^* is used to denote the set of all finite length event sequences over Σ, including the zero length event sequence ϵ. A member of Σ^* is a *trace* and a subset of Σ^* is a *language*. Given a language $K \subseteq \Sigma^*$, the *complement* of K, denoted $K^c \subseteq \Sigma^*$, is defined as $K^c := \Sigma^* - K$. The *supremal prefix-closed sublanguage* of K, denoted $supP(K) \subseteq K$, is defined as, $supP(K) := \{s \in K | pr(s) \subseteq K\}$. The *quotient* of K_1 with respect to K_2 is defined as $K_1/K_2 := \{s \in \Sigma^* | \exists t \in K_2 \text{ s.t. } st \in K_1\}$.

A DES is modeled as a *finite automaton* (FA)/*finite state machine* (FSM) G and is denoted by $G := (X, \Sigma, \alpha, x_0, X_m)$, where X is the set of states, Σ is the finite set of events, $x_0 \in X$ is the initial state, X_m is the marked state, and $\alpha : X \times \overline{\Sigma} \rightarrow 2^X$ is the transition function, where $\overline{\Sigma} := \Sigma \cup \{\epsilon\}$. G is said to be *deterministic* if $|\alpha(\cdot, \cdot)| \leq 1$ and $|\alpha(\cdot, \epsilon)| = 0$; otherwise, it is called *nondeterministic*. $(x, \sigma, x') \in X \times \overline{\Sigma} \times X$ is a transition of G if $x' \in \alpha(x, \sigma)$; it is an ϵ/silent-transition if $\sigma = \epsilon$. Throughout this dissertation, it is assumed that a silent transition (x, ϵ, x) is defined at each state x of a nondeterministic state machine. Letting $\epsilon^*(x)$ denote the set of states reachable from x in zero or more ϵ-transitions, the transition function α can

be extended from domain $X \times \overline{\Sigma}$ to domain $X \times \Sigma^*$ recursively as follows:

$$\forall x \in X, s \in \Sigma^*, \sigma \in \Sigma: \ \alpha(x, \epsilon) = \epsilon^*(x), \text{ and } \alpha(x, s\sigma) = \epsilon^*(\alpha(\alpha(x, s), \sigma))).$$

The *generated language* and the *marked language* of G can respectively be defined as follows: $L(G) := \{s \in \Sigma^* | \alpha(x_0, s) \text{ is defined}\}$, and $L_m(G) := \{s \in L(G) | \alpha(x_0, s) \in X_m\}$. (In this dissertation, if we are only concerned with generated behavior, we omit the marked states from the tupe-notation.) G is said to be *accessible* if all states in G are reachable from the initial state. G is said to be *co-accessible* if at least one marked state is reachable from each state of G. G is said to be *trim* if it is accessible and co-accessible. The prefix-closure of a language $K \subseteq \Sigma^*$, denoted $pr(K)$, is the set of all prefixes of traces in K. K is said to be *prefix-closed* if $pr(K) = K$. A language K is said to be *relative-closed* (with respect to $L_m(G)$) if $pr(K) \cap L_m(G) = K$. The set of *deadlocking traces* of G are those traces from which no further extensions exist in G, i.e., $s \in L(G)$ is deadlocking trace if $\{s\}\Sigma^* \cap L(g) = \{s\}$. States reached by execution of deadlocking traces in $L(G)$ are called deadlocking states. A *path* in G is a sequence of transitions $(x_1, \sigma_1, x_2, \cdots, \sigma_{n-1}, x_n)$, where $\sigma_i \in \overline{\Sigma}$ and $x_{i+1} \in \alpha(x_i, \sigma_i)$ for all $i \in \{1, \cdots, n-1\}$. The path is called a *cycle* if $x_1 = x_n$.

Given an automaton $G = \{X, \Sigma, \alpha, x_0\}$, the complete model of G is defined as $\overline{G} = \{\overline{X}, \Sigma, \overline{\alpha}, x_0\}$, where $\overline{X} := X \cup \{F\}$, and $\overline{\alpha}$ is defined as follows. $\forall \overline{x} \in \overline{X}, \sigma \in \Sigma, \overline{\alpha}(\overline{x}, \sigma) :=$

$$\begin{cases} \alpha(\overline{x}, \sigma), & \text{if } [\overline{x} \in X] \wedge [\alpha(\overline{x}, \sigma) \neq \emptyset] \\ F, & \text{if } [\overline{x} = F] \vee [\alpha(\overline{x}, \sigma) = \emptyset] \end{cases}.$$

Since all events are defined at each state, the complete model $\overline{G}$ generates the language Σ^*, i.e., $L(\overline{G}) = \Sigma^*$.

Given two automata $G = (X, \Sigma, \alpha, x_0)$ and $R = (Y, \Sigma, \beta, y_0)$, the *synchronous composition* of G and R is defined as, $G||R = (X \times Y, \Sigma, \gamma, (x_0, y_0))$ such that

$$\forall (x, y) \in X \times Y, \sigma \in \overline{\Sigma}, \gamma((x, y), \sigma) := \begin{cases} \alpha(x, \sigma) \times \beta(y, \sigma), & \text{if } \sigma \neq \epsilon; \\ (\alpha(x, \epsilon) \times \{y\}) \cup (\{x\} \times \beta(y, \epsilon)), & \text{otherwise.} \end{cases}$$

It is easy to see that $L(G_1 \| G_2) = L(G_1) \cap L(G_2)$.

If the system execution is observed through a single global observer, we can define a *global observation mask* as $M : \Sigma \cup \{\epsilon\} \rightarrow \Lambda \cup \{\epsilon\}$ satisfying $M(\epsilon) = \epsilon$, where $\epsilon \notin \Lambda$ and Λ is the set of observed symbols. An event mapped to ϵ is an unobservable event. The definition of M can be extended from events to event sequences inductively as follows:

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

$M^{-1}M(s) := \{t \in \Sigma^* | M(s) = M(t)\}$ denotes the set of traces which are indistinguishable from s. Given an automaton G and mask M, $M(G)$ is the *masked automaton* of G with each transition (x, σ, x') of G replaced by $(x, M(\sigma), x')$. The *local observation masks* associated with different local observers are defined as $M_i : \overline{\Sigma} \rightarrow \overline{\Lambda}_i$ $(i \in I = \{1, \cdots, m\})$, where m is the number of local observers, $\overline{\Lambda}_i := \Lambda_i \cup \{\epsilon\}$ and Λ_i is the set of locally observed symbols.

2.2 Failure Diagnosis of DESs

Let $G = (X, \Sigma, \alpha, x_0)$ and $R = (Y, \Sigma, \beta, y_0)$ represent the *plant* and the *specification* models, respectively. Then the generated language of the plant, $L = L(G)$, represents the *feasible* behavior of the system, whereas the specification language, $K = L(R)$, represents the *fault-free* behavior of the system. The *completed specification model* $\overline{R}$ is constructed from R by adding an additional failure state "F", which when reached due to the execution of a trace feasible in the system indicates the occurrence of a failure. Formally, $\overline{R} := (\overline{Y}, \Sigma, \overline{\beta}, y_0)$, where $\overline{Y} := Y \cup \{F\}$, and $\overline{\beta}$ is defined as: $\forall \overline{y} \in \overline{Y}, \sigma \in \Sigma$,

$$\overline{\beta}(\overline{y}, \sigma) := \begin{cases} \beta(\overline{y}, \sigma), & \text{if } [\overline{y} \in Y] \wedge [\beta(\overline{y}, \sigma) \neq \emptyset], \\ F, & \text{if } [\overline{y} = F] \vee [\beta(\overline{y}, \sigma) = \emptyset]. \end{cases}$$

The failure diagnosis problem is to detect and diagnose any failure behavior in $L - K$ within a bounded delay of its execution. Execution of any such behavior is viewed as the occurrence of a fault. When there does not exist any communication among the local diagnoser sites, it

is called a *decentralized* failure diagnosis problem; otherwise, it is called a *distributed* failure diagnosis problem.

2.3 Supervisory Control of DESs

For the purpose of control, the event set Σ is partitioned into two disjoint subsets: $\Sigma = \Sigma_c \cup \Sigma_u$, where Σ_c is the set of controllable events and Σ_u is the set of uncontrollable events. Let $\Gamma \subseteq 2^{\Sigma}$ denote the set of control actions (set of enabled events) with $\Sigma_u \subseteq \gamma$ for each $\gamma \in \Gamma$. A supervisor $S : L(G) \to \Gamma$ is a function from the generated language $L(G)$ to the set of control actions. The system under the control of S is denoted by S/G. $L(S/G)$ denotes the controlled generated language, whereas the controlled marked language is defined as $L_m(S/G) := L(S/G) \cap L_m(G)$.

Let a nonempty language $K \subseteq L(G)$ represent the desired behavior. A basic control problem is to design a supervisor S such that $L(S/G) = K$, and the nonblocking control problem is to design a supervisor S such that $L_m(S/G) = K$ and $L(S/G) = pr(L_m(S/G))$. A supervisor S satisfying $L(S/G) = pr(L_m(S/G))$ is called a *nonblocking supervisor*. K is said to be $(L(G), \Sigma_u)$-*controllable* if $pr(K)\Sigma_u \cap L(G) \subseteq pr(K)$, and it is said to be $(L(G), M)$-*observable* if

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

There exists a supervisor S such that $L(S/G) = K$ (resp., $L_m(S/G) = K$ and S is nonblocking) if and only if K is prefix-closed (resp., relative-closed), $(L(G), \Sigma_u)$-controllable, and $(L(G), M)$-observable [38].

In the decentralized setting, the control decision is generated from several local supervisors. Assume there are n local supervisors, and let $I := \{1, \cdots, n\}$ denote the index set of all local supervisors. Each local supervisor has its own controllable event set Σ_{ci} and observation mask $M_i : \Sigma \cup \{\epsilon\} \to \Lambda_i \cup \{\epsilon\}$. Collectively, the overall controllable event set is $\Sigma_c = \bigcup_{i \in I} \Sigma_{ci}$, and the uncontrollable event set is $\Sigma_u = \Sigma - \Sigma_c$. For each $s \in L(G)$, the decision made by the local

supervisor i is $S_i(M_i(s))$. The global control decisions are obtained by fusing decisions of local supervisors according to some fusion rules.

Conventional decentralized control is based on the *conjunctive* fusion rule, where a controllable event to be enabled needs to be enabled by all supervisors which can control that event [54, 9, 57]. Given a plant G, a nonempty language K is said to be $(L(G), \Sigma_{ci}, M_i)$-*coobservable* if the following condition holds:

$$\forall \sigma \in \Sigma_c, i \in I_c(\sigma), s_i, t \in pr(K) : M_i(s_i) = M_i(t), s_i\sigma \in pr(K), t\sigma \in L(G) \Rightarrow t\sigma \in pr(K),$$

where $I_c(\sigma) := \{i \in I | \sigma \in \Sigma_{ci}\}$ is the index set of all local supervisors which can control the event σ. Coobservability together with controllability is a necessary and sufficient condition for the conjunctive decentralized control. Let $inf\overline{PC_{\Sigma_{ui}}O_{M_i}}(K)$ $(i \in I)$ be the infimal prefix-closed, $(L(G), \Sigma_u)$-controllable, and $(L(G), M_i)$-observable superlanguage of K (such superlanguages are known to exist [57]). Local supervisors for the conjunctive decentralized control can be chosen to be generators of $inf\overline{PC_{\Sigma_{ui}}O_{M_i}}(K)$, and $K = \bigcap_{i \in I} inf\overline{PC_{\Sigma_{ui}}O_{M_i}}(K)$ whenever K is $(L(G), \Sigma_u)$-controllable and $(L(G), \Sigma_{ci}, M_i)$-coobservable [34].

Prosser *et al.* first investigated a non-conjunctive fusion rule based decentralized control [51]. Later a general *conjunctive+disjunctive* fusion architecture for decentralized control was studied comprehensively in [75], where the controllable event set Σ_c is partitioned into two disjoint sets: $\Sigma_c = \Sigma_{c,d} \dot{\cup} \Sigma_{c,e}$. The conjunctive rule is applied for decision fusion in $\Sigma_{c,d}$, and the disjunctive rule is applied for decision fusion in $\Sigma_{c,e}$. In that architecture, the following notion of C&P $\vee$ D&A-coobservability was introduced for the existence of conjunction+disjunction-based decentralized control.

Definition 1 [75] Given a prefix-closed language L, the controllable event sets Σ_{ci}, the observation masks M_i $(i \in I)$, and the decision fusion partition sets $\Sigma_{c,d}$ and $\Sigma_{c,e}$ of Σ_c, K is said to be $(L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})$-*C&P* $\vee$ *D&A-coobservable* if

1. *C&P-coobservability*: $\forall s \in pr(K), \sigma \in \Sigma_{c,d}, s\sigma \in L - pr(K) : \exists i \in I_c(\sigma)$ s.t. $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) = \emptyset$,

2. *D&A-coobservability*: $\forall s \in pr(K), \sigma \in \Sigma_{c,e}, s\sigma \in pr(K) : \exists i \in I_c(\sigma)$ s.t. $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap L \subseteq pr(K)$.

This dissertation uses PSC for decentralized control. In PSC, a priority set is associated with each interacting system, and for an event to be enabled in the interconnected system, it must be enabled in all systems whose priority sets contain that event. For a system with two components, PSC is defined as follows [17].

Definition 2 Given two systems $G_i = (X_i, \Sigma, \alpha_i, x_{i,0}, X_{i,m})$, let $A_i \subseteq \Sigma$ denote the priority event sets of G_i $(i \in I)$. The PSC of G_1 and G_2 is defined as $G_1 \,{}_{A_1}\|_{A_2}\, G_2 := (X, \Sigma, \alpha, x_0, X_m)$, where $X := X_1 \times X_2$, $x_0 := (x_{1,0}, x_{2,0})$, $X_m := X_{1,m} \times X_{2,m}$ and the transition function $\alpha : X \times \Sigma \to X$ is defined as follows. $\forall x = (x_1, x_2) \in X, \sigma \in \Sigma$:

$$\alpha(x,\sigma) := \begin{cases} (\alpha_1(x_1,\sigma), \alpha_2(x_2,\sigma)) & \text{if } \alpha_1(x_1,\sigma) \text{ defined}, \alpha_2(x_2,\sigma) \text{ defined} \\ (\alpha_1(x_1,\sigma), x_2) & \text{if } \alpha_1(x_1,\sigma) \text{ defined}, \alpha_2(x_2,\sigma) \text{ undefined}, \sigma \notin A_2 \\ (x_1, \alpha_2(x_2,\sigma)) & \text{if } \alpha_1(x_1,\sigma) \text{ undefined}, \alpha_2(x_2,\sigma) \text{ defined}, \sigma \notin A_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

CHAPTER 3. DECENTRALIZED FAILURE DIAGNOSIS

In this chapter, we study the decentralized diagnosis problem. We first introduce a definition of codiagnosability and provide an algorithm of complexity polynomial in the size of the system and the non-fault specification for testing codiagnosability, computing the bound in delay of diagnosis, off-line synthesis of individual diagnosers, and on-line diagnosis using them. Then extensions are made for the case of multiple specification languages and specifications of failure events. Next, a notion of strong-codiagnosability is introduced to capture the ability of being certain about the failure or non-failure conditions in a system within bounded delay. Finally, we study the property of being able to react safely to failures in a decentralized setting. For this purpose a notion of safe-codiagnosability is introduced by extending the notion of safe-diagnosability [46] to the decentralized setting.

3.1 Codiagnosability

In a decentralized diagnosis system, there are multiple local diagnosers, and each of them performs diagnosis based on its own set of sensors without communicating to each other. For a decentralized system with two local diagnosers, Figure 3.1 shows the system architecture.

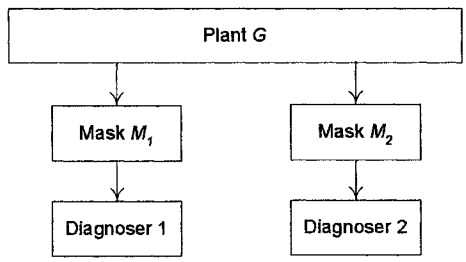


Figure 3.1 Architecture of a decentralized failure diagnosis system

In the definition below L represents the generated language of a system and is prefix-closed,

and $K \subseteq L$ represents a specification language. Since K can capture safety as well as progress properties, it need not be prefix-closed. A failure is said to have occurred if the system executes a trace in $L - pr(K)$ violating the specification. Thus although the "system specification" is K, the "specification for non-failures" is $pr(K)$ — a prefix-closed language.

Definition 3 Let L be the prefix-closed language generated by a system and K be the specification language contained in L ($K \subseteq L$). Assume there are m local sites with observation masks $M_i : \overline{\Sigma} \rightarrow \overline{\Lambda}_i$ ($i \in I = \{1, \cdots, m\}$). (L, K) is said to be codiagnosable with respect to $\{M_i\}$ if

$$(\exists n \in \mathcal{N})(\forall s \in L - pr(K))(\forall st \in L - pr(K), |t| \geq n \text{ or } st \text{ deadlocks}) \Rightarrow$$

$$(\exists i \in I)(\forall u \in M_i^{-1} M_i(st) \cap L, u \in L - pr(K)).$$

The above definition of codiagnosability has the following meaning. Let s be a trace in the "faulty language" $L - pr(K)$, and t be either a sufficiently long extension in $L - pr(K)$ after s (with at least n transitions), or st be a deadlocking trace. There exists at least one local site i such that any trace in L indistinguishable to st for site i belongs to the faulty language $L - pr(K)$. A local site is *ambiguous* if its past observations indicate the possible occurrence of a failure but not with complete certainty; otherwise, it is unambiguous. Informally, Definition 3 means that for any faulty trace, there exists at least one local site that can unambiguously detect the occurrence of the failure within finite transitions. It follows from this definition that (L, K) is codiagnosable if and only if $(L, pr(K))$ is codiagnosable.

3.1.1 Verification of Codiagnosability

To facilitate the development of an algorithm for testing codiagnosability, we first present a lemma for the condition of non codiagnosability, which is derived by negating the codiagnosability condition of Definition 3. (The basic idea of our algorithm is to check if there exists a situation that violates the definition of codiagnosability.)

Lemma 1 Let L be the prefix-closed language generated by a system and K be the specification language contained in L ($K \subseteq L$). Assume there are m local sites with observation masks $M_i : \overline{\Sigma} \to \overline{\Lambda}_i$ ($i \in I = \{1, \cdots, m\}$). (L, K) is not codiagnosable with respect to $\{M_i\}$ if and only if

$$(\forall n \in \mathcal{N})(\exists s \in L - pr(K))(\exists st \in L - pr(K), |t| \geq n \text{ or } st \text{ deadlocks}) \Rightarrow$$

$$(\forall i \in I)(\exists u_i \in M_i^{-1} M_i(st) \cap L, u_i \in pr(K)).$$

Lemma 1 states that (L, K) is not codiagnosable if and only if there exist a faulty trace $s \in L - pr(K)$ possessing an arbitrarily long extended (or deadlocking) trace st, and for each local site a st-indistinguishable non-faulty trace $u_i \in pr(K)$.

Remark 1 Note that if there exist deadlocks in the system, we can add a self-loop at each deadlocking state on ϵ. In this way, the deadlocking system is converted into a deadlock free system in such a way that the observation generated by any deadlocking trace t does not change: $M_i(t) = M_i(t\epsilon^*)$. With this observation in mind, in the following discussion the system to be diagnosed is assumed to be deadlock free.

The following algorithm checks whether codiagnosability is violated. Without loss of generality, assume there are two local sites (i.e., $I = \{1, 2\}$), and as mentioned in Remark 1 the system is deadlock-free. The idea behind the algorithm is to construct a testing automaton that tracks a faulty trace, and for each local site a corresponding indistinguishable non-faulty trace. Presence of a cycle involving such a tuple of traces in the testing automaton is equivalent to the violation of codiagnosability. For simplicity of presentation, we assume the generators G and R of L and $pr(K)$, respectively, are both deterministic. However, the algorithm of the paper continues to hold even when G is nondeterministic and either R is a subautomaton of G or R is deterministic.

Algorithm 1 Let $G = (X, \Sigma, \alpha, x_0)$ be a deterministic finite state machine (DFSM) system model and $L = L(G)$ be its generated language. For the specification language $K \subseteq L$, let $R = (Y, \Sigma, \beta, y_0)$ be the DFSM model with $L(R) = pr(K)$.

Step 1: Construct augmented specification automaton $\overline{R}$

We first augment the state set of R by adding a new state F, which indicates occurrence of a failure. For each $y \in Y$, we add a new transition to the failure state F on each event $\sigma \in \Sigma$ if $\beta(y, \sigma)$ is undefined. At the failure state F, we introduce a self-loop transition for any event $\sigma \in \Sigma$. The resulting automaton is denoted by $\overline{R} = \{\overline{Y}, \Sigma, \overline{\beta}, y_0\}$, where $\overline{Y} = Y \cup \{F\}$ and $\overline{\beta}$ is defined as:

$$\forall \overline{y} \in \overline{Y}, \sigma \in \Sigma,\ \overline{\beta}(\overline{y}, \sigma) := \begin{cases} \beta(\overline{y}, \sigma), & \text{if } [\overline{y} \in Y] \wedge [\beta(\overline{y}, \sigma) \neq \emptyset] \\ F, & \text{if } [\overline{y} = F] \vee [\beta(\overline{y}, \sigma) = \emptyset] \end{cases}$$

It follows from the above construction procedure that all events in Σ are defined at each state in $\overline{R}$. Therefore, $\overline{R}$ generates the language Σ^*, i.e., $L(\overline{R}) = \Sigma^*$. Also, execution of any trace outside $pr(K)$ reaches the failure state F.

Step 2: Construct codiagnosability testing automaton T

The codiagnosability testing automaton $T = (Z, \Sigma^T, \gamma, z_0)$ is defined as follows:

. $Z = X \times \overline{Y} \times Y \times Y$.

. $z_0 = (x_0, y_0, y_0, y_0)$.

. $\Sigma^T = \overline{\Sigma}^3$, where $\overline{\Sigma} = \Sigma \cup \{\epsilon\}$.

. $\gamma : Z \times \overline{\Sigma}^3 \to Z$ is defined as:

$\forall z = (x, \overline{y}, y^1, y^2) \in Z,\ \sigma^T = (\sigma, \sigma^1, \sigma^2) \in \Sigma^T - \{(\epsilon, \epsilon, \epsilon)\}$,

$$\gamma(z, \sigma^T) := (\alpha(x, \sigma), \overline{\beta}(\overline{y}, \sigma), \beta(y^1, \sigma^1), \beta(y^2, \sigma^2))$$

if and only if

$$[M_1(\sigma) = M_1(\sigma^1), M_2(\sigma) = M_2(\sigma^2)] \wedge [\alpha(x, \sigma), \beta(y^1, \sigma^1), \beta(y^2, \sigma^2) \neq \emptyset].$$

The state space of T is $X \times \overline{Y} \times Y \times Y$, and T tracks a triplet of traces $s \in L(G) \cap L(\overline{R}) = L(G), u_1 \in L(R), u_2 \in L(R)$ with the property, $M_1(s) = M_1(u_1), M_2(s) = M_2(u_2)$. The

first component of T tracks trace s in G, the second component the trace s in $\overline{R}$, the third component the trace u_1 in R, and the last component the trace u_2 in R. Purpose of tracking s in G and $\overline{R}$ is to determine whether s is a faulty trace belonging to $L \cap (pr(K))^c = L - pr(K)$.

Step 3: Check violation of codiagnosability

We first define a cycle cl^T in the testing automaton T as follows:

$$cl^T := (z_k, \sigma_k^T, z_{k+1}, \cdots, z_l, \sigma_l^T, z_k), (l \geq k \geq 0)$$

where $z_i = (x_i, \overline{y}_i, y_i^1, y_i^2) \in Z$ and $\sigma_i^T = (\sigma_i, \sigma_i^1, \sigma_i^2) \in \Sigma^T (i = k, k+1, \cdots, l)$.

Then, we check if there exists a cycle cl^T in T satisfying

$$\exists i \in [k, l] \text{ such that } (\overline{y}_i = F) \wedge (\sigma_i \neq \epsilon).$$

If the answer is yes, then (L, K) is not codiagnosable with respect to the observation masks $\{M_i\}$. Otherwise, (L, K) is codiagnosable. The condition $(\overline{y}_i = F)$ indicates that a failure has occurred. Since σ_i is a transition in the system G, the condition $(\sigma_i \neq \epsilon)$ requires that the system execute at least one event in the cycle cl^T. This requirement originates from the fact G must execute an extension after a failure has occurred in order to allow for its diagnosis.

Remark 2 If there are no unobservable-event cycles in the system, then the second condition $(\sigma_i \neq \epsilon)$ is redundant since in absence of unobservable cycles, this condition automatically holds. But the test needs to be strengthened with this condition in the presence of unobservable-event cycles.

The following example illustrates the construction used in the algorithm for testing codiagnosability.

Example 1 A system model G and a specification model R are given in Figure 3.2 with $L(G) = L$ and $L(R) = pr(K)$. Assume there are two local diagnosers, i.e., $I = \{1, 2\}$. The set of events is $\Sigma = \{a, b, c, \sigma_u, \sigma_f\}$, where σ_u and σ_f are two unobservable events, i.e., $\forall i \in I, M_i(\sigma_u) = M_i(\sigma_f) = \epsilon$. The event a can be observed by both diagnosers $(M_1(a) =$

$M_2(a) = a$), the event b can be observed by only the first diagnoser, whereas the event c can be observed by only the second diagnoser ($M_1(b) = b, M_1(c) = \epsilon, M_2(b) = \epsilon, M_2(c) = c$).

The augmented specification automaton $\overline{R}$ is constructed according to Algorithm 1 and is shown in Figure 3.2. Only a part of the testing automaton T is shown in Figure 3.2 to track a specific transition sequence. The transition sequence "$aaa, \sigma_f \epsilon b$" causes T to reach the state "5F12". This implies the trace $s = a\sigma_f \in L - pr(K)$ is a failure trace. In Figure 3.2, T eventually reaches the state "4F44" possessing a self-loop, that violates the codiagnosability condition mentioned in Step 3 of Algorithm 1. Therefore, (L, K) is not codiagnosable with respect to the observation masks $\{M_i\}$.

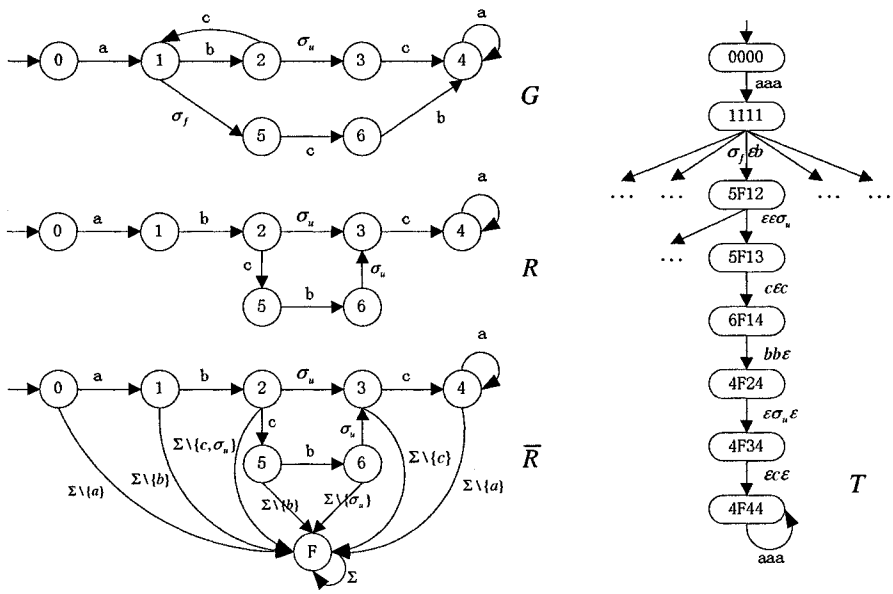


Figure 3.2 Algorithm 1 illustrated (G: plant model; R: specification model; $\overline{R}$: augmented automaton R; T: testing automaton)

The following theorem proves the correctness of Algorithm 1.

Theorem 1 Given a prefix-closed system language L and a specification language $K \subseteq L$, let $G = (X, \Sigma, \alpha, x_0)$ and $R = (Y, \Sigma, \beta, y_0)$ be automata with $L(G) = L$ and $L(R) = pr(K)$. Assume there are m local sites with observation masks M_i ($i \in I$). (L, K) is not codiagnosable with respect to $\{M_i\}$ if and only if there exists a cycle $cl^T = (z_k, \sigma_k^T, z_{k+1}, \cdots, z_l, \sigma_l^T, z_k)$ in the

testing automaton $T = (Z, \Sigma^T, \gamma, z_0)$ such that:

$$\exists i \in [k, l] \text{ s.t. } (\overline{y}_i = F) \wedge (\sigma_i \neq \epsilon), \tag{3.1}$$

where $\overline{y}_i$ is the second coordinate of state z_i, and σ_i is the first coordinate of $\sigma_i^T \in \Sigma^T$.

Proof: ($\Leftarrow$) Suppose there exists a cycle $cl^T = (z_k, \sigma_k^T, z_{k+1}, \cdots, z_l, \sigma_l^T, z_k)$ satisfying condition (3.1). Let *path* be a path in T ending with the cycle cl^T:

$$path = (z_0, \sigma_0^T, \cdots, z_{k-1}, \sigma_{k-1}^T, z_k, \sigma_k^T, \cdots, z_l, \sigma_l^T, z_k).$$

Therefore, for any $n \in N$ there exist the following event traces:

$$st = \sigma_0 \cdots (\sigma_k \cdots \sigma_l)^n \in L(G) = L,$$

$$u_1 = \sigma_0^1 \cdots (\sigma_k^1 \cdots \sigma_l^1)^n \in L(R) = pr(K),$$

$$u_2 = \sigma_0^2 \cdots (\sigma_k^2 \cdots \sigma_l^2)^n \in L(R) = pr(K),$$

where $M_1(st) = M_1(u_1)$ and $M_2(st) = M_2(u_2)$.

Since there exists $i \in (k, \cdots, l)$ such that $\overline{y}_i = F$, from the definition of $\overline{R}$, there exists a transition $\beta(y_p, \sigma_p)$ $(0 \leq p < i)$ which is not defined in R. Therefore, $st \notin L(R) = pr(K)$. Further once $\overline{R}$ reaches the failure state F, it remains there, and so $\overline{y}_i = F$ for each $i \in [k, l]$. We can choose $s = \sigma_0 \cdots \sigma_{k-1}$ and $t = (\sigma_k \cdots \sigma_l)^n$. Then since G is deadlock free and $\sigma_i \neq \epsilon$, it follows that $|t| \geq n$. From Lemma 1, (L, K) is not codiagnosable with respect to $\{M_i\}$.

($\Rightarrow$) Suppose (L, K) is not codiagnosable with respect to $\{M_i\}$. Then from Lemma 1, there exists a faulty trace $s \in L - pr(K)$ and its extended trace $st \in L - pr(K)$ such that the following holds: $\exists u_1, u_2$,

$$u_1 \in M_1^{-1} M_1(st) \cap pr(K), u_2 \in M_2^{-1} M_2(st) \cap pr(K).$$

Since $s \in L - pr(K)$, according to the definition of $\overline{R}$, $\overline{\beta}(y_0, s) = F$ and $\overline{R}$ remains at the failure state F on any further transition. Let us execute the trace tr in T:

$$tr = \sigma_0^T \cdots \sigma_l^T, \text{ where } \sigma_i^T = (\sigma_i, \sigma_i^1, \sigma_i^2) \ (i \in 0, \cdots, l)$$

such that $st = \sigma_0 \cdots \sigma_l$, $u_1 = \sigma_0^1 \cdots \sigma_l^1$, and $u_2 = \sigma_0^2 \cdots \sigma_l^2$. Let $|Z|$ be the number of states in the testing automaton T. If $|st| > |Z|$, then since T is a finite state machine (FSM), there will be a cycle $cl^T = (z_k, \sigma_k^T, z_{k+1}, \cdots, z_l, \sigma_l^T, z_k)$ along the trace tr, where $l \geq k \geq 0$, such that for some $i \in [k, l]$, $\overline{y}_i = F$ and $\sigma_i \neq \epsilon$. Therefore, the sufficiency holds. ■

The computation complexity of Algorithm 1 is analyzed as follows.

Remark 3 Let $|X|$ and $|Y|$ be the number of states of G and R, and $|\Sigma|$ be the number of events of G and R. Assume there are m local sites in the system. Table 3.1 lists the maximum number of states and transitions of various automata in Algorithm 1. Since $\overline{R}$ and R have same order of states and transitions, i.e., $\mathcal{O}(|Y|)$ and $\mathcal{O}(|Y| \times |\Sigma|)$ respectively, we do not differentiate the number of states and transitions of R and $\overline{R}$ for complexity analysis. Since there are $m + 1$ coordinates in a transition of T, the number of transitions at each state of T is at most $(|\Sigma| + 1)^{m+1}$.

The complexity of Step 1 and Step 2 is linear in the number of states and transitions of R and T respectively. The complexity of Step 3, which is to detect the presence of a certain "offending" cycle in the testing automaton T, is also linear in the number of states and transitions of T. Therefore, the complexity of Algorithm 1 is $\mathcal{O}(|X| \times |Y|^{m+1} \times |\Sigma|^{m+1})$.

Table 3.1 Computational complexity of Algorithm 1

	number of states	number of transitions
G	$\|X\|$	$\|X\| \times \|\Sigma\|$
R	$\|Y\|$	$\|Y\| \times \|\Sigma\|$
T	$\|X\| \times \|Y\|^{m+1}$	$\|X\| \times \|Y\|^{m+1} \times (\|\Sigma\| + 1)^{m+1}$
complexity	$\mathcal{O}(\|X\| \times \|Y\|^{m+1} \times \|\Sigma\|^{m+1})$	

Remark 4 For the special case when the specification model $R = (Y, \Sigma, \beta, y_0)$ is a subautomaton of the system model $G = (X, \Sigma, \alpha, x_0)$, the operation of testing codiagnosability can be performed with less complexity. Since R is a subautomaton of G, we have $Y \subseteq X$, $L(R) \subseteq L(G)$ and for all $s \in L(R)$, $\beta(y_0, s) = \alpha(x_0, s)$. Then for each trace in the testing automaton T, if the system model G reaches a state $x \in X$, then the augmented automaton $\overline{R}$ can only reach either the same state $y = x$ or the failure state $y = F$. Then the first two components of T, namely $G||\overline{R}$, has state size at most $2|X|$ and transition size at most $2|X| \times |\Sigma|$. Thus number of states and transitions in T is at most $2|X|^{m+1}$ and $2|X|^{m+1} \times |\Sigma|^{m+1}$, respectively. The complexity of Algorithm 1 for the special case is $\mathcal{O}(|X|^{m+1} \times |\Sigma|^{m+1})$, which is one order less than the general case.

Remark 5 For the centralized case, diagnosability can be defined similarly in the specification language setting by letting $m = 1$ in Definition 3. It follows that diagnosability requires a weaker condition than codiagnosability, i.e., some diagnosable systems may not be codiagnosable. For example, in Example 1, if we set the central observation mask as the "union" of the local observation masks, i.e., $M(a) = a, M(b) = b, M(c) = c$, and $M(\sigma_u) = M(\sigma_f) = \epsilon$, then we can verify that (L, K) is diagnosable with respect to M. See the automaton T for testing diagnosability shown in Figure 3.3, which does not contain any offending cycle.

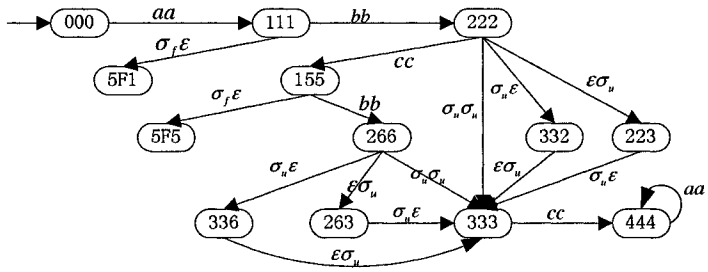


Figure 3.3 Testing of diagnosability under a global/central observer

3.1.2 Delay of Codiagnosability

In the above section, we discussed codiagnosability and its verification methods. Codiagnosability guarantees that once a failure occurs, there exists at least one local diagnoser that

can unambiguously detect and diagnose the failure within bounded delay. In this section we compute the value of delay and also compute the maximum delay possible. This information is very important because one purpose of failure diagnosis is to detect and isolate failures so as to activate some failure recovery procedures. Even for a codiagnosable system, if the diagnosis result of a failure arrives late, some recovery deadlines may be missed and the failure may cause catastrophic results. Since the case of failure events can be transformed to the case of specification language, we only consider the case of specification language.

The computation of delay of codiagnosability is based on the testing automaton $T = (Z, \overline{\Sigma}^3, \gamma, z_0)$ constructed in Algorithm 1. Once a failure occurs in the system model G, the second coordinate of T reaches the failure state "F" and stays there forever. Define a set of states containing all the "failure" states as:

$$Z_F = \{(x, \bar{y}, y^1, y^2) \in Z | \bar{y} = F\}.$$

From analysis in Section 3.1, we know that if a system (L, K) is codiagnosable, there does not exist any cycle among states in Z_F. Since T is a finite automaton, some deadlocking states will be reached eventually in Z_F. Then we can define the delay of codiagnosability as follows.

Definition 4 The delay of codiagnosability for $z \in Z_F$ is defined by

$$d(z) = \max_{\{(s,u_1,u_2) \in (\overline{\Sigma}^3)^*:\ \gamma(z,(s,u_1,u_2))=\emptyset\}} (EC(s)) + 1,$$

where for $s \in \Sigma^*$, $EC(s)$ is the *event count* of s, i.e., the number of elements in s that are not ϵ.

$d(z)$ is the number of non-silent transitions in the system that must be executed to lead failure state z to a deadlocking state. Since it is not possible to sustain the ambiguity of failure beyond deadlocking states (as no further execution possible), an extra transition will resolve ambiguity of failure, and so an extra one is added to $EC(s)$ above. Based on Definition 4, we can define the delay of codiagnosability of a system as follows.

Definition 5 Let L be the system language and K ($K \subseteq L$) be the specification language. The delay of codiagnosability of (L, K) with respect to $\{M_i\}$ is defined as:

$$d(L, K) = \max_{z \in Z_F} d(z).$$

The above definition states that the delay of codiagnosability of (L, K) is the maximum value of all delays of codiagnosability for any state in Z_F. To compute the delay of codiagnosability of (L, K), we present the following algorithm that computes the length of the longest path over Z_F and adds an extra one to it.

Algorithm 2 Given a system language L and a specification language K ($K \subseteq L$), let G be the system model with $L(G) = L$ and R be the specification model with $L(R) = pr(K)$.

1. Construct a testing automaton $T = (Z, \overline{\Sigma}^3, \gamma, z_0)$ as in Algorithm 1.

2. Initialize the delay of codiagnosability for all states $z \in Z_F$: $d^0(z) = 1$, and a counter $k = 0$.

3. Execute one-step forward search and update the delay of codiagnosability for each state $z \in Z_F$ as follows:

$$d^{k+1}(z) := \begin{cases} \max[d^k(z), d^k(z') + 1], & \text{if } \exists(\sigma, \sigma^1, \sigma^2) \in \Sigma \times (\overline{\Sigma})^2 \text{ s.t. } z' \in \gamma(z, (\sigma, \sigma^1, \sigma^2)) \\ d^k(z), & \text{otherwise.} \end{cases}$$

4. Repeat Step 3 until $\forall z \in Z_F, d^{k+1}(z) = d^k(z)$, each time incrementing k by 1.

5. Compute the maximum delay of codiagnosability:

$$d(L, K) = \max_{z \in Z_F} d^k(z).$$

The following example illustrates this algorithm.

Example 2 Figure 3.4 shows a system model G and a specification model R. The system has two local diagnosers with observation masks defined as: $M_1(a) = a, M_1(b) = b, M_1(c) = M_1(\sigma_f) = \epsilon, M_1(d) = d$, and $M_2(a) = a, M_2(c) = c, M_2(b) = M_2(d) = M_2(\sigma_f) = \epsilon$.

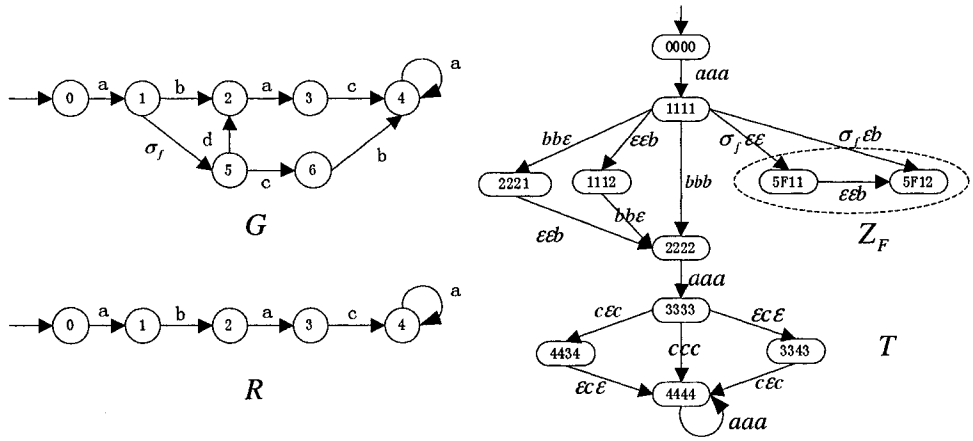


Figure 3.4 Delay of codiagnosability (G: system model; R: specification model; T: Testing automaton)

From the testing automaton T of Figure 3.4, it is easy to verify that the system (L, K) is codiagnosable with respect to the observation masks $\{M_i\}$. The set of failure states Z_F includes two states, (5F11) and (5F12). We first assign an initial value "1" to these two states. Then we perform the one-step forward search recursively until the termination condition is satisfied. Though (5F12) is a successor state of

(5F11), the first component of the transition between them is ϵ. Therefore, state (5F11) remains labeled by "1" and the maximum delay of codiagnosability equals 1. This means that if a failure occurs in the system, the failure will be detected after the system executes at most one more transition.

Remark 6 The complexity of Step 3 of Algorithm 2 is linear in the number of states of Z_F and transitions among them, which in worst case is of order the number of states and transitions in T. Since there are no cycles among states in Z_F for a codiagnosable system, Step 3 may be repeated at most $|Z_F|$ times. Therefore, the complexity of Algorithm 2 is $\mathcal{O}(|X|^2 \times |Y|^{2m+2} \times |\Sigma|^{m+1})$.

Remark 7 Failure diagnosis can be viewed as a type of *convergence* or *stability* [4, 44, 29, 72] problem. Informally, the convergence or stability specifies the eventual behavior of a system, i.e., a system is said to be convergent or stable if the system state can eventually converge to a legal set of states regardless of the current state of the system. Codiagnosability of a system requires that once a failure occurs, the testing system state (extended appropriately to evolve over $X \times \overline{Y}^3$ as opposed to $X \times \overline{Y} \times Y^2$) eventually converges to a "diagnosis region" where the failure can be unambiguously diagnosed by at least one local diagnoser. This diagnosis region can be defined over $X \times \overline{Y}^3$ as:

$$Z_D = \{(x, \overline{y}, \overline{y}^1, \overline{y}^2) | \overline{y} = \overline{y}^1 = F \text{ or } \overline{y} = \overline{y}^2 = F\}.$$

Then, the maximum delay of codiagnosability equals one plus the event count associated with the first component of the longest path from Z_F to Z_D, where Z_F is the set of failure states, and Z_D is the set of diagnosis states.

3.1.3 Diagnoser Synthesis and On-Line Diagnosis

In the previous sections, we discussed verification of codiagnosability and computation of maximum delay of codiagnosability. In this section, we discuss how to construct local diagnosers that can diagnose the system failures. Since the setting of failure events can be transformed to the setting of a specification language by constructing the equivalent specification model R as introduced in Section 3.1.5, we only discuss diagnoser synthesis and on-line diagnosis in the setting of specification language.

