2009

Fault-tolerant supervisory control of discrete-event systems

Qin Wen
Iowa State University

Fault-tolerant supervisory control of discrete-event systems

by

Qin Wen

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
Samik Basu
Nicola Elia
Manimaran Govindarasu
Arun K. Somani

Iowa State University

Ames, Iowa

2009

DEDICATION

To my wife Ying Wang, and my parents Shiyan Wen and Guangqi Zeng.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, I would like to give my sincere thanks to Dr. Ratnesh Kumar for his guidance, patience and support throughout this research and the writing of this dissertation. His insights have often renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Samik Basu, Dr. Nicola Elia, Dr. Manimaran Govindarasu, and Dr. Arun K. Somani. I would also like to thank my officemates, Wenbin Qiu, Changyan Zhou, Jing Huang, Haifeng Liu, Licheng Jin, Songyan Xu, Saayan Mitra, and Herman Sahota. Without their kind help, the finish of my research work would be impossible.

CHAPTER 1. INTRODUCTION

1.1 Discrete Event Systems

This dissertation is a study of a special class of systems, discrete-event systems, which is a good model for many systems that contain discrete changes. Even the continuous systems are sometimes modeled approximately into discrete-event systems on purpose for easy study. In recent years, more and more efforts are put on the research of discrete-event systems due to their wide use. Examples of discrete-event systems include communication channels, computer networks, manufacturing systems, etc.

1.1.1 The Concept of Discrete-Event Systems

Before going into the detail of discrete-event systems, it is necessary to first introduce the definition of event. But, to mention "event", it is also necessary to give the definition of state first. A *state* of a system is a unique configuration of information that can be used, together with system inputs, to determine the system output. For a simple example, a light bulb can have two states, ON and OFF. With the actions we put on the switch, we can implement the transitions between these two states. These actions are events.

It is hard to give the strict definition, but generally an *event* is thought to occur instantaneously and cause transition from one state to another[Cassandras, C. G. and Lafortune, S. (1999)]. An event should be identified with a specific action, e.g. turning on/off the switch, the arrival of a customer in a queue. The time of the occurrence of the event sometimes is not so important, depending on the type of the system.

A *Discrete-Event System* (DES) is a dynamic system which evolves according to asynchronous occurrence of certain discrete changes (events) [Kumar, R. and Garg, V. K. (1995)].

Examples of discrete-event systems include many man-made systems such as computer and communication networks, robotics and manufacturing systems, and automated traffic system. A DES has a discrete set of states which, unlike a physical system, may take symbolic values rather then real values, for example, a machine is either idle, working or broken. States transitions in such systems occur at asynchronous discrete instants of time in response to events. Model of DESs can be classified into untimed and timed models based on whether they ignore the timing information. The untimed models have information about order of state and event pairs, but not about their timing. Untimed models are used to control and coordinate orderly occurrence of states and events so that the system under study meets certain qualitative goals [Kumar, R. and Garg, V. K. (1995)]. In this thesis, we are concerned with studying only the untimed models.

From the definition above, we can see that the discrete-event system has two basic properties:

1. The set of states is discrete.

2. The state transition is event-driven.

The first property differentiates DES with continuous-state system. The second property indicates that the system can only change at discrete points, which are the occurrences of asynchronously generated discrete events. Discrete-event system models many real systems. Even if a system is not a discrete-state system, for the study interests, it may be helpful to have a discrete-state point of view. An example is power system. Even it is continuous, it is easy to analyze the system behavior if we describe it as a discrete-event system with many working states. In this dissertation, a simplified power system will be modeled into a discrete-event system to analyze its behavior after some faults and to synthesize the control actions.

Here is another simple example of discrete event system. Suppose we have an elevator in a three-story building. The elevator connects the first, the second and the third floor. Naturally, in the discrete event model of the elevator, there are three discrete states, each of which represents a different floor that the elevator will stop at. From lower floor to higher floor,

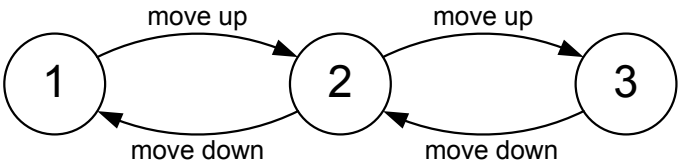

Figure 1.1 The discrete event model of the elevator

the action is moving up, and to the opposite direction, the action is moving down. Figure 1.1 illustrates the discrete event model of the elevator. Circles are the states, with the numbers representing the floors. Arrows are the transitions, and the names of the transition actions are next to the arrows.