The local diagnoser at site i is the state machine $G||\overline{R}$ masked by observation mask M_i, which is defined as

$$D_i = M_i(G||\overline{R}) = (X \times \overline{Y}, \Delta_i, \delta_i, (x_0, y_0)).$$

Note that D_i is in general nondeterministic. We use D_i to perform on-line failure diagnosis as follows. At each local site i, we maintain a set of possible current states of D_i reached by the observation sequence executed so far, denoted $Reach_i(\cdot)$. $Reach_i(\cdot)$ is computed recursively

upon each observation as follows.

$$Reach_i(\epsilon) = \epsilon^*_{D_i}((x_0, y_0));\ Reach_i(\tau\eta) = \epsilon^*_{D_i}(\delta_i(Reach_i(\tau), \eta)), \tau \in \Delta_i^*, \eta \in \Delta_i.$$

After each update of $Reach_i(\cdot)$, we check all states in $Reach_i(\cdot)$. If all states have second coordinate as "F", then a failure is detected at site i, i.e.,

$$\text{Failure detected by } D_i \Leftrightarrow Reach_i(\cdot) \subseteq X \times \{F\}.$$

If some but not all the states in $Reach_i(\cdot)$ have 2nd coordinate as F, then D_i is said to be ambiguous. For a codiagnosable system, there exists at least one local diagnoser which can detect/diagnose a failure within a bounded delay.

Remark 8 The complexity for local diagnoser construction and on-line failures detection is analyzed as follows. The number of states and transitions of $G||\overline{R}$ is of order $|X| \times |Y|$ and $|X| \times |Y| \times |\Sigma|$ respectively. The complexity of off-line construction of D_i is linear in the number of transitions of $G||\overline{R}$, i.e., for m local diagnosers, the complexity is $\mathcal{O}(m \times |X| \times |Y| \times |\Sigma|)$. During on-line failure diagnosis, update of $Reach_i(\cdot)$ is required following each new observation at site i. The complexity of such an update operation is linear in the number of states and transitions of D_i since it involves a certain reachability computation. Therefore, for a system with m local diagnosers, the complexity of performing each step of on-line failure diagnosis is $\mathcal{O}(m \times |X| \times |Y| \times |\Sigma|)$, the same as the complexity of constructing local diagnosers.

If the specification model R is a subautomaton of the system model G, then as analyzed before the number of states and transitions in $G||\overline{R}$ is of order $2|X|$ and $2|X| \times |\Sigma|$ respectively. So the complexity of off-line diagnoser construction and of each step of on-line failure diagnosis is $\mathcal{O}(m \times |X| \times |\Sigma|)$.

Example 3 Consider the codiagnosable system introduced in Example 2. Figure 3.5 shows the two local diagnosers $D_i = M_i(G||\overline{R})$ $(i \in \{1, 2\})$, where G is the system model and R is the specification model.

Assume that the system G executes the event trace $s = a\sigma_f cba^n$ $(n \in \mathcal{N})$. Due to the presence of partial observation, the traces observed in local diagnosers are $M_1(s) = aba^n$ and $M_2(s) = aca^n$ respectively. Figure 3.5 shows all $Reach_i(\cdot)$ computations during the on-line diagnosis procedure. Notice that following the observation of the trace ac by diagnoser D_2, all states in $Reach_2(ac)$ have second coordinate F. So D_2 detects and reports a failure at this point. On the other hand, all along the observation trace aba^n, D_1 remains ambiguous about the occurrence of a fault since in each $Reach_1(\cdot)$ set containing a state with second coordinate F, exists another state with second coordinate different from F. Therefore, the execution of the failure trace $s = a\sigma_f cba^n$ can only be diagnosed by the second diagnoser D_2. Similarly, it can be verified that the execution of failure trace $s = a\sigma_f daca^n$ in the system can be diagnosed by D_1, but not by D_2.

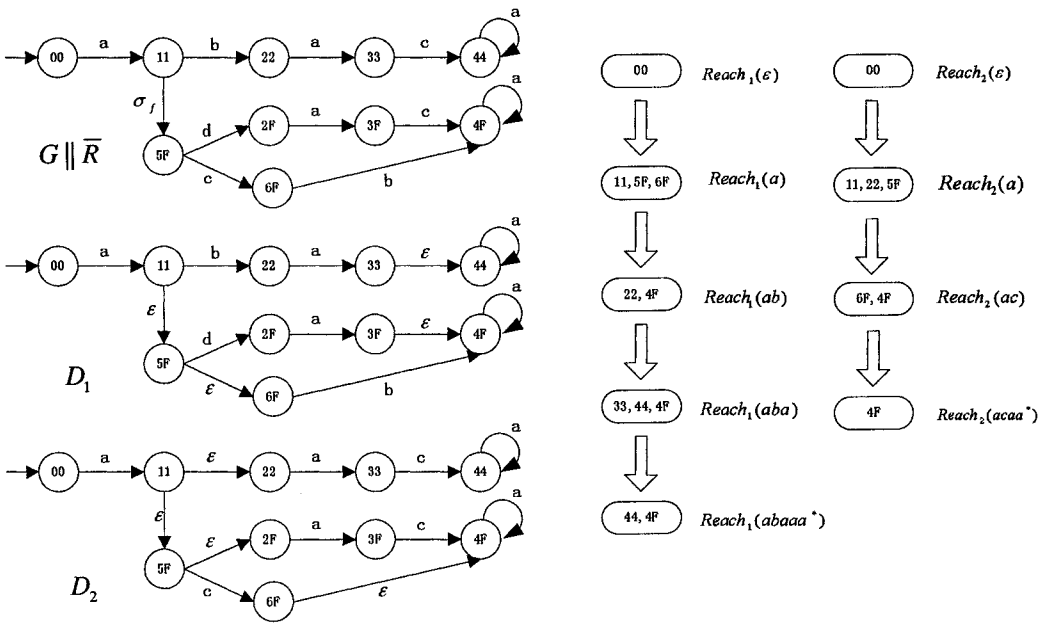


Figure 3.5 Off-line diagnoser synthesis and on-line diagnosis

Remark 9 Failure diagnosis introduced in [60] and [13] is based on the construction of centralized/local diagnosers, which are DFSMs (as opposed to NFSMs in our setting). As analyzed in [21], having DFSMs for diagnosers results in an exponential complexity. In contrast, our methods for diagnoser synthesis and on-line failure diagnosis have linear complexity.

3.1.4 Multiple Sub-Specification Languages

In some applications, it is more convenient to capture the desired behavior of a system by a set of sub-specification languages than by one single specification language. The system behavior is said to be faulty if it violates one of the sub-specification languages. In addition to detecting a failure, i.e., detecting the violation of the overall specification, we further need to detect which sub-specification has been violated. Such diagnosis result is helpful to isolate/locate failures and perform appropriate failure recovery operations.

Let L be a language generated by a system. Assume there are l sub-specification languages K_j ($j \in I_K = \{1, \cdots, l\}$), where $K_j \subseteq L$. The nominal behavior K is a conjunct of all prefix closures of sub-specification languages, i.e., $K = \bigcap_{j \in I_K} pr(K_j)$. It follows that K is prefix-closed. Then, the faulty behavior of the system can be captured by the language

$$L - K = L - \bigcap_{j \in I_K} pr(K_j) = \bigcup_{j \in I_K} (L - pr(K_j)).$$

In other words, a system behavior is faulty if it is faulty with respect to any of the sub-specification language K_j. Therefore, instead of detecting the system failures with respect to K, we can detect the system failures with respect to each sub-specification language K_j individually.

Definition 6 Let L be the prefix-closed language generated by a system, and $\mathcal{K}$ be the set of all sub-specification languages K_j, i.e., $\mathcal{K} = \{K_j | j \in I_K = \{1, \cdots, l\}\}$. Assume there are m local sites with observation masks M_i ($i \in I = \{1, \cdots, m\}$). $(L, \mathcal{K})$ is said to be codiagnosable with respect to $\{M_i\}$ if

$$\forall j \in I_K, (L, K_j) \text{ is codiagnosable with respect to } \{M_i\}.$$

Let $K = \bigcap_{j \in I_K} pr(K_j)$. $(L, \mathcal{K})$ is said to be codetectable if (L, K) is codiagnosable.

The above definition requires that a system with a set of sub-specification languages $(L, \mathcal{K})$ is codiagnosable if the system with each individual sub-specification language (L, K_j) is codi-

agnosable. Since codiagnosability of (L, K) allows detection of a failure unambiguously but it may not allow isolation of the failure, we refer to it as codetectability of $(L, \mathcal{K})$. For the desired system behavior $K = \bigcap_{j \in I_K} pr(K_j)$, we have the following property about the relationship between the codiagnosability of $(L, \mathcal{K})$ and (L, K). The proof can be easily derived from Definitions 3 and 6, and Lemma 1.

Property 1 $(L, \mathcal{K})$ codiagnosable $\Rightarrow$ (L, K) codiagnosable $\Leftrightarrow$ $(L, \mathcal{K})$ codetectable.

Given a system language L with a set of multiple sub-specification languages $\mathcal{K} = \{K_j | j \in I_K\}$, Algorithm 1 can be used to check the codiagnosability of $(L, \mathcal{K})$. To test the codiagnosability of $(L, \mathcal{K})$, we apply Algorithm 1 to each sub-specification language K_j $(j \in I_K)$ individually. Similarly, diagnoser synthesis and failure diagnosis can be performed for each sub-specification language individually. When the system is in operation, if a local diagnoser i $(i \in I)$ unambiguously detects a failure that violates a sub-specification language K_j $(j \in I_K)$, then the diagnoser i reports that failure, i.e., the violated sub-specification language K_j.

For the case of multiple sub-specification languages, the complexities of codiagnosability testing, diagnoser synthesis and on-line failure diagnosis are linear in the number of sub-specification languages. That is, for l sub-specification languages, these complexities are $\mathcal{O}(l \times |X| \times |Y|^{m+1} \times |\Sigma|^{m+1})$, $\mathcal{O}(l \times m \times |X| \times |Y| \times |\Sigma|)$ and $\mathcal{O}(l \times m \times |X| \times |Y| \times |\Sigma|)$ respectively. Here $|X|$ is the number of states in the system model and $|Y| = \max_{j \in I_K} |Y_j|$, where $|Y_j|$ is the number of states in the sub-specification model for K_j. For the special case where all automata models of sub-specification languages are subautomata of the system model, these complexity are $\mathcal{O}(l \times |X|^{m+1} \times |\Sigma|^{m+1})$, $\mathcal{O}(l \times m \times |X| \times |\Sigma|)$ and $\mathcal{O}(l \times m \times |X| \times |\Sigma|)$ respectively.

To compute the delay of codiagnosability in a system with multiple sub-specification languages, we can apply Algorithm 2 for each sub-specification language. After computing each delay bound $d(L, K_j)$ $(j \in I_K)$, the maximum value over all sub-specifications is taken as the maximum delay of codiagnosability, i.e.,

$$d(L, \mathcal{K}) = \max_{j \in I_K} d(L, K_j).$$

3.1.5 Failure As Occurrence of Failure Events

In this subsection, we study codiagnosability in the failure event framework. Let $\Sigma_f \subseteq \Sigma$ be the set of all failure events, $\mathcal{F} = \{f_1, \cdots, f_j\}$ be the set of all failure types, and $\psi : \Sigma \to \mathcal{F} \cup \{\emptyset\}$ be the failure assignment function. Then the set Σ_f is partitioned into several disjoint sets corresponding to different failure types: $\Sigma_f = \Sigma_{f_1} \cup \cdots \cup \Sigma_{f_l}$, where for all $\sigma \in \Sigma_{f_j} (j \in I_F = \{1, \cdots, l\})$, $\psi(\sigma) = f_j$. Since occurrence of any observable failure event can be immediately diagnosed, we only consider unobservable failure events, i.e.,

$$\forall \sigma \in \Sigma, \psi(\sigma) \neq \emptyset \Rightarrow \forall i \in I, M_i(\sigma) = \epsilon.$$

We first discuss a system with a single failure type f, i.e., $\mathcal{F} = \{f\}$. The following definition defines codiagnosability in the failure event setting.

Definition 7 Given a system model $G = (X, \Sigma, \alpha, x_0)$ with a singleton failure types set $\mathcal{F} = \{f\}$, let $L = L(G)$ be the generated language of G and $\psi : \Sigma \to \mathcal{F} \cup \{\emptyset\}$ be a failure assignment function. Assume there are m local sites with observation masks M_i $(i \in I)$. $(L, \mathcal{F})$ is said to be codiagnosable with respect to $\{M_i\}$ if

$$(\exists n \in \mathcal{N})(\forall s \in L, \psi(s_f) = f)(\forall st \in L, |t| \geq n \text{ or } st \text{ deadlocks}) \Rightarrow$$

$$(\exists i \in I)(\forall u \in M_i^{-1} M_i(st) \cap L)(\exists v \in pr(u), \psi(v_f) = f),$$

where s_f and v_f denote the last events in traces s and v respectively. The system G is said to be codiagnosable if $(L(G), \mathcal{F})$ is codiagnosable.

The verification of codiagnosability of $(L, \mathcal{F})$ is performed as follows. Given a failure assignment function $\psi : \Sigma \to \mathcal{F} \cup \{\emptyset\}$, we define a failure event set as $\Sigma_f = \{\sigma \in \Sigma | \psi(\sigma) = f\}$. Next, we construct a single state automaton R_0 with a self-loop labeled by all non-faulty events $\Sigma - \Sigma_f$. Generated language of R_0 is $(\Sigma - \Sigma_f)^*$, i.e., all traces containing no failure events. We define an equivalent specification model R as the synchronous composition of G and R_0. $R := G||R_0$, which generates all traces of G that contain no failure events. Clearly, $(L, \mathcal{F})$ is

codiagnosable if and only if $(L, L(R))$ is codiagnosable. As in Algorithm 1, we build a testing automata T and check the presence of "offending" cycles.

Since R_0 has a single state, $R = G||R_0$ is a subautomaton of G. So for a deterministic model G, the complexity of testing codiagnosability of $(L, \mathcal{F})$ is same as the subautomaton case analyzed in Remark 3, i.e., the complexity is $\mathcal{O}(|X|^{m+1} \times |\Sigma|^{m+1})$, where m is the number of local diagnosers. If G is nondeterministic, one transition may lead to different states. The number of transitions of G and T is at most $|X|^2 \times |\Sigma|$ and $4|X|^{2(m+1)} \times |\Sigma|^{m+1}$ respectively. Therefore, the complexity of testing codiagnosability for a nondeterministic G is $\mathcal{O}(|X|^{2(m+1)} \times |\Sigma|^{m+1})$. It should be noted that even when G is nondeterministic, $R = G||R_0$ is a subautomaton of G and $\overline{R} = G||\overline{R_0}$.

The following example illustrates the testing approach for the failure event case.

Example 4 Consider the system model G introduced in Example 1. The system only has one failure event σ_f, which belongs to the failure type f, i.e., $\Sigma_f = \{\sigma_f\}$ and $\psi(\sigma_f) = f$. Also, there are two diagnosers in the system with observation masks defined as before: $M_1(a) = a, M_1(b) = b, M_1(c) = M_1(\sigma_u) = M_1(\sigma_f) = \epsilon$, and $M_2(a) = a, M_2(c) = c, M_2(b) = M_2(\sigma_u) = M_2(\sigma_f) = \epsilon$.

The single state automaton R_0 is shown in Figure 3.6. Then we construct the equivalent specification model $R = G||R_0$ and build the augmented automaton $\overline{R}$ by adding a failure state F. Figure 3.6 shows a branch of transitions in T, where a "bad" state "4F44" possessing a self-loop is reached. Therefore, $(L, \mathcal{F})$ is not codiagnosable with respect to $\{M_i\}$. ■

When a system has events of multiple failure types, the definition of codiagnosability is given as follows.

Definition 8 Given a system model $G = (X, \Sigma, \alpha, x_0)$ with a set of failure types $\mathcal{F} = \{f_1, \cdots, f_l\}$, let $L = L(G)$ be the generated language of G and $\psi : \Sigma \to \mathcal{F} \cup \{\emptyset\}$ be a failure assignment function. Assume there are m local diagnosers with observation masks M_i $(i \in I = \{1, \cdots, m\})$. $(L, \mathcal{F})$ is said to be codiagnosable with respect to $\{M_i\}$ if for all $j \in I_F$, $(L, \{f_j\})$ is codiagnosable with respect to $\{M_i\}$.

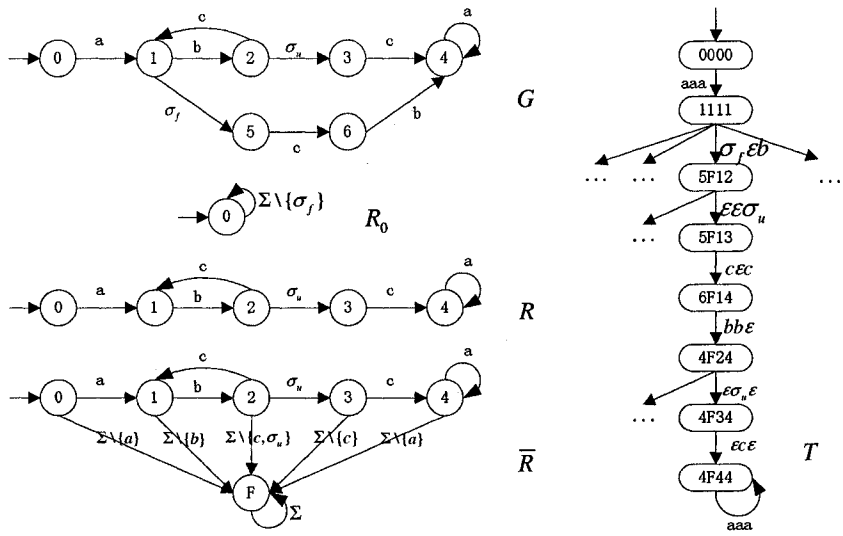


Figure 3.6 Automata in Example 4 (G: plant model; R_0: single state automaton; R: specification model; $\overline{R}$: augmented specification model; T: testing automaton)

To test the codiagnosability of $(L, \mathcal{F})$ as well as for the synthesis of the local diagnosers, we can apply the method developed for each single failure type individually. When we check the codiagnosability of $(L, \{f_j\})$, failure events of other failure types are treated as normal unobservable events. To construct local diagnosers for each failure of type j, we first define an equivalent sub-specification model R_j for each failure type f_j ($j \in I_F$), and then apply the approach introduced in Section 3.1.3 to (L, R_j). When a system is in operation, all such diagnosers are running concurrently. Each diagnoser is responsible for detecting one type of failure. A f_j-type failure is reported if there exists a diagnoser at a certain local site that is responsible for detecting f_j-type failures and reaches an unambiguous set of states.

The complexities of codiagnosability testing, off-line diagnoser synthesis and on-line failure diagnosis are linear in the number of failure types. That is, for a set $\mathcal{F}$ with l failure types, if the system model G is given as a DFSM, then the complexities of these operations are $\mathcal{O}(l \times |X|^{m+1} \times |\Sigma|^{m+1})$, $\mathcal{O}(l \times m \times |X| \times |\Sigma|)$ and $\mathcal{O}(l \times m \times |X| \times |\Sigma|)$ respectively. If G is given as a NFSM, the complexities are $\mathcal{O}(l \times |X|^{2(m+1)} \times |\Sigma|^{m+1})$, $\mathcal{O}(l \times m \times |X|^2 \times |\Sigma|)$ and $\mathcal{O}(l \times m \times |X|^2 \times |\Sigma|)$ respectively.

Also, after defining the equivalent sub-specification model R_j for failure type f_j, the delay

of codiagnosability in a system with multiple failure types can be computed as follows. We first apply Algorithm 2 to compute the delay for each (L, R_j). The delay of codiagnosability with respect to multiple failure types is the maximum value of all these delays, i.e.,

$$d(L, \mathcal{F}) = \max_{j \in I_F} d(L, R_j).$$

3.2 Strong-(Co)diagnosability

The notion of codiagnosability guarantees that occurrence of any failure is detected within finite delay, but there is no guarantee that non-occurrence of failure is unambiguously known within bounded delay. The following example illustrates the situation.

Example 5 Consider the system introduced in Example 2, Table 3.2 shows three traces st executed in the system model G and their local observations $M_1(st)$ and $M_2(st)$. From the analysis in Example 3, we know that when the system executes either the failure trace $a\sigma_f daca^n \in L(G) - pr(K)$ or $a\sigma_f cba^{n+1} \in L(G) - pr(K)$ $(n \in \mathcal{N})$, there exists one local diagnoser that can unambiguously detect that failure. However, when the system executes the non-faulty trace $abaca^n \in pr(K)$, both local diagnosers cannot unambiguously detect that no failure has happened. This is because regardless of what n is, $abaca^n \in pr(K)$ has the same observation as $a\sigma_f cba^{n+1} \in L(G) - pr(K)$ at the first local site, and as $a\sigma_f daca^n \in L(G) - pr(K)$ at the second local site.

Table 3.2 Possible system executions and their local observations

st	$M_1(st)$	$M_2(st)$
$abaca^n$	aba^{n+1}	$aaca^n$
$a\sigma_f daca^n$	ada^{n+1}	$aaca^n$
$a\sigma_f cba^{n+1}$	aba^{n+1}	aca^{n+1}

To capture the additional property of being able to be unambiguous about non-faulty executions, we introduce the notion of strong-codiagnosability. The purpose of this notion is that if a system executes an infinitely long non-faulty trace, then there must exist infinitely many

instances when it is known without ambiguity that no failure has occurred. Strong codiagnosability captures the ability of being certain about failure as well as non-failure conditions within bounded delay.

Definition 9 Let L be a system language and K be a specification language ($K \subseteq L$). Assume there are m local diagnosers with observation masks M_i ($i \in I = \{1, \cdots, m\}$). (L, K) is said to be strongly-codiagnosable with respect to $\{M_i\}$ if it is codiagnosable and

$$(\exists n \in \mathcal{N})(\forall s \in pr(K))(\forall st \in pr(K), |t| \geq n \text{ or } st \text{ deadlocks}) \Rightarrow$$

$$(\exists i \in I)(\forall u \in M_i^{-1} M_i(st) \cap L, u \in pr(K)). \tag{3.2}$$

Note that if an offending trace $s \in pr(K)$ exists, then non-faulty condition of s is not known for ever for some non-faulty extension beyond s. I.e., non-faulty condition for this extension is known at only finitely many instances, namely for some prefixes of s. In addition to the codiagnosability condition, the strong-codiagnosability requires that if s is a non-faulty trace in L and st is a non-faulty trace extended by sufficient number of transitions (or st deadlocks), then at least at one local site all st-indistinguishable traces are non-faulty as well.

To verify the strong-codiagnosability of (L, K), we need to check condition (3.2) as well as the codiagnosability of (L, K). Let $L(G) = L$ and $L(R) = pr(K)$. To check condition (3.2), we can construct a testing automaton to track traces $s \in L(R)$, $u_1 \in L(G) - L(R)$ and $u_2 \in L(G) - L(R)$ in the space $Y \times (X \times \overline{Y}) \times (X \times \overline{Y})$ with the properties $M_1(s) = M_1(u_1)$ and $M_2(s) = M_2(u_2)$. Then following the similar approach as in Algorithm 1 condition (3.2) can be verified by checking the absence of any "offending" cycles. Also, the computational complexity of this verification procedure is polynomial in the number of states and transitions of its testing automaton.

Remark 10 The notion of strong-diagnosability for centralized case can be defined as in Definition 9 with the local observation mask M_i replaced by the central observation mask M. We provide an example to illustrate that a diagnosable system may not be strongly-diagnosable even in the centralized case. Let $L = \{\epsilon, ab^*, a\sigma_f c^*\}$ and $K = \{\epsilon, ab^*\}$, where

$M(a) = a$, $M(c) = c$ and $M(b) = M(\sigma_f) = \epsilon$. It follows that (L, K) is diagnosable owing to the observability of event c. However, since $M(ab^*) = M(a\sigma_f) = a$, (L, K) is not strongly-diagnosable.

3.3 Safe-Codiagnosability

In this section, we introduce the notion of *safe-codiagnosability*, extending the notion of safe-diagnosability [46] to the decentralized setting. For a system, a certain sub-behavior is deemed safe (captured via a safety specification), and a further sub-behavior is deemed non-faulty (captured via a non-fault specification). Safe-codiagnosability requires that when the system executes a trace that is faulty, there exists at least one diagnoser that can detect this within bounded delay and also before the safety specification is violated. The above notion of safe-codiagnosability may also be viewed as an extension of the notion of codiagnosability. We show that safe-codiagnosability is equivalent to codiagnosability together with "zero-delay codiagnosability" of "boundary safe traces". (A safe trace is a boundary safe trace, if exists a single-event extension that is unsafe.) We give an algorithm of polynomial complexity for verifying safe-codiagnosability. For a safe-codiagnosable system, the same methods discussed for codiagnosable systems can be applied for off-line synthesis of individual diagnosers, as well as for on-line diagnosis using them.

3.3.1 Definition of Safe-Codiagnosability

Before introducing the definition of safe-codiagnosability, we first provide an alternative definition of codiagnosability in the following lemma. Without loss of generality, we assume a system to be diagnosed , a "plant", to be deadlock free.

Lemma 2 Let L and K be prefix-closed plant and non-fault specification languages respectively, and for $i \in I$, M_i be observation mask of site i. Then (L, K) is codiagnosable with respect to $\{M_i\}$ if and only if

$$\exists n \in \mathcal{N} : [(L - K)\Sigma^{\geq n} \cap L] \bigcap_{i \in I} M_i^{-1} M_i(K) = \emptyset.$$

Proof: The condition

$$(\exists i \in I)(\forall u \in M_i^{-1}M_i(st) \cap L, u \in L - K)$$

in the definition of codiagnosability requires that there exists a local site i such that any st-indistinguishable u at site i is faulty ($u \in L - K$). This can be rephrased as saying that it is not the case that for each site i exists a st-indistinguishable non-faulty trace $u_i \in K$, i.e.,

$$\neg(\forall i \in I)(\exists u_i \in M_i^{-1}M_i(st) \cap L, u_i \in K) \tag{3.3}$$

The set of traces,

$$\{w \mid \forall i \in I, \exists u_i \in M_i^{-1}M_i(w) \cap L, u_i \in K\}$$

is same as the set of traces $\bigcap_{i \in I} M_i^{-1}M_i(K)$. Thus the condition (3.3) can be equivalently written as, $st \notin \bigcap_{i \in I} M_i^{-1}M_i(K)$.

Further since $st \in L$ is a feasible extension of a faulty trace $s \in L - K$ with length of t at least the delay bound n, $st \in L \cap (L-K)\Sigma^{\geq n}$. It follows that the definition of codiagnosability of (L, K) may be rephrased as,

$$\exists n \in \mathcal{N} : [(L - K)\Sigma^{\geq n} \cap L] \bigcap_{i \in I} M_i^{-1}M_i(K) = \emptyset.$$

■

Remark 11 We can introduce the notion of "zero delay codiagnosability" by setting $n = 0$ in the definition of codiagnosability provided by Lemma 2. Then (L, K) is said to be *zero-delay codiagnosable* with respect to $\{M_i\}$ if

$$(L - K) \bigcap_{i \in I} M_i^{-1}M_i(K) = \emptyset. \tag{3.4}$$

We say a faulty sublanguage $H \subseteq L - K$ is zero-delay codiagnosable with respect to $\{M_i\}$ if

$H \bigcap_{i \in I} M_i^{-1} M_i(K) = \emptyset$. Note that (3.4) is equivalent to,

$$\bigcap_{i \in I} M_i^{-1} M_i(K) \cap L \subseteq K,$$

i.e., (L, K) is zero-delay codiagnosable if and only if the non-faulty behavior K is *decomposable* [57] with respect to the non-faulty+faulty (plant) behavior L.

Codiagnosability captures the system property that a failure event can be diagnosed within bounded delay after its occurrence by at least one of the local sites. In order to react to a failure in a timely fashion, it is also needed that a failure be detected before system behavior becomes "unsafe". Safe behavior includes all of non-faulty behavior and some of post-fault behavior where system performance may be degraded but still tolerable. The safety specification, denoted K_S, is another prefix-closed sublanguage of plant language, containing the non-fault specification, i.e., $K \subseteq K_S \subseteq L$. Then the notion of safe-codiagnosability can be formalized as follows.

Definition 10 Let L be the prefix-closed language generated by a plant, and K and K_S be prefix-closed non-fault and safety specification languages contained in L, respectively ($K \subseteq K_S \subseteq L$). Assume there are m local sites with observation masks $M_i : \overline{\Sigma} \rightarrow \overline{\Lambda}_i$ ($i \in I = \{1, \cdots, m\}$). (L, K, K_S) is said to be *safe-codiagnosable* with respect to $\{M_i\}$ if

$$\begin{aligned} &(\exists n \in \mathcal{N})(\forall s \in L - K)(\forall st \in L - K, |t| \geq n) \Rightarrow \\ &(\exists i \in I)(\exists v \in pr(st) \cap K_S)(\forall u \in M_i^{-1} M_i(v) \cap L, u \in L - K) \end{aligned} \tag{3.5}$$

Definition 10 has the following meaning. A system is safe-codiagnosable if there exists a delay bound n such that for all faulty trace $s \in L - K$ and all extension t of s with length longer than delay bound ($|t| \geq n$), there exists a site i and a safe prefix v of st such that for all v-indistinguishable u at site i, u is a faulty trace in $L - K$. Informally, Definition 10 means that for any faulty trace, there exists at least one local site that can unambiguously detect that failure within bounded delay and before safety is violated.

Just as we provided an alternative definition of codiagnosability in Lemma 2, we provide an alternative definition of safe-codiagnosability in the following lemma.

Lemma 3 Let L, K, and K_S be prefix-closed plant, non-fault specification, and safety specification languages respectively, and for $i \in I$, M_i be observation mask of site i. Then (L, K, K_S) is safe-codiagnosable with respect to $\{M_i\}$ if and only if

$$\exists n \in \mathcal{N} : [(L - K)\Sigma^{\geq n} \cap L] \cap supP[\bigcap_{i \in I} M_i^{-1} M_i(K) \cup K_S^c] = \emptyset.$$

Proof: The condition (3.5) in definition of safe-codiagnosability requires that exists a local site i and a safe prefix $v \leq st$ such that any v-indistinguishable u at site i is faulty ($u \in L - K$). This can be rephrased as saying that it is not the case that for each site i for each safe prefix $v \leq st$ exists a v-indistinguishable non-faulty trace $u_i \in K$, i.e.,

$$\neg(\forall i \in I)(\forall v \in pr(st) \cap K_S)(\exists u_i \in M_i^{-1} M_i(v) \cap L, u_i \in K) \tag{3.6}$$

The set of traces,

$$\{w \mid \forall i \in I, \forall v \in pr(w) \cap K_S,\ \exists u_i \in M_i^{-1} M_i(v) \cap L, u_i \in K\}$$

is same as the set of traces,

$$\{w \mid pr(w) \cap K_S \subseteq \bigcap_{i \in I} M_i^{-1} M_i(K)\},$$

which is the same set of traces,

$$\{w \mid pr(w) \subseteq \bigcap_{i \in I} M_i^{-1} M_i(K) \cup K_S^c\},$$

which is the set

$$supP[\bigcap_{i \in I} M_i^{-1} M_i(K) \cup K_S^c],$$

where $supP(\cdot)$ denotes the supremal prefix-closed language. Note that a trace w belongs to this last set if and only if $pr(w) \subseteq [\bigcap_{i \in I} M_i^{-1} M_i(K) \cup K_S^c]$, i.e., each prefix of w has the property that it is either unsafe (belongs to K_S^c) or for each i exists M_i-indistinguishable trace $u_i \in K$.

Thus the condition (3.5) can be equivalently written as, $st \notin supP[\bigcap_{i \in I} M_i^{-1} M_i(K) \cup K_S^c]$. On the other hand, based on the arguments used in the proof of Lemma 2, we know that $st \in L \cap (L - K)\Sigma^{\geq n}$. It follows that the definition of safe-codiagnosability of (L, K) may be rephrased as,

$$\exists n \in \mathcal{N} : [(L - K)\Sigma^{\geq n} \cap L] \cap supP[\bigcap_{i \in I} M_i^{-1} M_i(K) \cup K_S^c] = \emptyset.$$

■

3.3.2 Separatability of Safe-Codiagnosability

To facilitate the development of a test for safe-codiagnosability, we show that the property of safe-codiagnosability can be separated into codiagnosability together with zero-delay codiagnosability of set of boundary safe traces, where a boundary safe trace is a safe trace for which exists a single-event extension that is unsafe.

Definition 11 Given prefix-closed plant language L and safety specification language K_S, a safe trace $s \in K_S$ is called a *boundary safe trace* if exists $\sigma \in \Sigma$ such that $s\sigma \in L - K_S$, i.e., $s \in [(L - K_S)/\Sigma] \cap K_S$. The set of all boundary safe traces is called the *boundary safe language*, denoted K_S^∂, and is given by $K_S^\partial = [(L - K_S)/\Sigma] \cap K_S$.

We need the result of the following lemma before establishing the main "separation" result.

Lemma 4 Consider the prefix-closed non-fault specification language K and the observation masks $\{M_i\}$ $(i \in I)$. Then $\bigcap_{i \in I} M_i^{-1} M_i(K)$ is prefix-closed.

Proof: Prefix-closure of K implies prefix-closure of $M_i^{-1} M_i(K)$ for each $i \in I$. So the result follows since prefix-closure is preserved under intersection. ■

The following theorem presents the "separation property" of safe-codiagnosability, based on which we develop the test for safe-codiagnosability.

Theorem 2 Let L, K and K_S be plant language, non-fault specification language, and safety specification language, respectively. (L, K, K_S) is safe-codiagnosable with respect to $\{M_i\}$ if and only if

1. (L, K) codiagnosable with respect to $\{M_i\}$: $\exists n \in \mathcal{N} : [(L-K)\Sigma^{\geq n} \cap L] \bigcap_{i\in I} M_i^{-1}M_i(K) = \emptyset$;
2. K_S^{∂} zero-delay codiagnosable with respect to $\{M_i\}$: $K_S^{\partial} \bigcap_{i\in I} M_i^{-1}M_i(K) = \emptyset$.

Proof: (if) From the property of codiagnosability exists a delay bound n such that condition of codiagnosability is satisfied. We claim that the same delay bound works for the definition of safe-codiagnosability. To see this, pick $s \in L - K$ and $t \in \Sigma^*$ such that $|t| \geq n$ and $st \in L$. Then $st \in [(L-K)\Sigma^{\geq n} \cap L]$. We need to show that $st \notin supP[\bigcap_{i\in I} M_i^{-1}M_i(K) \cup K_S^c]$, i.e., exists a prefix $v \leq st$ such that $v \notin \bigcap_{i\in I} M_i^{-1}M_i(K) \cup K_S^c$. Since $L - K = (K_S - K) \cup (L - K_S)$, $st \in L - K$ implies either $st \in K_S - K$ or $st \in L - K_S$.

For the first case ($st \in K_S - K$), we can set $v = st$. Then v is a prefix of st, and also since $v = st \in K_S$, it holds that $v \notin K_S^c$. It remains to be shown that $v = st \notin \bigcap_{i\in I} M_i^{-1}M_i(K)$, which holds from the property of codiagnosability since $st \in [(L-K)\Sigma^{\geq n} \cap L]$ and $[(L-K)\Sigma^{\geq n} \cap L] \bigcap_{i\in I} M_i^{-1}M_i(K) = \emptyset$.

For the second case ($st \in L - K_S$), suppose for contradiction that for every prefix $v \leq st$, it holds that $v \in \bigcap_{i\in I} M_i^{-1}M_i(K) \cup K_S^c$. Since $st \in L - K_S$, exists a prefix $w \leq st$ that is a boundary safe trace, i.e., $w \in K_S^{\partial}$. From our supposition, $w \in \bigcap_{i\in I} M_i^{-1}M_i(K) \cup K_S^c$. So,

$$\begin{aligned} w &\in [(L - K_S)/\Sigma \cap K_S] \cap [\bigcap_{i\in I} M_i^{-1}M_i(K) \cup K_S^c] \\ &= [(L - K_S)/\Sigma \cap K_S] \cap [\bigcap_{i\in I} M_i^{-1}M_i(K)]. \end{aligned}$$

Then we arrive at a contradiction to the condition: $K_S^{\partial} \bigcap_{i\in I} M_i^{-1}M_i(K) = \emptyset$.

(only if) From Lemma 3 we have,

$$\exists n \in \mathcal{N} : [(L - K)\Sigma^{\geq n} \cap L] \cap supP[\bigcap_{i\in I} M_i^{-1}M_i(K) \cup K_S^c] = \emptyset.$$

This implies,

$$\exists n \in \mathcal{N} : [(L - K)\Sigma^{\geq n} \cap L] \cap supP[\underset{i \in I}{\cap} M_i^{-1} M_i(K)] = \emptyset.$$

Further from Lemma 4, $supP[\underset{i \in I}{\cap} M_i^{-1} M_i(K)] = \underset{i \in I}{\cap} M_i^{-1} M_i(K)$. So we also have

$$\exists n \in \mathcal{N} : [(L - K)\Sigma^{\geq n} \cap L] \underset{i \in I}{\cap} M_i^{-1} M_i(K) = \emptyset,$$

establishing the codiagnosability. ■

Next to show the zero-delay codiagnosability of boundary safe traces, pick a boundary safe trace $w \in K_S^{\partial}$. Then exists $\sigma \in \Sigma$ such that $w\sigma \in L - K_S$, and we need to show that $w \notin \underset{i \in I}{\cap} M_i^{-1} M_i(K)$. Set $s = w\sigma \in L - K_S \subseteq L - K$, and pick t such that $|t| \geq n$ and $st \in L$ (which is possible from our underlying assumption of plant being deadlock free). Then $st \in [(L - K)\Sigma^{\geq n} \cap L]$. From the assumption of safe-codiagnosability, $st \notin supP[\underset{i \in I}{\cap} M_i^{-1} M_i(K) \cup K_S^c]$, which implies every prefix of st, including $w \notin \underset{i \in I}{\cap} M_i^{-1} M_i(K) \cup K_S^c$. From this it follows that $w \notin \underset{i \in I}{\cap} M_i^{-1} M_i(K)$, as desired. ■

3.3.3 Verification of Safe-Codiagnosability

The algorithm for verifying safe-codiagnosability is based upon checking whether there exists a situation that violates the conditions of safe-codiagnosability. From Theorem 2, we know that safe-codiagnosability can be verified by checking codiagnosability of (L, K) together with zero-delay codiagnosability of K_S^{∂}, the set of boundary safe traces.

Algorithm 3 Consider the finite state machine models, $G = (X, \Sigma, \alpha, x_0)$, $R = (Y, \Sigma, \beta, y_0)$, and $R_S = (Y_S, \Sigma, \beta_S, y_0^S)$, respectively, of the plant, the non-fault specification, and the safety specification. The corresponding plant, non-fault specification, and safety specification languages are $L = L(G), K = L(R)$, and $K_S = L(R_S)$, respectively, where $K \subseteq K_S \subseteq L$. Let M_i be the observation mask of site i ($i \in I$). To check the safe-codiagnosability of (L, K, K_S), perform the following steps:

Step 1: Check the codiagnosability of (L, K)

Construct a testing automaton $T = (G\|\overline{R}) \times R \times R$ for verifying the codiagnosability of (L, K). This automaton is defined as $T = (Z, \Sigma_T, \gamma, z_0)$, where

. $Z = (X \times \overline{Y}) \times Y \times Y$.

. $\Sigma_T = \overline{\Sigma}^3$, where $\overline{\Sigma} = \Sigma \cup \{\epsilon\}$.

. $z_0 = ((x_0, y_0), y_0, y_0)$.

. $\gamma : Z \times \overline{\Sigma}^3 \to Z$ is defined as: $\forall z = ((x, y), y_1, y_2) \in Z, \sigma_T = (\sigma, \sigma_1, \sigma_2) \in \Sigma_T - \{(\epsilon, \epsilon, \epsilon)\}$, $\gamma(z, \sigma_T) := ((\alpha(x, \sigma), \overline{\beta}(y, \sigma)), \beta(y_1, \sigma_1), \beta(y_2, \sigma_2))$ if and only if

$$M_1(\sigma) = M_1(\sigma_1)] \wedge [M_2(\sigma) = M_2(\sigma_2)]$$
$$\wedge[(\alpha(x, \sigma) \neq \emptyset) \vee (\beta(y, \sigma) \neq \emptyset) \vee (\beta(y_1, \sigma_1) \neq \emptyset) \vee (\beta(y_2, \sigma_2) \neq \emptyset)]$$

Note that the silent-transition ϵ is defined at each state of any automaton as a self loop by default. The testing automaton T is used to track if exists a triplet of traces s, u_1 and u_2 such that u_i is a s-indistinguishable non-fault trace under mask M_i ($i \in \{1, 2\}$).

Then check if exists an "offending cycle" $cl_T = (z^k, \sigma_T^k, z^{k+1}, \cdots, z^l, \sigma_T^l, z^k)$ such that

$$\exists i \in [k, l] \text{ s.t. } (\overline{y}^i = F) \wedge (\sigma^i \neq \epsilon), \tag{3.7}$$

where $z^i = ((x^i, \overline{y}^i), y_1^i, y_2^i) \in Z$, and $\sigma_T^i = (\sigma^i, \sigma_1^i, \sigma_2^i) \in \Sigma_T$. If the answer is yes, then (L, K) is not codiagnosable, and (L, K, K_S) is not safe-codiagnosable as well. Otherwise, go to the next step.

Step 2: Compute the set of "boundary safe states" B in $G\|R_S$

Construct the composition $G\|R_S$, and define the set of *boundary safe states* as, $B := \{(x, y_S) \in X \times Y_S | \exists \sigma \in \Sigma : \alpha(x, \sigma) \neq \emptyset, \beta_S(y_s, \sigma) = \emptyset\}$. Note that if $s \in L(G\|R_S) = L(G) \cap L(R_S) = L \cap K_S = K_S$ is such that execution of s results in reaching a state $(x, y_S) \in B$, then exists $\sigma \in \Sigma$ such that $s\sigma \in L - K_S$, i.e., $s \in (L - K_S)/\Sigma$. It follows that $s \in K_S^\partial$.

Step 3: Check the zero-delay codiagnosability of K_S^∂ with respect to $\{M_i\}$

Construct a testing automaton $T_S = (G\|R_S) \times R \times R$ for verifying the zero-delay codiagnosability of K_S^∂, where T_S is obtained by replacing $\overline{R}$ by R_S in the testing automaton T

constructed above. Let $T_S = (Z_S, \Sigma_T, \gamma_s, z_0^S)$, where Z_S, γ_s, and z_0^S of T_S are defined similarly as Z, γ, and z_0 of T, respectively (with $\overline{R}$ replaced by R_S). Then check if exists an "offending state" $((x, y_s), y_1, y_2)$ in T_S with $(x, y_s) \in B$. K_S^∂ is zero-delay codiagnosable if and only if the answer is no. If K_S^∂ is zero-delay codiagnosable, then (L, K, K_S) is safe-codiagnosable as well (since (L, K) was determined to be codiagnosable above). Otherwise, (L, K, K_S) is not safe-codiagnosable.

Since the correctness of the test for checking codiagnosability was established in Theorem 1, in the following theorem we show the correctness of the test for checking zero-delay codiagnosability of K_S^∂.

Theorem 3 K_S^∂ is not zero-delay codiagnosable with respect to $\{M_i\}$ if and only if there exists a state $z_s = ((x, y_s), y_1, y_2)$ in the testing automaton T_S with $(x, y_s) \in B$.

Proof: ($\Leftarrow$) If there is a state $((x, y_s), y_1, y_2)$ in T_S such that $(x, y_s) \in B$, then exist traces $s \in L(G \| R_S)$, $u_i \in L(R) = K$ such that (i) $s \in K_S^\partial = (L - K_S)/\Sigma \cap K_S$, and (ii) $M_i(s) = M_i(u_i)$. This implies that $s \in K_S^\partial \bigcap_{i \in I} M_i^{-1} M_i(K)$, i.e., $K_S^\partial \bigcap_{i \in I} M_i^{-1} M_i(K) \neq \emptyset$. Thus, K_S^∂ is not zero-delay codiagnosable with respect to $\{M_i\}$.

($\Rightarrow$) If K_S^∂ is not zero-delay codiagnosable with respect to $\{M_i\}$, then exists a boundary safe trace $s \in K_S^\partial$ such that $s \in \bigcap_{i \in I} M_i^{-1} M_i(K)$, which implies that for $i = 1, 2$, there exist $u_i \in K$ such that $M_i(u_i) = M_i(s)$. Then execution of the trace triple (s, u_1, u_2) in T_S results in a state $((x, y_s), y_1, y_2)$. Since $s \in K_S^\partial$, $(x, y_s) \in B$, proving the assertion. ∎

From Theorem 3 and Theorem 1, we get the following corollary showing the correctness of Algorithm 3.

Corollary 1 Let $G = (X, \Sigma, \alpha, x_0)$, $R = (Y, \Sigma, \beta, y_0)$ and $R_S = (Y_S, \Sigma, \beta_S, y_0^S)$ be the plant, non-fault specification and safety specification models, respectively, with $[K = L(R)] \subseteq [K_S = L(R_S)] \subseteq [L = L(G)]$. Let M_i be the observation mask of site i ($i \in I$). (L, K, K_S) is *not* safe-codiagnosable with respect to $\{M_i\}$ if and only if one of the following conditions holds:

1. There exists an "offending" cycle $cl_T = (z^k, \sigma_T^k, z^{k+1}, \cdots, z^l, \sigma_T^l, z^k)$ as defined in (3.7) in the testing automaton T;

2. There exists a "offending" state $z_s = ((x, y_s), y_1, y_2)$ in the testing automaton T_S with $(x, y_s) \in B$.

Remark 12 Let $|X|$, $|Y|$ and $|Y_S|$ be the number of states in plant G, non-fault specification R, and safety specification R_S respectively, and $|\Sigma|$ be the number of events. $L = L(G), K = L(R), K_S = L(R_S)$. Assume there are m local sites. It was shown in [52] that the complexity for constructing the testing automaton T and checking codiagnosability of (L, K) is $\mathcal{O}(|X| \times |Y|^{m+1} \times |\Sigma|^{m+1})$. Using a similar analysis, we can verify that the complexity for constructing the testing automaton T_S and checking the zero-delay codiagnosability of K_S^∂ is $\mathcal{O}(|X| \times |Y_S| \times |Y|^m \times |\Sigma|^{m+1})$. It follows that overall complexity of checking safe-codiagnosability of (L, K, K_S) is, $\mathcal{O}(|X| \times (|Y| + |Y_S|) \times |Y|^m \times |\Sigma|^{m+1})$.

Remark 13 In Algorithm 3, we use two testing automata T and T_S to verify the safe-codiagnosability of (L, K, K_S). These two testing automata can be combined into a testing automaton $T' = (G \| R_S \| \overline{R}) \times R \times R$ by replacing $\overline{R}$ by $R_S \| \overline{R}$ in T. Then, (L, K, K_S) is not safe-codiagnosable if and only if there exists an "offending cycle" containing a state with the third coordinate labeled by "F", or exists an "offending state" with its first pair of coordinates contained in B. However, in this case, the complexity is $\mathcal{O}(|X| \times |Y_S| \times |Y|^{m+1} \times |\Sigma|^{m+1})$, which is an order higher. Thus the "separation" result obtained in Theorem 2 provides an order reduction in the complexity of testing safe-codiagnosability.

Once a system is deemed safe-codiagnosable, the same methods for codiagnosable systems can be applied for the synthesis of local diagnosers as well as for on-line diagnosis using them. This is because a diagnoser simply observes the plant behavior and reports a fault when it becomes certain about it. The property of safe-codiagnosability guarantees that at least one diagnoser become certain within bounded delay of the occurrence of a fault and prior to the system behavior becoming unsafe. The following example illustrates how to verify the safe-codiagnosability using Algorithm 3.

Example 6 Figure 3.7 (a), (b) and (c) show a plant model G, a non-fault specification model R, and a safety specification model R_S. The set of events is given by $\Sigma = \{a, b, f\}$. There are

two local sites, with their observation masks given as follows:

$$M_1(a) = a, M_1(b) = M_1(f) = \epsilon; M_1(b) = b, M_2(a) = M_2(f) = \epsilon.$$

It can be verified that $(L(G), L(R))$ is codiagnosable with respect to $\{M_i\}$ by constructing a testing automaton $T = (G\|\overline{R}) \times R \times R$, which is omitted here.

Since $L = L(G) = pr(ab^* + faab^*)$ and $K_{S_1} = pr(ab^* + fa)$, the boundary safe language $K^{\partial}_{B_1} = [(L - K_{S_1})/\Sigma] \cap K_{S_1} = \{fa\}$. Following the trace fa, state "3" in G and state "3" in R_{S_1} are reached. Thus, the set of boundary safe states is given by, $B_1 = \{(3,3)\}$. Figure 3.7 (d) shows a part of the testing automaton $T_{S_1} = (G\|R_{S_1}) \times R \times R$, where an offending state $((3,3),1,1)$ is reached. Therefore, $K^{\partial}_{B_1}$ is not zero-delay codiagnosable with respect to $\{M_i\}$, and thus (L, K, K_{S_1}) is not safe-codiagnosable with respect to $\{M_i\}$ as well.

Now, if we relax the safety requirement by considering a new enlarged safety specification model R_{S_2} as shown in Figure 3.8 (a), the system becomes safe-codiagnosable. To see this, since $K_{S_2} = pr(ab^* + faa)$, the boundary safe language is given by, $K^{\partial}_{B_2} = [(L - K_{S_2})/\Sigma] \cap K_{S_2} = \{faa\}$. Thus, the set of boundary safe states is given by, $B_2 = \{(4,4)\}$. The new testing automaton $T_{S_2} = (G\|R_{S_2}) \times R \times R$ is shown in Figure 3.8 (b), where no offending states (states with first pair of coordinates being $(4,4)$) are reached. Therefore, $K^{\partial}_{B_2}$ is zero-delay codiagnosable with respect to $\{M_i\}$, and thus (L, K, K_{S_2}) is safe-codiagnosable with respect to $\{M_i\}$ as well.

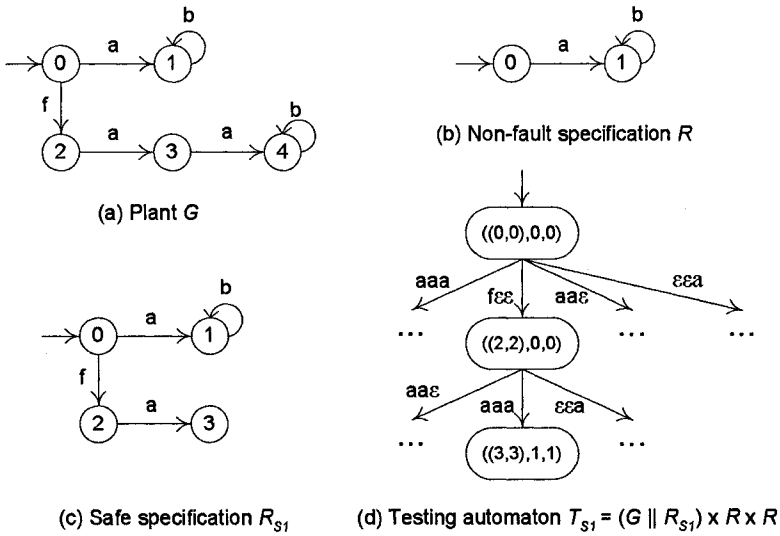


Figure 3.7 Models G, R and R_{S_1}, and testing automaton T_{S_1}

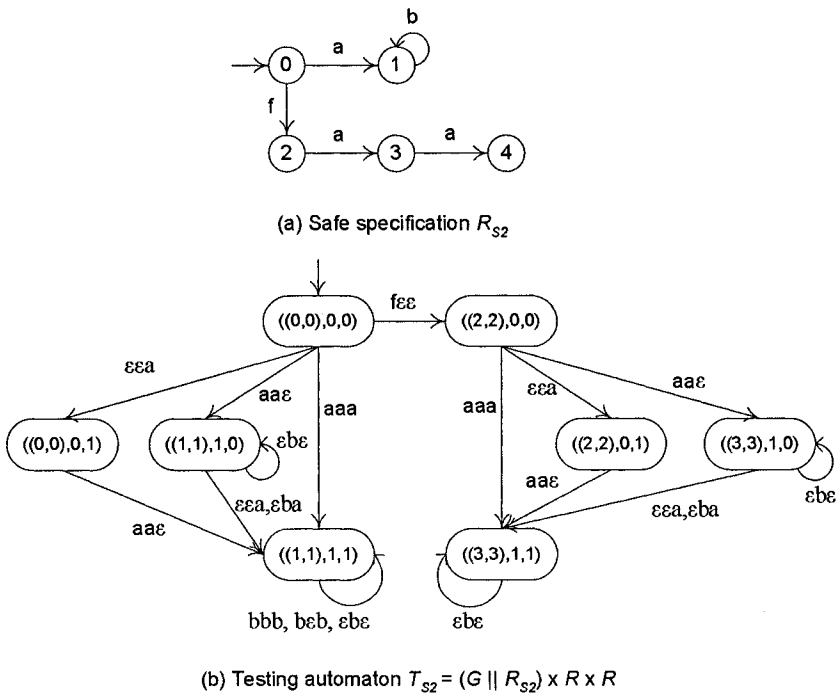


Figure 3.8 Safe specification model R_{S_2} and testing automaton T_{S_2}

CHAPTER 4. DISTRIBUTED FAILURE DIAGNOSIS

In this chapter, we study *distributed* failure diagnosis under bounded/unbounded communication delay. We first discuss communication protocols for distributed failure diagnosis. Then a notion of *joint$_k$-diagnosability* is introduced so that any failure can be diagnosed within a bounded delay of its occurrence by one of the local sites using its own observations and the communicated information from other local sites. Next, diagnosis using an "immediate observation passing (iop)" protocol is discussed, and is converted to a decentralized diagnosis problem using extended system and specification models. Results from Chapter 3 are applied for verifying joint$_k^{iop}$-diagnosability, and for synthesizing local diagnosers. Comparison is given among various notions of diagnosability, codiagnosability, and joint$_k^{iop}$-diagnosability. Next, we discuss the problem of distributed diagnosis under unbounded communication, and show that it is decidable by establishing the equivalence between joint$_\infty$-diagnosability and codiagnosability.

4.1 Communication Protocols

Figure 4.1 shows the architecture of a distributed failure diagnosis system with two local sites. Site i contains three modules: observation mask M_i, communication protocol i, and diagnoser i. The observation mask module M_i is a map $M_i : \Sigma^* \to \Lambda_i^*$. The protocol module for site i decides how to share information among various diagnosers. The diagnoser module for site i performs failure diagnosis based on the local observations and the communicated information received from other sites j ($j \neq i$). Information is communicated among various sites over communication channels that are loss-free and order-preserving but introduce bounded delays.

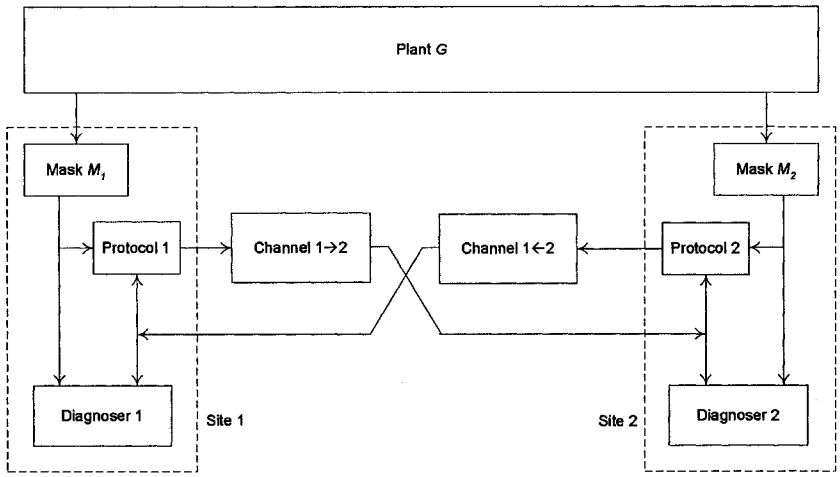


Figure 4.1 Architecture of a distributed failure diagnosis system

A communication protocol is a causal (prefix-preserving) map from history of all information received to history of all information transmitted. The communication protocol at site i can be implemented as a dynamical system as shown in Figure 4.2.

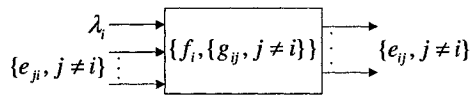


Figure 4.2 Model of a general communication protocol

The inputs to this dynamical system consist of the *local observations* $\lambda_i \in \Lambda_i$, and the *communicated informations* from each diagnoser $\{e_{ji} | j \neq i\}$. The output of this dynamical system is the *information to be transmitted* to other diagnosers $\{e_{ij} | j \neq i\}$. The protocol maintains an internal state e_i, called the *protocol state.* The formats of e_i, e_{ji}, and e_{ij} $(j \neq i)$ are specific to a protocol. Formally, a general communication protocol is given by $P^{gen} := \{P_i^{gen}, i \in I\}$, where each P_i^{gen} is modeled by a set of maps $\{f_i, \{g_{ij}, j \neq i\}\}$ as follows:

$$P_i^{gen} : \begin{cases} e_i = f_i(e_i, \lambda_i, \{e_{ji}, j \neq i\}) & \text{(protocol state update),} \\ e_{ij} = g_{ij}(e_i, \lambda_i, \{e_{ji}, j \neq i\}) & \text{(protocol output computation).} \end{cases} \tag{4.1}$$

f_i is the *protocol state update map* at site i, which updates the protocol state based on its current value and newly received information, λ_i or $\{e_{ji}, j \neq i\}$. (Events in $\{\lambda_i\} \cup \{e_{ji}, j \neq$

$i\}\cup\{e_{ij}, j \neq i\}$ occur asynchronously.) g_{ij} is the *protocol-output map* at site i, which determines the information to be transmitted to site j ($j \neq i$). The set of protocols of the form specified in (4.1) is denoted $\mathcal{P}^{gen}$. The setting of decentralized diagnosis involving no communication can be represented by a "null-communication" protocol, $P^{\emptyset}$, for which the output is always null.

If a protocol allows diagnosers to transmit only their local observations, it is called an *observation passing protocol*, denoted $P^{op} := \{P_i^{op}, i \in I\}$. The dynamic model of P_i^{op} is captured by a set of maps $\{f_i, \{g_{ij}, j \neq i\}\}$ defined as follows. ("\" denotes the "after" operation.)

$$P_i^{op} : \begin{cases} e_i = e_i\lambda_i \backslash \{e_{ij}, j \neq i\} & \text{(protocol state update)}, \\ e_{ij} = g_{ij}(e_i, \lambda_i) \leq e_i\lambda_i & \text{(protocol output computation)}, \end{cases} \tag{4.2}$$

where the protocol state e_i is a "vector" whose j^{th} entry stores observations that are not yet transmitted to site j, and e_{ij} is the newly transmitted observation to site j, which is a prefix of $e_i\lambda_i$, the concatenation of the observation trace not yet transmitted and the newly arrived observation. The class of protocols of the form specified in (4.2) comprise the class of observation passing protocols, denoted $\mathcal{P}^{op}$.

When $g_{ij}(e_i, \lambda_i) = e_i\lambda_i = \epsilon\lambda_i = \lambda_i$ in (4.2), the protocol model simplifies to the one given in (4.3), and the corresponding protocol is called the *immediate observation passing protocol*, denoted $P^{iop} := \{P_i^{iop}, i \in I\}$, where P_i^{iop} is defined as follows:

$$P_i^{iop} : \begin{cases} e_i = \epsilon & \text{(protocol state update)}, \\ e_{ij} = \lambda_i & \text{(protocol output computation)}. \end{cases} \tag{4.3}$$

In this protocol any local observation is transmitted immediately to other sites, and there is no observation that is not transmitted but stored as the protocol state. Therefore, the protocol state update map f_i is trivial, and the information to be transmitted e_{ij} simply equals the local observation λ_i in the protocol output map g_{ij}. The distributed diagnosis architecture under the immediate observation passing protocol is shown in Figure 4.3(a), and a rearrangement of

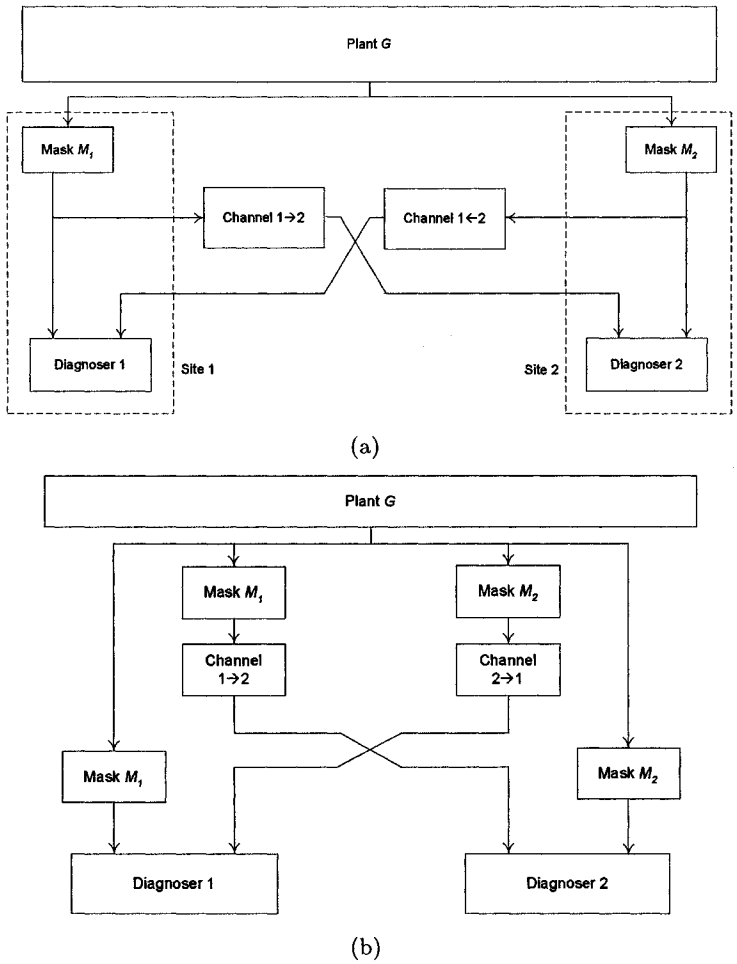


Figure 4.3 Distributed diagnosis architecture under protocol P^{iop}

the same is shown in Figure 4.3(b).

4.2 Joint$_k$-Diagnosability

In this section, we introduce a notion of *joint$_k$-diagnosability*, which captures the property of a system in which any failure can be diagnosed within a bounded delay of its occurrence by one of the local sites using its own observations and the delayed information sent from other local sites communicating using a general communication protocol $P^{gen} \in \mathcal{P}^{gen}$. When the communication is P^{iop}-based, the notion of distributed diagnosability is denoted by *joint$_k^{iop}$-diagnosability*, which will be discussed in the next section.

Let E_{ij} $(i \neq j, i, j \in I)$ denote the set of output symbols communicated from site i to site j. Then any symbol received at site i lies in the set $\Lambda_i \cup_{j \neq i} E_{ji}$, called the set of *aggregate observation symbols* at site i. Assuming local sites commute over loss-free, FIFO, and k-bounded delay channels under a general communication protocol $P^{gen} \in \mathcal{P}^{gen}$, the execution of a trace s by the system results in the reception at site i of a sequence of observation symbols in Λ_i interleaved with a sequence of communication symbols in $\cup_{j \neq i} E_{ji}$. Due to the asynchronous nature of the communication channels and the introduction of bounded but random delays by them, execution of a trace s by the system can result in the reception of one of many possible sequences of observed and communicated symbols at site i. Also, any such sequence of observed and communicated symbols arrives in its entirety at site i within a bounded-delay of the execution of trace s.

To characterize the set of sequences of aggregate observations received at site i under protocol $P^{gen} \in \mathcal{P}^{gen}$ immediately at the time the system has executed a trace s, we define a map $O_i^{gen,k} : \Sigma^* \rightarrow 2^{(\Lambda_i \cup_{j \neq i} E_{ji})^*}$, where k is the communication delay bound. We call this map to be the *P^{gen}-based aggregate observations map* for site i under k-bounded communication delay. Similarly, we can define $O_i^{op,k}$ (resp., $O_i^{iop,k}$ and $O_i^{\emptyset}$) to be the *P^{op}(resp., P^{iop} and $P^{\emptyset}$)-based aggregate observations map* for site i under k-bounded communication delay. Since $P^{\emptyset}$ represents the "null communication" protocol, it is obvious that $O_i^{\emptyset}(s) = \{M_i(s)\}$ for any $s \in L(G)$. For protocol P^{iop}, the aggregate observations map $O_i^{iop,k}$ can be formally defined through the extended observation mask $\mathcal{M}_i$ as follows.

Definition 12 Given a plant G, let $\mathcal{G}^k = G \| C_{12}^k \| C_{21}^k$ be the extended plant model, where C_{ij}^k $(i, j \in \{1, 2\}, i \neq j)$ is the k-delaying&masking model from site i to site j. Consider the extended observation mask $\mathcal{M}_i : \Sigma \cup \Lambda_1 \cup \Lambda_2 \rightarrow \overline{\Lambda}_1 \cup \overline{\Lambda}_2$ $(i \in \{1, 2\})$, and the inverse projection $\Pi_\Sigma^{-1} : \Sigma^* \rightarrow (\Sigma \cup \Lambda_1 \cup \Lambda_2)^*$. The *$P^{iop}$-based aggregate observations map* for site i under k-bounded communication delay, $O_i^{iop,k} : \Sigma^* \rightarrow 2^{(\cup_{j \in I} \Lambda_j)^*}$, is given by:

$$\forall s \in L(G) : O_i^{iop,k}(s) := \mathcal{M}_i(\Pi_\Sigma^{-1}(s) \cap L(\mathcal{G}^k)).$$

Two traces s and t are indistinguishable if they possess a common aggregate observations sequence. This is captured by the following definition of indistinguishability predicate over the set of traces.

Definition 13 Given a plant G, let $O_i^{gen,k} : \Sigma^* \to 2^{(\Lambda_i \cup_{j \neq i} E_{ji})^*}$ be the aggregate observations map for site i. Then the *P^{gen}-based indistinguishability predicate* for site i under k-bounded communication delay is defined as follows:

$$\forall s,t \in L(G), \Upsilon_i^{gen,k}(s,t) := \begin{cases} 1 & O_i^{gen,k}(s) \cap O_i^{gen,k}(t) \neq \emptyset \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

The *P^{op}(resp., P^{iop} and $P^{\emptyset}$)-based indistinguishability predicate* for site i under k-bounded communication delay, denoted $\Upsilon_i^{op,k}$ (resp., $\Upsilon_i^{iop,k}$ and $\Upsilon_i^{\emptyset}$), are defined similarly by replacing $O_i^{gen,k}$ in (4.4) with $O_i^{op,k}$ (resp., $O_i^{iop,k}$ and $O_i^{\emptyset}$).

The following proposition shows the relationship between indistinguishability predicates $\Upsilon_i^{\emptyset}$ and $\Upsilon_i^{iop,k}$, and observation masks $\{M_i\}$ and $\{\mathcal{M}_i\}$.

Proposition 1 Given a plant G, and the extended plant model $\mathcal{G}^k$, let $\Upsilon_i^{iop,k}$ (resp., $\Upsilon_i^{\emptyset}$) be the P^{iop}(resp., $P^{\emptyset}$)-based indistinguishability predicates for site i under k-bounded communication delay, $M_i : \Sigma \to \overline{\Lambda}_i$ be the observation mask for site i, and $\mathcal{M}_i : \Sigma \cup \Lambda_1 \cup \Lambda_2 \to \overline{\Lambda}_1 \cup \overline{\Lambda}_2$ be the extended observation mask for site i. Then, we have that

- $\forall s,t \in L(G)$: $[\Upsilon_i^{\emptyset}(s,t) = 1] \Leftrightarrow [M_i(s) = M_i(t)]$, and
- $\forall s,t \in L(G)$: $[\Upsilon_i^{iop,k}(s,t) = 1] \Leftrightarrow \exists s' \in \Pi_{\Sigma}^{-1}(s) \cap L(\mathcal{G}^k), t' \in \Pi_{\Sigma}^{-1}(t) \cap L(\mathcal{G}^k) : \mathcal{M}_i(s') = \mathcal{M}_i(t')$.

where $\Pi_{\Sigma}^{-1} : \Sigma^* \to (\Sigma \cup \Lambda_1 \cup \Lambda_2)^*$ is the inverse projection defined in (4.8).

Proof: Since $O_i^{\emptyset}(s) = \{M_i(s)\}$ for any $s \in L(G)$, the first assertion readily follows from the definition of $\Upsilon_i^{\emptyset}$.

According to Definition 13, $\Upsilon_i^{iop,k}(s,t) = 1$ if and only if $O_i^{iop,k}(s) \cap O_i^{iop,k}(t) \neq \emptyset$. Since $O_i^{iop,k}(s) = \mathcal{M}_i(\Pi_{\Sigma}^{-1}(s) \cap L(\mathcal{G}^k))$ and $O_i^{iop,k}(t) = \mathcal{M}_i(\Pi_{\Sigma}^{-1}(t) \cap L(\mathcal{G}^k))$, it follows that there

exist s' and t' in $L(\mathcal{G}^k)$ such that $[\Pi_\Sigma(s') = s] \wedge [\Pi_\Sigma(t') = t] \wedge [\mathcal{M}_i(s') = \mathcal{M}_i(t')]$, proving the correctness of the second assertion. ■

Using the notion of indistinguishability predicate, we next give the definition of joint$_k$-diagnosability.

Definition 14 Let L be the prefix-closed language generated by a plant, and K be a prefix-closed specification language contained in L ($K \subseteq L$). Assume there are m local sites ($I = \{1, \cdots, m\}$) communicating over k-bounded delay FIFO channels under protocol $P^{gen} \in \mathcal{P}^{gen}$. Then (L, K) is said to be joint$_k$-diagnosable under P^{gen}, called *joint$_k^{gen}$-diagnosable*, if

$$(\exists n \in \mathcal{N})(\forall s \in L - K)(\forall st \in L, |t| \geq n \text{ or } st \text{ deadlocks}) \Rightarrow$$

$$(\exists i \in I)(\forall u \in L, \Upsilon_i^{gen,k}(st, u) = 1 \Rightarrow u \in L - K), \tag{4.5}$$

where $\Upsilon_i^{gen,k}$ is the P^{gen}-based indistinguishability predicate for site i under k-bounded communication delay defined in (4.4). The joint$_k$-diagnosability with respect to protocol $P^{op} \in \mathcal{P}^{op}$ and P^{iop}, denoted *joint$_k^{op}$-diagnosability* and *joint$_k^{iop}$-diagnosability*, respectively, are defined by replacing $\Upsilon_i^{gen,k}$ in (4.5) with $\Upsilon_i^{op,k}$ and $\Upsilon_i^{iop,k}$, respectively.

The above definition has the following meaning. For any faulty trace s ($s \in L - K$) that is extended by a sufficiently long trace ($|t| \geq n$) or is extended to a deadlocking trace (st deadlocks), there exists at least one local site i such that any trace u in L that is indistinguishable from st at site i under protocol P^{gen} (resp., P^{op}, P^{iop}) and k-bounded communication delay, i.e. $\Upsilon_i^{gen,k}(st, u) = 1$ (resp., $\Upsilon_i^{op,k}(st, u) = 1$, $\Upsilon_i^{iop,k}(st, u) = 1$), belongs to the faulty language $L - K$.

4.3 P^{iop}-Based Distributed Diagnosis Under Bounded-Delay Communication

In this section, we discuss distributed diagnosis under bounded-delay communication based on an immediate observation passing protocol.

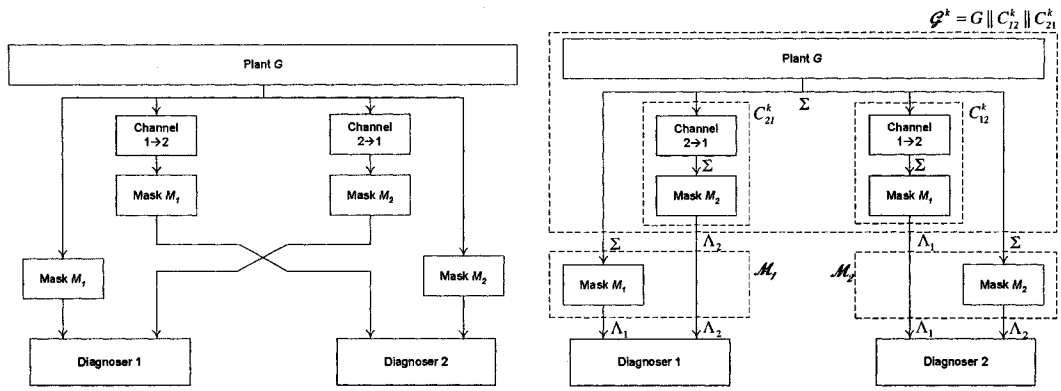


Figure 4.4 Equivalent system architecture under protocol P^{iop}

4.3.1 Communication Delay & Extended System/Specification Models

We first model the effect of communication delay when using P^{iop}-based distributed diagnosis, and introduce the extended system and specification models. The idea is to use the extended models to convert the P^{iop}-based distributed failure diagnosis problem to an instance of a decentralized failure diagnosis problem, where no communication is exchanged among local diagnosers. The former problem can then be solved using the methods for the latter problem developed in Chapter 3.

In a channel with k-bounded communication delay, there can be at most k events executed by the plant between the transmission and the reception of a message on the channel. Since the operations of masking and delaying can be interchanged, the behavior under the block diagram of Figure 4.3(b) is equivalent to that of Figure 4.4(a), which can be rearranged to obtain the block diagram of Figure 4.4(b).

It is clear by comparing Figure 3.1 and Figure 4.4(b) that P^{iop}-based distributed diagnosis can be converted to a decentralized diagnosis having an extended plant $\mathcal{G}^k$, and local diagnosers having the extended observation masks $\{\mathcal{M}_i\}$. The extended plant is given by $\mathcal{G}^k = G \| C_{12}^k \| C_{21}^k$, where C_{ij}^k models the k-bounded delaying and masking operation. We call C_{ij}^k to be the *k-delaying&masking* model. The *extended plant* $\mathcal{G}^k$ "generates" events in $\Sigma \cup \Lambda_1 \cup \Lambda_2$, which are observed by diagnoser i through an extended observation mask $\mathcal{M}_i : \Sigma \cup \Lambda_1 \cup \Lambda_2 \to \overline{\Lambda}_1 \cup \overline{\Lambda}_2$. $\mathcal{M}_i$ is same as M_i for events in Σ, whereas it acts as an identity

mask for events in Λ_j $(j \neq i)$ and blocks the events in Λ_i. Formally, it is defined as follows:

$$\mathcal{M}_i(\sigma) := \begin{cases} M_i(\sigma), & \sigma \in \Sigma; \\ \sigma, & \sigma \in \Lambda_j \ (j \neq i); \\ \epsilon, & \sigma \in \Lambda_i. \end{cases} \tag{4.6}$$

The k-delaying&masking model C_{ij}^k can be modeled as a finite automaton, which evolves whenever a new event occurs in the plant or a transmitted observation is delivered to a destination diagnoser, and such arrival and departure of events occur asynchronously. The state of the model is portion of the event trace executed by the plant whose observed value is pending to be delivered to a destination diagnoser. Formally, the k-delaying&masking model from diagnoser i to diagnoser j $(i \neq j)$ is defined as,

$$C_{ij}^k = (Z_{ij}^k, \Sigma \cup \Lambda_i, \gamma_{ij}^k, z_0).$$

Z_{ij}^k is the set of states, which are represented by the event traces executed in the plant but their observed values not yet received by the destination diagnoser. For $z \in Z_{ij}^k$, $|z|$ denotes the length of trace z. Since the message arrivals and departures in a communication channel occur asynchronously, for a k-delaying&masking model, we have $|z| \leq k+1$. $\Sigma \cup \Lambda_i$ is the event set, where Σ is the set of input events and Λ_i is the set of output events. Without loss of generality, we assume that $\Sigma \cap \Lambda_i = \emptyset$ and $\Lambda_i \cap \Lambda_j = \emptyset$ $(i \neq j)$ (otherwise, we can simply rename some of the symbols). $z_0 = \epsilon$ is the initial state. The transition function γ_{ij}^k is defined as follows.

- "Arrival" due to an event execution in the plant: $\forall z \in Z_{ij}^k$, $\forall \sigma \in \Sigma$, if $|z| \leq k$, then $\gamma_{ij}^k(z, \sigma) = z\sigma$,
- "Departure" due to a reception at the destination diagnoser: $\forall z \in Z_{ij}^k$, $\forall \lambda_i \in \Lambda_i$, if $M_i(head(z)) = \lambda_i$, then $\gamma_{ij}^k(z, \lambda_i) = z \backslash head(z)$,
- Undefined, otherwise,

where $head(z)$ is the first event in trace z, and the after operator "\" in $z \backslash head(z)$ returns the trace after removing the initial event $head(z)$ from the trace z. The transition function γ_{ij}^k can be understood as follows. The state z is extended at the "tail" upon each plant execution $\sigma \in \Sigma$ (provided the current length does not exceed the delay bound k), while shortened at the "head" upon each reception $\lambda_i \in \Lambda_i$. It follows from the above definition that the k-delaying&masking model C_{ij}^k is a subgraph of the $(k+1)$-delaying&masking model C_{ij}^{k+1}.

Having introduced the model C_{ij}^k, we next introduce several "extended" models (extended to capture the effect of k-delaying&masking). These extended models are obtained by performing a synchronous composition between an "unextended" original model and certain k-delaying&masking models.

- *extended plant model*: $\mathcal{G}^k = G \| C_{12}^k \| C_{21}^k$.
- *extended specification model*: $\mathcal{R}^k = R \| C_{12}^k \| C_{21}^k$, which consists of traces in extended plant $\mathcal{G}^k$ whose projection over Σ do not violate the original specification R.
- *refined extended specification model*: $\overline{\mathcal{R}}^k = \overline{R} \| C_{12}^k \| C_{21}^k$, where $\overline{R}$ is the completed specification model with an additional failure state "F".
- *refined extended plant model*: $\overline{\mathcal{G}}^k = \mathcal{G}^k \| \overline{\mathcal{R}}^k = G \| \overline{R} \| C_{12}^k \| C_{21}^k$, which generates the same langauge as $\mathcal{G}^k$, but the execution of those traces that are "faulty" reaches a state with second coordinate "F".
- *extended local specification model*: $\mathcal{R}_i^k = R \| C_{ji}^k$.
- *refined extended local specification model*: $\overline{\mathcal{R}}_i^k = \overline{R} \| C_{ji}^k$.

Note that in the above construction, any event in the set Σ is synchronized among all participating components, while any event in the set $\Lambda_1 \cup \Lambda_2$ is executed asynchronously.

The following example illustrates the construction of k-delaying&masking models C_{ij}^k, extended plant model $\mathcal{G}^k$, extended specification model $\mathcal{R}^k$, and extended local specification models $\mathcal{R}_i^k$.

Example 7 A plant model G and a specification model R are shown in Figure 4.5(a) and Figure 4.5(b), respectively, with $L(G) = pr(aabc^* + baac^*)$ and $L(R) = pr(aabc^*)$. Suppose the observation masks of two local sites are defined as follows:

- $M_1(a) = a'$, $M_1(b) = M_1(c) = \epsilon$, and
- $M_2(b) = b'$, $M_2(a) = M_2(c) = \epsilon$.

For delay = 1, Figure 4.5(c) and Figure 4.5(d) show the models C_{12}^1 and C_{21}^1 respectively. The two models have the same structure and the same set of states, while some of the transitions are labeled differently. If we follow the trace aba' in the first model C_{12}^1, the states ϵ, a, ab and b are traversed sequentially. This corresponds to the situation in which site 1 sends out its observation a' to site 2 after the occurrence of ab in the plant, whereas event b is executed in the plant but its observation is not yet received at site 2.

By taking the synchronous composition between the original system/specification models and the k-delaying&masking models, we obtain the refined extended plant model $\overline{\mathcal{G}}^1 = \mathcal{G}^1 \| \overline{\mathcal{R}}^1 = G \| \overline{R} \| C_{12}^1 \| C_{21}^1$ and the extended specification model $\mathcal{R}^1 = R \| C_{12}^1 \| C_{21}^1$ as shown in Figure 4.5(e). The extended local specification models $\mathcal{R}_1^1 = R \| C_{21}^1$ and $\mathcal{R}_2^1 = R \| C_{12}^1$ are shown in Figure 4.5(f) and Figure 4.5(g), respectively.

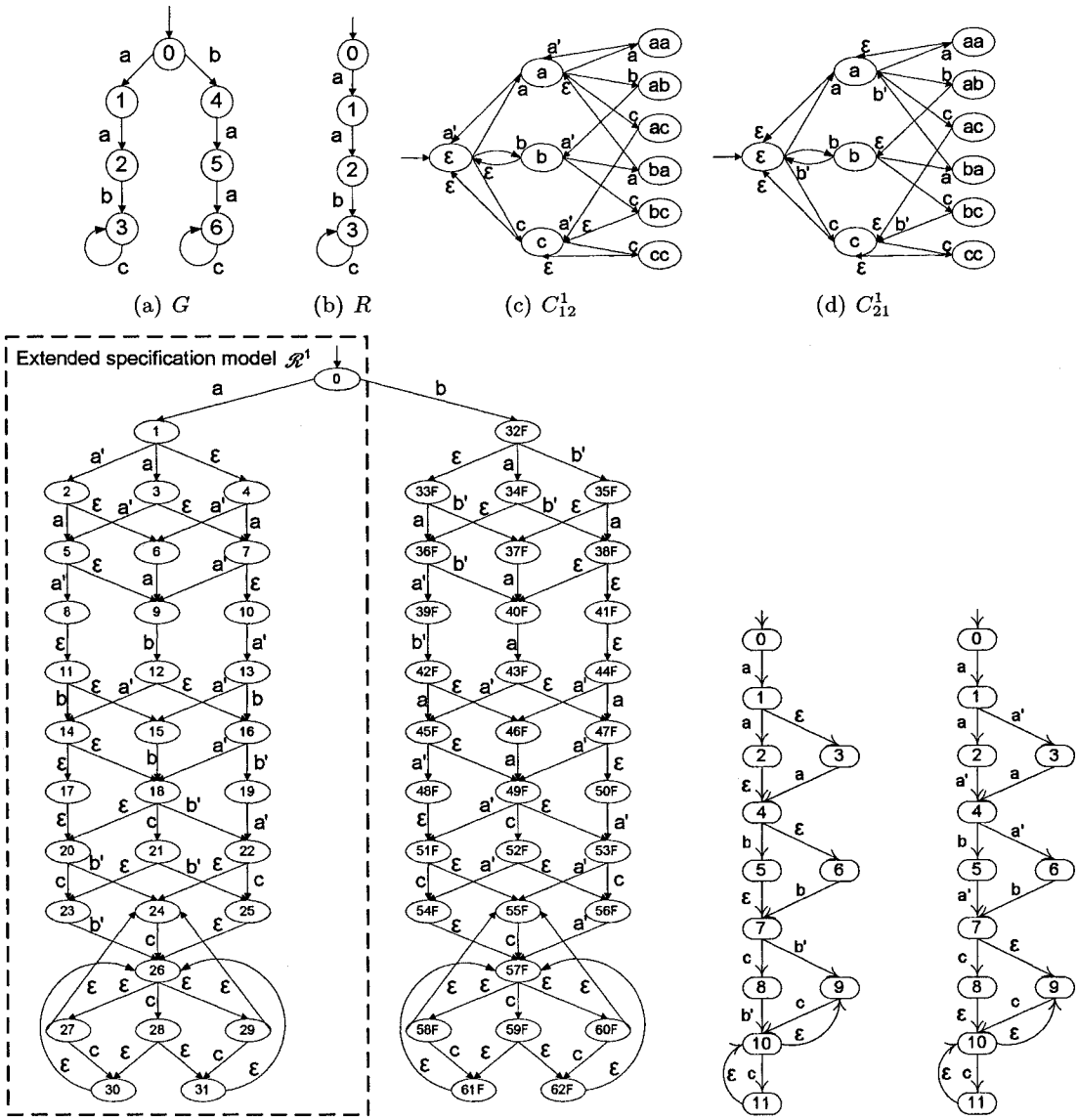
For delay = 2, the communication delay models C_{12}^2 and C_{21}^2, refined extended plant model $\overline{\mathcal{G}}^2$, extended specification model $\mathcal{R}^2$, and extended local specification models $\mathcal{R}_1^2$ and $\mathcal{R}_2^2$ are shown in Figure 4.6.