1.2 Fault Tolerance

1.2.1 Introduction

In modern world, people rely more and more on the availability and reliability of the complex systems. An small error in the system may cause a disaster. An example is the traffic light systems. The traffic lights are supposed to work continuously and correctly. The malfunction of the traffic light, such as stop functioning or false directing, may lead to an unnecessary jam or serious accidents. For some projects involving human lives, the dependability is extremely critical due to the serious consequences.

With the development of industry, people pay more and more effort on studying trusty systems. Systems are expected to work dependably under anticipation. The concept of dependable computing first appeared in 1830s [Avižienis, A. and Laprie, J.-C. and Randell, B. (2000)]. After the invention of electronic computer, practical techniques were used to improve its reliability. The concept of failure-tolerant system was introduced by W.H. Pierce in 1965 [Pierce, W. H. (1965)]. The research on the fault-tolerant system develops fast, since people rely more and more on the machines. Although it may be expensive to use fault-tolerant techniques, they are applied in aircrafts and railway systems, due to the extremely high cost that failures may cause in transportation. Fault-tolerant software are applied in a lot of well-known projects, such as Ariane 5, Airbus A-320, Boeing B-777, Elektra Austrian

railway signaling system, French train systems, NASA space shuttle, etc. [Voas, J. (2001)].

1.2.2 The Concepts about Fault Tolerance

1.2.2.1 Dependability

Fault tolerance is used to enhance the dependability of the system. Dependability is a composed concept, which contains four attributes: reliability, availability, safety and security [Jalote, P. (1998)].

- *Reliability* is the ability of a system, when it is needed, to perform its service correctly.
- *Availability* means the system is available to perform its service when it is needed.
- *Safety* is a characteristic that qualifies the ability to avoid catastrophic failures that might involve human life or excessive costs.
- *Security* is the ability of a system to prevent unauthorized access.

Fault tolerance is the ability of a system to work correctly, even in case that some components can not behave as desired. The purpose is to prove reliability and availability, such that the system can work continuously and correctly. Although fault tolerance does not have direct relation with safety and security, some researches [Weber, D. G. (1989)] can still make them associated. A system can either be safe but not fault-tolerant, such as fail-safe systems which fail in a safe state, or fault-tolerant but not safe, such as non-masking fault-tolerant systems, which may temporarily enter an unsafe state after a fault occurs.

1.2.2.2 Faults

There are three kinds of threats to the dependability: failure, error and fault. Although these three concepts are used undistinguished in many articles, it is necessary to make it clear of the difference between them. A *failure* occurs when an actual running system deviates from its specified behavior. An *error* is the part of the system state that is liable to lead to a failure. A *fault* is the cause of an error. For example, a stuck-at-zero fault occurs on one bit in the

memory. It has no harm as long as that bit is not used, or only value 0 is restored. This fault may cause an error when value 1 is restored. And, it becomes a failure if this bit is read. It is clear that these three threats have decreasing level of catastrophic influences on the systems. That's why so many efforts has been put on handling faults in order to prevent them becoming failures.

Faults can be categorized into various classes, according to diverse attributes and sources. Table 1.1 shows the classification according to some criteria.

Table 1.1 Fault classification

Duration	transient, persistent, intermittent
Cause	design, operational
Behavior	crash, omission, timing, Byzantine

The last classification is made according to the behavior of the failed system. A *crash fault* will cause the component to halt. An *omission fault* makes the component not to respond to some inputs. If the component responds to the inputs either too early or too late, it has a *timing fault*. And a *Byzantine fault* will lead the component to behave in an arbitrary manner. Unlike the other classifications, the classes in this classification are not disjoint. A crash fault is also an omission fault, and omission fault is a special case of timing fault. The class of Byzantine fault contains all kinds of faults, such that the component with Byzantine fault may have any kind of behavior, which makes the Byzantine fault the most difficult to cope with. Therefore, if a fault-tolerant system can take care of Byzantine faults, it has the ability to deal with any kind of faults. [Lamport, L. and Shostak, R. and Pease, M. (1980)] is the first paper published about the Byzantine faults. Later, [Lamport, L. and Shostak, R. and Pease, M. (1982)] has a systematic study of the Byzantine general problem, which helps to manage Byzantine faults in distributed components. With respect to the same number of faults, the study continues to reduce the number of rounds in reaching the agreement, as well as to reduce the algorithm's complexity in communication and computation. [Berman, P. and Garay, J. A. and Kerry, K. J. (1989)], [Berman, P. and Garay, J. A. (1991)] etc. made some

improvement on [Lamport, L. and Shostak, R. and Pease, M. (1980)], while [Garay, J. A. and Moses, Y. (1998)], with polynomial complexity, reached agreement in $t+1$ rounds among $3t+1$ components, where t is the number of faults.