Let us define a projection $\Pi_\Sigma : \Sigma \cup \Lambda_1 \cup \Lambda_2 \to \Sigma$ as follows:

$$\forall \sigma \in \Sigma \cup \Lambda_1 \cup \Lambda_2, \Pi_\Sigma(\sigma) := \begin{cases} \sigma, & \sigma \in \Sigma; \\ \epsilon, & \sigma \in \Lambda_1 \cup \Lambda_2. \end{cases} \tag{4.7}$$

This projection can be inductively extended from event to event traces to obtain, $\Pi_\Sigma : (\Sigma \cup \Lambda_1 \cup \Lambda_2)^* \to \Sigma^*$, as follows:

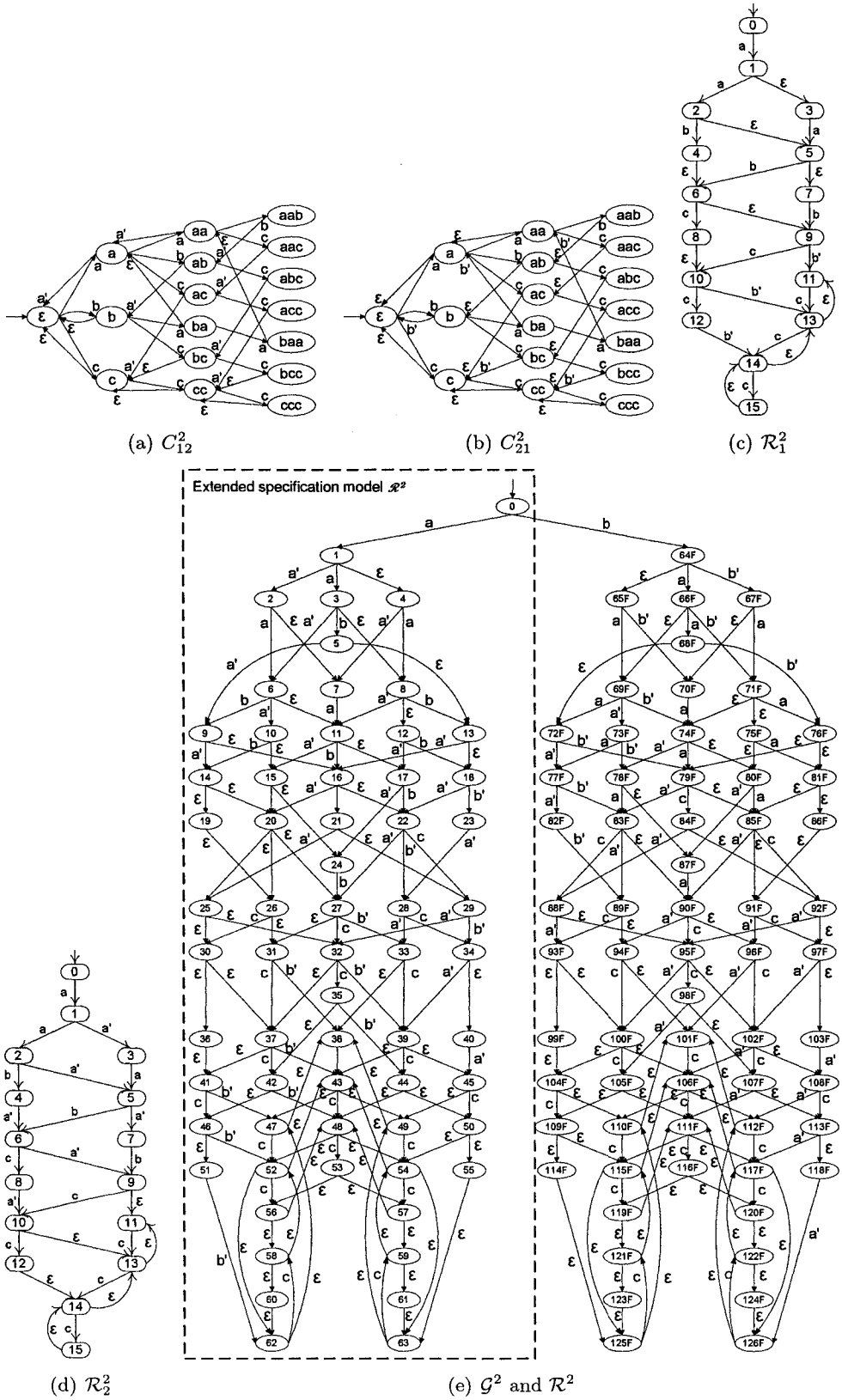
$$\Pi_\Sigma(\epsilon) = \epsilon;\ \forall s \in (\Sigma \cup \Lambda_1 \cup \Lambda_2)^*, \sigma \in \Sigma \cup \Lambda_1 \cup \Lambda_2, \Pi_\Sigma(s\sigma) = \Pi_\Sigma(s)\Pi_\Sigma(\sigma).$$



(a) G (b) R (c) C_{12}^1 (d) C_{21}^1

(e) $\mathcal{G}^1$ and $\mathcal{R}^1$ (f) $\mathcal{R}_1^1$ (g) $\mathcal{R}_2^1$

Figure 4.5 Illustrating Example 7: System models with delay $k = 1$



(a) C_{12}^2 (b) C_{21}^2 (c) $\mathcal{R}_1^2$

(d) $\mathcal{R}_2^2$ (e) $\mathcal{G}^2$ and $\mathcal{R}^2$

Figure 4.6 Illustrating Example 7: System models with delay $k = 2$

The inverse projection of Π_Σ, $\Pi_\Sigma^{-1} : \Sigma^* \rightarrow (\Sigma \cup \Lambda_1 \cup \Lambda_2)^*$, is defined as follows:

$$\forall u \in \Sigma^*, \Pi_\Sigma^{-1}(u) := \{s \in (\Sigma \cup \Lambda_1 \cup \Lambda_2)^* \mid \Pi_\Sigma(s) = u\}. \tag{4.8}$$

Lemma 5 Given a plant G and a specification model R with $L(R) \subseteq L(G)$, let $\mathcal{G}^k$ and $\mathcal{R}^k$ be the extended plant model and the extended specification model, respectively. Then,

$$\forall s \in L(\mathcal{R}^k), t \in (\Lambda_1 \cup \Lambda_2)^* : st \in L(\mathcal{G}^k) \Rightarrow st \in L(\mathcal{R}^k).$$

Proof: Since $\mathcal{G}^k = G \| C_{12}^k \| C_{21}^k$, $\mathcal{R}^k = R \| C_{12}^k \| C_{21}^k$, and $L(R) \subseteq L(G)$, it follows that $L(\mathcal{R}^k) \subseteq L(\mathcal{G}^k)$. So, $s \in L(\mathcal{G}^k)$ for any $s \in L(\mathcal{R}^k)$. Since events in Λ_i are only executed by C_{ij}^k $(i, j \in 1, 2,\ i \neq j)$, it follows that if $t \in (\Lambda_1 \cup \Lambda_2)^*$ is executable beyond s in $\mathcal{G}^k = G \| C_{12}^k \| C_{21}^k$, then t is executable beyond s in $C_{12}^k \| C_{21}^k$, and is executable beyond s in $\mathcal{R}^k = R \| C_{12}^k \| C_{21}^k$ as well, i.e., $st \in L(\mathcal{R}^k)$. ∎

Proposition 2 Given a plant G and a specification model R with $L(R) \subseteq L(G)$, let $\mathcal{G}^k$ and $\mathcal{R}^k$ be the extended plant model and the extended specification model, respectively, and Π_Σ and Π_Σ^{-1} be the projection and the inverse projection as defined in (4.7) and (4.8), respectively. Then the original system behaviors and the extended system behaviors have the following relationship:

- $\forall s \in L(\mathcal{G}^k) : \Pi_\Sigma(s) \in L(G)$,
- $\forall s \in L(\mathcal{R}^k) : \Pi_\Sigma(s) \in L(R)$, and
- $\forall s \in L(R) : \Pi_\Sigma^{-1}(s) \cap L(\mathcal{G}^k) \subseteq L(\mathcal{R}^k)$.

Proof: We prove the above assertions inductively on the length of trace s, denoted $|s|$. For the first assertion, if $|s| = 0$, i.e., $s = \epsilon$, then it is obvious that $\Pi_\Sigma(s) = \epsilon \in L(G)$. Assume that the first assertion holds for any trace $s \in L(\mathcal{G}^k)$ with $|s| = n$. Then, consider trace $s\sigma \in L(\mathcal{G}^k)$ with $|s| = n$ and $\sigma \in \Sigma \cup \Lambda_1 \cup \Lambda_2$. If $\sigma \in \Lambda_1 \cup \Lambda_2$, then $\Pi_\Sigma(s\sigma) = \Pi_\Sigma(s) \in L(G)$ from the hypothesis. On the other hand, if $\sigma \in \Sigma$, then $\Pi_\Sigma(s\sigma) = \Pi_\Sigma(s)\sigma$. Since all events in set Σ

are executed synchronously in $\mathcal{G}^k$ among G, C_{12}^k, and C_{21}^k, it follows that σ is executable in G after trace $\Pi_\Sigma(s)$, which implies that $\Pi_\Sigma(s\sigma) = \Pi_\Sigma(s)\sigma \in L(G)$. Thus, the first assertion holds. The second assertion can be proved similarly.

For the third assertion, if $|s| = 0$, i.e., $s = \epsilon$, then $\Pi_\Sigma^{-1}(s) \cap L(\mathcal{G}^k) = (\Lambda_1 \cup \Lambda_2)^* \cap L(\mathcal{G}^k) \subseteq L(\mathcal{R}^k)$, where the last inclusion follows from Lemma 5. Assume that the third assertion holds for $|s| = n$. For any trace $s\sigma \in L(R)$ with $|s| = n$ and $\sigma \in \Sigma$, we have that

$$\Pi_\Sigma^{-1}(s\sigma) \cap L(\mathcal{G}^k) = \{t_1\sigma t_2 \mid t_1 \in L(\mathcal{R}^k), \Pi_\Sigma(t_1) = s, t_2 \in (\Lambda_1 \cup \Lambda_2)^*, t_1\sigma t_2 \in L(\mathcal{G}^k)\}.$$

Since $L(\mathcal{G}^k)$ is prefix-closed, it follows that $t_1\sigma \in \mathcal{G}^k$. I.e., σ is enabled after t_1 in $\mathcal{G}^k = G\|C_{12}^k\|C_{21}^k$. Since $\mathcal{R}^k = R\|C_{12}^k\|C_{21}^k$ and σ is enabled after $\Pi_\Sigma(t_1) = s$ in R, it follows from the synchronization of σ in $\mathcal{G}^k$ and $\mathcal{R}^k$ that σ is enabled after t_1 in $\mathcal{R}^k$ as well. I.e., $t_1\sigma \in L(\mathcal{R}^k)$. Then for any $t_1\sigma t_2 \in \Pi_\Sigma^{-1}(s\sigma) \cap L(\mathcal{G}^k)$ with $t_2 \in (\Lambda_1 \cup \Lambda_2)^*$, it follows from Lemma 5 that $t_1\sigma t_2 \in L(\mathcal{R}^k)$. This completes the proof for the third assertion. ■

From Proposition 2, we can get the following corollary.

Corollary 2 Given a plant G and a specification model R with $L(R) \subseteq L(G)$, let $\mathcal{G}^k$ and $\mathcal{R}^k$ be the extended plant model and the extended specification model, respectively, and Π_Σ and Π_Σ^{-1} be the projection and the inverse projection defined in (4.7) and (4.8), respectively. Then,

- $\forall s \in L(G) - L(R) : \Pi_\Sigma^{-1}(s) \cap L(\mathcal{G}^k) \subseteq L(\mathcal{G}^k) - L(\mathcal{R}^k)$, and
- $\forall s \in L(\mathcal{G}^k) - L(\mathcal{R}^k) : \Pi_\Sigma(s) \in L(G) - L(R)$.

Proof: Suppose for the purpose of contradiction that there exist $s \in L(G) - L(R)$ and $t \in (\Sigma \cup \Lambda_1 \cup \Lambda_2)^*$ such that $t \in \Pi_\Sigma^{-1}(s) \cap L(\mathcal{G}^k)$, but $t \notin L(\mathcal{G}^k) - L(\mathcal{R}^k)$, i.e., $t \in L(\mathcal{R}^k)$. It follows from the second assertion of Proposition 2 that $\Pi_\Sigma(t) = s \in L(R)$, which contradicts the condition $s \in L(G) - L(R)$. This proves the fist assertion. The second assertion can be proved similarly from the third assertion of Proposition 2. ■

In the following proposition, we present a relationship between languages $L(\mathcal{G}^k)$ and $L(\mathcal{R}^k)$, and languages $L(\mathcal{G}^{k+1})$ and $L(\mathcal{R}^{k+1})$.

Proposition 3 Given a plant G and a specification model R with $L(R) \subseteq L(G)$, consider $\mathcal{G}^k = G \| C_{12}^k \| C_{21}^k$ and $\mathcal{R}^k = R \| C_{12}^k \| C_{21}^k$. Then, we have that

- $L(\mathcal{G}^k) \subseteq L(\mathcal{G}^{k+1})$, $L(\mathcal{R}^k) \subseteq L(\mathcal{R}^{k+1})$, $L(\mathcal{R}_i^k) \subseteq L(\mathcal{R}_i^{k+1})$, and
- $L(\mathcal{G}^k) - L(\mathcal{R}^k) \subseteq L(\mathcal{G}^{k+1}) - L(\mathcal{R}^{k+1})$.

Proof: Since C_{ij}^k is a subgraph of C_{ij}^{k+1}, all system models obtained by composing with $\{C_{ij}^k\}$ are subgraphs of corresponding system models obtained by composing with $\{C_{ij}^{k+1}\}$, implying the correctness of the first assertion.

To show the second assertion, suppose for the sake of contradiction that

$$\exists s : [s \in L(\mathcal{G}^k) - L(\mathcal{R}^k)] \wedge [s \notin L(\mathcal{G}^{k+1}) - L(\mathcal{R}^{k+1})].$$

Since $s \in L(\mathcal{G}^k) \subseteq L(\mathcal{G}^{k+1})$, it follows that if $s \notin L(\mathcal{G}^{k+1}) - L(\mathcal{R}^{k+1})$, then $s \in L(\mathcal{R}^{k+1})$, which combined with the second assertion of Proposition 2 implies that $\Pi_\Sigma(s) \in L(R)$. This leads to a contradiction that since $s \in L(\mathcal{G}^k) - L(\mathcal{R}^k)$, and so it follows from the second assertion of Corollary 2 that $\Pi_\Sigma(s) \in L(G) - L(R)$. ■

4.3.2 Joint$_k^{iop}$-Diagnosability

Next, we present a theorem to reduce the P^{iop}-based distributed diagnosis problem to an instance of the decentralized diagnosis problem.

Theorem 4 Given a plant $G = (X, \Sigma, \alpha, x_0)$ and a specification model R with $L(R) \subseteq L(G)$, define the extended plant and extended specification models $\mathcal{G}^k = G \| C_{12}^k \| C_{21}^k$ and $\mathcal{R}^k = R \| C_{12}^k \| C_{21}^k$ respectively, where C_{ij}^k $(i, j \in \{1, 2\}, i \neq j)$ is the k-delaying&masking model from site i to site j. Let $M_i : \Sigma \to \overline{\Lambda}_i$ and $\mathcal{M}_i : \Sigma \cup \Lambda_1 \cup \Lambda_2 \to \overline{\Lambda}_1 \cup \overline{\Lambda}_2$ be the observation mask and the extended observation mask for site i, respectively. Then,

$$(L(G), L(R)) \text{ joint}_k^{iop}\text{-diagnosable} \Leftrightarrow (L(\mathcal{G}^k), L(\mathcal{R}^k)) \text{ codiagnosable with respect to } \{\mathcal{M}_i\}.$$

Proof: ($\Rightarrow$) If $(L(G), L(R))$ is joint$_k^{iop}$-diagnosable, then

$$(\exists n' \in \mathcal{N})(\forall s' \in L(G) - L(R))(\forall s't' \in L(G), |t'| \geq n' \text{ or } s't' \text{ deadlocks}) \Rightarrow \tag{4.9}$$

$$(\exists i \in \{1,2\})(\forall u' \in L(G), \Upsilon_i^{iop,k}(s't', u') = 1 \Rightarrow u' \in L(G) - L(R)).$$

In order to prove the codiagnosability of $(L(\mathcal{G}^k), L(\mathcal{R}^k))$, we need to show the existence of a bound $n \in \mathcal{N}$ such that

$$(\forall s \in L(\mathcal{G}^k) - L(\mathcal{R}^k))(\forall st \in L(\mathcal{G}^k), |t| \geq n \text{ or } st \text{ deadlocks}) \Rightarrow$$

$$(\exists i \in \{1,2\})(\forall u \in L(\mathcal{G}^k), \mathcal{M}_i(st) = \mathcal{M}_i(u) \Rightarrow u \in L(\mathcal{G}^k) - L(\mathcal{R}^k)).$$

We claim that $n = 3n'$ can be chosen as such a bound, where n' is the diagnosis bound specified in (4.9).

Pick any $s \in L(\mathcal{G}^k) - L(\mathcal{R}^k)$. By Corollary 2, $\Pi_\Sigma(s) \in L(G) - L(R)$. For all $st \in L(\mathcal{G}^k)$, it follows from Proposition 2 that $\Pi_\Sigma(st) \in L(G)$. Since $\mathcal{G}^k = G \| C_{12}^k \| C_{21}^k$ is extended from G by interleaving events in Σ with their observations or communicated observations in $\overline{\Lambda}_1 \cup \overline{\Lambda}_2$, and each observation in $\overline{\Lambda}_1 \cup \overline{\Lambda}_2$ occurs after its corresponding event in Σ, if $|t| \geq n = 3n'$ or if st deadlocks, we must have that $|\Pi_\Sigma(t)| \geq n'$ or $\Pi_\Sigma(st)$ deadlocks.

Then, pick any trace $u \in L(\mathcal{G}^k)$. By Proposition 2, $\Pi_\Sigma(u) \in L(G)$. Since $\Pi_\Sigma(s) \in L(G) - L(R)$, and $\Pi_\Sigma(st) \in L(G)$ with $|\Pi_\Sigma(t)| \geq n'$ or $\Pi_\Sigma(st)$ deadlocks, it follows from the joint$_k^{iop}$-diagnosability of $(L(G), L(R))$ that

$$\exists i \in \{1,2\} : \Upsilon_i^{iop,k}(\Pi_\Sigma(st), \Pi_\Sigma(u)) = 1 \Rightarrow \Pi_\Sigma(u) \in L(G) - L(R).$$

From Proposition 1, we know that if $\mathcal{M}_i(st) = \mathcal{M}_i(u)$, then $\Upsilon_i^{iop,k}(\Pi_\Sigma(st), \Pi_\Sigma(u)) = 1$, and thus $\Pi_\Sigma(u) \in L(G) - L(R)$, which implies $u \in \Pi_\Sigma^{-1}[\Pi_\Sigma(u)] \cap L(\mathcal{G}^k) \subseteq L(\mathcal{G}^k) - L(\mathcal{R}^k)$ by Corollary 2. Thus $(L(\mathcal{G}^k), L(\mathcal{R}^k))$ is codiagnosable with respect to $\{\mathcal{M}_i\}$.

($\Leftarrow$) If $(L(\mathcal{G}^k), L(\mathcal{R}^k))$ is codiagnosable with respect to $\{\mathcal{M}_i\}$, then

$$(\exists n' \in \mathcal{N})(\forall s' \in L(\mathcal{G}^k) - L(\mathcal{R}^k))(\forall s't' \in L(\mathcal{G}^k), |t'| \geq n' \text{ or } s't' \text{ deadlocks}) \Rightarrow \tag{4.10}$$

$$(\exists i \in \{1,2\})(\forall u' \in L(\mathcal{G}^k), \mathcal{M}_i(s't') = \mathcal{M}_i(u') \Rightarrow u' \in L(\mathcal{G}^k) - L(\mathcal{R}^k)).$$

In order to prove the joint$_k^{iop}$-diagnosability of $(L(G), L(R))$, we need to show the existence of a bound n such that

$$(\forall s \in L(G) - L(R))(\forall st \in L(G), |t| \geq n \text{ or } st \text{ deadlocks}) \Rightarrow$$

$$(\exists i \in \{1,2\})(\forall u \in L(G), \Upsilon_i^{iop,k}(st, u) = 1 \Rightarrow u \in L(G) - L(R)).$$

We claim that the same bound n' specified in (4.10) works for the joint$_k^{iop}$-diagnosability as well, i.e., $n = n'$.

Pick any $s \in L(G) - L(R)$. By Corollary 2, $\Pi_\Sigma^{-1}(s) \cap L(\mathcal{G}^k) \subseteq L(\mathcal{G}^k) - L(\mathcal{R}^k)$. For all $st \in L(G)$ with $|t| \geq n$ or st deadlocks, it follows that

$$\exists s' \in \Pi_\Sigma^{-1}(s) \cap L(\mathcal{G}^k), t' \in \Pi_\Sigma^{-1}(t) \cap L(\mathcal{G}^k) : s't' \in \Pi_\Sigma^{-1}(st) \cap L(\mathcal{G}^k), |t'| \geq n \text{ or } s't' \text{ deadlocks}.$$

Consider any $u \in L(G)$. For any $u' \in \Pi_\Sigma^{-1}(u) \cap L(\mathcal{G}^k)$, it follows from the codiagnosability of $(L(\mathcal{G}^k), L(\mathcal{R}^k))$ that

$$\exists i \in \{1,2\} : \mathcal{M}_i(s't') = \mathcal{M}_i(u') \Rightarrow u' \in L(\mathcal{G}^k) - L(\mathcal{R}^k).$$

From Proposition 1, we know that if $\Upsilon_i^{iop,k}(st, u) = 1$, then there exist $s't' \in \Pi_\Sigma^{-1}(st) \cap L(\mathcal{G}^k)$ and $u' \in \Pi_\Sigma^{-1}(u) \cap L(\mathcal{G}^k)$ such that $\mathcal{M}_i(s't') = \mathcal{M}_i(u')$. This indicates that there exists $i \in \{1,2\}$ such that

$$\forall u \in L(G) : (\Upsilon_i^{iop,k}(st, u) = 1) \Rightarrow (\exists u' \in \Pi_\Sigma^{-1}(u) \cap L(\mathcal{G}^k), u' \in L(\mathcal{G}^k) - L(\mathcal{R}^k)),$$

where $u' \in L(\mathcal{G}^k) - L(\mathcal{R}^k)$ implies that $u = \Pi_\Sigma(u') \in L(G) - L(R)$ by Corollary 2. This completes the proof. ■

An implication of Theorem 4 is that the methods presented in Chapter 3 for studying decentralized diagnosis can be applied to study P^{iop}-based distributed diagnosis. Before discussing the algorithm to verify joint$_k^{iop}$-diagnosability, we present the following proposition relating the extended global specification and the extended local specification models, which is used to reduce the complexity of the verification algorithm.

Proposition 4 Given a specification model R, define $\mathcal{R}^k := R\|C_{12}^k\|C_{21}^k$ and $\mathcal{R}_i^k := R\|C_{ji}^k$ $(i,j \in \{1,2\}, i \neq j)$, the extended specification and the extended local specification models, respectively, where C_{ij}^k is the k-delaying&masking model from site i to site j. Let $\mathcal{M}_i : \Sigma \cup \Lambda_1 \cup \Lambda_2 \to \overline{\Lambda}_1 \cup \overline{\Lambda}_2$ be the extended observation mask for site i. Then, we have that

$$\mathcal{M}_i(L(\mathcal{R}^k)) = \mathcal{M}_i(L(\mathcal{R}_i^k)).$$

Proof: $(\subseteq)$ It suffices to show that for any trace $s \in L(\mathcal{R}^k)$, there exists a trace $s' \in L(\mathcal{R}_i^k)$ such that $\mathcal{M}_i(s) = \mathcal{M}_i(s')$. Pick any trace $s = a_1 \cdots a_l \in L(\mathcal{R}^k) = L(\mathcal{R}^k)$. Since $\Lambda_1 \cap \Lambda_2 = \emptyset$, we can define a trace $s' = a_1' \cdots a_l'$ with

$$\forall q \in \{1, \cdots, l\}, a_q' := \begin{cases} a_q & \text{if } a_q \in \Sigma \cup \Lambda_j; \\ \epsilon & \text{if } a_q \in \Lambda_i. \end{cases}$$

It follows from $\mathcal{M}_i(\Lambda_i) = \{\epsilon\}$ that $\mathcal{M}_i(s) = \mathcal{M}_i(s')$. Since in $\mathcal{R}^k$ and $\mathcal{R}_i^k$, all events in Σ are executed synchronously, and all events in $\Lambda_1 \cup \Lambda_2$ are executed asynchronously, and since $\Pi_\Sigma(C_{ji}^k) = \Sigma^*$, it follows that $s' \in L(\mathcal{R}_i^k)$. Thus, $\mathcal{M}_i(L(\mathcal{R}^k)) \subseteq \mathcal{M}_i(L(\mathcal{R}_i^k))$.

$(\supseteq)$ It suffices to show that for any trace $s \in L(\mathcal{R}_i^k) = R\|C_{ji}^k$, there exists a trace $s' \in L(\mathcal{R}^k)$ such that $\mathcal{M}_i(s) = \mathcal{M}_i(s')$. Pick any trace $s = a_1 \cdots a_l \in L(\mathcal{R}_i^k)$. Let $C_{ji}^k = (Z_{ji}^k, \Sigma \cup \Lambda_j, \gamma_{ji}^k, z_0)$ $(i,j \in \{1,2\}, i \neq j)$. Since C_{12}^k and C_{21}^k share the same structure and the same state space $(Z_{21}^k = Z_{12}^k)$, and only transitions labeled by local observation symbols from $\overline{\Lambda}_1 \cup \overline{\Lambda}_2$ are

different, we can define $s' = a'_1 \cdots a'_l$ with

$$\forall q \in \{1, \cdots, l\}, a'_q := \begin{cases} a_q & \text{if } a_q \in \Sigma; \\ a_q b_q & \text{if } a_q \in \overline{\Lambda}_j, \end{cases}$$

where $b_q \in \overline{\Lambda}_i$ and $\gamma^k_{ji}(z, a_q) = \gamma^k_{ij}(z, b_q)$ for $z \in Z^k_{21} = Z^k_{12}$. Since in $\mathcal{R}^k$ and $\mathcal{R}^k_i$, all events in Σ are executed synchronously, and all events in $\Lambda_1 \cup \Lambda_2$ are executed asynchronously, and since $\Lambda_1 \cap \Lambda_2 = \emptyset$, it follows that $s' \in L(\mathcal{R}^k)$. Since $\mathcal{M}_i(b_q) = \epsilon$ for $b_q \in \overline{\Lambda}_i$, we have that $\mathcal{M}_i(s) = \mathcal{M}_i(s')$. Thus, $\mathcal{M}_i(L(\mathcal{R}^k)) \supseteq \mathcal{M}_i(L(\mathcal{R}^k_i))$. ■

An algorithm for verifying the codiagnosability was given in Algorithm 1 of Chapter 3, where a testing automaton $T = (G \| \overline{R}) \times R \times R$ was constructed to track a triplet of traces s, u_1 and u_2 with the following property:

$$\forall i \in \{1, 2\}, M_i(s) = M_i(u_i), s \in L(G) \cap L(\overline{R}) = L(G), u_i \in L(R),$$

and without loss of generality G was plant to be deadlock-free. A similar testing automaton is defined in the following verification algorithm for testing joint$^{iop}_k$-diagnosability, assuming again without loss of generality that G is deadlock-free.

Note that the application of codiagnosability test in Algorithm 1 of Chapter 3 would result in the construction of the testing automaton $\overline{\mathcal{G}}^k \times \mathcal{R}^k \times \mathcal{R}^k$ that tracks a trace-triple satisfying

$$\forall i \in \{1, 2\}, \mathcal{M}_i(s) = \mathcal{M}_i(u_i), s \in L(\overline{\mathcal{G}}^k) = L(\mathcal{G}), u_i \in L(\mathcal{R}^k) \tag{4.11}$$

Since $\mathcal{M}_i(L(\mathcal{R}^k)) = \mathcal{M}_i(L(\mathcal{R}^k_i))$ as shown in Proposition 4, it suffices to construct the testing automaton $\mathcal{T}^k = \overline{\mathcal{G}}^k \times \mathcal{R}^k_1 \times \mathcal{R}^k_2$. The advantage is that $\mathcal{T}^k$ has a smaller state space than $\overline{\mathcal{G}}^k \times \mathcal{R}^k \times \mathcal{R}^k$.

Algorithm 4 Given a (deadlock-free) plant $G = (X, \Sigma, \alpha, x_0)$ and a specification model $R = (Y, \Sigma, \beta, y_0)$, consider a distributed diagnosis systems with two local sites, which communicate with each other using the immediate observation passing protocol P^{iop}. Perform the following

operations:

1. Construct the k-delaying&masking models C_{12}^k and C_{21}^k;

2. Construct the extended plant model $\mathcal{G}^k = G\|C_{21}^k\|C_{21}^k$, the refined extended plant model $\overline{\mathcal{G}}^k = G\|\overline{R}\|C_{12}^k\|C_{21}^k$, the extended specification model $\mathcal{R}^k = R\|C_{12}^k\|C_{21}^k$, and the extended local specification model $\mathcal{R}_i^k = R\|C_{ij}^k$ $(i,j \in \{1,2\}, i \neq j)$;

3. Construct a testing automaton $\mathcal{T}^k$ for checking joint$_k^{iop}$-diagnosability:

$$\begin{aligned}\mathcal{T}^k &= (\mathcal{G}\|\overline{\mathcal{R}}^k) \times \mathcal{R}_1^k \times \mathcal{R}_2^k \\ &= \overline{\mathcal{G}}^k \times \mathcal{R}_1^k \times \mathcal{R}_2^k \\ &= (G\|\overline{R}\|C_{12}^k\|C_{21}^k) \times (R\|C_{21}^k) \times (R\|C_{12}^k).\end{aligned}$$

   Note that $(\epsilon,\epsilon,\epsilon)$-transition is allowed in the testing automaton if it is not performed as a self loop. The testing automaton $\mathcal{T}^k$ tracks all triplet of traces $s, u_1, u_2 \in (\Sigma \cup \Lambda_1 \cup \Lambda_2)^*$ satisfying the following property:

$$\forall i \in \{1,2\}, \mathcal{M}_i(s) = \mathcal{M}_i(u_i), s \in L(\overline{\mathcal{G}}^k) = L(\mathcal{G}), u_i \in L(\mathcal{R}_i^k).$$

4. Check the existence of any "offending" cycle in $\mathcal{T}^k$: The system is not joint$_k^{iop}$-diagnosable if and only if any state in a cycle contains the label "F".

Theorem 5 Algorithm 4 is correct. I.e., (L,K) is joint$_k^{iop}$-diagnosable if and only if the testing automaton $\mathcal{T}^k$ in Algorithm 4 does not contain cycles with states labeled by "F".

Proof: By Theorem 4, (L,K) is joint$_k^{iop}$-diagnosable if and only if $(L(\mathcal{G}^k), L(\mathcal{R}^k))$ is codiagnosable with respect to $\{\mathcal{M}_i\}$. By Theorem 1 of Chapter 3, codiagnosability of $(L(\mathcal{G}^k), L(\mathcal{R}^k))$ with respect to $\{\mathcal{M}_i\}$ can be checked by checking presence of cycles containing states labeled "F" in the testing automaton $\overline{\mathcal{G}}^k \times \mathcal{R}^k \times \mathcal{R}^k$. Since $\overline{\mathcal{G}}^k \times \mathcal{R}^k \times \mathcal{R}^k$ tracks triplets of traces satisfying 4.11, and since $\mathcal{M}_i(L(\mathcal{R}^k)) = \mathcal{M}_i(L(\mathcal{R}_i^k))$ from Proposition 4, $\mathcal{T}^k = \overline{\mathcal{G}}^k \times \mathcal{R}_1^k \times \mathcal{R}_2^k$

also tracks the same triplets of traces. So the correctness of Algorithm 4 follows from the correctness of codiagnosability test given in Theorem 1 and Proposition 4. ■

To illustrate Algorithm 4, we present the following example to verify joint$_k^{iop}$-diagnosability of system discussed in Example 7.

Example 8 Consider the system introduced in Example 7. For the delay bound $k = 1$, the extended plant/specification models are shown in Figure 4.5. We construct the testing automaton as shown in Figure 4.7(a). Note at the initial state of $\mathcal{G}$, the transition on a leads the state machine to a "good" region satisfying the specification, where no state is labeled by "F", and there is no possibility of leaving that region. For the failure diagnosis purpose, we do not need to track traces in that "good" region. Thus we omit the corresponding part from the testing automation $\mathcal{T}^1$. It is seen that no "offending" cycle exist in $\mathcal{T}^1$. Thus, the system is joint$_1^{iop}$-diagnosable.

For the same system, if the delay bound is $k = 2$, then it can be verified that the system is not joint$_2^{iop}$-diagnosable. The extended plant/specification models are shown in Figure 4.6. We can construct the corresponding testing automaton $\mathcal{T}^2 = \overline{\mathcal{G}}^2 \times \mathcal{R}_1^2 \times \mathcal{R}_2^2$, part of which is shown in Figure 4.7(b), where $\underline{\epsilon}$ is used to denote a non self-loop ϵ transition appearing in $\overline{\mathcal{G}}^2$ or $\mathcal{R}^2$. It is seen that there is an "ambiguous" cycle formed between the states $(82F, 5, 11)$ and $(82F, 5, 13)$, indicating that the system is not joint$_2^{iop}$-diagnosable. Thus a larger delay results in a loss of diagnosability, which is not unexpected. Since the system is not joint$_2^{iop}$-diagnosable, it can be further concluded that the system is not codiagnosable (recall codiagnosability requires no communication at all.)

4.3.3 Diagnoser Synthesis

In the setting of decentralized diagnosis, a local diagnoser at site i was taken to be $D_i = M_i(G\|\overline{R})$. Analogously, we can define an extended local diagnoser for site i to be $\mathcal{D}_i^k := \mathcal{M}_i(\mathcal{G}^k\|\overline{\mathcal{R}}^k) = \mathcal{M}_i(G\|\overline{\mathcal{R}}^k)$ for the diagnosis of a joint$_k^{iop}$-diagnosable system. As is the case with the verification, certain complexity reduction is also possible for the synthesis of local diagnosers. We define a "reduced" diagnoser at site i to be $\widetilde{\mathcal{D}}_i^k := \mathcal{M}_i(G\|\overline{\mathcal{R}}_i^k)$. We show in

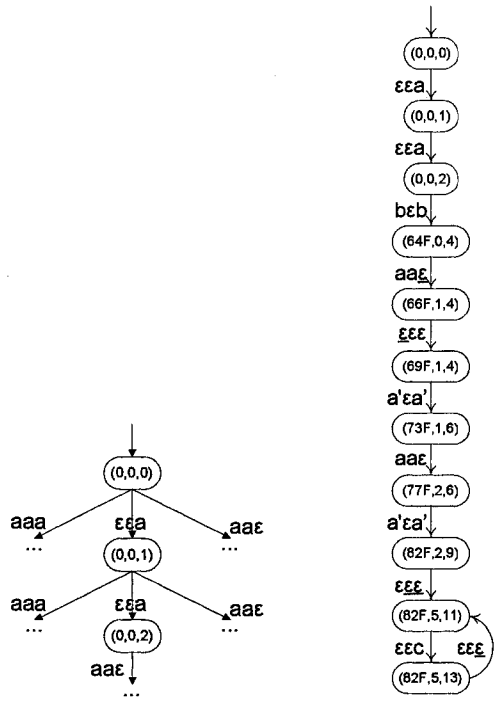


Figure 4.7 Testing automata in Example 8

the following that the two diagnosers produce the same diagnosis result. I.e., diagnoser $\widetilde{\mathcal{D}}_i^k$ detects a failure if and only if diagnoser $\mathcal{D}_i^k$ detects a failure. The following lemma shows that diagnosers $\mathcal{D}_i^k$ and $\widetilde{\mathcal{D}}_i^k$ generate the same language.

Lemma 6 Given a plant G and a specification model R with $L(R) \subseteq L(G)$, let $\mathcal{G}^k$, $\mathcal{R}^k$, and $\mathcal{R}_i^k$ ($i \in \{1,2\}$) be the extended plant model, extended specification model, and extended local specification model for site i, respectively. Consider the local diagnosers for site i, $\mathcal{D}_i^k := \mathcal{M}_i(\mathcal{G}^k \| \overline{\mathcal{R}}^k) = \mathcal{M}_i(G \| \overline{\mathcal{R}}^k)$ and $\widetilde{\mathcal{D}}_i^k := \mathcal{M}_i(G \| \overline{\mathcal{R}}_i^k)$, where $\mathcal{M}_i : \Sigma \cup \Lambda_1 \cup \Lambda_2 \to \overline{\Lambda}_1 \cup \overline{\Lambda}_2$ is the extended observation mask for site i. Then, $\mathcal{D}_i^k$ and $\widetilde{\mathcal{D}}_i^k$ generate the same language, i.e., $L(\mathcal{D}_i^k) = L(\widetilde{\mathcal{D}}_i^k)$.

Proof: Since $L(\mathcal{D}_i^k) = L(\mathcal{M}_i(G \| \overline{\mathcal{R}}^k)) = \mathcal{M}_i(L(G \| \overline{\mathcal{R}}^k))$ and $L(\widetilde{\mathcal{D}}_i^k) = L(\mathcal{M}_i(G \| \overline{\mathcal{R}}_i^k)) = \mathcal{M}_i(L(G \| \overline{\mathcal{R}}_i^k))$, we need to show that $\mathcal{M}_i(L(G \| \mathcal{R}^k)) = \mathcal{M}_i(L(G \| \mathcal{R}_i^k))$, which can be proved by showing $\mathcal{M}_i(L(G \| \mathcal{R}^k)) \subseteq \mathcal{M}_i(L(G \| \mathcal{R}_i^k))$ and $\mathcal{M}_i(L(G \| \mathcal{R}^k)) \supseteq \mathcal{M}_i(L(G \| \mathcal{R}_i^k))$ as in Proposition 4. ∎

The *reachability set* $Reach_{\widetilde{\mathcal{D}}_i^k}(\cdot)$ (resp., $Reach_{\mathcal{D}_i^k}(\cdot)$) denotes the set of possible states of $\widetilde{\mathcal{D}}_i^k$ (resp., $\mathcal{D}_i^k$) reached by an execution of a trace in $L(\widetilde{\mathcal{D}}_i^k)$ (resp., $L(\mathcal{D}_i^k)$). Let $x^0_{\widetilde{\mathcal{D}}_i^k}$ denote the initial state of diagnoser $\widetilde{\mathcal{D}}_i^k$, and $\delta_{\widetilde{\mathcal{D}}_i^k}$ denote its transition function. The reachability set $Reach_{\widetilde{\mathcal{D}}_i^k}(\cdot)$ is computed recursively upon each incoming information as follows.

- $Reach_{\widetilde{\mathcal{D}}_i^k}(\varepsilon) = \varepsilon^*_{\widetilde{\mathcal{D}}_i^k}(x^0_{\widetilde{\mathcal{D}}_i^k})$;
- $\forall s \in L(\widetilde{\mathcal{D}}_i^k), \sigma \in \Lambda_1 \cup \Lambda_2 : Reach_{\widetilde{\mathcal{D}}_i^k}(s\sigma) = \varepsilon^*_{\widetilde{\mathcal{D}}_i^k}(\delta_{\widetilde{\mathcal{D}}_i^k}(Reach_{\widetilde{\mathcal{D}}_i^k}(s), \sigma))$.

The reachability set $Reach_{\mathcal{D}_i^k}(\cdot)$ is computed similarly. Let $(\Omega_F)_{\widetilde{\mathcal{D}}_i^k}$ denote the set of "failure states" of $\widetilde{\mathcal{D}}_i^k$, i.e., $(\Omega_F)_{\widetilde{\mathcal{D}}_i^k} = X \times \{F\} \times Z^k_{ji}$. Similarly, $(\Omega_F)_{\mathcal{D}_i^k} = X \times \{F\} \times Z^k_{12} \times Z^k_{21}$ denotes the failure states of diagnoser $\mathcal{D}_i^k$. The equivalence of diagnosers $\widetilde{\mathcal{D}}_i^k$ and $\mathcal{D}_i^k$ in the sense of their diagnosis capabilities is presented in the following theorem.

Theorem 6 Given a plant G and a specification model R with $L(R) \subseteq L(G)$, let $\mathcal{G}^k$, $\mathcal{R}^k$, and $\mathcal{R}_i^k$ ($i \in \{1,2\}$) be the extended plant model, extended specification model, and extended local specification model for site i, respectively. Consider the local diagnosers for site i, $\mathcal{D}_i^k := \mathcal{M}_i(\mathcal{G}^k \| \overline{\mathcal{R}}^k) = \mathcal{M}_i(G \| \overline{\mathcal{R}}^k)$ and $\widetilde{\mathcal{D}}_i^k := \mathcal{M}_i(G \| \overline{\mathcal{R}}_i^k)$, where $\mathcal{M}_i : \Sigma \cup \Lambda_1 \cup \Lambda_2 \to \overline{\Lambda}_1 \cup \overline{\Lambda}_2$ is the extended observation mask for site i. Let $(\Omega_F)_{\mathcal{D}_i^k}$ and $(\Omega_F)_{\widetilde{\mathcal{D}}_i^k}$ denote the set of failure states in $\mathcal{D}_i^k$ and $\widetilde{\mathcal{D}}_i^k$, respectively. Then,

$$\forall s \in L(\mathcal{D}_i^k) = L(\widetilde{\mathcal{D}}_i^k) : Reach_{\mathcal{D}_i^k}(s) \subseteq (\Omega_F)_{\mathcal{D}_i^k} \Leftrightarrow Reach_{\widetilde{\mathcal{D}}_i^k}(s) \subseteq (\Omega_F)_{\widetilde{\mathcal{D}}_i^k}.$$

Proof: Pick any trace $s \in L(\mathcal{D}_i^k) = L(\widetilde{\mathcal{D}}_i^k)$. To prove the above theorem, we need to show that

$$\forall x \in Reach_{\mathcal{D}_i^k}(s) : x \in (\Omega_F)_{\mathcal{D}_i^k} \Rightarrow \forall x' \in Reach_{\widetilde{\mathcal{D}}_i^k}(s) : x' \in (\Omega_F)_{\widetilde{\mathcal{D}}_i^k}, \tag{4.12}$$

and

$$\forall x' \in Reach_{\widetilde{\mathcal{D}}_i^k}(s) : x' \in (\Omega_F)_{\widetilde{\mathcal{D}}_i^k} \Rightarrow \forall x \in Reach_{\mathcal{D}_i^k}(s) : x \in (\Omega_F)_{\mathcal{D}_i^k}. \tag{4.13}$$

Since whether a state in $\mathcal{D}_i^k = \mathcal{M}_i(G \| \overline{\mathcal{R}}^k)$ or $\widetilde{\mathcal{D}}_i^k = \mathcal{M}_i(G \| \overline{\mathcal{R}}_i^k)$ possesses a failure label or not only depends on the event trace in Σ^*, to prove (4.12) and (4.13), it is enough to show

that

$$\Pi_\Sigma(\mathcal{M}_i^{-1}(s) \cap L(G\|\overline{\mathcal{R}}^k)) = \Pi_\Sigma(\mathcal{M}_i^{-1}(s) \cap L(G\|\overline{\mathcal{R}}_i^k)).$$

We first show that $\Pi_\Sigma(L(G\|\overline{\mathcal{R}}^k)) = \Pi_\Sigma(L(G\|\overline{\mathcal{R}}_i^k))$. Since $G\|\overline{\mathcal{R}}^k = (G\|\overline{\mathcal{R}}_i^k)\|C_{ij}^k$ and events in Σ are synchronized among $G\|\overline{\mathcal{R}}_i^k$ and C_{ij}^k, it follows that $\Pi_\Sigma(L(G\|\overline{\mathcal{R}}^k)) \subseteq \Pi_\Sigma(L(G\|\overline{\mathcal{R}}_i^k))$. Similarly as in Proposition 4, it can be shown that any possible transition in $G\|\overline{\mathcal{R}}_i^k$ would not be blocked by the additional component C_{ij}^k in $G\|\overline{\mathcal{R}}^k$. Therefore, $\Pi_\Sigma(L(G\|\overline{\mathcal{R}}^k)) \supseteq \Pi_\Sigma(L(G\|\overline{\mathcal{R}}_i^k))$. From $\Pi_\Sigma(L(G\|\overline{\mathcal{R}}^k)) = \Pi_\Sigma(L(G\|\overline{\mathcal{R}}_i^k))$, it is easy to get that $\Pi_\Sigma(\mathcal{M}_i^{-1}(s) \cap L(G\|\overline{\mathcal{R}}^k)) = \Pi_\Sigma(\mathcal{M}_i^{-1}(s) \cap L(G\|\overline{\mathcal{R}}_i^k))$. ■

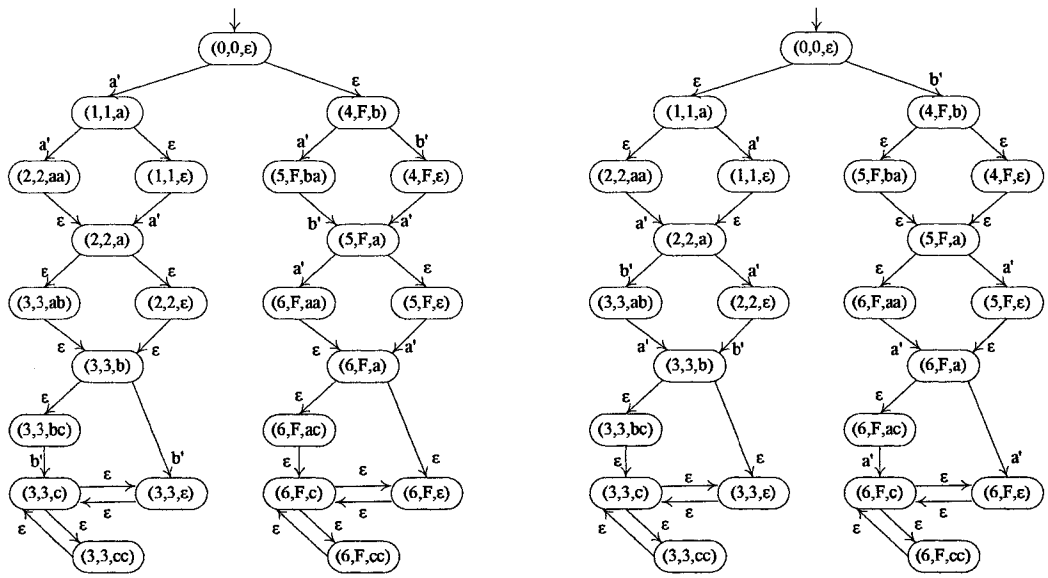
An implication of Theorem 6 is that we can use local diagnoser for site i to be $\widetilde{\mathcal{D}}_i^k = \mathcal{M}_i(G\|\overline{\mathcal{R}}_i^k) = \mathcal{M}_i(G\|\overline{R}\|C_{ji}^k)$ for performing on-line diagnosis for a joint$_k^{iop}$-diagnosable system. Diagnoser $\widetilde{\mathcal{D}}_i^k$ computes the reachability set $Reach_{\widetilde{\mathcal{D}}_i^k}$ each time it observes a new event in the plant or receives a communicated observation from another diagnoser $\widetilde{\mathcal{D}}_j^k$ $(j \neq i)$. Whenever all states in $Reach_{\widetilde{\mathcal{D}}_i^k}$ contain the label "F", diagnoser $\widetilde{\mathcal{D}}_i^k$ reports that a failure is detected.

The following example illustrates how to construct local diagnoser $\widetilde{\mathcal{D}}_i^k$, and how to perform on-line diagnosis using the reachability set $Reach_{\widetilde{\mathcal{D}}_i^k}$.

Example 9 Let us revisit the system presented in Example 7. For the unit-delay case, we know from Example 8 that the system is joint$_1^{iop}$-diagnosable. Figure 4.8 shows the two local diagnosers $\widetilde{\mathcal{D}}_1^1$ and $\widetilde{\mathcal{D}}_2^1$. Each state in diagnoser $\widetilde{\mathcal{D}}_i^1 = \mathcal{M}_i(G\|\overline{\mathcal{R}}_i^1) = \mathcal{M}_i(G\|\overline{R}\|C_{ji}^1)$ $(i, j \in \{1, 2\}$, $i \neq j)$ contains three coordinates, which correspond to states in G, $\overline{R}$, and C_{ji}^1, respectively.

Assume that the plant executes a trace $s = ba$. Diagnoser 1 observes event a' followed by a communicated observation b' from diagnoser 2 with unit-delay. Diagnoser 2 observes event b' followed by a communicated observation a' from diagnoser 1 with 0 or 1 delay. The reachability sets $Reach_{\widetilde{\mathcal{D}}_1^1}$ and $Reach_{\widetilde{\mathcal{D}}_2^1}$ are computed as follows:

- $Reach_{\widetilde{\mathcal{D}}_1^1}(\epsilon) = \{(0, 0, \epsilon), (4, F, b)\}$,
  $Reach_{\widetilde{\mathcal{D}}_1^1}(a') = \{(1, 1, a), (1, 1, \epsilon), (5, F, ba)\}$,
  $Reach_{\widetilde{\mathcal{D}}_1^1}(a'b') = \{(5, F, a), (5, F, \epsilon)\}$;
- $Reach_{\widetilde{\mathcal{D}}_2^1}(\epsilon) = \{(0, 0, \epsilon), (1, 1, a), (2, 2, aa)\}$,



(a) $\widetilde{\mathcal{D}}_1^1$ (b) $\widetilde{\mathcal{D}}_2^1$

Figure 4.8 Local diagnosers in Example 9

$$Reach_{\widetilde{\mathcal{D}}_2^1}(b') = \{(4, F, b), (5, F, ba), (4, F, \epsilon), (5, F, a), (6, F, aa)\},$$
$$Reach_{\widetilde{\mathcal{D}}_2^1}(b'a') = \{(5, F, \epsilon), (6, F, a), (6, F, ac)\}.$$

When diagnoser 1 observes event a', it cannot determine whether the plant is at a "normal" state 1 or the "failure" state 5, and thus is ambiguous about whether a failure has occurred or not. After receiving the communicated event b' from diagnoser 2, diagnoser 1 is sure that the plant is at state 5, and a failure has occurred. On the other hand, since all elements in $Reach_{\widetilde{\mathcal{D}}_2^1}(b')$ have their second coordinate labeled "F", diagnoser 2 is certain about the occurrence of a failure after observing event b'.

Remark 14 Let $|X|$ and $|Y|$ be the number of states in plant G and specification model R, respectively, and $|\Sigma|$ be the number of events. Assume that there are m local sites ($I = \{1, \cdots, m\}$), each of them associated with a local observation mask $M_i : \Sigma \to \overline{\Lambda}_i$. We analyze the number of states and transitions in various automata as follows, and the results are summarized in Table 4.1.

- C_{ij}^k: The number of states of a k-delaying&masking model C_{ij}^k $(i, j \in I, i \neq j)$ is at most $1 + |\Sigma| + |\Sigma|^2 + \cdots + |\Sigma|^{k+1} = \mathcal{O}(|\Sigma|^{k+2})$. Since there are at most $1 + |\Sigma|$ possible

transitions at each state, the number of transitions in C_{ij}^k is $\mathcal{O}(|\Sigma|^{k+3})$.

- $\mathcal{G}^k$: Since there are $m(m-1)$ k-delaying&masking models C_{ij}^k involved in constructing $\mathcal{G}^k = G\|_{i,j\in I, i\neq j} C_{ij}^k$, the number of states in $\mathcal{G}^k$ is $\mathcal{O}(|X| \times |\Sigma|^{m(m-1)(k+2)})$. From the synchronization of events in $|\Sigma|$ among G and $\{C_{ij}^k\}$, and the non-synchronization of events in $\bigcup_{i\in I} \Lambda_i$ among $\{C_{ij}^k\}$, we know that there are at most $|\Sigma|+m(m-1)$ transitions at each state of $\mathcal{G}^k$. Thus, the number of transitions in $\mathcal{G}^k$ is $\mathcal{O}((|\Sigma| + m(m-1)) \times |X| \times |\Sigma|^{m(m-1)(k+2)})$.

- $\mathcal{R}_i^k$: Since there are $(m-1)$ k-delaying&masking models $\{C_{ij}^k\}$ involved in constructing $\mathcal{R}_i^k = R\|_{j\in I, j\neq i} C_{ji}^k$, the number of states and transitions in $\mathcal{R}_i^k$ are $\mathcal{O}(|Y| \times |\Sigma|^{(m-1)(k+2)})$ and $\mathcal{O}((|\Sigma| + (m-1)) \times |Y| \times |\Sigma|^{(m-1)(k+2)})$, respectively.

- $\mathcal{T}^k$: Since there are m local specification models $\mathcal{R}_i^k$ $(i \in I)$ involved in constructing the testing automaton $\mathcal{T}^k = \overline{\mathcal{G}}^k \times_{i\in I} \mathcal{R}_i^k$, the number of states and transitions are $\mathcal{O}(|X| \times |Y|^m \times |\Sigma|^{2m(m-1)(k+2)})$, and $\mathcal{O}((|\Sigma| + m(m-1)) \times (|\Sigma| + (m-1))^m \times |X| \times |Y|^m \times |\Sigma|^{2m(m-1)(k+2)})$, respectively.

- $\widetilde{\mathcal{D}}_i^k$: For the local diagnoser $\widetilde{\mathcal{D}}_i^k = \mathcal{M}_i(G\|\mathcal{R}_i^k)$, the number of states and transitions are $\mathcal{O}(|X| \times |Y| \times |\Sigma|^{(m^2-1)(k+2)})$, and $\mathcal{O}((|\Sigma| + m(m-1)) \times (|\Sigma| + (m-1)) \times |X| \times |Y| \times |\Sigma|^{(m^2-1)(k+2)})$, respectively.

Table 4.1 Summary of complexity analysis in Remark 14

	number of states	number of transitions
C_{ij}^k	$\mathcal{O}(\|\Sigma\|^{k+2})$	$\mathcal{O}(\|\Sigma\|^{k+3})$
$\mathcal{G}^k$	$\mathcal{O}(\|X\| \times \|\Sigma\|^{m(m-1)(k+2)})$	$\mathcal{O}((\|\Sigma\| + m(m-1)) \times \|X\| \times \|\Sigma\|^{m(m-1)(k+2)})$
$\mathcal{R}_i^k$	$\mathcal{O}(\|Y\| \times \|\Sigma\|^{(m-1)(k+2)})$	$\mathcal{O}((\|\Sigma\| + (m-1)) \times \|Y\| \times \|\Sigma\|^{(m-1)(k+2)})$
$\mathcal{T}^k$	$\mathcal{O}(\|X\| \times \|Y\|^m \times \|\Sigma\|^{2m(m-1)(k+2)})$	$\mathcal{O}((\|\Sigma\|+m(m-1)) \times (\|\Sigma\|+(m-1))^m \times \|X\| \times \|Y\|^m \times \|\Sigma\|^{2m(m-1)(k+2)})$
$\widetilde{\mathcal{D}}_i^k$	$\mathcal{O}(\|X\| \times \|Y\| \times \|\Sigma\|^{(m^2-1)(k+2)})$	$\mathcal{O}((\|\Sigma\|+m(m-1)) \times (\|\Sigma\|+(m-1)) \times \|X\| \times \|Y\| \times \|\Sigma\|^{(m^2-1)(k+2)})$

From the above analysis, it can be concluded that the complexity of Algorithm 4 for verifying joint$_k^{iop}$-diagnosability is $\mathcal{O}((|\Sigma| + m(m-1)) \times (|\Sigma| + (m-1))^m \times |X| \times |Y|^m \times |\Sigma|^{2m(m-1)(k+2)})$, and the complexity of synthesizing local diagnosers is $\mathcal{O}((|\Sigma| + m(m-1)) \times (|\Sigma| + (m-1)) \times |X| \times |Y| \times |\Sigma|^{(m^2-1)(k+2)})$. It is clear that complexity is polynomial in the number of plant and specification states, but grows exponentially in delay bound and the number of local diagnosers.

4.3.4 Hierarchy of Various Diagnosabilities

In the following, we compare various notions of distributed/decentralized diagnosabilities. In particular, we establish the following hierarchy:

$$\begin{aligned}
\text{Codiagnosable} &\Leftrightarrow \text{Joint}_{\infty}^{iop}\text{-diagnosable} \\
&\overset{\Rightarrow}{\not\Leftarrow} \text{Joint}_{k+1}^{iop}\text{-diagnosable} \\
&\overset{\Rightarrow}{\not\Leftarrow} \text{Joint}_{k}^{iop}\text{-diagnosable} \\
&\overset{\Rightarrow}{\not\Leftarrow} \text{Joint}_{0}^{iop}\text{-diagnosable} \\
&\Leftrightarrow \text{Diagnosable,}
\end{aligned}$$

where joint$_\infty^{iop}$-diagnosability denotes the P^{iop}-based joint-diagnosability under unbounded-delay communication, and joint$_0^{iop}$-diagnosability denotes the P^{iop}-based joint-diagnosability under zero-delay communication.

We will consider the relationship of equivalence between codiagnosability and joint$_\infty^{iop}$-diagnosability in the next section, where we will show that "codiagnosability $\Leftrightarrow$ joint$_\infty^{gen}$-diagnosability" holds for a general protocol $P^{gen} \in \mathcal{P}^{gen}$. It follows that this relationship also holds for protocol P^{iop}.

The following proposition establishes the equivalence between joint$_0^{iop}$-diagnosability and diagnosability. For a collection of local masks $\{M_i\}$ $(i \in I = \{1, \cdots, m\})$, we define a global mask $\overrightarrow{M} = (M_1, \cdots, M_m)$ as follows:

- $\forall \sigma \in \Sigma : \overrightarrow{M}(\sigma) = (M_1(\sigma), \cdots, M_m(\sigma))$, and

- $\forall s \in \Sigma^*, \sigma \in \Sigma : \overrightarrow{M}(s\sigma) = \overrightarrow{M}(s)\overrightarrow{M}(\sigma)$.

Proposition 5 Given a plant G and a specification model R with $L(R) \subseteq L(G)$, let $\{M_i | i \in I\}$ be local masks, and $\overrightarrow{M} = (M_1, \cdots, M_m)$ be a global mask. Then,

$(L(G), L(R))$ is joint$_0^{iop}$-diagnosable $\Leftrightarrow$ $(L(G), L(R))$ is diagnosable with respect to $\overrightarrow{M}$.

Proof: It follows from definitions of joint$_0^{iop}$-diagnosability and diagnosability that we only need to show the following claim to prove the above proposition:

$$\forall s, t \in L(G), i \in I : \Upsilon_i^{iop,0}(s,t) = 1 \Leftrightarrow \overrightarrow{M}(s) = \overrightarrow{M}(t).$$

According to Definition 13, $\Upsilon_i^{iop,0}(s,t) = 1$ if and only if $O_i^{iop,0}(s) \cap O_i^{iop,0}(t) \neq \emptyset$, where $O_i^{iop,0}(s) = \mathcal{M}_i(\Pi_\Sigma^{-1}(s) \cap L(\mathcal{G}^0))$, $O_i^{iop,0}(t) = \mathcal{M}_i(\Pi_\Sigma^{-1}(t) \cap L(\mathcal{G}^0))$, and $\mathcal{G}^0 = G\|_{i \neq j} C_{ij}^0$. The structure of C_{ij}^0 is shown in Figure 4.9, where $a, b, c, \cdots \in \Sigma$, and their observations under M_i are $\underline{a}^i, \underline{b}^i, \underline{c}^i, \cdots \in \overline{\Lambda}_i$, respectively. Since events in Σ are executed synchronously and events in $\bigcup_{i \in I} \Lambda_i$ are executed asynchronously in $\mathcal{G}^0$, for $s = a_1 a_2 \cdots a_l \in L(G)$, it follows that

$$\Pi_\Sigma^{-1}(s) \cap L(\mathcal{G}^0) = a_1(\underline{a}_1^1 \| \cdots \| \underline{a}_1^m) \cdots a_l(\underline{a}_l^1 \| \cdots \| \underline{a}_l^m).$$

Applying observation mask $\mathcal{M}_i$ upon the above formula, we get

$$O_i^{iop,0}(s) = \mathcal{M}_i(\Pi_\Sigma^{-1}(s) \cap L(\mathcal{G}^0)) = \underline{a}_1^i(\|_{j \neq i} \underline{a}_1^j) \cdots \underline{a}_l^i(\|_{j \neq i} \underline{a}_l^j).$$

Thus, for $s = a_1 \cdots a_l \in L(G)$ and $t = b_1 \cdots b_{l'} \in L(G)$, $\Upsilon_1^{iop,0}(s,t) = 1$ if and only if

$$[\underline{a}_1^1(\underline{a}_1^2 \| \cdots \| \underline{a}_1^m) \cdots \underline{a}_l^1(\underline{a}_l^2 \| \cdots \| \underline{a}_l^m)] \cap [\underline{b}_1^1(\underline{b}_1^2 \| \cdots \| \underline{b}_1^m) \cdots \underline{b}_{l'}^1(\underline{b}_{l'}^2 \| \cdots \| \underline{b}_{l'}^m)] \neq \emptyset. \quad (4.14)$$

Since it is assumed that $\Lambda_i \cap \Lambda_j = \emptyset$, i.e., $\underline{a}^i \neq \underline{b}^j$ for $i \neq j$, it follows from (4.14) that

$$\underline{a}_1^1(\underline{a}_1^2 \cdots \underline{a}_1^m) \cdots \underline{a}_l^1(\underline{a}_l^2 \cdots \underline{a}_l^m) = \underline{b}_1^1(\underline{b}_1^2 \cdots \underline{b}_1^m) \cdots \underline{b}_{l'}^1(\underline{b}_{l'}^2 \cdots \underline{b}_{l'}^m),$$

which implies $\overrightarrow{M}(s) = \overrightarrow{M}(t)$. On the other hand, if $\overrightarrow{M}(s) = \overrightarrow{M}(t)$, we can similarly prove that $\Upsilon_i^{iop,0}(s,t) = 1$ for all $i \in I$. This completes the proof. ■

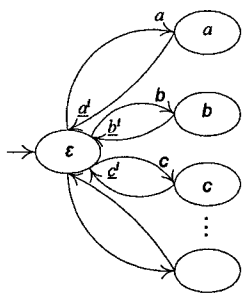


Figure 4.9 C_{ij}^0

Next, we present the relationship between joint$_k^{iop}$-diagnosability and joint$_{k+1}^{iop}$-diagnosability.

Proposition 6 Given a plant G and a specification model R with $L(R) \subseteq L(G)$,

$$(L(G), L(R)) \text{ is joint}_{k+1}^{iop}\text{-diagnosable} \overrightarrow{\not\Leftarrow} (L(G), L(R)) \text{ is joint}_k^{iop}\text{-diagnosable.}$$

Proof: ($\Rightarrow$) We prove this property by showing that if $(L(G), L(R))$ is not joint$_k^{iop}$-diagnosable, then it is not joint$_{k+1}^{iop}$-diagnosable as well. If $(L(G), L(R))$ is not joint$_k^{iop}$-diagnosable, then $(L(\mathcal{G}^k), L(\mathcal{R}^k))$ is not codiagnosable with respect to $\{\mathcal{M}_i\}$, i.e.,

$$\forall n \in \mathcal{N}, \forall i \in I : \exists s \in L(\mathcal{G}^k) - L(\mathcal{R}^k) \text{ with } |s| \geq n, \exists u \in L(\mathcal{R}^k) \text{ s.t. } \mathcal{M}_i(s) = \mathcal{M}_i(u_i).$$

By Proposition 3, $L(\mathcal{G}^k) - L(\mathcal{R}^k) \subseteq L(\mathcal{G}^{k+1}) - L(\mathcal{R}^{k+1})$ and $L(\mathcal{R}^k) \subseteq L(\mathcal{R}^{k+1})$. Since the extended observation masks $\{\mathcal{M}_i\}$ do not depend on the delay bound, we have that

$$\forall n \in \mathcal{N}, \forall i \in I : \exists s \in L(\mathcal{G}^{k+1}) - L(\mathcal{R}^{k+1}) \text{ with } |s| \geq n, \exists u \in L(\mathcal{R}^{k+1}) \text{ s.t. } \mathcal{M}_i(s) = \mathcal{M}_i(u_i).$$

I.e., $(L(\mathcal{G}^{k+1}), L(\mathcal{R}^{k+1}))$ is not codiagnosable with respect to $\{\mathcal{M}_i\}$. Thus, $(L(G), L(R))$ is not joint$_{k+1}^{iop}$-diagnosable.

($\not\Leftarrow$) Example 8 shows a system which is joint$_1^{iop}$-diagnosable, but not joint$_2^{iop}$-diagnosable. Thus, in general, joint$_k^{iop}$-diagnosability does not imply joint$_{k+1}^{iop}$-diagnosability. ■

4.4 Distributed Diagnosis Under Unbounded-Delay Communication

In this section, we discuss distributed diagnosis under unbounded-delay communication. This problem was first considered in [62], where the authors introduced a notion of decentralized-diagnosability, and showed that the problem is not decidable. In contrast to that result, our study shows that the property of decentralized-diagnosability is not strong enough to capture the problem. Instead, we use the notion of joint$_\infty^{gen}$-diagnosability as introduced in the last section, and show the decidability of distributed diagnosis under unbounded-delay communication by establishing the equivalence between codiagnosability and joint$_\infty^{gen}$-diagnosability.

Theorem 7 Let L be the prefix-closed language generated by a system and K be a prefix-closed specification language contained in L ($K \subseteq L$). (L, K) is joint$_\infty^{gen}$-diagnosable under a general protocol $P^{gen} \in \mathcal{P}^{gen}$ if and only if (L, K) is codiagnosable.

Proof: Since codiagnosability assumes no communication, whereas joint$_\infty^{gen}$-diagnosability does, the sufficiency follows from the fact that any communication can only provide an additional information.

Now suppose that (L, K) is joint$_\infty^{gen}$-diagnosability, but not codiagnosable. Then for some faulty trace $s \in L - K$, there exists no diagnoser which can detect the occurrence of the fault within a bounded delay when there is no communication among diagnosers. However, when there is communication among diagnosers, there exists a diagnoser i that detects the occurrence of the fault within a bounded delay, say n, i.e., by the time system executes some extension t of s with $|t| \geq n$. This means that $\Upsilon_i^{gen,\infty}(st, u) = 0$, which is equivalent to

$$\forall u \in K : O_i^{gen,\infty}(st) \cap O_i^{gen,\infty}(u) = \emptyset. \tag{4.15}$$

Due to the unbounded communication delay it is possible that immediately after the execution of the trace st or trace u no communication is received at site i from any other site $j \neq i$. This means that a possible sequence of extended observations immediately after the execution of trace st or trace u at site i is $M_i(st)$ or $M_i(u)$, respectively, i.e., $M_i(st) \in O_i^{gen,\infty}(st)$ and

$M_i(u) \in O_i^{gen,\infty}(u)$. From (4.15) it follows that $M_i(st) \neq M_i(u)$, which is contradictory to the fact that (L, K) is not codiagnosable. ■

Remark 15 In Chapter 3, we proposed an algorithm for verifying codiagnosability with complexity $\mathcal{O}(|G| \times |R|^{m+1})$, where G and R are the system and the non-fault specification automata models. It is implied by Theorem 7 that the same algorithm can be applied to verify joint$_\infty^{gen}$-diagnosability, which implies the decidability of joint$_\infty^{gen}$-diagnosability.

4.4.1 Joint$_\infty^{gen}$-Diagnosability vs. Decentralized-Diagnosability

We show that joint$_\infty^{gen}$-diagnosability is strictly stronger than decentralized-diagnosability introduced in [62] as an attempt to define distributed diagnosability under unbounded-delay communication.

Definition 15 [62] Let L be the prefix-closed language generated by a system and K be a prefix-closed specification language contained in L ($K \subseteq L$). Assume there are m local sites with observation masks $M_i : \overline{\Sigma} \to \overline{\Lambda}_i$ ($i \in I = \{1, \cdots, m\}$). (L, K) is said to be *decentralized-diagnosable* with respect to $\{M_i\}$ if

$$(\exists n \in \mathcal{N})(\forall s \in L - K)(\forall st \in L - K, |t| \geq n)(\forall u \in \bigcap_{i \in I} M_i^{-1} M_i(st) \cap L) \Rightarrow (u \in L - K).$$

It follows from the above definition that decentralized-diagnosability is violated if and only if there exists a non-faulty trace u that is indistinguishable from st to *all* diagnosers. But even when such a trace u does not exist, there may exist a set of non-faulty traces $\{u_i\}$, one for each diagnoser i, such that u_i is indistinguishable from st to diagnoser i. Then each diagnoser can not conclude on its own that a fault has occurred. Further a communication from another diagnoser may not be received by the time system executes st due to the unbounded-delay communication. So even when there does not exist a *single* non-faulty trace u that is indistinguishable from st to all diagnosers, it is possible that diagnosis cannot be performed. The following lemma shows that codiagnosability is strictly stronger than decentralized-diagnosability.

Lemma 7 Codiagnosability $\overset{\Rightarrow}{\not\Leftarrow}$ Decentralized-diagnosability.

Proof: We first prove the forward direction. If (L, K) is codiagnosable, then

$$(\exists n \in \mathcal{N})(\forall s \in L - K)(\forall st \in L - K, |t| \geq n) \Rightarrow$$

$$(\exists i \in I)(\forall u \in L, M_i(u) = M_i(st) \Rightarrow u \in L - K). \tag{4.16}$$

To prove decentralized-diagnosability, we claim that the same delay bound n works. To see this, pick $s \in L - K$ and t such that $st \in L - K$ and $|t| \geq n$. Also pick $u \in \bigcap_{i \in I} M_i^{-1}(st) \cap L$. Then $u \in L$ and $M_i(u) = M_i(st)$ for all $i \in I$. Suppose for contradiction that $u \notin L - K$, and define $u_i := u$ for all $i \in I$. Then $M_i(u_i) = M_i(st)$ and $u_i \notin L - K$ for all $i \in I$. This contradicts condition (4.16).

Next we establish the backward direction. Since the problem of verifying decentralized-diagnosability is undecidable [62], while the problem of verifying codiagnosability is decidable as shown in 3, the second assertion must hold. Otherwise, together with the first assertion, it will follow that decentralized-diagnosability is equivalent to codiagnosability, which contradicts their decidability properties. ■

The following example illustrates a system which is decentralized-diagnosable, but not codiagnosable.

Example 10 Let a system language be $L = pr(abc^* + ac^* + bc^*)$, and consider a specification language $K = pr(ac^* + bc^*)$. Then the traces in abc^* are faulty, whereas the traces in ac^* or bc^* are non-faulty. The event set is $\Sigma = \{a, b, c\}$, and the observation masks $\{M_i\}$ $(i = 1, 2)$ are defined as follows: $M_1(a) = a, M_1(b) = M_1(c) = \epsilon$; $M_2(b) = b, M_2(a) = M_2(c) = \epsilon$. Thus, we have that $M_1(abc^*) = M_1(ac^*) = a, M_1(bc^*) = \epsilon$, and $M_2(abc^*) = M_2(bc^*) = b, M_2(ac^*) = \epsilon$.

The local observations of sites 1 and 2 for the faulty traces in abc^* are given by a and b respectively. On the other hand, the local observations of site 1 and 2 for the non-faulty traces in ac^* are a and ϵ respectively, and those for the non-faulty traces in bc^* are ϵ and b respectively. Since the local observations of the two sites for the non-faulty traces are not the same as those from the faulty traces, the system is decentralized-diagnosable. In contrast, when faulty traces in abc^* are executed, the first diagnoser is "ambiguous" since the non-faulty

traces in ac^* generate the same observation, i.e., a. On the other hand the second diagnoser is also "ambiguous" since the non-faulty traces in bc^* generate the same observation, i.e., b. Thus the system is not codiagnosable.

Since joint$_\infty^{gen}$-diagnosability is equivalent to codiagnosability as shown in Theorem 7, the following theorem can be directly derived from Lemma 7.

Theorem 8 Joint$_\infty^{gen}$-diagnosability is strictly stronger than decentralized-diagnosability.

CHAPTER 5. PRIORITIZED SYNCHRONIZATION BASED DECENTRALIZED CONTROL

In this chapter, we study the decentralized control problem of discrete event systems via prioritized synchronous composition (PSC). PSC was introduced by Heymann to relax certain synchronization requirements on plant and supervisors. PSC was used for decentralized control for the first time by Kumar and Shayman with a restriction that priority sets of supervisors exhaust the controllable events set. We generalize that prior work by relaxing the above restriction, and introduces the notion of PSC-coobservability, which together with controllability serves as a necessary and sufficient condition for the existence of PSC-based decentralized supervisors. PSC-coobservability is more general than C&P $\vee$ D&A-coobservability introduced by Yoo and Lafortune, as it has the mechanism to support control-authority besides control-capability. (When there is flexibility in choosing the priority and conjunction/disjunction sets, the class of PSC-coobservable and C&P$\vee$D&A-coobservable languages coincide.) An algorithm for synthesizing supervisors is presented with complexity polynomial in plant and exponential in specification size. (Prior construction method of conjunctive setting does not work in our setting, whereas the synthesis method suggested by Yoo and Lafortune is exponential in both plant and specification.) Our synthesis algorithm is also applicable in the setting of centralized control under partial observation, and conjunction+disjunction-based decentralized control. A simple manufacturing example is presented to motivate and illustrate the PSC-based decentralized control.

5.1 PSC-Based Decentralized Control Policy

In this section, we study the PSC-based decentralized control policy. The priority sets A_1 and A_2 associated with two automata G_1 and G_2 can equivalently be viewed as a four-way partitioning of the event set [6]:

1. The *G_1-controlled event set*, $A_1 - A_2$, in which the events are executed at a state in the resultant automaton if and only if the event is permitted at the participating state of automaton G_1. This set of events we term as the Σ_{A_1} partition.

2. The *G_2-controlled event set*, $A_2 - A_1$, obtained in a similar way to the G_1-controlled event set. This set of events we term as the Σ_{A_2} partition.

3. The *$(G_1 \wedge G_2)$-controlled event set*, $A_1 \cap A_2$, in which the events are executed concurrently or not at all. This set of events we term as the $\Sigma_{\wedge}$ partition.

4. The *$(G_1 \vee G_2)$-controlled event set*, $\Sigma - (A_1 \cup A_2)$, in which the events are executed at a state in the resultant automaton if and only if at least one of the participating states of the automata, G_1 or G_2, permit it. This set of events we term as the $\Sigma_{\vee}$ partition.

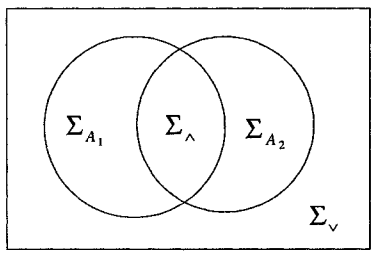


Figure 5.1 Partition sets under PSC

In PSC-based decentralized control, PSC is used to model the the interaction among local supervisors, and the interaction between the plant and the supervisors as well. We choose the priority set of the plant to be Σ, and the priority set of local supervisor S_i to be A_i. Note that Σ_{ci} denotes the set of events S_i can control, a subset A_i of such events necessarily require the participation of S_i for their global enablement. Thus the priority set A_i of S_i is naturally a subset of Σ_{ci}. Note that this implies, $\bigcup_i A_i \subseteq \bigcup_i \Sigma_{ci} = \Sigma_c$, which is weaker than the requirement

$\bigcup_i A_i = \Sigma_c$ imposed in [33]. Without loss of generality, we consider two local supervisors in the following discussion, i.e., $I = \{1, 2\}$.

Let $\Gamma_i \subseteq 2^{\Sigma}$ denote the set of control actions of supervisor $i \in I$. Assume that $L = pr(L) \neq \emptyset$ is the generated language of the plant. In the PSC-based decentralized control, the *local PSC-based control policy* for supervisor i is defined as $f_i : M_i(L) \rightarrow \Gamma_i$, a map from the locally observed system behavior to the locally enabled events. Partial observation leads to ambiguity in choosing control action since an event may be legal following a certain trace, but illegal following an indistinguishable trace. In the case of ambiguity, supervisors take certain default control actions. The default decision for the PSC-based decentralized control is stated as follows: *Whenever a supervisor is ambiguous, the default decision is to enable an event if and only if the event is in the priority set of that supervisor.* This allows the flexibility that the decisions by other supervisors, who are unambiguous, can override any default decisions. Consider the converse for the sake of argument: If a priority event were to be disabled by default by a local supervisor, then it would remain disabled globally regardless of the decisions of other local supervisors, even though they may be unambiguous. In other words, disablement of a priority event cannot be made a default decision for a local supervisor having priority over that event. The above default decision rule for the four partitions of Σ_c induced by two priority sets is summarized in Table 5.1, where f_1 and f_2 indicate the control decisions of supervisor 1 and supervisor 2, respectively. (The table also lists the fusion rules, to be discussed below.)

Table 5.1 Default decisions and fusion rules for PSC (1: enable; 0: disable)

	Default Decision	Fusion Rule
$\Sigma_\wedge$	$f_1 = 1, f_2 = 1$	$f_{PSC} = 1 \Leftrightarrow [f_1 = 1] \wedge [f_2 = 1]$
Σ_{A_1}	$f_1 = 1, f_2 = 0$	$f_{PSC} = 1 \Leftrightarrow [f_1 = 1]$
Σ_{A_2}	$f_1 = 0, f_2 = 1$	$f_{PSC} = 1 \Leftrightarrow [f_2 = 1]$
$\Sigma_\vee$	$f_1 = 0, f_2 = 0$	$f_{PSC} = 1 \Leftrightarrow [f_1 = 1] \vee [f_2 = 1]$

Let $K \subseteq L$ be a specification language, which specifies the desired closed-loop behavior. In accordance with the default decision rule, we define the local PSC-based control policy for

supervisor i as follows: $\forall s \in L$,

$$\begin{aligned} f_i(M_i(s)) := &\{\sigma \in A_i \mid [M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) \neq \emptyset\} \\ \cup &\{\sigma \in \Sigma_{ci} - A_i \mid [M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap L \subseteq pr(K)\}. \end{aligned} \tag{5.1}$$

The above control policy states that an event is enabled by a local supervisor if that event is a priority event and cannot be unambiguously disabled, or it is a non-priority event and can be unambiguously enabled. Note that a PSC-based supervisor does not need to participate in the occurrence of events which it cannot control, i.e., events in $\Sigma - \Sigma_{ci}$. No control decision is mentioned for such events, but by definition of PSC, they remain enabled.

We fuse local decisions together to get the overall control decision. Let $\Gamma \subseteq 2^{\Sigma}$ denote the global control actions. The *global PSC-based control policy* is defined as $f_{PSC} : L \rightarrow \Gamma$, a map from the system behavior to the set of global control actions. For two local supervisors, the decision fusion rule is defined for each partition set as follows. For an event in $\Sigma_\wedge := A_1 \cap A_2$, it is enabled if and only if both supervisors enable it; for an event in $\Sigma_{A_i} := A_i - A_j$ $(i, j \in I, i \neq j)$, it is enabled if and only if supervisor i enables it; and for an event in $\Sigma_\vee := \Sigma_c - (A_1 \cup A_2)$, it is enabled if and only if either supervisor enables it. This fusion rule is shown in Table 5.1, where f_{PSC} indicates the fused control decision.

Define $A := A_1 \cup A_2$ as the set of priority events and $I_A(\sigma) := \{i \in I | \sigma \in A_i\}$ as the index set of all local supervisors whose priority event set contains σ. Then, based on the decision fusion rule, the global PSC-based control policy is defined as follows: $\forall s \in L$,

$$f_{PSC}(s) = \{\sigma \in A \mid \sigma \in \bigcap_{i \in I_A(\sigma)} f_i(M_i(s))\} \cup \{\sigma \in \Sigma_c - A \mid \sigma \in \bigcup_{i \in I_c(\sigma)} f_i(M_i(s))\}. \tag{5.2}$$

The global control policy f_{PSC} indicates that $\sigma \in A$ is enabled if it is enabled by all local supervisors whose priority sets contain σ (*prioritized conjunctive rule*), and $\sigma \in \Sigma_c - A$ is enabled if it is enabled by at least one local supervisor which can control the event (*disjunctive rule*).

The closed-loop behavior achieved using the global PSC-based control policy is defined as follows.

Definition 16 Let L and L_m be the generated and marked languages of a plant, respectively. Then the generated and marked languages of the controlled plant under the global PSC-based control policy f_{PSC}, denoted by L/f_{PSC} and L_m/f_{PSC} respectively, are defined as follows:

- $\epsilon \in L/f_{PSC}$; $[s\sigma \in L/f_{PSC}] \Leftrightarrow [s \in L/f_{PSC}] \wedge [s\sigma \in L] \wedge [\sigma \in \Sigma_u \cup f_{PSC}(s)]$;
- $L_m/f_{PSC} := L/f_{PSC} \cap L_m$.

We show via the following example the generality of the PSC-based decentralized control considered in this paper when compared to that in [33].

Example 11 Consider the generated language of a plant $L = pr(c + ac + abc)$ and the specification language $K = pr(c + abc)$. Let $M_1(a) = a, M_1(b) = \epsilon, M_1(c) = c$, $M_2(a) = \epsilon, M_2(b) = b, M_2(c) = c$, $\Sigma_{c1} = \{a, c\}$, and $\Sigma_{c2} = \{b, c\}$. Choose $A_1 = \{a\}$ and $A_2 = \{b\}$. Then it is easy to verify that the specification language K can be achieved through the PSC-based control policies shown in Table 5.2. However, if we require that the priority sets satisfy the condition $A_1 \cup A_2 = \Sigma_c = \{a, b, c\}$ as required in [33], then it can be verified that K is not achievable no matter how the priority sets are assigned.

Table 5.2 PSC-based local and global control policies for Example 11

s	$f_1(M_1(s))$	$f_2(M_2(s))$	$f_{PSC}(s)$
ϵ	$\{a, c\}$	$\{b\}$	$\{a, b, c\}$
a	$\emptyset$	$\{b\}$	$\{b\}$
ab	$\emptyset$	$\{c\}$	$\{c\}$

5.2 PSC-Coobservability: Existence, Test, and Properties

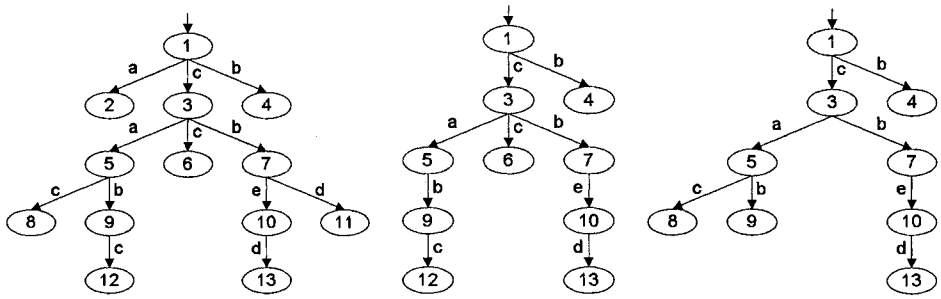
To capture the restriction imposed by the partial controllability and observability of the supervisors, and the restriction of prioritized composition, the property of PSC-coobservability is introduced as follows.

Definition 17 Given a prefix-closed language L, controllable event sets Σ_{ci}, observation masks M_i, and priority sets $A_i \subseteq \Sigma_{ci}$ $(i \in I = \{1,2\})$ with $A := A_1 \cup A_2$ and $\Sigma_c := \Sigma_{c1} \cup \Sigma_{c2}$, K is said to be $(L, \Sigma_{ci}, M_i, A_i)$-*PSC-coobservable* if

1. $\forall s \in pr(K), \sigma \in A, s\sigma \in L - pr(K) : \exists i \in I_A(\sigma)$ s.t. $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) = \emptyset$,
2. $\forall s \in pr(K), \sigma \in \Sigma_V, s\sigma \in pr(K) : \exists i \in I_c(\sigma)$ s.t. $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap L \subseteq pr(K)$.

Condition 1 indicates that for a priority event in A to be disabled it needs to be unambiguously disabled by at least one supervisor whose priority set contains that event. Condition 2 indicates that for a non-priority event in $\Sigma_V = \Sigma_c - A$ to be enabled it needs to be unambiguously enabled by at least one supervisor that controls the event. The notion of PSC-coobservability is illustrated through the following example.

Example 12 Consider a plant model G and a specification model R shown in Figure 5.2(a) and Figure 5.2(b), respectively. Let $L = L(G)$ and $K = L(R)$. The set of events is $\Sigma = \{a, b, c, d, e\}$. Assume there are two local supervisors with $M_1(a) = a$, $M_1(c) = c$, $M_1(b) = M_1(d) = M_1(e) = \epsilon$, $M_2(b) = b$, $M_2(e) = e$, $M_2(a) = M_2(c) = M_2(d) = \epsilon$, $\Sigma_{c1} = \{a, c, d\}$ and $\Sigma_{c2} = \{b, c, d\}$. Let $A_1 = \{a, d\}$ and $A_2 = \{b, d\}$. Thus, $\Sigma_\wedge = \{d\}$, $\Sigma_{A_1} = \{a\}$, $\Sigma_{A_2} = \{b\}$, and $\Sigma_V = \{c\}$. Specification requires that event a be disabled at state 1, event c at state 5, and event d at state 7. Event a can be unambiguously disabled at state 1 since $[M_1^{-1}M_1(\epsilon) \cap pr(K)]a \cap pr(K) = \emptyset$. Similarly, event d can be unambiguously disabled at state 7 since $[M_2^{-1}M_2(cb) \cap pr(K)]d \cap pr(K) = \emptyset$. For the non-priority event c, if it cannot be unambiguously enabled, it will be disabled by default. Since $cac \in L - K$, event c is disabled at state 5. We can also verify that event c can be unambiguously enabled at state 3 and 9 since $[M_1^{-1}M_1(c) \cap pr(K)]c \cap L = \{cc\} \subseteq pr(K)$ and $[M_2^{-1}M_2(b) \cap pr(K)]c \cap L = \{cabc\} \subseteq pr(K)$. Therefore, according to Definition 17, K is $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable. For another specification model R' shown in Figure 5.2(c), we verify below that $L(R')$ is not $(L(G), \Sigma_{ci}, M_i, A_i)$-PSC-coobservable.



(a) plant G (b) specification R (c) specification R'

Figure 5.2 Diagrams illustrating Example 12

Theorem 9 Consider a plant G with generated language L and marked language L_m. Let Σ_{ci}, M_i and $A_i \subseteq \Sigma_{ci}$ $(i \in I = \{1, 2\})$ be controllable event sets, observation masks and priority sets, respectively.

1. For nonempty $K \subseteq L$ there exists a PSC-based control policy f_{PSC} such that $L/f_{PSC} = K$ if and only if K is (L, Σ_u)-controllable, $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable and prefix-closed.

2. For nonempty $K \subseteq L_m$ there exists a nonblocking PSC-based control policy f_{PSC} such that $L_m/f_{PSC} = K$ if and only if K is (L, Σ_u)-controllable, $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable and relative-closed with respect to L_m.

Proof: We first prove the necessary condition of assertion 1. Suppose there exists f_{PSC} such that $L/f_{PSC} = K$. For all $s \in pr(K) = L/f_{PSC}, \sigma \in \Sigma_u$, and $s\sigma \in L$, we have $s\sigma \in L/f_{PSC}$, which indicates that K is (L, Σ_u)-controllable. The prefix-closure of K follows from Definition 16, which implies that L/f_{PSC} is prefix-closed. Next we prove the PSC-coobservability of K.

For all $s \in pr(K), \sigma \in A, s\sigma \in L - pr(K)$, we need to show that there exists $i \in I_A(\sigma)$ such that $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) = \emptyset$. For the purpose of contradiction, assume that for all $i \in I_A(\sigma)$, $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) \neq \emptyset$. I.e., there is no supervisor that can unambiguously disable σ. Then, all supervisors whose priority sets contain σ take the default decision to enable σ. According to the fusion rule, the fused decision is to enable σ, which conflicts with the condition $s\sigma \in L(G) - pr(K)$.

For all $s \in pr(K), \sigma \in \Sigma_c - A, s\sigma \in pr(K)$, it can be verified in a similar way that there exists $i \in I_c(\sigma)$ such that $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap L \subseteq pr(K)$. Therefore, K is $(L(G), \Sigma_{ci}, M_i, A_i)$-PSC-coobservable. This completes the proof of the necessary condition of assertion 1.

To prove the sufficient condition of assertion 1, we need to show that if K is controllable, PSC-coobservable and prefix-closed, then there exists f_{PSC} such that $L/f_{PSC} = K$. The proof is done by induction on the length of traces in L/f_{PSC} and K. It is trivial to show that $\epsilon \in L/f_{PSC} \Leftrightarrow \epsilon \in K$. (Left hand side follows from Definition 16, and right hand side from the facts that K is nonempty and prefix-closed.) Assume for all $|s| \leq n$, $s \in K$ if and only if $s \in L/f_{PSC}$. We need to show that

$$\forall \sigma \in \Sigma, s\sigma \in K \Leftrightarrow s\sigma \in L/f_{PSC}, \text{ where } |s| = n.$$

($\sigma \in \Sigma_u$) If $s\sigma \in K$, then $s \in K$. From the induction hypothesis, we have $s \in L/f_{PSC}$. It follows from Definition 16 that $s\sigma \in L/f_{PSC}$. On the other hand, if $s\sigma \in L/f_{PSC}$, then $s \in L/f_{PSC}$, and again from the induction hypothesis $s \in K$. It follows from the controllability and prefix-closure of K that $s\sigma \in K$.