Since faults may cause undesired deviation of the system, how to deal with the faults is the core problem in building a reliable system. Fault handling is a technique that covers a wide range of method to treat with the faults. It includes fault prevention (fault avoidance), fault tolerance, fault removal and fault forecasting [Avižienis, A. and Laprie, J.-C. and Randell, B. (2000)]. Different from fault tolerance, in which the system works with some failed components, *fault prevention* is achieved during deliberate design and manufacturing, and *fault removal* is to remove faults before they lead to disastrous accidents, while *fault forecasting* is to predict possible faults by concurrently evaluating the system performance.

1.2.2.3 Fault Tolerance

To provide fault tolerance, it is necessary for the system to have redundancy. Redundancy is the key to support fault tolerance, such that there can be no fault tolerance without redundancy [Gartner, F. C. (1999)]. *Redundancy* is defined as those parts of the system that are not needed for the correct functioning of the system [Jalote, P. (1998)]. There can be hardware redundancy (such as backup components) or software redundancy (such as reconfiguration algorithms). Since the system needs time to handle the faults after being aware of them, time redundancy, that is extra execution time, is also necessary.

In many cases, software and hardware are tied in achieving fault tolerance. In fact, hardware redundancy nearly always employs software redundancy. Although the names look similar, there are quite big differences between software fault-tolerance (fault-tolerant computing) and hardware fault-tolerance. [Laprie, J.-C. and Arlat, J. and Beounes, C. and Kanoun, K. (1990)] presents the comparison and contrast of them in architecture. Table 1.2 gives the contrast of them in other aspects.

In the presence of faults, different fault-tolerant systems may have different performance, according to the techniques they use. Three different types of fault tolerance are listed below.

Table 1.2 Software fault tolerance vs. Hardware fault tolerance

	Hardware FT	Software FT
Types of faults	physical faults	design faults
Origins of failures	hardware defects	design/implementation defects
Ways to tolerant faults	forward/backward recovery	design diversity

- Masking tolerance: Both safety and liveness are satisfied in the presence of faults.
- Non-Masking tolerance: Safety may be violated temporarily.
- Fail-Safe Tolerance: Only safety is satisfied in the presence of faults.

Fault masking has several backups for a single component, and uses normal components to "mask" the faulty components, preventing them from affecting the correct performance of the system. Fault masking is a widely used technique because of its simpleness and convenience. Well-known examples include Triple Modular Redundancy (TMR) and Redundant Array of Inexpensive Disks (RAID). According to the status of the backups during working, the backups can be hot standby, cold standby or warm standby [Selic, B. (2002)]. The advantage of fault masking is that the system can work continuously and safely after fault happens, but the cost may be expensive.

Unlike masking fault tolerance, non-masking does not use backups. After fault isolation, reconfiguration techniques are used to recover the system to normal performance. A controller is needed to guide the recovery within some given specifications. Non-masking fault tolerance provides liveness, but the system may violate safety specification during recovery, and the system performance may satisfy only a lower level of specification after recovery, which is known as graceful degradation. Note that masking fault tolerance guarantees liveness, but non-masking fault tolerance only guarantees it eventually. The advantage of non-masking fault tolerance is that, although it is strictly weaker than masking fault tolerance, it can still be used in cases when masking fault tolerance is too costly to implement or even provably impossible [Gartner, F. C. (1999)]. An interesting example is Denmark Ørsted project [Gartner, F. C. (1999)], where using hardware redundancy to obtain a fail-safe design is impossible,

due to cost and weight constraints. In addition, operator intervention from ground is limited to periods when the satellite passes Denmark. There is a 13-hour interval, during which the satellite is unattended. In this situation, autonomous fault tolerant control is a right choice.

Fail safe tolerance, that is failing in safe state, is another kind of property. Fail-stop processors have been studied since 1980's [Schlichting, R. D. and Schneider, F. B. (1983)], [Schneider, F. B. (1983)]. Instead of keeping on working after fault, they simply ceases functioning to preserve safety. It sacrifices liveness to ensure safety. Since the fail-safe systems stop working when a fault occurs, it is a weaker form of fault tolerance, such that it is still disputed (e.g. [Blanke, M. and Staroswiecki, M. and Wu, N. E. (2001)], [Blanke, M. and Izadi-Zamanabadi, R. and Bogh, S. A. and Lunau, C. P. (1997)],[Voas, J. (2001)]) whether fail-safe is a subset of fault tolerance. Since safety is more preferable than liveness, fail-safe tolerance is still an advisable technique, used, for example, in the ground control system of the Ariane 5 space missile project [Dega, J.-L. (1996)].