($\sigma \in A$) If $s\sigma \in K = pr(K)$, then from the definition of PSC-coobservability, we have that for all $i \in I_A(\sigma)$, $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) \neq \emptyset$, which implies that $\sigma \in f_i(M_i(s))$ for all $i \in I_A(\sigma)$. Since $f_{PSC}(s) = \{\sigma \in A \mid \sigma \in \bigcap_{i \in I_A(\sigma)} f_i(M_i(s))\}$, we have $\sigma \in f_{PSC}(s)$. It follows from Definition 16 that $s\sigma \in L/f_{PSC}$. On the other hand, if $s\sigma \in L/f_{PSC}$, let us assume $s\sigma \notin K$ for the purpose of contradiction. Then from definitions of PSC-coobservability, there exists $i \in I_A(\sigma)$ such that $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) = \emptyset$, which implies that $\sigma \notin f_i(s)$ for some i. Then from the definition of f_{PSC}, $\sigma \notin f_{PSC}(s)$, which conflicts with $s\sigma \in L/f_{PSC}$. Thus, $s\sigma \in K$.

($\sigma \in \Sigma_c - A$) The proof is similar to that for $\sigma \in A$. This completes the induction step, and the proof for assertion 1.

In the following, we present the proof for assertion 2. For necessity, suppose there exists nonblocking f_{PSC} such that $L_m/f_{PSC} = K$. Then $L/f_{PSC} = pr(L_m/f_{PSC}) = pr(K)$. From assertion 1, $pr(K)$ is nonempty, (L, Σ_u)-controllable and $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable,

implying K is nonempty, (L, Σ_u)-controllable and $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable. Also, $pr(K) \cap L_m = pr(L_m/f_{PSC}) \cap L_m = pr(L/f_{PSC} \cap L_m) \cap L_m \subseteq L/f_{PSC} \cap pr(L_m) \cap L_m = L/f_{PSC} \cap L_m = L_m/f_{PSC} = K$, i.e., K is relative-closed. For sufficiency, choose f_{PSC} such that $L/f_{PSC} = pr(K)$. Then $L_m/f_{PSC} = L/f_{PSC} \cap L_m = pr(K) \cap L_m = K$, where last equality follows from relative-closure of K. Finally, since $pr(L_m/f_{PSC}) = pr(K) = L/f_{PSC}$, f_{PSC} is nonblocking. ■

Remark 16 The proof of Theorem 9 implies that the same control policy f_{PSC} can be applied for both the basic control ($L/f_{PSC} = K$) and the nonblocking control ($L_m/f_{PSC} = K$ and $pr(L_m/f_{PSC}) = L/f_{PSC}$).

5.2.1 Verification of PSC-Coobservability

Next we show that PSC-coobservability is polynomially verifiable. In order to discuss the verification, we first present an equivalent statement of PSC-coobservability as follows.

Proposition 7 Given a prefix-closed language L, controllable event sets Σ_{ci}, observation masks M_i, and priority sets $A_i \subseteq \Sigma_{ci}$ $(i \in I = \{1, 2\})$ with $A := A_1 \cup A_2$, K is $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable if and only if for all $s, s_1, s_2 \in pr(K)$ with $[M_1(s) = M_1(s_1)] \wedge [M_2(s) = M_2(s_2)]$ and $\sigma \in \Sigma_c$ with $s\sigma \in L$ the following conditions hold:

1 $[\sigma \in \Sigma_\wedge] \wedge [s_1\sigma \in pr(K)] \wedge [s_2\sigma \in pr(K)] \Rightarrow s\sigma \in pr(K)$;

2 $[\sigma \in \Sigma_{A_i}] \wedge [s_i\sigma \in pr(K)] \Rightarrow s\sigma \in pr(K)$ $(i \in I)$;

3.a $[\sigma \in \Sigma_\vee] \wedge [\sigma \in \Sigma_{c1} \cap \Sigma_{c2}] \wedge [s_1\sigma \in L - pr(K)] \wedge [s_2\sigma \in L - pr(K)] \Rightarrow s\sigma \in L - pr(K)$,

3.b $[\sigma \in \Sigma_\vee] \wedge [\sigma \in \Sigma_{c_i} - \Sigma_{c_j}] \wedge [s_i\sigma \in L - pr(K)] \Rightarrow s\sigma \in L - pr(K)$ $(i, j \in I, i \neq j)$.

Proof: (Only if) Suppose K is $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable. We show that the second condition holds, and other conditions can be proved similarly. Assume for contradiction that the second condition fails, and without loss of generality let $i = 1$. I.e., $[\sigma \in \Sigma_{A_1}] \wedge [s_1\sigma \in pr(K)] \wedge [s\sigma \in L - pr(K)]$. According to Definition 17, there exists $i \in I_A(\sigma) = \{1\}$ such that

$[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) = \emptyset$, which conflicts with $[s_1\sigma \in pr(K)]$ since $M_1(s) = M_1(s_1)$. Thus, the second condition holds.

(If) We show that the first condition in Definition 17 holds in the following. (The second condition can be proved similarly.) For $\sigma \in A$ and $s\sigma \in L - pr(K)$, first consider the case $\sigma \in \Sigma_{A_1}$. For $[\sigma \in \Sigma_{A_1}] \wedge [s\sigma \in L - pr(K)]$, assume for contradiction that $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) \neq \emptyset$. I.e., there exists $s_1 \in pr(K)$ with $M_1(s) = M_1(s_1)$ and $s_1\sigma \in pr(K)$. Then we get a contradiction that $s\sigma \in pr(K)$ from the second condition in the above. Thus, $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) = \emptyset$. Similar proofs can be developed for cases $\sigma \in \Sigma_{A_2}$ and $\sigma \in \Sigma_\wedge$. ■

Based on the above expanded version of PSC-coobservability, we develop an algorithmic test as in [56, 75], where a certain testing automaton is constructed to track the bad traces which violate the coobservability condition. The only marked state in the testing automaton is the dump state. We do not present all the details of the testing automata in this paper since it is similar to that in [56, 75]. Briefly, the testing automaton has a transition function so that for all $s, s_1, s_2 \in pr(K)$, it tracks the ones with $[M_1(s) = M_1(s_1)] \wedge [M_2(s) = M_2(s_2)]$ (see for example [56, 75]). Here, we only discuss the violating condition for each partition set and the corresponding state space in each testing automaton. This is shown in Table 5.3, where X is the state set in the plant model G, Y is the state set in the specification model R, and d is the dump state.

Each violating condition causes a transition to the dump state. In each condition, we assume that the event σ in trace $s_i\sigma$ $(i \in I)$ can be controlled by the corresponding supervisor S_i $(i \in I)$. If σ is not controlled by a supervisor i, we ignore condition on $s_i\sigma$ $(i \in I)$. For example, if $\sigma \in \Sigma_\vee$ and $\sigma \in \Sigma_{c1} - \Sigma_{c2}$, we only check if the condition $[s\sigma \in pr(K)] \wedge [s_1\sigma \in L - pr(K)]$ is violated or not, and ignore the condition $s_2\sigma \in L - pr(K)$ shown in Table 5.3 for $\Sigma_\vee$.

Let the testing automata for events in $\Sigma_\wedge$, Σ_{A_1}, Σ_{A_2} and $\Sigma_\vee$ be $T_\wedge(G,R)$, $T_{A_1}(G,R)$, $T_{A_2}(G,R)$ and $T_\vee(G,R)$, respectively. Then we have the following result:

Theorem 10 Given a plant G and a specification R with $L(G) = L$ and $L(R) = K$, K is $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable if and only if $L_m(T_i(G,R)) = \emptyset$ for all $i \in \{\wedge, A_1, A_2, \vee\}$,

Table 5.3 Violating conditions and state spaces in the testing automata for verifying PSC-coobservability

Partition Set	Violating Conditions	State Space Required
$\sigma \in \Sigma_\wedge$	$[s\sigma \in L - pr(K)] \wedge [s_1\sigma \in pr(K)] \wedge [s_2\sigma \in pr(K)]$	$(X \times Y) \times Y \times Y \cup \{d\}$
$\sigma \in \Sigma_{A_1}$	$[s\sigma \in L - pr(K)] \wedge [s_1\sigma \in pr(K)]$	$(X \times Y) \times Y \cup \{d\}$
$\sigma \in \Sigma_{A_2}$	$[s\sigma \in L - pr(K)] \wedge [s_2\sigma \in pr(K)]$	$(X \times Y) \times Y \cup \{d\}$
$\sigma \in \Sigma_\vee$	$[s\sigma \in pr(K)] \wedge [s_1\sigma \in L - pr(K)] \wedge [s_2\sigma \in L - pr(K)]$	$Y \times (X \times Y) \times (X \times Y) \cup \{d\}$

i.e., the dump state d is not reachable in all testing automata $T_i(G, R)$.

Proof: The proof is similar to that in [56]. ∎

Remark 17 The maximum searching space of testing automata $T_i(G, R)$ $(i \in \{\wedge, A_1, A_2, \vee\})$ includes states in two copies of plant automata and three copies of specification automata. Therefore, the algorithm for testing PSC-coobservability is quadratic in plant states and cubic in specification states, which is the same as the computational complexity of the algorithm for testing C&P∨D&A coobservability in [75].

Remark 18 Though the testing for PSC-coobservability can be done by testing four different automata, some of them can be combined together. For example, we can use one testing automaton with the state space $(X \times Y) \times Y \times Y \cup \{d\}$ for events in $\Sigma_\wedge$ and Σ_{A_i}. Since this method does not enlarge the search space, it has the same computational complexity as explained earlier. All these testing automata can also be combined into one single testing automaton with the state space $(X \times Y)^3 \cup \{d\}$. However, since the search space is enlarged by introducing another plant automaton, this method will have a higher computational complexity (cubic both in plant and specification states).

The following example constructs a testing automaton to verify the PSC-coobservability of language $K' = L(R')$ discussed in Example 12.

Example 13 It was mentioned in Example 12 that the language $K' = L(R')$ is not $(L(G), \Sigma_{ci}, M_i, A_i)$-PSC-coobservable. Figure 5.3 shows the testing automaton $T_{\vee}(G, R')$, where each state is labeled by (y, x_1, y_1, x_2, y_2). For all $s \in pr(K)$, $s_1, s_2 \in L - pr(K')$ with $[M_1(s) = M_1(s_1)] \wedge [M_2(s) = M_2(s_2)]$ and $\sigma \in \Sigma_{\vee} = \{c\}$, the state coordinates y, (x_1, y_1), and (x_2, y_2) track if $s\sigma \in pr(K')$, $s_1\sigma \in L - pr(K')$ and $s_2\sigma \in L - pr(K')$ respectively. It can be seen from Figure 5.3 that $L_m(T_{\vee}(G, R')) \neq \emptyset$. Therefore, K' is not $(L(G), \Sigma_{ci}, M_i, A_i)$-PSC-coobservable.

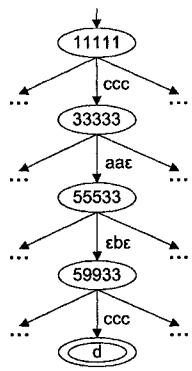


Figure 5.3 Testing automaton $T_{\vee}(G, R')$

5.2.2 PSC-Coobservability vs. C&P $\vee$ D&A-Coobservability

It is clear from the definition of PSC-coobservability that it bears similarity with C&P $\vee$ D&A-coobservability. Here we explore the similarity and differences between the two notions of coobservability. Comparing the definition of PSC-coobservability and C&P $\vee$ D&A-coobservability, we have the following property:

Proposition 8 Consider a prefix-closed language L, a sublanguage $K \subseteq L$, controllable event sets Σ_{ci}, and observation masks M_i $(i \in I = \{1, 2\})$. Given the priority set A_i $(i \in I)$ for the PSC architecture and the fusion partitions $\Sigma_{c,d}$ and $\Sigma_{c,e}$ for the conjunction+disjunction

architecture, the following properties hold:

1. $[\Sigma_{c,d} = A_1 \cup A_2 = A_1 = A_2] \Rightarrow [(L, \Sigma_{ci}, M_i, A_i)\text{-PSC-coobservability} \Leftrightarrow$
$(L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})\text{-C\&P} \vee \text{D\&A-coobservability}.$
2. $[\Sigma_{c,d} = A_1 \cup A_2, A_1 \neq A_2] \Rightarrow [(L, \Sigma_{ci}, M_i, A_i)\text{-PSC-coobservability} \Rightarrow$
$(L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})\text{-C\&P} \vee \text{D\&A-coobservability}].$
3. $[\Sigma_{c,d} \neq A_1 \cup A_2] \Rightarrow [(L, \Sigma_{ci}, M_i, A_i)\text{-PSC-coobservability} \not\Leftrightarrow$
$(L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})\text{-C\&P} \vee \text{D\&A-coobservability}].$

Proof: 1. When $\Sigma_{c,d} = A_1 = A_2$, there are only two partitions $\Sigma_\wedge$ and $\Sigma_\vee$ of Σ_c with $\Sigma_\wedge = \Sigma_{c,d}$ and $\Sigma_\vee = \Sigma_{c,e}$. Since the same default action (disable) and fusion rule (conjunctive rule) are applied for events in $\Sigma_\wedge$ and $\Sigma_{c,d}$, and the same default action (enable) and fusion rule (disjunctive rule) are applied for events in $\Sigma_\vee$ and $\Sigma_{c,e}$, it is easy to conclude the first assertion.

2. In the case $\Sigma_{c,d} = A_1 \cup A_2$ and $A_1 \neq A_2$, there are four PSC-based fusion partitions with $\Sigma_\wedge \subseteq \Sigma_{c,d}$ and $\Sigma_\vee = \Sigma_{c,e}$. Since we have $I_A(\sigma) = I_c(\sigma)$ for event $\sigma \in \Sigma_\wedge$, and $I_A(\sigma) \subseteq I_c(\sigma)$ for event $\sigma \in \Sigma_{A_i} - \Sigma_{A_j}$ $(i, j \in I = \{1, 2\}, i \neq j)$, it follows from the definitions of PSC-coobservability and C&P $\vee$ D&A-coobservability that

$$K\ (L, \Sigma_{ci}, M_i, A_i)\text{-PSC-coobservable} \Rightarrow K\ (L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})\text{-C\&P} \vee \text{D\&A-coobservable}.$$

However, the other direction may not be true. For example, suppose $\sigma \in A_1 - A_2$ and $\sigma \in \Sigma_{c1} \cap \Sigma_{c2}$. For $s \in pr(K)$ and $s\sigma \in L - K$ (i.e., σ needs to be disabled), consider the case that $[M_1^{-1} M_1(s) \cap pr(K)]\sigma \cap pr(K) \neq \emptyset$ and $[M_2^{-1} M_2(s) \cap pr(K)]\sigma \cap pr(K) = \emptyset$, i.e., only the second supervisor can disable σ unambiguously. According to the default rules and decision fusion rules in both architectures, it follows that K is $(L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})$-C&P$\vee$D&A-coobservable, but not $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable. This completes the proof of the second assertion.

3. For $\Sigma_{c,d} \neq A_1 \cup A_2$, the possible cases are: $\exists \sigma \in A_1 - \Sigma_{c,d}$, or $\exists \sigma \in A_2 - \Sigma_{c,d}$, or $\exists \sigma \in \Sigma_{c,d} - A$. In the first case, $\sigma \in A_1$ and $\sigma \in \Sigma_{c,e}$.

- Suppose $s\sigma \in pr(K)$. If $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap L \not\subseteq pr(K)$ holds for all $i \in I = \{1,2\}$, then σ is disabled in the conjunction+disjunction fusion architecture, which is in conflict with $s\sigma \in pr(K)$. Thus, K is not $(L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})$-C&P $\vee$ D&A-coobservable. On the other hand, since $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) \neq \emptyset$, it is not possible that σ is disabled in the PSC-based architecture. Thus, σ is enabled, and K is $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable.

- Suppose $s \in pr(K)$ and $s\sigma \in L - pr(K)$. If $[M_1^{-1}M_1(s) \cap pr(K)]\sigma \cap pr(K) \neq \emptyset$, then σ is enabled in the PSC-based architecture, which conflicts with $s\sigma \in L - pr(K)$. Thus K is not $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable. On the other hand, since $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap L \not\subseteq pr(K)$ holds for all $i \in I = \{1,2\}$, σ is disabled in the conjunction+disjunction fusion architecture. Thus, K is $(L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})$-C&P $\vee$ D&A-coobservable.

This shows that PSC-coobservability and C&P $\vee$ D&A-coobservability are incomparable in the case $\exists \sigma \in A_1 - \Sigma_{c,d}$. Similar arguments are applicable to the other two cases. This completes the proof of the third assertion. ■

The following corollary is immediate from Proposition 8, showing PSC-coobservability can subsume C&P $\vee$ D&A-coobservability.

Corollary 3 Consider a prefix-closed language L, a sublanguage $K \subseteq L$, controllable event sets Σ_{ci}, and observation masks M_i $(i \in I = \{1,2\})$.

1. Given the partition sets $\Sigma_{c,d}$ and $\Sigma_{c,e}$: $\exists A_1, A_2$, s.t.

$$K \text{ is } (L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})\text{-C\&P} \vee \text{D\&A-coobservable}$$

$$\Leftrightarrow K \text{ is } (L, \Sigma_{ci}, M_i, A_i)\text{-PSC-coobservable;}$$

2. Given a pair of priority sets A_1 and A_2: $\nexists \Sigma_{c,d}, \Sigma_{c,e}$ s.t.

$$K \text{ is } (L, \Sigma_{ci}, M_i, A_i)\text{-PSC-coobservable}$$

$$\Leftrightarrow K \text{ is } (L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})\text{-C\&P} \vee \text{D\&A-coobservable.}$$

Proof: The first conclusion follows from the first assertion in Proposition 8 by choosing $A_1 = A_2 = \Sigma_{c,d}$. The second conclusion follows from the second and third assertions in Proposition 8 that no matter how $\Sigma_{c,d}$ is chosen (whether equaling $A_1 \cup A_2$ or otherwise), the two coobservability properties do not coincide. ■

Although in general PSC-coobservability subsumes C&P $\vee$ D&A-coobservability, the following theorem shows that whenever there is flexibility of choosing the fusion partition sets and the priority sets, a PSC-based control exists if and only if a conjunction+disjunction-based control exists. (Such a flexibility may not always exist as is the case with the manufacturing example considered in Section 5.4.)

Theorem 11 Consider a prefix-closed language L, a sublanguage $K \subseteq L$, controllable event sets Σ_{ci}, and observation masks M_i $(i \in I = \{1,2\})$.

1. If the conjunction+disjunction fusion architecture and associated partitions $\Sigma_{c,d}$ and $\Sigma_{c,e}$ are chosen so that K is $(L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})$-C&P $\vee$ D&A-coobservable, then K is also $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable with $A_1 = A_2 = \Sigma_{c,d}$;

2. If the PSC fusion architecture and associated priority sets A_1 and A_2 are chosen so that K is $(L, \Sigma_{ci}, M_i, A_i)$-PSC-coobservable, then K is also $(L, \Sigma_{ci}, M_i, \Sigma_{c,d}/\Sigma_{c,e})$-C&P $\vee$ D&A-coobservable with $\Sigma_{c,d} = A_1 \cup A_2$ and $\Sigma_{c,e} = \Sigma_c - \Sigma_{c,d}$.

Proof: The first assertion readily follows from Corollary 3. The second assertion follows from the second assertion in Proposition 8 that PSC-coobservability is stronger than C&P $\vee$ D&A-coobservability when $\Sigma_{c,d} = A_1 \cup A_2$. ■

Remark 19 Theorem 11 shows that both the PSC and conjunction+disjunction architectures provide the same capability of decentralized control whenever there is flexibility of choosing the fusion partition sets and the priority sets. However, certain applications may require PSC-based decentralized control, in which case the second assertion in Corollary 3 indicates that it may not be possible to use conjunctive+disjunctive architecture as a replacement. Also, using the PSC-based control, the supervisors are not required to participate in the occurrence of all system events, which is its additional advantage.

5.3 Synthesis of PSC-Based Supervisors

Since in a complex system it is impossible to list control actions for all possible system observations, an off-line synthesis of PSC-based supervisors requires finding the automaton based representation of the PSC-based control policy. Then the supervisors restrict the behavior of a plant by way of prioritized synchronous composition with it.

Methods for realization of automata based supervisors do exist for centralized control, and for conjunctive decentralized control. For example, supervisors can be chosen to be generators of infimal prefix-closed, controllable and observable superlanguages with respect to local controllable events and local observation masks [34]. However, this method is not suitable for synthesizing PSC-based supervisors as shown in the following example, which advocates the need for a new synthesis method.

Example 14 Example 11 shows that there exist PSC-based control policies f_1, f_2 and f_{PSC} such that $L/f_{PSC} = K$. It is easy to verify that $inf\overline{PC_{\Sigma_{ui}}O_{M_i}}(K) = L = pr(c + ac + abc)$ for all $i \in I = \{1, 2\}$. Suppose we choose S_i to be a generator of $inf\overline{PC_{\Sigma_{ui}}O_{M_i}}(K) = L$. Then it follows that $L(S_1 \,{}_{A_1}\|_{A_2}\, S_2) = L$, and so $L(G \,{}_{\Sigma}\|_{\Sigma_c}\, S) = L$, where $S := S_1 \,{}_{A_1}\|_{A_2}\, S_2$. In other words, if a generator of $inf\overline{PC_{\Sigma_{ui}}O_{M_i}}(K)$ is chosen as supervisor S_i, then closed-loop behavior in the PSC-setting need not be K, unlike the conjunctive setting.

Not only the prior synthesis method is not applicable in the PSC setting, the conventional automaton model itself is not suitable for the PSC-based supervisor synthesis. In the PSC-based decentralized control, a locally disabled event may be enabled globally, and thus can be executed by the plant. If that event is locally observable, a supervisor may need to update its state when it is executed by the plant, but that will not be possible in an usual automaton model due to absence of any transition on a disabled event. To accommodate this new type of requirement, a more general automaton, called the *command-response* automaton, is introduced for the realization of PSC-based supervisors. A command-response automaton possesses two types of transitions: One the traditional type which determine the set of locally enabled events (command), and others that determine how to perform state-

updates when a locally disabled but globally enabled event is executed by the plant. Such an automaton is defined as $G^{C,R} := (X, \Sigma, \alpha^{C,R}, x_0, X_m)$, where X, Σ, x_0 and X_m are defined as before, and $\alpha^{C,R} := (\alpha^C, \alpha^R)$ includes two types of transitions. $\alpha^C : X \times \Sigma \to X$ defines the *command transitions* that determine the locally enabled events, whereas $\alpha^R : X \times \Sigma \to X$ defines the *response transitions* that determine the state updates on locally disabled but globally enabled events. Without loss of generality, the response transition function α^R is assumed to be a complete function. (If for some $x \in X$ and $\sigma \in \Sigma$, $\alpha^R(x, \sigma)$ is not defined, then a self-loop transition $\alpha^R(x, \sigma) = x$ can be added for the completeness.) Automaton $G^C = (X, \Sigma, \alpha^C, x_0, X_m)$ is called the basic automaton model of the command-response automaton model $G^{C,R} = (X, \Sigma, \alpha^{C,R}, x_0, X_m)$.

We extend the definition of PSC given in Definition 2 to cover command-response automata models as follows:

Definition 18 Given $G_i^{C,R} = (X_i, \Sigma, \alpha_i^{C,R}, x_{i,0}, X_{i,m})$ with $\alpha_i^{C,R} = (\alpha_i^C, \alpha_i^R)$, let $A_i \subseteq \Sigma$ denote the priority event sets of $G_i^{C,R}$ $(i \in I)$. The prioritized synchronous composition of $G_1^{C,R}$ and $G_2^{C,R}$ is defined as $G_1^{C,R}\ {}_{A_1}\|_{A_2}\ G_2^{C,R} := (X, \Sigma, \alpha^{C,R}, x_0, X_m)$, where $X := X_1 \times X_2$, $x_0 := (x_{1,0}, x_{2,0})$, $X_m := X_{1,m} \times X_{2,m}$ and the transition function $\alpha^{C,R} := (\alpha^C, \alpha^R)$ with α^C and α^R being defined as: $\forall x = (x_1, x_2) \in X, \sigma \in \Sigma$:

$$\alpha^C(x,\sigma) := \begin{cases} (\alpha_1^C(x_1,\sigma), \alpha_2^C(x_2,\sigma)) & \text{if } \alpha_1^C(x_1,\sigma) \text{ defined}, \alpha_2^C(x_2,\sigma) \text{ defined} \\ (\alpha_1^C(x_1,\sigma), \alpha_2^R(x_2,\sigma)) & \text{if } \alpha_1^C(x_1,\sigma) \text{ defined}, \alpha_2^C(x_2,\sigma) \text{ undefined}, \sigma \notin A_2 \\ (\alpha_1^R(x_1,\sigma), \alpha_2^C(x_2,\sigma)) & \text{if } \alpha_1^C(x_1,\sigma) \text{ undefined}, \alpha_2^C(x_2,\sigma) \text{ defined}, \sigma \notin A_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\alpha^R(x,\sigma) := (\alpha_1^R(x_1,\sigma), \alpha_2^R(x_2,\sigma)).$$

Definition 18 indicates that in the generalized PSC, an event is enabled if and only if it is enabled by all systems having priority over it. For a locally disabled and globally enabled event, a supervisor performs a state-update by taking the corresponding response transition. Definition 2 can be viewed as a special case of Definition 18 with all response transitions

appearing only as self-loops (since no state-update is possible in case a locally disabled but globally enabled event is executed by the plant).

Remark 20 The idea of having two types of transitions in an automaton model is not new, and first appeared in [64], where a system can contain two types of transitions, the *real transitions* and the *virtual transitions*. However, the distinction between the real and virtual transitions considered in [64] is not the same as the distinction between the command and response transitions.

Now we are ready to discuss the synthesis of PSC-based supervisors. Rewrite the local PSC-based control policy as: $\forall s \in L$

$$f_i(M_i(s)) = \{\sigma \in A_i \mid \{[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap L\} \cap pr(K) \neq \emptyset\}$$

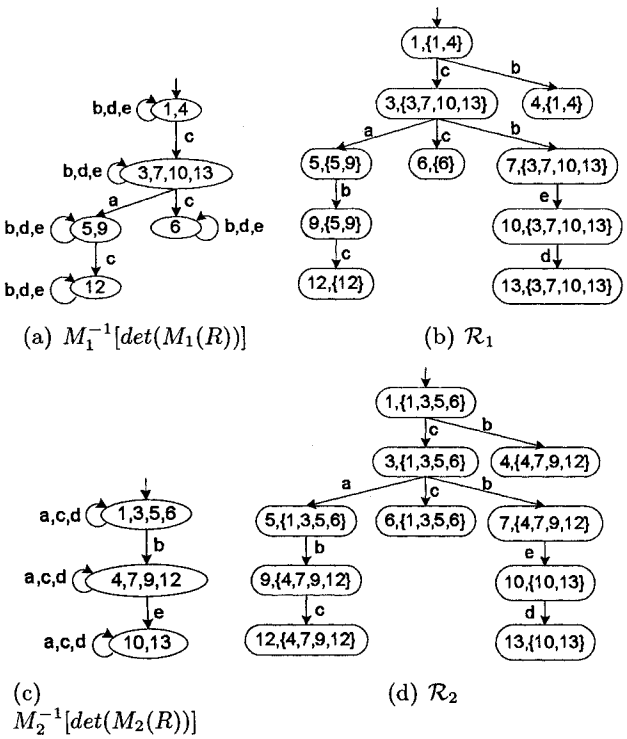
$$\cup\{\sigma \in \Sigma_{ci} - A_i \mid [M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap L \subseteq pr(K)\}.$$

To find an automaton representation of this control policy, we need to identify all traces in $pr(K)$ indistinguishable to s, i.e., $M_i^{-1}M_i(s) \cap pr(K)$. Given a specification model $R = (Y, \Sigma, \beta, y_0, Y_m)$ with $L(R) = pr(K)$, define the *refined specification model with respect to* M_i as $\mathcal{R}_i := R \| M_i^{-1}[det(M_i(R))]$. $\mathcal{R}_i$ generates the language $M_i^{-1}M_i(pr(K)) \cap pr(K) = pr(K) = L(R)$. The following algorithm, which is based on that presented in [27], describes how to construct the automaton $\mathcal{R}_i$.

Algorithm 5 Consider a specification model $R = (Y, \Sigma, \beta, y_0, Y_m)$.

1. Construct $M_i(R)$ by replacing each transition label $\sigma \in \Sigma$ by $M_i(\sigma) \in \Lambda \cup \{\epsilon\}$;
2. Construct a deterministic automaton $det(M_i(R))$ with its state set 2^Y;
3. Construct $M_i^{-1}[det(M_i(R))]$ by replacing each transition $\lambda \in \Lambda$ by events in $M_i^{-1}(\lambda) = \{\sigma \in \Sigma | M_i(\sigma) = \lambda\}$, and adding self-loops at each state on events in $M^{-1}(\epsilon) = \{\sigma \in \Sigma | M_i(\sigma) = \epsilon\}$. Note that $M_i^{-1}[det(M_i(R))]$ and $det(M_i(R)$ have the same state space;

4. Construct $\mathcal{R}_i = R \| M_i^{-1}[det(M_i(R))] = (Z, \Sigma, \gamma_i, z_0, Z_m)$ through the synchronous composition of R and $M_i^{-1}[det(M_i(R))]$.



(a) $M_1^{-1}[det(M_1(R))]$ (b) $\mathcal{R}_1$

(c) $M_2^{-1}[det(M_2(R))]$ (d) $\mathcal{R}_2$

Figure 5.4 Diagrams illustrating Example 15

The result of Algorithm 5 is an automaton with its state set $Z = Y \times 2^Y$ and marked state set $Z_m = Y_m \times 2^Y$. Note that in the automaton $M_i^{-1}[det(M_i(R))]$, all indistinguishable traces under mask M_i reach the same state. Consequently, the following is an immediate corollary of Algorithm 5.

Lemma 8 [27] Suppose for $s_1, s_2 \in pr(K) = L(\mathcal{R}_i)$ $z_1 = (y_1, \hat{y_1}) = \gamma_i(z_0, s_1)$ and $z_2 = (y_2, \hat{y_2}) = \gamma_i(z_0, s_2)$. If $M_i(s_1) = M_i(s_2)$, then $\hat{y_1} = \hat{y_2}$. (Such states z_1 and z_2 with $\hat{y_1} = \hat{y_2}$ are called *matching pair of states.*)

Algorithm 5 is illustrated through the following example.

Example 15 Consider the specification R shown in Example 12. Figure 5.4 (a) and (c) show the automata $M_1^{-1}[det(M_1(R))]$ and $M_2^{-1}[det(M_2(R))]$ respectively, where each state of

them contains all states of R which are matched to each other. Synchronously composing the specification R with those automata results in the refined specification models $\mathcal{R}_1$ and $\mathcal{R}_2$, shown in Figure 5.4 (b) and (d) respectively.

In the refined specification model $\mathcal{R}_i$ constructed in Algorithm 5, all states reached by indistinguishable traces are labeled with matched set of states. In the following algorithm, we first construct a *refined plant model with respect to the refined specification*, denoted $\mathcal{G}_i := G \| \overline{\mathcal{R}_i}$, where $\overline{\mathcal{R}_i}$ is obtained by "completing" the transition function of $\mathcal{R}_i$ by introducing a dump state "d" that is unmarked. Then $L(\mathcal{G}_i) = L(G) \cap L(\overline{\mathcal{R}_i}) = L(G) \cap \Sigma^* = L(G)$, and the states reached in $\mathcal{G}_i$ by illegal traces have their second coordinate labeled with the dump state "d". In order to construct the ith local supervisor, all the states reached in $\mathcal{G}_i$ on legal traces is partitioned into different classes based on the labels of matching states. Those equivalence classes form the states of the ith local supervisor since matching states are reached by indistinguishable traces and so the local supervisor cannot distinguish between them. Each local PSC-based supervisor is constructed using a command-response automaton $S_i^{C,R}$. The command transitions in $S_i^{C,R}$ represent the locally enabled events, which conform to the definition of f_i, i.e., a priority event $\sigma \in A_i$ is enabled if it cannot be disabled unambiguously ($[M_i^{-1} M_i(s) \cap pr(K)]\sigma \cap pr(K) \neq \emptyset$), and a non-priority event $\sigma \in \Sigma_{ci} - A_i$ is enabled if it can be enabled unambiguously ($[M_i^{-1} M_i(s) \cap pr(K)]\sigma \cap L \subseteq pr(K)$). The response transitions in $S_i^{C,R}$ represent the non-priority events disabled but observable locally. The global PSC-based supervisor is constructed by taking the PSC composition of local PSC-based supervisors.

Algorithm 6 Given a plant $G = (X, \Sigma, \alpha, x_0, X_m)$ and a specification $R = (Y, \Sigma, \beta, y_0, Y_m)$, consider two supervisors with observation mask M_i and priority set A_i ($i \in I = \{1, 2\}$). A finite automaton representing the local PSC-based control policy f_i is constructed as follows:

1. Construct the refined specification model $\mathcal{R}_i = (Z, \Sigma, \gamma_i, z_0, Z_m)$ using Algorithm 5;

2. Construct the "completed" model of $\mathcal{R}_i$, denoted $\overline{\mathcal{R}_i} := (\overline{Z}, \Sigma, \overline{\gamma}_i, z_0, Z_m)$, where $\overline{Z} =$

$Z \cup \{d\}$ and $d \notin Z$ is a dump state. The transition function $\overline{\gamma}_i$ is defined as follows:

$$\forall \overline{z} \in \overline{Z}, \sigma \in \Sigma : \overline{\gamma}_i(\overline{z}, \sigma) := \begin{cases} \gamma_i(\overline{z}, \sigma) & \text{if } [\overline{z} \neq d] \wedge [\gamma_i(\overline{z}, \sigma) \text{ defined}], \\ d & \text{otherwise.} \end{cases}$$

I.e., for any event $\sigma \in \Sigma$ not defined at state $z \in Z$, a transition on σ from the state z to the dump state d is added in $\overline{\mathcal{R}}_i$. Also, any $\sigma \in \Sigma$ appears as a self-loop at the dump state d;

3. Construct the *refined plant model with respect to refined specification*, denoted $\mathcal{G}_i := G \| \overline{\mathcal{R}}_i = (\hat{X}, \Sigma, \hat{\alpha}, \hat{x}_0, \hat{X}_m)$, where $\hat{X} := X \times \overline{Z}$, $\hat{x}_0 := (x_0, z_0)$, $\hat{X}_m := X_m \times Z_m$ and the transition function $\hat{\alpha}$ is defined as follows:

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

4. Construct the local supervisor S_i representing the local PSC-based control policy f_i:

   (a) First partition the state set of $\mathcal{G}_i$ into a set of classes. A class consists either of a set of legal states that are matched to each other, or the set of illegal states. The class consisting of the initial state of $\mathcal{G}_i$ is called the *initial class*, denoted C_0, and the class consisting of illegal states is called the *dump class*, denoted C_d. The set of all classes is denoted $\mathcal{C}_i$;

   (b) Construct local supervisor $S_i^{C,R} := (\mathcal{C}_i, \Sigma, \rho_i^{C,R}, C_0, \mathcal{C}_i)$, where the transition function $\rho_i^{C,R} := (\rho_i^C, \rho_i^R)$ is defined by considering transitions in $\mathcal{G}_i$ as follows.

   - If a priority event $\sigma \in A_i$ has a transition from a state in class C to a state in class C' $(C' \neq C_d)$, then $\rho_i^C(C, \sigma) = C'$;
   - If a non-priority controllable event $\sigma \in \Sigma_{ci} - A_i$ has a transition from a state in class C to a state in class C' $(C' \neq C_d)$, and there is no transition on σ from any state in class C to a state in the dump class C_d, then $\rho_i^C(C, \sigma) = C'$;

- If a locally uncontrollable event $\sigma \in \Sigma_{ui} = \Sigma - \Sigma_{ci}$ has a transition from a state in class C to a state in class C' ($C' \neq C_d$), then $\rho_i^C(C, \sigma) = C'$; (Note that controllability guarantees there is no uncontrollable event with a transition from a state in class C to a state in the dump class C_d.)
- If a non-priority controllable event $\sigma \in \Sigma_{ci} - A_i$ has a transition from a state in class C to a state in class C' ($C' \neq C_d$), and also has a transition from a state in class C to a state in the dump class C_d, then $\rho_i^R(C, \sigma) = C'$.

Formally, the transition functions ρ_i^C and ρ_i^R are defined as follows: $\forall C \in \mathcal{C}_i, \sigma \in \Sigma$,

$$\rho_i^C(C,\sigma) := \begin{cases} C' & \text{if } [C' \neq C_d] \wedge [\hat{\alpha}(C,\sigma) \cap C' \neq \emptyset] \wedge [(\sigma \in A \cup \Sigma_{ui}) \vee \\ & ((\sigma \in \Sigma_{ci} - A_i) \wedge (\hat{\alpha}(C,\sigma) \cap C_d = \emptyset))] \\ \text{undefined} & \text{otherwise,} \end{cases}$$

$$\rho_i^R(C,\sigma) := \begin{cases} C' & \text{if } [C' \neq C_d] \wedge [\hat{\alpha}(C,\sigma) \cap C' \neq \emptyset] \\ & \wedge [(\sigma \in \Sigma_{ci} - A_i) \wedge (\hat{\alpha}(C,\sigma) \cap C_d \neq \emptyset)] \\ C & \text{otherwise.} \end{cases}$$

A command-response automaton representing the global PSC-based control policy f_{PSC} is constructed through the PSC composition, namely, $S_{PSC}^{C,R} = S_1^{C,R} \,{}_{A_1}\|_{A_2}\, S_2^{C,R}$. Since the interaction between the plant and the global PSC-based supervisor is captured by the basic PSC composition, we only need the basic automaton model of the global PSC-based supervisor for the control purpose. By deleting all response transitions and computing the trim version of the resulting automaton, we can obtain the corresponding basic automaton model of f_{PSC}, denoted S_{PSC}.

Remark 21 Algorithm 6 is also applicable to synthesize supervisors in centralized, conjunctive decentralized, and non-conjunctive decentralized settings. In the centralized or conjunctive decentralized setting, a response transition will never be executed since a locally disabled event is also globally disabled. And so in centralized or conjunctive decentralize setting, the response transitions may be omitted without affecting the overall closed-loop behavior. As a

result in those settings, the supervisors can be realized as conventional automata.

Remark 22 The complexity of Algorithm 6 is analyzed as follows. The complexity of Algorithm 5 to construct the refined specification model $\mathcal{R}_i$ is $\mathcal{O}(|Y| \times 2^{|Y|} \times |\Sigma|)$. In Algorithm 6, the construction of local PSC-based supervisor $S_i^{C,R}$ is based on the refined plant model $\mathcal{G}_i$, whose construction has a complexity of $\mathcal{O}(|X| \times |Y| \times 2^{|Y|} \times |\Sigma|)$. The complexity of doing the partitioning and defining the transitions in step 4 of Algorithm 6 is linear in the number of states and the number of transitions of $\mathcal{G}_i$. Thus, the overall complexity to construct each local PSC-based supervisor is $\mathcal{O}(|X| \times |Y| \times 2^{|Y|} \times |\Sigma|)$. Note from the work in [69] it is known that *a polynomial complexity algorithm for synthesis of supervisors does not exist* (unless P=NP). (The NP-completeness of supervisor synthesis reported in [69] is for the centralized setting but is also applicable to the decentralized setting, the centralized control being an instance of the decentralized control.) In contrast, the observer based synthesis procedure sketched in [75] has complexity $\mathcal{O}(2^{|X|\times|Y|} \times |\Sigma|)$ for the size of the observer, whereas the algorithmic description of how to compute the control actions for each of the observer states is not given, and so any extra associated complexity is not quantifiable. When the specification R is a subgraph of the plant G, then since the second coordinate of a state in $\mathcal{G}_i = G||\overline{R||M_i^{-1}[det(M_i(R))]}$ is either the same as the first coordinate or is the dump state d, the complexity of constructing the local supervisors is reduced to $\mathcal{O}(|X| \times 2^{|Y|} \times |\Sigma|)$.

To illustrate Algorithm 6, we construct the PSC-based supervisors for plant G and specification R introduced in Example 12 as follows.

Example 16 Consider plant G and specification R introduced in Example 12. Based on the refined specification models $\mathcal{R}_1$ and $\mathcal{R}_2$ constructed in Example 15, Figure 5.5 shows the supervisor synthesis procedure. The complete refined specification models $\mathcal{R}_1$ and $\mathcal{R}_2$ are shown in Figure 5.5 (a) and (d), respectively. And the refined plant model $\mathcal{G}_1$ and $\mathcal{G}_2$ are shown in Figure 5.5 (b) and (e), respectively, where states with second coordinate d are unreachable through the legal traces of the specification R. Each non-dump class in $\mathcal{G}_1$ and $\mathcal{G}_2$ is surrounded by a dotted spline curve. Following the construction described in step 4(b)

of Algorithm 6, we construct the local PSC-based supervisors $S_1^{C,R}$ and $S_2^{C,R}$ as shown in Figure 5.5 (c) and (f) respectively. Note all command transitions are represented by solid lines, and all response transitions are represented by dotted lines. For the global PSC-based supervisor, its extended automaton model and basic automaton model are shown in Figure 5.5 (g) and (h), respectively. It can be verified that the controlled system under S_{PSC} achieves the specification R, i.e., $G \,{}_{\Sigma}\|_{\Sigma_c}\, S_{PSC} = R$.

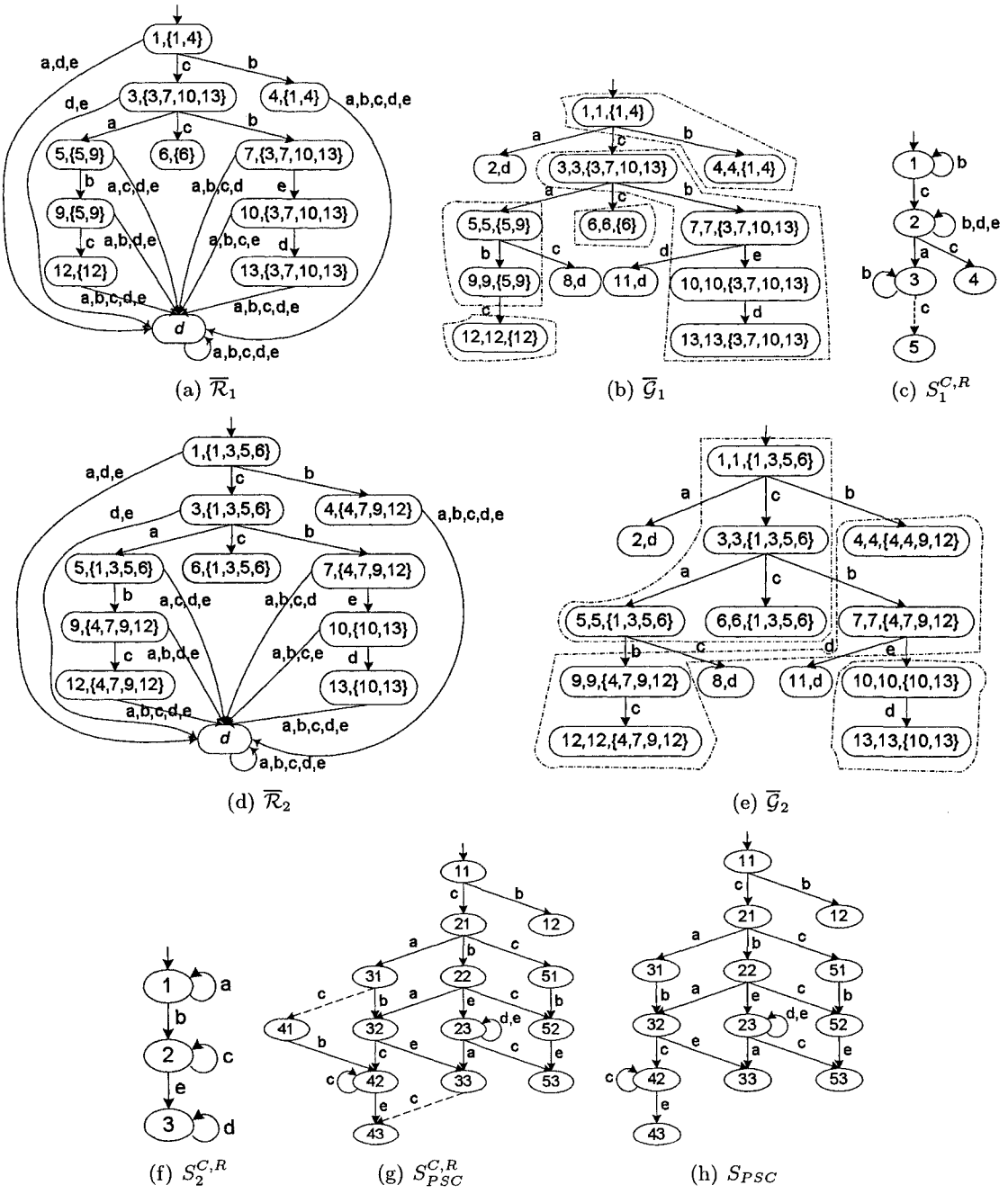
The following lemma establishes the relationship between the local PSC-based control policy and the local PSC-based supervisor represented by a command-response automaton.

Lemma 9 Consider a plant G with generated language L, and a specification R with $L(R) = pr(K)$. Let f_i be the local PSC-based control policy defined in Equation (5.2), and $S_i^{C,R} = (\mathcal{C}_i, \Sigma, \rho_i^{C,R}, C_0, \mathcal{C}_i)$ be the supervisor constructed in Algorithm 6. For all $s \in pr(K)$ let C_s be the non-dump class reached in $S_i^{C,R}$.

$$\forall \sigma \in \Sigma_{ci} : [\sigma \in f_i(M_i(s))] \Leftrightarrow [\rho_i^C(C_s, \sigma) \text{ is defined}].$$

Proof: (Only if) If $\sigma \in f_i(M_i(s))$ for $\sigma \in A_i$, then it follows from the definition of f_i that $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap pr(K) \neq \emptyset$. Thus there exists t such that $M_i(s) = M_i(t)$ and $t\sigma \in pr(K)$. So $t\sigma$ (as well as t) reaches a non-dump state in the completed specification model $\overline{\mathcal{R}}_i$. Then the state reached by executing $t\sigma$ (as well as t) in $S_i^{C,R}$ is contained in a non-dump class. Let C_t denote the non-dump class reached by t in $S_i^{C,R}$. Then since $t\sigma$ also reaches a non-dump class in $S_i^{C,R}$ and $\sigma \in A_i$, it follows from definition of $S_i^{C,R}$ that $\rho_i^C(C_t, \sigma)$ is defined. Since $M_i(s) = M_i(t)$, it follows from Lemma 8 that s and t lead to a pair of matching states in $\mathcal{R}_i$, and thus to states contained in the same non-dump class $C_s = C_t$ in $S_i^{C,R}$. Further $s\sigma$ being indistinguishable to $t\sigma$, $\rho_i^C(C_s, \sigma) = \rho_i^C(C_t, \sigma)$, implying that $\rho_i^C(C_s, \sigma)$ is defined.

For $\sigma \in \Sigma_{ci} - A_i$, it follows from the definition of f_i that $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap L \subseteq pr(K)$. I.e., $s\sigma \in pr(K)$, and for all $t \in pr(K)$ with $M_i(t) = M_i(s)$ and $t\sigma \in L$, we have $t\sigma \in pr(K)$. Using a similar argument as that for $\sigma \in A_i$, we can verify that all these traces do not reach any state in the dump class. Consequently $\rho_i^C(C_s, \sigma)$ is defined. This completes the proof of



(a) $\overline{\mathcal{R}}_1$ (b) $\overline{\mathcal{G}}_1$ (c) $S_1^{C,R}$

(d) $\overline{\mathcal{R}}_2$ (e) $\overline{\mathcal{G}}_2$

(f) $S_2^{C,R}$ (g) $S_{PSC}^{C,R}$ (h) S_{PSC}

Figure 5.5 Diagrams illustrating Example 16

the necessary condition.

(If) If $\rho_i^C(C_s, \sigma)$ is defined for $\sigma \in A_i$, then according to the definition of ρ_i^C, there exists a state $C \in \mathcal{C}_i$ such that $\rho_i^C(C_s, \sigma) = C$. It indicates that there exists a trace $t \in pr(K)$ (t may be s) such that $M_i(t) = M_i(s)$ and the dump state d is not reachable by executing $t\sigma$ in the completed specification model $\overline{\mathcal{R}}_i$, i.e., $t\sigma \in L(\mathcal{R}_i) = L(R) = pr(K)$. Thus $M_i^{-1}M_i(s)\sigma \cap pr(K) \neq \emptyset$, and then $\sigma \in f_i(M_i(s))$.

For $\sigma \in \Sigma_{ci} - A_i$, if $\rho_i^C(C_s, \sigma)$ is defined, then for all $t \in pr(K)$ with $M_i(t) = M_i(s)$, $\hat{\alpha}(x, t\sigma) \notin C_d$. Thus all reachable states by executing t in $\overline{\mathcal{R}}_i$ are non-dump states. It follows that $t\sigma \in L(\mathcal{R}_i) = L(R) = pr(K)$, which implies that $[M_i^{-1}M_i(s) \cap pr(K)]\sigma \cap L \subseteq pr(K)$. Then, $\sigma \in f_i(M_i(s))$. This completes the proof of the sufficiency condition and concludes the proof. ■

Based on Lemma 9, we show in the following theorem that the PSC-based control policy f_{PSC} and the PSC-based supervisor S_{PSC} constructed in Algorithm 6 are equivalent in the sense that they achieve the same controlled behaviors.

Theorem 12 Consider a plant G with generated language L and marked language L_m.

1. For nonempty $K \subseteq L$, $L/f_{PSC} = K$ if and only if $L(G \,_{\Sigma}\|_{\Sigma_c} S_{PSC}) = K$;
2. For nonempty $K \subseteq L_m$, $L_m/f_{PSC} = K$ and $pr(L_m/f_{PSC}) = L/f_{PSC}$ if and only if $L_m(G \,_{\Sigma}\|_{\Sigma_c} S_{PSC}) = K$ and $pr(L_m(G \,_{\Sigma}\|_{\Sigma_c} S_{PSC})) = L(G \,_{\Sigma}\|_{\Sigma_c} S_{PSC})$.

Proof: 1. We prove the first assertion by showing that a trace is enabled in L/f_{PSC} if and only if that trace is enabled in $L(G \,_{\Sigma}\|_{\Sigma_c} S_{PSC})$. This is done by induction on the length of traces in the languages L/f_{PSC} and $L(G \,_{\Sigma}\|_{\Sigma_c} S_{PSC})$. It is trivial that $\epsilon \in L/f_{PSC}$ and $\epsilon \in L(G \,_{\Sigma}\|_{\Sigma_c} S_{PSC})$. Assume that for $|s| \leq n$, $s \in L/f_{PSC}$ if and only if $s \in L(G \,_{\Sigma}\|_{\Sigma_c} S_{PSC})$. We need to show that for event $\sigma \in \Sigma$, $s\sigma \in L/f_{PSC}$ if and only if $s\sigma \in L(G \,_{\Sigma}\|_{\Sigma_c} S_{PSC})$, where $|s| = n$.

(Only if) If $s\sigma \in L/f_{PSC}$, it follows from Definition 16 that $s\sigma \in L$ and $\sigma \in \Sigma_u \cup f_{PSC}(s)$. If $\sigma \in \Sigma_u$, since the priority set of S_{PSC} is Σ_c, according to the definition of PSC, $s\sigma \in L(G \,_{\Sigma}\|_{\Sigma_c} S_{PSC})$.

If $\sigma \in f_{PSC}(s)$, then $\sigma \in [\underset{i \in I_A(\sigma)}{\cap} f_i(M_i(s))] \cap A$, or $\sigma \in [\underset{i \in I_c(\sigma)}{\cup} f_i(M_i(s))] \cap [\Sigma_c - A]$. In the first case, $\sigma \in f_i(M_i(s))$ for all $i \in I_A(\sigma)$. It follows from Lemma 9 that $\rho_i^C(C_s, \sigma)$ is defined in all $S_i^{C,R}$ $(i \in I_A(\sigma))$, and thus $s\sigma$ is defined in S_{PSC}. Since $s\sigma \in L$, we have that $s\sigma \in L(G \,{}_{\Sigma}\|_{\Sigma_c} S_{PSC})$. In the second case, there exists $i \in I_c(\sigma)$ such that $\sigma \in f_i(M_i(s))$. From Lemma 9, $\rho_i^C(C_s, \sigma)$ is defined in at least one supervisor $S_i^{C,R}$ $(i \in I_c(\sigma))$, and thus $s\sigma$ is defined in S_{PSC}. Since $s\sigma \in L$, it follows that $s\sigma \in L(G \,{}_{\Sigma}\|_{\Sigma_c} S_{PSC})$.

(If) If $s\sigma \in L(G \,{}_{\Sigma}\|_{\Sigma_c} S_{PSC})$, then from the definition of PSC, we have that $s\sigma \in L$ and either $\sigma \in \Sigma_u$ or it is enabled by S_{PSC} at the state reached by executing s. Using the similar argument as above, it can be verified that $\sigma \in f_{PSC}$.

2. Based on the first assertion, the second assertion readily follows from the definitions of marked language and L_m/f_{PSC}. ■

5.4 Illustrative Example

In this section, we present a simple manufacturing example to illustrate PSC-based decentralized control. Figure 5.6 shows a simple manufacturing system, which contains three machines, two supervisors, two sensors (observation masks), one input port and one output port. Machine M_{n0} is a pre-processing machine, which pre-processes workpieces before they are fed into machine M_{n1} and machine M_{n2}. Machines M_{n1} and M_{n2} are major processing machines producing two different types of products. We use the following event labels to represent the actions that can occur in this system.

u: a workpiece is taken by M_{n0} from the input port;

a_1: a workpiece is sent from M_{n0} to M_{n1};

a_2: a workpiece is sent from M_{n0} to M_{n2};

b_1: a product is taken away from M_{n1} to the output port;

b_2: a product is taken away from M_{n2} to the output port.

Both supervisors control the operations of three machines using short-range wireless communication. Event u is under the control of both supervisors, and machine M_{n0} takes a workpiece from the input port if either supervisor issues that command. However, only the

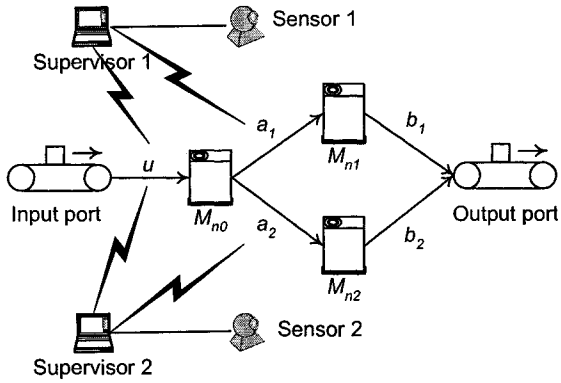


Figure 5.6 A simple manufacturing system

first (resp., second) supervisor has the *authority* to command machine M_{n1} (resp., M_{n2}) to take a workpiece from machine M_{n0}. The output actions from both machine M_{n1} and machine M_{n2} are beyond the control of both supervisors. I.e.,

$$\Sigma_{c1} = \{u, a_1\}, \Sigma_{u1} = \{a_2, b_1, b_2\}; \Sigma_{c2} = \{u, a_2\}, \Sigma_{u2} = \{a_1, b_1, b_2\}.$$

Also due to the authority requirement of control, $A_1 = \{a_1\}$, and $A_2 = \{a_2\}$, and the four PSC-induced partitions are

$$\Sigma_\wedge = \emptyset, \Sigma_{A_1} = \{a_1\}, \Sigma_{A_2} = \{a_2\}, \Sigma_\vee = \{u\}.$$

Supervisor 1 and supervisor 2 obtain information about system behaviors through sensor 1 and sensor 2, respectively. Sensor 1 (resp., sensor 2) can observe the events u, a_1, b_1 (resp., u, a_2, b_2), i.e.,

$$M_1(u) = u, M_1(a_1) = a_1, M_1(b_1) = b_1, M_1(a_2) = M_1(b_2) = \epsilon,$$
$$M_2(u) = u, M_2(a_2) = a_2, M_2(b_2) = b_2, M_2(a_1) = M_2(b_1) = \epsilon.$$

Each machine has two states, "I" (Idle), and "W" (Working). Machines M_{n0}, M_{n1} and M_{n2} are modeled by automata G_1, G_2 and G_3, respectively. The model for the whole system is the synchronous composition of the three machine models, i.e., $G = G_1 \| G_2 \| G_3$. All these system models are shown in Figure 5.7.

Due to the restriction of shared resources, only one machine is allowed to operate at one time. It is also required by the manufacturing procedure that M_{n1} and M_{n2} operate alternately, and each time M_{n1} should start to work first. The corresponding specification model R is also shown in Figure 5.7.

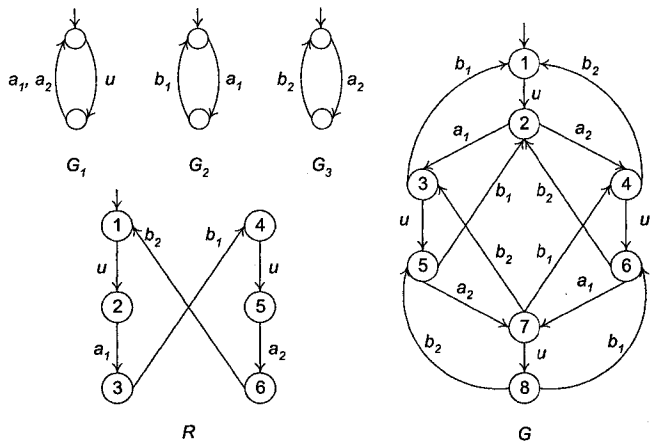


Figure 5.7 Plant and specification

Comparing the plant and specification models, we know that the desired behavior of the system should repeatedly traverse states 1, 2, 3 and 4 of the plant in the following sequence: $1 \to 2 \to 3 \to 1 \to 2 \to 4 \to \cdots$. It is required that events a_2 and a_1 be disabled at state 2 alternately, and event u be constantly enabled at state 1 and disabled at states 3 and 4. Using the testing methods introduced in Section 5.2, one can verify that the specification language $L(R)$ is $(L(G), \Sigma_{ci}, M_i, A_i)$-PSC-coobservable.

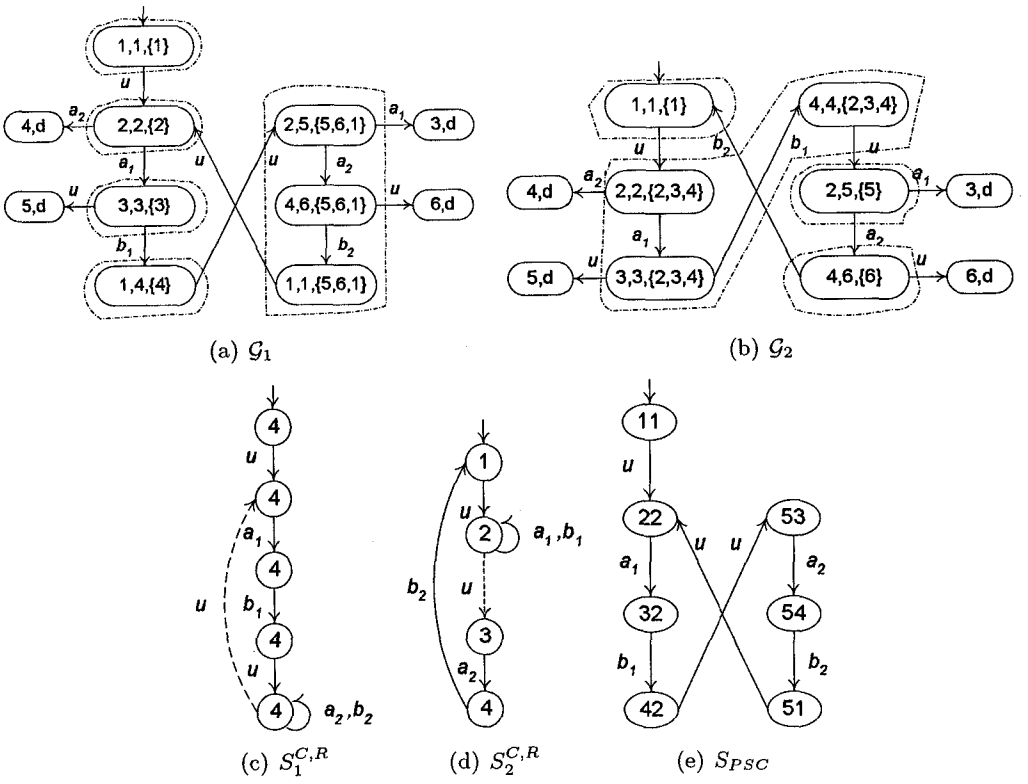
We construct the PSC-based supervisors for this system in the following. Using Algorithm 6, the refined plant models $\mathcal{G}_i$ $(i \in I = \{1, 2\})$ are constructed in Figure 5.8, where each non-dump class is surrounded by a closed dotted spline curve. Note that we only draw those dump states which are reachable within one transition from non-dump states since transitions within the dump class do not effect the supervisor synthesis result. Then, the local PSC-based supervisors $S_i^{C,R}$ are constructed from $\mathcal{G}_i$, where the response transitions are represented by dotted lines.

Using the two local supervisors, we can control the plant to achieve the specification as follows. At the initial state, both supervisors enable event u. Then the first supervisor $S_1^{C,R}$

sequentially enables events a_1, b_1 and u without ambiguity. During these steps, the second supervisor $S_2^{C,R}$ cannot observe events a_1 and b_1, and thus cannot make unambiguous decision for event u, which needs to be disabled at state 3 and enabled at state 1. Thus, $S_2^{C,R}$ takes the default action to disable event u. However, since the non-priority event u may be enabled by $S_1^{C,R}$, $S_2^{C,R}$ needs the capability to track event u, which justifies the response transition labeled by u in $S_2^{C,R}$. According to the above control commands, the plant executes the trace ua_1b_1u and traverses states in the sequence of $1 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 2$. Then, supervisor $S_2^{C,R}$ sequentially enables events a_2, b_2 and u, and supervisor $S_1^{C,R}$ tracks event u during this phase through a response transition. This way the controlled plant generates the specification language $pr((ua_1b_1ua_2b_2)^*)$. The existence of PSC-based local supervisors that enforce the specification also establishes its PSC-coobservability (and controllability).

The global PSC-based supervisor is constructed by taking PSC composition of $S_1^{C,R}$ and $S_2^{C,R}$, i.e., $S_{PSC}^{C,R} = S_1^{C,R} \,{}_{A_1}\|_{A_2}\, S_2^{C,R}$. Since all transitions in $S_{PSC}^{C,R}$ are command transitions, $S_{PSC} = S_{PSC}^{C,R}$. It is easy to verify that $L(G \,{}_{\Sigma}\|_{\Sigma_c}\, S_{PSC}) = L(R)$.

As a final remark, owing to the presence of control-authority requirements in this application, this decentralized control problem cannot be cast in the conjunctive+disjunctive setting as it does not have the mechanism to capture the notion of control-authority. This fact was formally captured as the second part of Corollary 3.



(a) $\mathcal{G}_1$ (b) $\mathcal{G}_2$

(c) $S_1^{C,R}$ (d) $S_2^{C,R}$ (e) S_{PSC}

Figure 5.8 Synthesis of PSC-based supervisors

CHAPTER 6. NONDETERMINISTIC DECENTRALIZED CONTROL

In this chapter, we study the decentralized supervisory problem using nondeterministic supervisors. Nondeterministic supervisory control under partial observation was first proposed in [31] by allowing nondeterminism in control policies, which lead to the existence condition of *achievability* that is weaker than controllability and observability combined and is algebraically better behaved. In this chapter, we extend nondeterministic control to the decentralized setting. Using the simple conjunctive rule of decision fusion, we present a definition of *coachievability* to characterize the languages which can be supervisors. Coachievability is weaker than controllability and co-observability combined, which serve as the necessary and control. Other properties of coachievable languages are presented, and the relationship between achievability and coachievability is established. We derive necessary and sufficient conditions for target and range control problems. The existence conditions are polynomially verifiable, and also local supervisors can be polynomially synthesized. supervisors can be polynomially synthesized.

6.1 Centralized Nondeterministic Control

In this section, we first present results for centralized nondeterministic supervisory control, which are needed for the development of decentralized nondeterministic control.

Definition 19 [31] Let $S = (Y, \Sigma, \beta, Y_0)$ be a nondeterministic state machine, $\Sigma_u \subseteq \Sigma$ be the set of uncontrollable events, and $M : \overline{\Sigma} \to \overline{\Lambda}$ be the observation mask.

- S is called Σ_u-compatible if $\forall y \in Y, \forall \sigma \in \Sigma_u$, $\beta(y, \sigma) \neq \emptyset$.
- S is called M-compatible if $\forall y \in Y, \forall \sigma, \sigma' \in \overline{\Sigma}$, $M(\sigma) = M(\sigma')$ and $\beta(y, \sigma), \beta(y, \sigma') \neq \emptyset$ implies $\beta(y, \sigma) = \beta(y, \sigma')$.

- S is called (Σ_u, M)-compatible if S is Σ_u-compatible and M-compatible.

Each nondeterministic supervisor needs to be (Σ_u, M)-compatible with respect to its own uncontrollable event set Σ_u and observation mask M. The notion of (Σ_u, M)-compatibility captures the fact that no uncontrollable events can be disabled at any of the states of a supervisor, and further the state update on any pair of indistinguishable events should be identical. The notion of (Σ_u, M)-achievability with respect to Σ^* is defined as follows.

Definition 20 [31] Let $K \subseteq \Sigma^*$, then K is said to be (Σ_u, M)-achievable with respect to Σ^* if

1. **Controllability/Achievability-1** (C): $s \in pr(K) \Rightarrow s\Sigma_u^* \subseteq pr(K)$,
2. **Recognizability/Achievability-2** (R): $sat \in pr(K), M(a) = \epsilon \Rightarrow sa^*t \subseteq pr(K)$,
3. **Achievability-3** (A): $sat \in pr(K) \Rightarrow s[M^{-1}M(a) \cap \Sigma_u^*]t \subseteq pr(K)$.

(Σ_u, M)-achievability characterizes languages that are generated by (Σ_u, M)-compatible nondeterministic supervisors.

Theorem 13 [31] Let $K \subseteq \Sigma^*$, then there exists a (Σ_u, M)-compatible nondeterministic state machine S_K such that $L(S_K) = pr(K)$ if and only if K is (Σ_u, M)-achievable with respect to Σ^*.

The following theorem gives the closure properties of (Σ_u, M)-achievability under union and intersection. The latter property guarantees the existence of infimal prefix-closed and (Σ_u, M)-achievable superlanguage, which is defined in Definition 21.

Theorem 14 [31] (Σ_u, M)-achievability with respect to Σ^* is preserved under union over arbitrary languages and under intersection over prefix-closed languages.

Definition 21 For $K \subseteq \Sigma^*$, the class of prefix-closed and (Σ_u, M)-achievable superlanguages of K with respect to Σ^* is defined as:

$$\overline{PA}_{\Sigma^*}(K) := \{K' \supseteq K \mid K' = pr(K'), K' \text{ is } (\Sigma_u, M)\text{-achievable wrt } \Sigma^*\}.$$

The infimal prefix-closed and (Σ_u, M)-achievable superlanguage of K with respect to Σ^* is denoted $inf\overline{PA}_{\Sigma^*}(K)$.

The following theorem follows directly from Definition 20 and Definition 21, and presents a property of (Σ_u, M)-achievable languages with respect to Σ^*.

Theorem 15 [31] For $K \subseteq \Sigma^*$, K is (Σ_u, M)-achievable with respect to Σ^* if and only if $pr(K) = inf\overline{PA}_{\Sigma^*}(K)$.

Theorem 15 can be thought of as an alternative definition of (Σ_u, M)-achievability with respect to Σ^*, and can be used to define (Σ_u, M)-achievability with respect to a plant language as follows:

Definition 22 [31] Let $K \subseteq L = pr(L)$, then K is said to be (Σ_u, M)-achievable with respect to L if $pr(K) = inf\overline{PA}_{\Sigma^*}(K) \cap L$.

The following is an algorithm of linear complexity to compute $inf\overline{PA}_{\Sigma^*}(K)$. This algorithm is used to synthesize nondeterministic supervisors.

Algorithm 7 [31] Let $S = (Y, \Sigma, \beta, Y_0)$ be a trim deterministic state machine such that $L(S) = pr(K)$. Then we have the following algorithm for the computation of $inf\overline{PA}_{\Sigma^*}(K)$.

1. Separate the states in S: for every transition (y, b, y') with either $M(b) = \epsilon$ or $\exists (y, b', y'')$ s.t. $M(b) = M(b') \neq \epsilon$, replace (y, b, y') by a pair of transitions $(y, \epsilon, \hat{y})$ and $(\hat{y}, b, y')$, where $\hat{y}$ is a newly added state.

2. For every transition (y, b, y') with $M(b) = \epsilon$, add transitions (y, b, y) and (y, ϵ, y').

3. For every state y and every event $b \in \Sigma_u \cap M^{-1}(\epsilon)$, if b is not defined at y, then add b-labeled transitions to let $\beta(y, b) = \beta(y, \epsilon)$.

4. For every state x, every event $b \in \Sigma_u \cap (\Sigma - M^{-1}(\epsilon))$, and every transition (y, a, y') with $M(a) = M(b)$, add a transition (y, b, y').

5. For every state y and every event $b \in \Sigma_u \cap (\Sigma - M^{-1}(\epsilon))$, if no b-indistinguishable event a is defined at y, then add a transition $(y, b, dump)$, where $dump$ is an added state such that $\forall \sigma \in \Sigma_u$, $\beta(dump, \sigma) = \{dump\}$.

6. Return the modified S as the generator S_K of $inf\overline{PA}_{\Sigma^*}(K)$.

6.2 Coachievability with respect to Σ^*

The notion of coachievability is defined to characterize the class of languages that can be achieved as the behavior of a pair of (Σ_{ui}, M_i)-compatible nondeterministic supervisors S_i operating in synchrony, i.e., the behavior of $S_1 \| S_2$. It follows from Theorem 13 that the language generated by each local supervisor is (Σ_{ui}, M_i)-achievable with respect to Σ^*. By considering all possible combinations of the two achievability conditions, we come up with a definition of (Σ_{ui}, M_i)-coachievability with respect to Σ^* as follows:

Definition 23 $K \subseteq \Sigma^*$ is said to be (Σ_{ui}, M_i)-coachievable with respect to Σ^* if

1. $(C_1\text{-}C_2)$ $s_1, s_2 \in pr(K) \Rightarrow \bigcap_{i=1}^{2} s_i \Sigma_{ui}^* \subseteq pr(K)$,

2. $(R_1\text{-}R_2)$ $s_1 a_1 t_1, s_2 a_2 t_2 \in pr(K), M_1(a_1) = M_2(a_2) = \epsilon \Rightarrow \bigcap_{i=1}^{2} s_i a_i^* t_i \subseteq pr(K)$,

3. $(A_1\text{-}A_2)$ $s_1 a_1 t_1, s_2 a_2 t_2 \in pr(K) \Rightarrow \bigcap_{i=1}^{2} s_i [M_i^{-1} M_i(a_i) \cap \Sigma_{ui}^*] t_i \subseteq pr(K)$,

4. a. $(C_1\text{-}R_2)$ $s_1, s_2 a_2 t_2 \in pr(K), M_2(a_2) = \epsilon \Rightarrow s_1 \Sigma_{u1}^* \cap s_2 a_2^* t_2 \subseteq pr(K)$,

   b. $(R_1\text{-}C_2)$ $s_1 a_1 t_1, s_2 \in pr(K), M_1(a_1) = \epsilon \Rightarrow s_1 a_1^* t_1 \cap s_2 \Sigma_{u2}^* \subseteq pr(K)$,

5. a. $(C_1\text{-}A_2)$ $s_1, s_2 a_2 t_2 \in pr(K) \Rightarrow s_1 \Sigma_{u1}^* \cap s_2 [M_2^{-1} M_2(a_2) \cap \Sigma_{u2}^*] t_2 \subseteq pr(K)$,

   b. $(A_1\text{-}C_2)$ $s_1 a_1 t_1, s_2 \in pr(K) \Rightarrow s_1 [M_1^{-1} M_1(a_1) \cap \Sigma_{u1}^*] t_1 \cap s_2 \Sigma_{u2}^* \subseteq pr(K)$,

6. a. $(R_1\text{-}A_2)$ $s_1 a_1 t_1, s_2 a_2 t_2 \in pr(K) \Rightarrow s_1 a_1^* t_1 \cap s_2 [M_2^{-1} M_2(a_2) \cap \Sigma_{u2}^*] t_2 \subseteq pr(K)$,

   b. $(A_1\text{-}R_2)$ $s_1 a_1 t_1, s_2 a_2 t_2 \in pr(K) \Rightarrow s_1 [M_1^{-1} M_1(a_1) \cap \Sigma_{u1}^*] t_1 \cap s_2 a_2^* t_2 \subseteq pr(K)$.

The closure properties of (Σ_{ui}, M_i)-coachievability with respect to Σ^* under intersection and union are given in the following theorem.

Theorem 16 (Σ_{ui}, M_i)-coachievability with respect to Σ^* is preserved under intersection over prefix-closed languages.

Proof: Assume $\{K_j, j \in J\}$ is a set of (Σ_{ui}, M_i)-coachievable languages with respect to Σ^*. We need to prove that $\bigcap_{j\in J} K_j$ is also (Σ_{ui}, M_i)-coachievable with respect to Σ^*, i.e., to verify that $\bigcap_{j\in J} K_j$ satisfies the conditions in Definition 23. Since each K_j is prefix-closed, $pr(\bigcap_{j\in J} K_j) = \bigcap_{j\in J} pr(K_j) = \bigcap_{j\in J} K_j$.

For condition (C_1-C_2), pick two traces $s_1, s_2 \in \bigcap_{j\in J} pr(K_j)$. Then, for all $j \in J$, $s_1, s_2 \in pr(K_j)$. Since K_j is (Σ_{ui}, M_i)-coachievable with respect to Σ^*, it follows from the definition of coachievability that $s_1\Sigma^*_{u1} \cap s_2\Sigma^*_{u2} \subseteq pr(K_j)$. Therefore, we have $s_1\Sigma^*_{u1} \cap s_2\Sigma^*_{u2} \subseteq \bigcap_{j\in J} pr(K_j)$, which implies that $\bigcap_{j\in J} K_j$ satisfies condition 1. Similarly, we can verify that other conditions in Definition 23 also hold for $\bigcap_{j\in J} K_j$. ■

The closure of (Σ_{ui}, M_i)-coachievability with respect to Σ^* under intersection over prefix-closed languages guarantees the existence of the infimal prefix-closed and (Σ_{ui}, M_i)-coachievable superlanguage with respect to Σ^*. For $K \subseteq \Sigma^*$, the class of prefix-closed and (Σ_{ui}, M_i)-coachievable superlanguage of K with respect to Σ^* is defined as:

$$\overline{PCoA}_{\Sigma^*}(K) = \{K' \supseteq K \mid K' = pr(K'), K' \text{ is } (\Sigma_{ui}, M_i)\text{-coachievable with respect to } \Sigma^*\}.$$

The infimal prefix-closed and (Σ_{ui}, M_i)-coachievable superlanguage of K with respect to Σ^* is denoted as $inf\overline{PCoA}_{\Sigma^*}(K)$. The following theorem provides a characterization of (Σ_{ui}, M_i)-coachievability with respect to Σ^*.

Theorem 17 $K \subseteq \Sigma^*$ is (Σ_{ui}, M_i)-coachievable with respect to Σ^* if and only if $pr(K) = inf\overline{PCoA}_{\Sigma^*}(K)$.

Proof: (If) If $pr(K) = inf\overline{PCoA}_{\Sigma^*}(K)$, then since $inf\overline{PCoA}_{\Sigma^*}(K)$ is (Σ_{ui}, M_i)-coachievable with respect to Σ^*, so is $pr(K)$, and hence so is K.

(Only If) Since K is (Σ_{ui}, M_i)-coachievable with respect Σ^*, so is the superlanguage $pr(K)$. Since $inf\overline{PCoA}_{\Sigma^*}(K)$ is the infimal such superlanguage, we have $pr(K) \supseteq inf\overline{PCoA}_{\Sigma^*}(K)$.

Also, it follows from the definition of $inf\overline{PCoA}_{\Sigma^*}(K)$ that $pr(K) \subseteq inf\overline{PCoA}_{\Sigma^*}(K)$. Therefore, $pr(K) = inf\overline{PCoA}_{\Sigma^*}(K)$. ■

The following lemma follows directly from Definition 20 and Definition 23, and establishes a relationship between (Σ_{ui}, M_i)-achievability with respect to Σ^* and (Σ_{ui}, M_i)-coachievability with respect to Σ^*.

Lemma 10 If $K_i \subseteq \Sigma^* (i = 1, 2)$ is prefix-closed and (Σ_{ui}, M_i)-achievable with respect to Σ^*, then $K_1 \cap K_2$ is prefix-closed and (Σ_{ui}, M_i)-coachievable with respect to Σ^*.

Proof: Since prefix-closure is preserved under intersection, $K_1 \cap K_2$ is prefix-closed. Assume $K_i \subseteq \Sigma^* (i = 1, 2)$ is (Σ_{ui}, M_i)-achievable with respect to Σ^*. Then all conditions of achievability listed in Definition 20 are held for K_i. To verify the (Σ_{ui}, M_i)-coachievability of $K_1 \cap K_2$ with respect to Σ^*, we need to check all conditions in Definition 23. Consider the first condition, $(C_1$–$C_2)$. Pick $s_1, s_2 \in pr(K_1 \cap K_2) = pr(K_1) \cap pr(K_2)$, then $s_1 \in pr(K_1)$ and $s_2 \in pr(K_2)$. It follows from the achievability of K_i that $s_1\Sigma_{u1}^* \subseteq pr(K_1)$ and $s_2\Sigma_{u2}^* \subseteq pr(K_2)$. So we have $s_1\Sigma_{u1}^* \cap s_2\Sigma_{u2}^* \subseteq pr(K_1) \cap pr(K_2) = pr(K_1 \cap K_2)$. Similarly, other conditions of Definition 23 can be verified. ■

Based on Lemma 10, we obtain the following theorem to compute $inf\overline{PCoA}_{\Sigma^*}(K)$.

Theorem 18 For $K \subseteq \Sigma^*$,

$$inf\overline{PCoA}_{\Sigma^*}(K) = inf\overline{PA}^1_{\Sigma^*}(K) \cap inf\overline{PA}^2_{\Sigma^*}(K),$$

where $inf\overline{PA}^1_{\Sigma^*}(K)$ is the prefix-closed, (Σ_{u1}, M_1)-achievable superlanguage of K with respect to Σ^*, and $inf\overline{PA}^2_{\Sigma^*}(K)$ is the prefix-closed, (Σ_{u2}, M_2)-achievable superlanguage of K with respect to Σ^*.

Proof: For notational simplicity, we define $K^1 = inf\overline{PA}^1_{\Sigma^*}(K)$, $K^2 = inf\overline{PA}^2_{\Sigma^*}(K)$ and $K^{12} = inf\overline{PCoA}_{\Sigma^*}(K)$.

$(\subseteq)$ Clearly, $K \subseteq K^i$ $(i = 1, 2)$, which implies $K \subseteq K^1 \cap K^2$. Further from Lemma 10, $K^1 \cap K^2$ is prefix-closed and (Σ_{ui}, M_i)-coachievable with respect to Σ^*. Since K^{12} is the infimal such superlanguage of K, we have that $K^{12} \subseteq K^1 \cap K^2$.

($\supseteq$) We need to show that $K^{12} \supseteq K^1 \cap K^2$. This proof is done inductively on the length of traces. Without loss of generality, $K \neq \emptyset$ (otherwise the theorem trivially holds). Since the zero length trace ϵ belongs to K^{12} as well as $K^1 \cap K^2$, we need to show that any non-zero length trace in $K^1 \cap K^2$ also belongs to K^{12}. I.e.,

$$\forall s \in K^{12} \cap K^1 \cap K^2, \forall a \in \Sigma, sa \in K^1 \cap K^2 \Rightarrow sa \in K^{12}.$$

Obviously if $sa \in pr(K)$, then $sa \in K^{12}$. On the other hand, if $sa \in K^1 \cap K^2 - pr(K)$, then sa is a trace that gets created due to the introduction of new transitions in the state machine S when Algorithm 7 is applied to compute $S_K^i (i = 1, 2)$ with $L(S_K^i) = K^i$.

We first examine the centralized case and discuss all possible ways in which transitions added by Algorithm 7 can create a trace $sa \in L(S_K)$ given that $s \in L(S_K)$. From the construction of Algorithm 7, the trace sa can be created by inserting, deleting, or replacing an event in the middle of a trace that ends in event a, or by appending the event a at the end of s. So the possible cases are given by,

1. (Insert) $s = ubbv \in L(S_K), \exists t = ubva \in pr(K), M(b) = \epsilon \Rightarrow sa = ubbva \in L(S_K)$
2. (Delete) $s = uv \in L(S_K), \exists t = ubva \in pr(K), M(b) = \epsilon \Rightarrow sa = uva \in L(S_K)$
3. (Replace) $s = ubv \in L(S_K), \exists t \in u[M^{-1}M(b) \cap \Sigma_u]va \cap pr(K) \Rightarrow sa = ubva \in L(S_K)$
4. (Append-C) $s \in L(S_K), a \in \Sigma_u \Rightarrow sa \in L(S_K)$
5. (Append-R) $s = ua \in L(S_K), M(a) = \epsilon \Rightarrow sa = uaa \in L(S_K)$

(Note in cases 4 and 5 trace s is extended based on controllability and recognizability requirements, respectively. This is the reason we denote them by 'Append-C' and 'Append-R' respectively.)

By considering the two state machines S_K^1 and S_K^2 (that generate the languages K^1 and K^2 respectively), and considering all possible pairwise combinations of the above 5 cases, we can list all possible ways a trace sa can be created in $K^1 \cap K^2 = L(S_K^1) \cap L(S_K^2)$ given that $s \in K^1 \cap K^2 = L(S_K^1) \cap L(S_K^2)$:

(1-1) $s = u_1b_1b_1v_1 \in K^1, \exists t_1 = u_1b_1v_1a \in pr(K), M_1(b_1) = \epsilon \Rightarrow sa = u_1b_1b_1v_1a \in K^1$

$s = u_2b_2b_2v_2 \in K^2, \exists t_2 = u_2b_2v_2a \in pr(K), M_2(b_2) = \epsilon \Rightarrow sa = u_2b_2b_2v_2a \in K^2$

(1-2) $s = u_1b_1b_1v_1 \in K^1, \exists t_1 = u_1b_1v_1a \in pr(K), M_1(b_1) = \epsilon \Rightarrow sa = u_1b_1b_1v_1a \in K^1$

$s = u_2v_2 \in K^2, \exists t_2 = u_2b_2v_2a \in pr(K), M_2(b_2) = \epsilon \Rightarrow sa = u_2v_2a \in K^2$

(1-3) $s = u_1b_1b_1v_1 \in K^1, \exists t_1 = u_1b_1v_1a \in pr(K), M_1(b_1) = \epsilon \Rightarrow sa = u_1b_1b_1v_1a \in K^1$

$s = u_2b_2v_2 \in K^2, \exists t_2 \in u_2[M_2^{-1}M_2(b_2) \cap \Sigma_{u2}]v_2a \cap pr(K) \Rightarrow sa = u_2b_2v_2a \in K^2$

(1-4) $s = u_1b_1b_1v_1 \in K^1, \exists t_1 = u_1b_1v_1a \in pr(K), M_1(b_1) = \epsilon \Rightarrow sa = u_1b_1b_1v_1a \in K^1$

$s \in K^2, a \in \Sigma_{u2} \Rightarrow sa \in K^2$

(1-5) $s = u_1b_1b_1v_1 \in K^1, \exists t_1 = u_1b_1v_1a \in pr(K), M_1(b_1) = \epsilon \Rightarrow sa = u_1b_1b_1v_1a \in K^1$

$s = u_2a \in K^2, M_2(a) = \epsilon \Rightarrow sa = u_2aa \in K^2$

(2-2) $s = u_1b_1b_1v_1 \in K^1, \exists t_1 = u_1b_1v_1a \in pr(K), M_1(b_1) = \epsilon \Rightarrow sa = u_1b_1b_1v_1a \in K^1$

$s = u_2b_2b_2v_2 \in K^2, \exists t_2 = u_2b_2v_2a \in pr(K), M_2(b_2) = \epsilon \Rightarrow sa = u_2b_2b_2v_2a \in K^2$

(2-3) $s = u_1b_1b_1v_1 \in K^1, \exists t_1 = u_1b_1v_1a \in pr(K), M_1(b_1) = \epsilon \Rightarrow sa = u_1b_1b_1v_1a \in K^1$

$s = u_2b_2v_2 \in K^2, \exists t_2 \in u_2[M_2^{-1}M_2(b_2) \cap \Sigma_{u2}]v_2a \cap pr(K) \Rightarrow sa = u_2b_2v_2a \in K^2$

(2-4) $s = u_1b_1b_1v_1 \in K^1, \exists t_1 = u_1b_1v_1a \in pr(K), M_1(b_1) = \epsilon \Rightarrow sa = u_1b_1b_1v_1a \in K^1$

$s \in K^2, a \in \Sigma_{u2} \Rightarrow sa \in K^2$

(2-5) $s = u_1b_1b_1v_1 \in K^1, \exists t_1 = u_1b_1v_1a \in pr(K), M_1(b_1) = \epsilon \in K^1$

$s = u_2a \in K^2, M_2(a) = \epsilon \Rightarrow sa = u_2aa \in K^2$

(3-3) $s = u_1b_1v_1 \in K^1, \exists t_1 \in u_1[M_1^{-1}M_1(b_1) \cap \Sigma_{u1}]v_1a \cap pr(K) \Rightarrow sa = u_1b_1v_1a \in K^1$

$s = u_2b_2v_2 \in K^2, \exists t_2 \in u_2[M_2^{-1}M_2(b_2) \cap \Sigma_{u2}]v_2a \cap pr(K) \Rightarrow sa = u_2b_2v_2a \in K^2$

(3-4) $s = u_1b_1v_1 \in K^1, \exists t_1 \in u_1[M_1^{-1}M_1(b_1) \cap \Sigma_{u1}]v_1a \cap pr(K) \Rightarrow sa = u_1b_1v_1a \in K^1$

$s \in K^2, a \in \Sigma_{u2} \Rightarrow sa \in K^2$

(3-5) $s = u_1b_1v_1 \in K^1, \exists t_1 \in u_1[M_1^{-1}M_1(b_1) \cap \Sigma_{u1}]v_1a \cap pr(K) \Rightarrow sa = u_1b_1v_1a \in K^1$

$s = u_2a \in K^2, M_2(a) = \epsilon \Rightarrow sa = u_2aa \in K^2$

(4-4) $s \in K^1 \cap K^2, a \in \Sigma_{u1} \cap \Sigma_{u2} \Rightarrow sa \in K^1 \cap K^2$

(4-5) $s \in K^1, a \in \Sigma_{u1} \Rightarrow sa \in K^1$

$s = u_2 a \in K^2, M_2(a) = \epsilon \Rightarrow sa = u_2 aa \in K^2$

(5-5) $s = ua \in K^1 \cap K^2, M_1(a) = M_2(a) = \epsilon \Rightarrow sa = uaa \in K^1 \cap K^2$

The additional combinations (2-1), (3-1), (3-2), (4-1), (4-2), (4-3), (5-1), (5-2), (5-3), and (5-4) are also possible, and can be defined symmetrically to the combinations (1-2), (1-3), (2-3), (1-4), (2-4), (3-4), (1-5), (2-5), (3-5), and (4-5), respectively.

Since K^{12} is (Σ_{ui}, M_i)-coachievable with respect to Σ^* and $s \in K^{12}$ (induction hypothesis), it follows from the Definition 23 that $sa \in K^{12}$ in each of the above cases. To further illustrate the proof, we list the proof for the first five cases in Table 6.1. (Proof for other cases can be obtained in a similar way and have been omitted.) This proves the induction step and completes the proof. ■

Case ID	Corresponding rule in Definition 23	Define sub-traces of s as	Extended trace sa
(1-1)	(R_1-R_2)	$s = s_1 a_1 t_1 : s_1 = u_1, a_1 = b_1, t_1 = v_1 a$ $s = s_2 a_2 t_2 : s_2 = u_2, a_2 = b_2, t_2 = v_2 a$	$sa = u_1 b_1 b_1 v_1 a$ $= u_2 b_2 b_2 v_2 a$
(1-2)	(R_1-R_2)	$s = s_1 a_1 t_1 : s_1 = u_1, a_1 = b_1, t_1 = v_1 a$ $s = s_2 a_2 t_2 : s_2 = u_2, a_2 = b_2, t_2 = v_2 a$	$sa = u_1 b_1 b_1 v_1 a$ $= u_2 v_2 a$
(1-3)	(R_1-A_2)	$s = s_1 a_1 t_1 : s_1 = u_1, a_1 = b_1, t_1 = v_1 a$ $s = s_2 a_2 t_2 : s_2 = u_2, a_2 = c_2, t_2 = v_2 a$	$sa = u_1 b_1 b_1 v_1 a$ $= u_2 b_2 v_2 a$
(1-4)	(R_1-C_2)	$s = s_1 a_1 t_1 : s_1 = u_1, a_1 = b_1, t_1 = v_1 a$ $s, a \in \Sigma_{u2}$	$sa = u_1 b_1 b_1 v_1 a$
(1-5)	(R_1-R_2)	$s = s_1 a_1 t_1 : s_1 = u_1, a_1 = b_1, t_1 = v_1 a$ $s = s_2 a_2 t_2 : s_2 = u_2, a_2 = a, t_2 = \epsilon$	$sa = u_1 b_1 b_1 v_1 a$ $= u_2 aa$

Table 6.1 Proof table for Theorem 18

The following corollary follows from Theorems 17 and 18.

Corollary 4 $K \subseteq \Sigma^*$ is (Σ_{ui}, M_i)-coachievable with respect to Σ^* if and only if $pr(K) = inf\overline{PA}^1_{\Sigma^*}(K) \cap inf\overline{PA}^2_{\Sigma^*}(K)$.

Remark 23 Corollary 4 provides a way to test (Σ_{ui}, M_i)-coachievability with respect to Σ^* as follows: Using Algorithm 7 compute $S_K^i (i = 1, 2)$ with $L(S_K^i) = inf\overline{PA}^i_{\Sigma^*}(K)$, and check whether $L(S_K^1 \| S_K^2) \subseteq pr(K)$. The complexity of this test is cubic in the number of states of the acceptor of K.

Now that we have an efficient way of testing coachievability based on Corollary 4, we use it in the following example to illustrate that coachievability is not preserved under union.

Example 17 Consider languages K_1, K_2, and $K_1 \cup K_2$ generated by state machines shown in Fig. 6.1, with $K_1 = pr(ac + a^*), K_2 = pr(abc + a^*)$ and $K_1 \cup K_2 = pr(ac + abc + a^*)$. Suppose $M_1(a) = \epsilon, M_1(b) = b, M_1(c) = c, M_2(a) = M_2(b) = a, M_2(c) = c, \Sigma_{u1} = \emptyset$, and $\Sigma_{u2} = \{a\}$. Fig. 6.1 shows the prefix-closed, (Σ_{ui}, M_i)-achievable superlanguages of K_1, K_2 and $K_1 \cup K_2$ with respect to Σ^* based on Algorithm 7. It is easy to verify that

$$inf\overline{PA}^1_{\Sigma^*}(K_1) = pr(a^*c + a^*),$$

$$inf\overline{PA}^2_{\Sigma^*}(K_1) = pr(aca^* + a^*),$$

$$inf\overline{PA}^1_{\Sigma^*}(K_2) = pr(a^*bc + a^*),$$

$$inf\overline{PA}^2_{\Sigma^*}(K_2) = pr(abca^* + aaca^* + aba^* + a^*),$$

$$inf\overline{PA}^1_{\Sigma^*}(K_1 \cup K_2) = pr(a^*c + a^*bc + a^*), \text{ and}$$

$$inf\overline{PA}^1_{\Sigma^*}(K_1 \cup K_2) = pr(abca^* + aaca^* + aba^* + a^*).$$

Therefore, $pr(K_1) = inf\overline{PA}^1_{\Sigma^*}(K_1) \cap inf\overline{PA}^2_{\Sigma^*}(K_1)$, $pr(K_2) = inf\overline{PA}^1_{\Sigma^*}(K_2) \cap inf\overline{PA}^2_{\Sigma^*}(K_2)$, but $pr(K_1 \cup K_2) \neq inf\overline{PA}^1_{\Sigma^*}(K_1 \cup K_2) \cap inf\overline{PA}^2_{\Sigma^*}(K_1 \cup K_2)$. Then, it follows from Corollary 4 that K_1 and K_2 are (Σ_{ui}, M_i)-coachievable with respect to Σ^*, but $K_1 \cup K_2$ does not have that property. ■

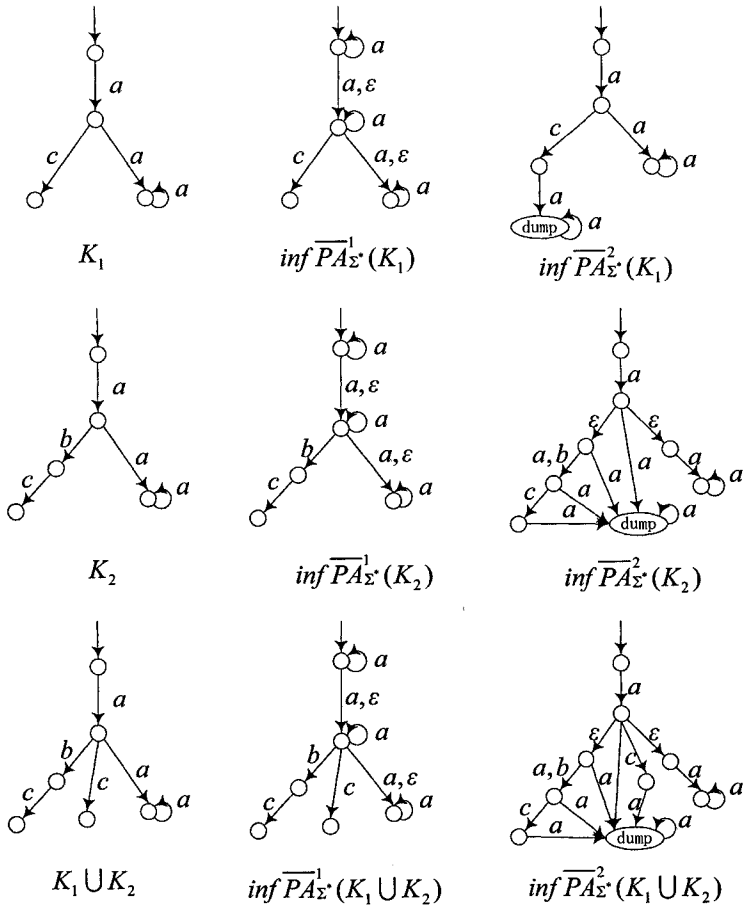


Figure 6.1 Illustration for Example 17

6.3 Decentralized Control Using Nondeterministic Supervisors

In this section, we study the decentralized control problem using nondeterministic supervisors, including target control and range control problems. First, we introduce the definition of (Σ_{ui}, M_i)-coachievability with respect to a plant language L.

Definition 24 $K \subseteq L = pr(L)$ is said to be (Σ_{ui}, M_i)-coachievable with respect to L if $pr(K) = inf\overline{PCoA}_{\Sigma^*}(K) \cap L$.

When $L = \Sigma^*$, coachievability with respect to L is reduced to coachievability with respect to Σ^*, and $pr(K) = inf\overline{PCoA}_{\Sigma^*}(K)$, which is stated in Theorem 15. The following theorem establishes a relationship between (Σ_{ui}, M_i)-coachievability with respect to Σ^* and (Σ_{ui}, M_i)-coachievability with respect to $L = pr(L) \subseteq \Sigma^*$.

Theorem 19 $K \subseteq L = pr(L)$ is (Σ_{ui}, M_i)-coachievable with respect to L if and only if there exists $K' \subseteq \Sigma^*$ such that K' is (Σ_{ui}, M_i)-coachievable with respect to Σ^* and $pr(K) = pr(K') \cap L$.

Proof: The necessity follows directly from Definition 24 by choosing $K' = inf\overline{PCoA}_{\Sigma^*}(K)$. For sufficiency, we need to prove $pr(K) = inf\overline{PCoA}_{\Sigma^*}(K) \cap L$. Since it always holds that $pr(K) \subseteq inf\overline{PCoA}_{\Sigma^*}(K) \cap L$, we only need to show $pr(K) \supseteq inf\overline{PCoA}_{\Sigma^*}(K) \cap L$. Since K' is (Σ_{ui}, M_i)-coachievable with respect to Σ^*, so is $pr(K')$. Also since $pr(K) = pr(K') \cap L$, $pr(K')$ is a superlanguage of K. So we must have $pr(K') \supseteq inf\overline{PCoA}_{\Sigma^*}(K)$. This implies $pr(K) = pr(K') \cap L \supseteq inf\overline{PCoA}_{\Sigma^*}(K) \cap L$, as desired. ■

The following property of (Σ_{ui}, M_i)-coachievability with respect to L can be obtained from Theorems 14 and 19.

Theorem 20 (Σ_{ui}, M_i)-coachievability with respect to L is preserved under intersection over prefix-closed languages, but it is not preserved under union.

Proof: First, we prove that (Σ_{ui}, M_i)-coachievability with respect to L is preserved under intersection over prefix-closed languages. Assume $\{K_j, j \in J\}$ is a set of (Σ_{ui}, M_i)-coachievable languages with respect to L. It follows from Theorem 19 that for each $j \in J$ there exists K'_j such that K'_j is (Σ_{ui}, M_i)-coachievable languages with respect to Σ^* and $pr(K_j) = pr(K'_j) \cap L$. Then from Theorem 14, $\bigcap_{j \in J} K'_j$ is (Σ_{ui}, M_i)-coachievable with respect to Σ^*. Also, since $pr(\bigcap_{j \in J} K_j) = \bigcap_{j \in J} pr(K_j) = \bigcap_{j \in J} (pr(K'_j) \cap L) = pr(\bigcap_{j \in J} K'_j) \cap L$, it follows from another application of Theorem 19 that $\bigcap_{j \in J} K_j$ is (Σ_{ui}, M_i)-coachievable with respect to L.

Since (Σ_{ui}, M_i)-coachievability with respect to Σ^* is not preserved under union (Example 17), (Σ_{ui}, M_i)-coachievability with respect to L has the same property. Otherwise, we can let $L = \Sigma^*$ and arrive at a contradiction. ■

The closure property of (Σ_{ui}, M_i)-coachievability with respect to L under intersection guarantees the existence of the infimal prefix-closed and (Σ_{ui}, M_i)-coachievable superlanguage with respect to L. For $K \subseteq L = pr(L)$, the class of prefix-closed and (Σ_{ui}, M_i)-coachievable super-

language of K with respect to L is defined as:

$$\overline{PCoA}_L(K) = \{K \subseteq K' \subseteq L \mid K' = pr(K'), K' \text{ is } (\Sigma_{ui}, M_i)\text{-coachievable with respect to } L\}.$$

The infimal prefix-closed and (Σ_{ui}, M_i)-coachievable superlanguage of K with respect to L is denoted $inf\overline{PCoA}_L(K)$.

The relationship between $inf\overline{PCoA}_L(K)$ and $inf\overline{PCoA}_{\Sigma^*}(K)$ is stated in the following theorem.

Theorem 21 For all $K \subseteq L = pr(L)$, $inf\overline{PCoA}_L(K) = inf\overline{PCoA}_{\Sigma^*}(K) \cap L$.

Proof: ($\supseteq$) It follows from Theorem 19 that there exists $K' \subseteq \Sigma^*$ such that K' is (Σ_{ui}, M_i)-coachievable with respect to Σ^* and $inf\overline{PCoA}_L(K) = pr(K') \cap L$. Since $pr(K')$ is prefix-closed, (Σ_{ui}, M_i)-coachievable with respect to Σ^*, we have $pr(K') \supseteq inf\overline{PCoA}_{\Sigma^*}(K)$. Therefore, $inf\overline{PCoA}_L(K) \supseteq inf\overline{PCoA}_{\Sigma^*}(K) \cap L$.

($\subseteq$) It follows from Definition 24 that $inf\overline{PCoA}_{\Sigma^*}(K) \cap L$ is (Σ_{ui}, M_i)-coachievable with respect to L. Since $inf\overline{PCoA}_{\Sigma^*}(K) \cap L$ is also prefix-closed, it follows that $inf\overline{PCoA}_L(K) \subseteq inf\overline{PCoA}_{\Sigma^*}(K) \cap L$. ■

The following corollary follows directly from Definition 24, Theorems 18, and 21.

Corollary 5 $K \subseteq L = pr(L)$ is (Σ_{ui}, M_i)-coachievable with respect to L if and only if $pr(K) = inf\overline{PCoA}_L(K) = inf\overline{PCoA}_{\Sigma^*}(K) \cap L = inf\overline{PA}^1_{\Sigma^*}(K) \cap inf\overline{PA}^2_{\Sigma^*}(K) \cap L$.

Remark 24 Since the complexity of computing $inf\overline{PA}_{\Sigma^*}(K)$ is linear in the number of states in the acceptor of K, the complexity of computing $inf\overline{PCoA}_L(K) = inf\overline{PA}^1_{\Sigma^*}(K) \cap inf\overline{PA}^2_{\Sigma^*}(K) \cap L$ is quadratic in the number of states of the acceptor of K and linear in the number of states of the generator for L. So the complexity of checking $inf\overline{PCoA}_L(K) \subseteq pr(K)$ to verify the (Σ_{ui}, M_i)-coachievability of K with respect to L is cubic in the number of states of the acceptor of K and linear in the number of states of the generator for L.

The following theorem provides a necessary and sufficient condition for the existence of decentralized nondeterministic supervisors for "target" control.

Theorem 22 Given a plant G and the specification language $K \subseteq L(G)$, there exist (Σ_{ui}, M_i)-compatible nondeterministic supervisors S_i $(i = 1, 2)$ such that the controlled system behavior $L(G\|(S_1\|S_2)) = pr(K)$ if and only if K is nonempty and (Σ_{ui}, M_i)-coachievable with respect to $L(G)$.

Proof: (If) If K is (Σ_{ui}, M_i)-coachievable with respect to $L(G)$, it follows from Corollary 5 that $pr(K) = inf\overline{PA}^1_{\Sigma^*}(K) \cap inf\overline{PA}^2_{\Sigma^*}(K) \cap L(G)$. Further K nonempty implies $inf\overline{PA}^i_{\Sigma^*}(K) \neq \emptyset$ $(i = 1, 2)$. So we can construct (Σ_{ui}, M_i)-compatible nondeterministic supervisors S_i such that $L(S_i) = inf\overline{PA}^i_{\Sigma^*}(K)$. Then we have that

$$pr(K) = L(S_1) \cap L(S_2) \cap L(G) = L(G\|(S_1\|S_2)).$$

(Only if) If there exist (Σ_{ui}, M_i)-compatible nondeterministic supervisors S_i $(i = 1, 2)$ such that $pr(K) = L(G\|(S_1\|S_2))$, then clearly, $K \neq \emptyset$. Further, it suffices to show that $L(G\|(S_1\|S_2))$ is (Σ_{ui}, M_i)-coachievable with respect to $L(G)$. Since $L(G\|(S_1\|S_2) = L(S_1) \cap L(S_2) \cap L(G)$, by Theorem 19, it suffices to prove that $L(S_1) \cap L(S_2)$ is (Σ_{ui}, M_i)-coachievable with respect to Σ^*. It follows from Definition 23 that we need to consider the following languages $K_1, \ldots, K_6$ and show that $\bigcup_{i=1}^{6} K_i \subseteq L(S1) \cap L(S_2)$:

$$K_1 = \{s_1\Sigma^*_{u1} \cap s_2\Sigma^*_{u2} \mid s_1, s_2 \subseteq L(S_1) \cap L(S_2)\},$$

$$K_2 = \{s_1a_1^*t_1 \cap s_2a_2^*t_2 \mid s_1a_1t_1, s_2a_2t_2 \in L(S_1) \cap L(S_2), M_1(a_1) = M_2(a_2) = \epsilon\},$$

$$K_3 = \{s_1[M_1^{-1}M_1(a_1) \cap \Sigma^*_{u1}]t_1 \cap s_2[M_2^{-1}M_2(a_2) \cap \Sigma^*_{u2}]t_2 \mid s_1a_1t_1, s_2a_2t_2 \in L(S_1) \cap L(S_2)\},$$

$$K_4 = \{s_i\Sigma^*_{ui} \cap s_ja_j^*t_j \mid s_i, s_ja_jt_j \in L(S_1) \cap L(S_2), M_j(a_j) = \epsilon\},$$

$$K_5 = \{s_i\Sigma^*_{ui} \cap s_j[M_j^{-1}M_j(a_j) \cap \Sigma^*_{uj}]t_j \mid s_i, s_ja_jt_j \in L(S_1) \cap L(S_2)\},$$

$$K_6 = \{s_ia_i^*t_i \cap s_j[M_j^{-1}M_j(a_j) \cap \Sigma^*_{uj}]t_j \mid s_ia_it_i, s_ja_jt_j \in L(S_1) \cap L(S_2)\}.$$

where $i, j \in \{1, 2\}, i \neq j$.

In order to show $\bigcup_{n=1}^{6} K_n \subseteq L(S_1) \cap L(S_2)$, we review the following result from centralized setting taken from [31]. Suppose S is a (Σ_u, M)-compatible nondeterministic state machine.

Define H_1, H_2 and H_3 as follows:

$$H_1 = \{s\Sigma_u^* | s \in L(S)\},$$
$$H_2 = \{sa^*t | sat \in L(S), M(a) = \epsilon\},$$
$$H_3 = \{s[M^{-1}M(a) \cap \Sigma_u^*]t | sat \in L(S))\}.$$

Then using (Σ_u, M)-achievability with respect to Σ^* of $L(S)$, it is shown in [31] that $\bigcup_{i=1}^{3} H_i \subseteq L(S)$. Extending this observation to the decentralized case yields the desired containment, $\bigcup_{i=1}^{6} K_i \subseteq L(S_1) \cap L(S_2)$. ■

Remark 25 From the proof of Theorem 22, we know that the existence of decentralized supervisors for the target control is cubic in the number of states of the acceptor of K and linear in the number of the plant states. When the existence condition is met, local nondeterministic supervisors $S_i (i = 1, 2)$ can be constructed such that $L(S_i) = inf\overline{PA}^i_{\Sigma^*}(K)$. It follows that the complexity of synthesizing each local supervisor is linear in the number of states of the acceptor of K.

Next we present an example of a language that is controllable and coachievable but not co-observable.

Example 18 Consider plant G and specification R state machines shown in Fig. 6.2, with $L(G) = pr(ac + bc)$ and $K = L(R) = pr(ac + b)$. Suppose all events are controllable to both supervisors and $M_1(a) = M_1(b) = M_2(a) = M_2(b) \neq \epsilon$. Then clearly K is controllable but not co-observable. (Both supervisors have the same mask function, and since a and b are indistinguishable, and c is permitted after only a, both supervisors will be ambiguous about disabling c following the occurrence of b.) It follows that the specification language K cannot be enforced using deterministic supervisors.

In order to test the existence of nondeterministic supervisors, we construct nondeterministic state machines S_i $(i = 1, 2)$ with generated languages $inf\overline{PA}^i_{\Sigma^*}(K)$ as shown in Fig. 6.2. Then it is easy to verify that $pr(K) = inf\overline{PA}^1_{\Sigma^*}(K) \cap inf\overline{PA}^2_{\Sigma^*}(K) \cap L(G)$. Then, it follows

from Corollary 5 and Theorem 22 that the specification language can be enforced using the nondeterministic supervisors S_i's. ■

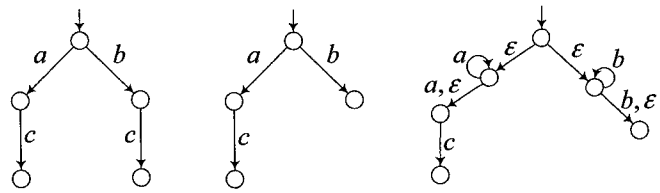


Figure 6.2 G (left), R (middle) and S_i (right)

The result of Theorem 22 can be extended to solve the range control problem. Without loss of generality we take the lower bound to be nonempty (if empty, it can be replaced by $\{\epsilon\}$ since the controlled behavior must at least generate the ϵ trace).

Theorem 23 Given a plant G and specification bounds $A \subseteq E \subseteq L(G)$, there exist (Σ_{ui}, M_i)-compatible nondeterministic supervisors S_i $(i = 1, 2)$ such that $A \subseteq L(G \| (S_1 \| S_2)) \subseteq E$ if and only if $inf\overline{PCoA}_{L(G)}(A) \subseteq E$.

Proof: (If) When $inf\overline{PCoA}_{L(G)}(A) \subseteq E$, it follows from Theorem 22 that there exist (Σ_{ui}, M_i)-compatible nondeterministic supervisors S_i $(i = 1, 2)$ such that $L(G \| (S_1 \| S_2)) = inf\overline{PCoA}_{L(G)}(A) \subseteq E$. Since $A \subseteq inf\overline{PCoA}_{L(G)}(A)$ (from the definition of $inf\overline{PCoA}_{L(G)}(A)$), we have $A \subseteq L(G \| (S_1 \| S_2)) \subseteq E$.

(Only if) When there exist (Σ_{ui}, M_i)-compatible nondeterministic supervisors S_i $(i = 1, 2)$ such that $A \subseteq L(G \| (S_1 \| S_2)) \subseteq E$, it follows from Theorem 22 that $L(G \| (S_1 \| S_2))$ is prefix-closed, (Σ_{ui}, M_i)-coachievable with respect to $L(G)$, and is a superlanguage of A. Since $inf\overline{PCoA}_{L(G)}(A)$ is the infimal such superlanguage, $inf\overline{PCoA}_{L(G)}(A) \subseteq L(G \| (S_1 \| S_2)) \subseteq E$.

■

CHAPTER 7. CONCLUSION

7.1 Summary of Dissertation

The main contributions of this dissertation on decentralized/distributed failure diagnosis of DESs include:

1. In this dissertation, we formalize the problem of decentralized diagnosis by introducing a notion of codiagnosability. Algorithms of polynomial complexity in the size of system/specificatioin model are proposed for verifying codiagnosability, synthesizing local diagnosis, and on-line diagnosis using them. Compared to methods used in [60, 13, 46], which have exponential complexity in the size of system/specificiaton models, our methods are more computationally efficientive.

2. We introduce the properties of strong-diagnosability and strong-codiagnosability in this dissertation, which specify capabilities of a system being certain about its fault/non-fault behaviors within bounded delays in centralized and decentralized settings, respectively. Such properties are useful for system health monitoring, as well as failure diagnosis.

3. To study the property of being able to react safely to failures in a decentralized setting, we introduce a notion of safe-codiagnosability by extending the notion of safe-diagnosability [46] to the decentralized setting. Safe-codiagnosability captures the property that when a system executes a trace that is faulty, there exists at least one diagnoser that can detect this within bounded delay and also before the system behavior becomes "unsafe". Necessary and sufficient conditions for safe-codiagnosability are established, showing that safe-codiagnosability can be separated into the properties of codiagnosability together with "zero-delay codiagnosability" of "boundary safe traces". Algorithm with polyno-

mial complexity is provided for verifying safe-codiagnosability. For a safe-codiagnosable system, the same methods as those for a codiagnosable system are applicable for the synthesis of local diagnosers as well as for on-line diagnosis using them.

4. We establish a unified dynamic model for general communication protocols in distributed diagnosis. Based on that modeling framework, distributed diagnosis via various protocols, with bounded or unbounded communication delays, and event decentralized diagnosis can be formulated and analyzed in a unified manner.

5. We propose finite automata models to capture communication delays in an immediate observation passing protocol. The advantage of this modeling mechanism is that communication delays can be systematically analyzed by composing such delay models with system and specification models, which are automata models as well.

6. By using automata models of communication delays, we convert distributed diagnosis under an immediate observation passing protocol to an instance of decentralized diagnosis. Then algorithms of polynomial complexity for the latter problem are applied to solve the former problem. Methods of reducing complexity of verification and synthesis are presented.

7. For distributed diagnosis under unbounded-delay communication, we establish its decidability. This work clarifies misunderstanding existing in the literature due to a prior work [68], where the authors proposed a notion of decentralized-diagnosability to capture this problem. Our study shows that decentralized-diagnosability is not an adequate property. Instead, a stronger notion of joint$_\infty$-diagnosability is needed, which is shown to be equivalent to codiagnosability.

The main contributions of this dissertation on decentralized supervisory control of DESs include:

1. In this dissertation, we employ PSC for decentralized control. As opposed to SSC, PSC based decision fusion mechanism removes the requirement on supervisors that feasible

uncontrollable events be synchronously executed (i.e., the requirement of "supervisor completeness" or "Σ_u-compatibility" [64]).

2. A notion of PSC-coobservability is introduced to capture conditions for PSC based decentralized control. Restriction in a prior work [33], which requires that priority sets of the supervisors exhaust the controllable events set, is relaxed in our method. Also, we show that PSC-coobservability is more general than C&P $\vee$ D&A-coobservability [75], plus when there is flexibility to choose the priority and conjunction/disjunction sets, the class of PSC-coobservable languages coincides with the class of C&P$\vee$D&A-coobservable languages.

3. We show through construction that when plant and specification languages are regular, the local supervisors can be represented as finite automata, with complexity polynomial in plant states and exponential in specification states. Prior work on C&P $\vee$ D&A-coobservability presented the local supervisors as observers and did not specify an algorithm for the computation of control actions for each observer state. Also, the size of the observer is exponential in both the plant and specification states. While the realization of local supervisors is well known in the conjunctive setting, the same construction does not work in the PSC setting (as illustrated by an example). A new construction method has been presented in this paper that works also in all other settings, namely, centralized, conjunctive, disjunctive, and conjunctive+disjunctive.

4. "Command-response" automata for the realization of supervisors is introduced. Unlike the conjunctive setting, in the setting of PSC or non-conjunction, a locally disabled event may be globally enabled, and a local supervisor may need to track such an event when it is executed by the plant. On one hand, a disablement requires that a transition on the disabled event be undefined, and on the other hand, tracking requires the transition be defined. So, this leads to introduction of more general, "command-response" automata. Such automata possess two types of transitions, one which determines the local supervisors' enablement decisions (command), and others which determine how to

perform state-updates when a locally disabled but globally enabled event is executed by the plant (response). Accordingly, the definition of PSC has been extended to over "command-response" automata.

5. By extending nondeterministic control from the centralized setting to the decentralized setting, we obtain a weaker condition of existence of decentralized supervisors, introduced as co-achievability. When the local supervisors are required to be deterministic, a stronger condition of co-observability together with controllability serves as a necessary and sufficient condition for the existence.

6. For decentralized target control and range control problems, we develop algorithms of polynomial complexity for the existence test and synthesis of decentralized *nondeterministic* supervisors. This is in contrast to decentralized *deterministic* control, where both the existence and synthesis are of exponential complexity.

7.2 Directions of Future Research

For distributed diagnosis under bounded communication delays, we discuss an immediate observation passing protocol in this dissertation. A future direction can be an approach for more general distributed diagnosis where communicated information is not necessarily limited to local observations, but local sites may compute local state estimates and exchange them to further refine the estimations to come up with their diagnosis decisions. Since communication channels have bounded but random delays, an important issue is how to synchronize diagnosis results of multiple diagnosers.

A direction to reduce computational complexity of diagnosis algorithms is to consider *modular* diagnosis. Our discussion on decentralized and distributed failure diagnosis is based on *monolithic* system and specification models. In some applications, it would be more convenient to construct models for subsystems or local specifications. If we construct global models by synchronously composing local models, such operation would have exponential complexity in the number of local sites. To achieve computational efficiency, modular diagnosis using lo-

cal system/specification models needs to be explored. [48, 49] studied distributed diagnosis without constructing global models, but the authors did not provide conditions for a system to be diagnosable in that setting. [14, 11] considered conditions for modular diagnosis to be equivalent to centralized diagnosis. However, to verify such conditions, a global model has to be constructed first, which eliminates the advantage of modular diagnosis.

For decision fusion mechanisms in decentralized control, it is possible to have more general interface mechanisms such as one that accounts for not only the control limitations but also the observation limitations. Generalizations of PSC to account for observation limitations have been proposed in the literature [64, 30], and also been used for centralized control [30, 23]. In the future it will also be instructive to use a generalized version of PSC that delegates the effects of observation limitations, as well as control limitations, from the logic part to the interface part.

Also, future research may consider including general decision fusion rules into nondeterministic decentralized control. We study in this dissertation nondeterministic decentralized control based on *conjunctive* decision fusion rules. From our discussion on decision fusion mechanisms in deterministic decentralized control, we know that a larger specification set can be achieved by utilizing more general rules, such as conjunctive+disjunctive, and PSC-based decision fusion rules. Similarly, by extending such general rules to nondeterministic decentralized control, a larger specification set is expected to be achieved.

Another important research direction is fault-tolerant control. After the detection and isolation of a failure, we need to initiate some failure recovery or system reconfiguration procedures to guarantee certain basic services in spite of the presence of failures.

In this dissertation, we are focused on untimed DESs, which considers logic behaviors of a system. To consider timing properties of a system, especially for real-time systems, we need to consider timed DESs. For example, a direction is to study how to perform failure diagnosis while meeting certain real-time deadlines by inspecting timing properties in timed DESs.

An extended direction of DESs is hybrid systems, where system dynamics are described by both continuous and discrete-event models. For example, in an embedded system, control

actions of actuators may be described by continuous models, while digital logics can be modeled by DESs. Utilizing some abstraction techniques, DESs theory can be used for modeling, verification, diagnosis, and control of hybrid systems. Those topics are interesting for future research.

BIBLIOGRAPHY

[1] A. Beneveniste, E. Fabre, S. Haar, and C. Jard. Diagnosis of asynchronous discrete-event systems: A net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, 2003.

[2] R. K. Boel and J. H. van Schuppen. Decentralized failure diagnosis for discrete-event systems with constrained communication between diagnosers. In *Proceedings of International Workshop on Discrete Event Systems*, 2002.

[3] A. Bouloutas, G. W. Hart, and M. Schwartz. Simple finite-state fault detectors for communication networks. *IEEE Trans. on Communications*, 40(3):477–479, March 1992.

[4] Y. Brave and M. Heymann. On stabilization of discrete event processes. *International Journal of Control*, 51(5):1101–1117, 1990.

[5] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, Norwell, MA, 1995.

[6] V. Chandra, Z. Hunag, W. Qiu, and R. Kumar. Prioritized composition with exclusion and generation for the interaction and control of discrete event systems. *Mathematical and Computer Modeling of Dynamical Systems*, 9(3):255–280, 2003.

[7] V. Chandra and R. Kumar. A discrete event system modeling formalism based on event occurrence rules and precedences. *IEEE Transactions on Robotics and Automation*, 17(6):785–794, December 2001.

[8] V. Chandra and R. Kumar. A event occurrence rules-based compact modeling formalism

for a class of discrete event systems. *Mathematical and Computer Modeling of Dynamical Systems*, 8(1):49–73, 2002.

[9] 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.

[10] S. R. Das and L. E. Holloway. Characterizing a confidence space for discrete event timings for fault monitoring using discrete sensing and actuation signals. *IEEE Transactions on Systems, Man, and Cybernetics—Part A: Systems and Humans*, 30(1):52–66, 2000.

[11] R. Debouk. Diagnosis of discrete event systems: A modular approach. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pages 306 – 311, 2003.

[12] R. Debouk, S. Lafortune, and D. Teneketzis. On the effect of communication delays in failure diagnosis of decentralized discrete event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 13(3):263–289.

[13] R. Debouk, S. Lafortune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamical Systems: Theory and Applications*, 10:33–79, 2000.

[14] R. Debouk, R. Malik, and B. Brandin. A modular architecture for diagnosis of discrete event systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, pages 417 – 422, 2002.

[15] D. N. Godbole, J. Lygeros, E. Singh, A. Deshpande, and A. E. Lindsey. Communication protocols for a fault-tolerant automated highway system. *IEEE Transactions on Control Systems Technology*, 8(5):787–800, September 2000.

[16] C.N. Hadjicostis and G.C. Verghese. Power system monitoring based on relay and circuit breaker information. In *Proceedings of the 2001 IEEE International Symposium on Circuits and Systems*, volume 2, pages 197–200, May 2001.

[17] M. Heymann. Concurrency and discrete event control. *IEEE Control Systems Magazine*, 10(4):103–112, 1990.

[18] M. Heymann and G. Meyer. Algebra of discrete event processes. Technical Report NASA 102848, NASA Ames Research Center, Moffett Field, CA, June 1991.

[19] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1985.

[20] K. Inan. Nondeterministic supervision under partial observations. In Guy Cohen and Jean-Pierre Quadrat, editors, *Lecture Notes in Control and Information Sciences 199*, pages 39–48. Springer-Verlag, New York, 1994.

[21] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial time algorithm for diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 46(8):1318–1321, 2001.

[22] S. Jiang and R. Kumar. Failure diagnosis of discrete event systems with linear-time temporal logic fault specifications. In *Proceedings of 2002 American Control Conference*, pages 128–133, Anchorage, AK, May 2002.

[23] S. Jiang and R. Kumar. Supervisory control of nondeterministic discrete event systems with driven events via masked prioritized synchronization. *IEEE Transactions on Automatic Control*, 47(9):1438–1449, 2002.

[24] S. Jiang and R. Kumar. Diagnosis of repeated failures for discrete event systems with linear-time temporal logic specifications. In *Proceedings of IEEE Conference on Decision and Control*, pages 3221–3226, Maui, Hawaii, 2003.

[25] S. Jiang, R. Kumar, and H. E. Garcia. Diagnosis of repeated/intermittent failures in discrete event systems. *IEEE Transactions on Automatic Control*, 19(2):310–323, 2003.

[26] R. Kumar and V. K. Garg. *Modeling and Control of Logical Discrete Event Systems.* Kluwer Academic Publishers, Boston, MA, 1995.

[27] R. Kumar and V. K. Garg. Optimal supervisory control of discrete event dynamical systems. *SIAM Journal of Control and Optimization*, 33(2):419–439, March 1995.

[28] 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.

[29] R. Kumar, V. K. Garg, and S. I. Marcus. Language stability and stabilizability of discrete event dynamical systems. *SIAM Journal of Control and Optimization*, 31(5):1294–1320, September 1993.

[30] R. Kumar and M. Heymann. Masked prioritized synchronization for interaction and control of discrete event systems. *IEEE Transactions on Automatic Control*, 45(11):1970–1982, 2000.

[31] R. Kumar, S. Jiang, C. Zhou, and W. Qiu. Polynomial synthesis of supervisor for partially observed discrete-event systems by allowing nondeterminism in control. *IEEE Transaction on Automatic Control*, 50(4):463–475.

[32] R. Kumar and M. A. Shayman. Nonblocking supervisory control of nondeterministic systems via prioritized synchronization. *IEEE Transactions on Automatic Control*, 41(8):1160–1175, August 1996.

[33] R. Kumar and M. A. Shayman. Centralized and decentralized supervisory control of nondeterministic systems under partial observation. *SIAM Journal of Control and Optimization*, 35(2):363–383, March 1997.

[34] R. Kumar and M. A. Shayman. Formulae relating controllability, observability, and co-observability. *Automatica*, 34(2):211–215, 1998.

[35] H. Lamouchi and J. Thistle. Effective control synthesis for DES under partial observations. In *Proceedings of IEEE Conference on Decsision and Control*, pages 22–28, Sydney, Australia, 2000.

[36] F. Lin. Analysis and synthesis of discrete event systems using temporal logic. *Control Theory and Advanced Technologies*, 9(1):341–350, 1993.

[37] F. Lin. Diagnosability of discrete event systems and its applications. *Discrete Event Dynamic Systems: Theory and Applications*, 4(1):197–212, 1994.

[38] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(3):173–198, 1988.

[39] J.-Y. Lin and D. Ionescu. Verifying a class of nondeterministic discrete event systems in a generalized temporal logic. *IEEE Transactions on Systems, Man and Cybernetics*, 22(6):1461–1469, 1992.

[40] J.-Y. Lin and D. Ionescu. Reachability synthesis procedure for discrete event systems in a temporal logic. *IEEE Transactions on Systems, Man and Cybernetics*, 24(9):1397–1406, 1994.

[41] J. Lygeros, D. N. Godbole, and M. Broucke. A fault tolerant control architecture for automated highway system. *IEEE Transactions on Control Systems Technology*, 8(2):205–219, March 2000.

[42] R. E. Miller and A. K. Arisha. Fault identification in networks by passive testing. In *Proceedings of the IEEE Annual Simulation Symposium*, pages 277–284, 2001.

[43] J. S. Ostroff. Temporal logic and extended state machines in discrete-event control. In M. J. Denham and A. J. Laub, editors, *Advanced Computing Concepts and Techniques in Control Engineering*, volume F47 of *NATO ASI*, pages 215–236. Springer-Verlag, 1988.

[44] C. M. Ozveren, A. S. Willsky, and P. J. Antsaklis. Stability and stabilizability of discrete event dynamical systems. *Journal of ACM*, 38(3):730–752, July 1991.

[45] D. Pandalai and L. Holloway. Template languages for fault monitoring of timed discrete event processes. *IEEE Transactions on Automatic Control*, 45(5):868–882, May 2000.

[46] A. Paoli and S. Lafortune. Safe diagnosability of discrete event systems. In *Proceedings of IEEE Conference on Decision and Control*, volume 3, pages 2658–2664, Hawaii, USA, December 2003.

[47] D. M. R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. 5th GI Conference (Lecture Notes in Computer Science, 104)*, pages 167–183. Springer-Verlag, New York, 1981.

[48] Y. Pencolé. Decentralized diagnoser approach: application to telecommunication networks. In *Proceedings of the International Workshop on Principles of Diagnosis*, pages 185–192.

[49] Y. Pencolé, M. O. Cordier, and L. Rozé. Incremental decentralized diagnosis approach for the supervision of a telecommunication network. In *Proceedings of the 12th International Workshop on Principles of Diagnosis*, pages 151–158.

[50] A. D. Pouliezos and G. S. Stavrakakis. *Real time fault monitoring of industrial processes.* Kluwer Academic Publishers, Boston, MA, 1994.

[51] J. H. Prosser, M. Kam, and H. G. Kwatny. Decision fusion and supervisor synthesis in decentralized descrete-event systems. In *Proceedings of 1997 American Control Conference*, pages 2251–2255, 1997.

[52] W. Qiu and R. Kumar. Decentralized failure diagnosis of discrete event systems. In *Proceedings of 2004 International Workshop on Discrete Event Systems*, Reim, France, September 2004.

[53] 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.

[54] 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.

[55] S. L. Ricker and J. H. van Schuppen. Decentralized failure diagnosis with asynchronous communication between supervisors. In *Proceedings of the European Control Conference*, pages 1002–1006, 2001.

[56] K. Rudie and J. C. Willems. The computational complexity of decentralized discrete-event control problems. *IEEE Transactions on Automatic Control*, 40(7):1313–1319, July 1995.

[57] K. Rudie and W. M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, November 1992.

[58] R. H. Kwong S. H. Zad and W. M. Wonham. Fault diagnosis in timed discrete-event systems. In *Proceedings of the 38th IEEE Conference on Decision and Control*, pages 1756–1761, Phoenix, AZ, 1999.

[59] M. Sampath and S. Lafortune. Active diagnosis of discrete event systems. *IEEE Transactions on Automatic Control*, 43(7):908–929, 1998.

[60] M. Sampath, R. Sengupta, S. Lafortune, K. Sinaamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, September 1995.

[61] M. Sampath, R. Sengupta, S. Lafortune, K. Sinaamohideen, and D. Teneketzis. Failure diagnosis using discrete event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124, March 1996.

[62] R. Sengupta and S. Tripakis. Decentralized diagnosis of regular language is undecidable. In *Proceedings of IEEE Conference on Decision and Control*, pages 423–428, Las Vegas, NV, December 2002.

[63] M. A. Shayman and R. Kumar. Supervisory control of nondeterministic systems with driven events via prioritized synchronization and trajectory models. *SIAM Journal of Control and Optimization*, 33(2):469–497, March 1995.

[64] M. A. Shayman and R. Kumar. Process objects/masked composition: An object oriented approach for modeling and control of discrete event systems. *IEEE Transactions on Automatic Control*, 44(10):1864–1869, 1999.

[65] R. Su, W. M. Wonham, J. Kurien, and X. Koutsoukos. Distributed diagnosis for qualitative systems. In *Proceedings of International Workshop on Discrete Event Systems*, 2002.

[66] J. G. Thistle and W. M. Wonham. Control problems in temporal logic framework. *International Journal of Control*, 44(4):943–976, 1986.

[67] S. Tripakis. Undecidable problems of decentralized control and observation. In *Proceedings of 2001 IEEE Conference on Decision and Control*, Orlando, FL, December 2001.

[68] S. Tripakis. Decentralized control of discrete-event systems with bounded or unbounded delay communication. *IEEE Transactions on Automatic Control*, 49(9):1489–1501, 2004.

[69] J. N. Tsitsiklis. On the control of discrete event dynamical systems. *Mathematics of Control Signals and Systems*, 2(2):95–107, 1989.

[70] G. Westerman, R. Kumar, C. Stroud, and J. R. Heath. Discrete event systems approach for delay fault analysis in digital circuits. In *Proceedings of 1998 American Control Conference*, Philadelphia, PA, 1998.

[71] Y. Willner and M. Heymann. Supervisory control of concurrent discrete-event systems. *International Journal of Control*, 54(5):1143–1169, 1991.

[72] Y. Willner and M. Heymann. Language convergence in controlled discrete-event systems. *IEEE Transactions on Automatic Control*, 40(4):616–627, 1995.

[73] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, 1987.

[74] T. Yoo and H. E. Garcia. Event diagnosis of discrete-event systems with uniformly and nonuniformly bounded diagnosis delays. In *Proceedings of 2004 American Control Conference*, pages 5102–5107, Boston, MA, June 2004.

[75] T. S. Yoo and S. Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 12(3):335–377, July 2002.

[76] T. S. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, 47(9):1491–1495, 2002.

[77] S. H. Zad, R. H. Kwong, and W. M. Wonham. Fault diagnosis in discrete-event systems: Framework and model reduction. *IEEE Transactions on Automatic Control*, 48(7):1199–1212, 2003.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere thanks to my advisor Dr. Ratnesh Kumar for providing excellent research environment to me, for his inspiration and guidance in my research work, and for his support, patient and encouragement which help me through difficult time.

I also want to thank my committee members: Dr. Nicola Elia and Dr. Murti Salapaka in the department of Electrical and Computer Engineering, and Dr. Andrew Miner and Dr. Simanta Mitra in the department of Computer Science, for their valuable feedback and comments on my research work and this dissertation.

Last but not lease, I would like to thank my wife, my parents and the rest of my families for their endless love and encouragement through these years.