1.3 Fault Tolerant Control

Fault-tolerant control integrates diagnosis with control methods to handle faults. The aim of fault-tolerant control is to prevent faults from being developed into serious failures ,and therefore increase the availability and reliability of the system and reduce the risk of loss. Various methods of fault-tolerant control have been developed. Most of fault-tolerant control methods share a common structure, which is depicted in Figure 1.2 [Blanke, M. and Kinnaert, M. and Lunze, J. and Staroswiecki, M. (2003)]:

A supervisor is added in fault-tolerant control, which contains the diagnostic block and controller re-design block. The re-design block uses diagnosed information of the fault and adjusts the controller accordingly. The re-design of the controller may change the controller parameters, or give a new controller configuration. The single arrows represent signals, and the double arrow refers information flow.

Without fault, the system runs mainly in the execution level. The nominal controller, which is designed for the faultless system, attenuates the disturbance d and ensures set-point

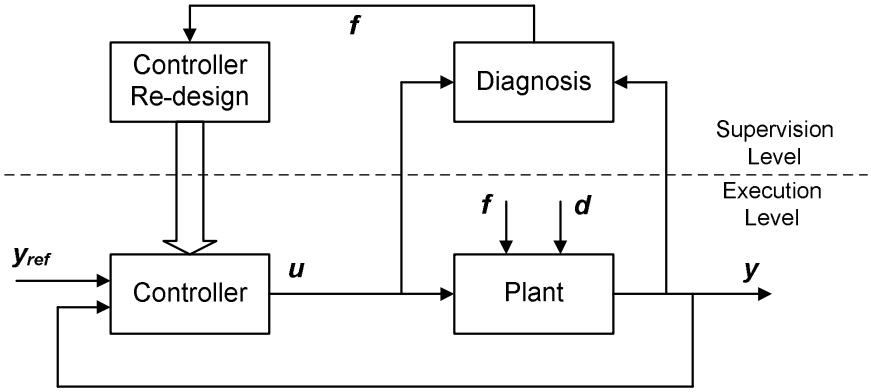

Figure 1.2 The architecture of fault-tolerant control

following and other requirements on the closed-loop system. In this situation, the diagnostic block recognizes that the closed-loop system is faultless and no change of the execution is necessary. If a fault f occurs, the supervision level makes the control loop fault-tolerant. The diagnostic block identifies the fault and the controller re-design block adjusts the controller to the new situation. Afterwards, the execution level alone continues to satisfy the control aims [Blanke, M. and Kinnaert, M. and Lunze, J. and Staroswiecki, M. (2003)].

1.3.1 Controller Re-Design

In case of faults, the controller re-design block will compute the next controller from the diagnosis result. There are two kinds of controller re-design methods: fault accommodation and control reconfiguration.

In fault accommodation, the controller parameters are adapted to the dynamical properties of the faulty plant. Before and after fault accommodation, the input and output of the plant used in the control loop remain the same. A simple way of fault accommodation is to use predesigned controllers, which are selected off-line for a specific fault. The re-design step is to switch among different control laws. This step is quick and can meet strong real-time constraints. However, it is for small systems, otherwise it is difficult to design in advance for all possible faults before the system starts operating, and hard to store all possible controllers in the control software (see more in [Blanke, M. and Kinnaert, M. and Lunze, J. and

Staroswiecki, M. (2003)]).

Because it is fast and real-time, fault accommodation is used in many applications. The method using fault accommodation is called fault adaptive control. Some recent works are about fault adaptive control. A switching hybrid model is used in [Abdelwahed, S. and Wu, J. and Biswas, G. and Ramirez, J. and Manders, E. (2005)] to represent the dynamics of the system components and their interactions, and a controller scheme is designed and implemented for efficient resource management in Advanced Life Support Systems. In [Ji, M. and Zhang, Z. and Biswas, G. and Sarkar, N. (2003)], a hierarchical control accommodation framework is developed. It provides switching stability among a set of trajectory tracking controllers. Fault-adaptive control can also be found in many applications, such as aircraft fuel systems [Karsai, G. and Biswas, G. and Pasternak, T. and Narasimhan, S. and Pecili, G. and Simon, G. and Kovacshazy, T. (2001)], [Simon, G. and Karsai, G. and Biswas, G. and Abdelwahed, S. and Mahadevan, N. and Szemethy, T. and Pecili, G. and Kovacshazy, T. (2003)],aircraft roll control systems [Simon, G. and Kovacshazy, T. and Pecili, G. and Szemethy, T. and Karsai, G. and Ledeczi, A. (2002)].

When fault accommodation is impossible, the entire control loop need to be reconfigured. The input-output relations between controller and plant are changed. Reconfiguration includes the selection of a new control configuration where alternative input and output signals are used. The selection of these signals depends upon the existing faults. A new control law has to be designed on-line. The necessity of control reconfiguration becomes obvious when sensor or actuator faults are considered. If these components fail completely, the fault leads to a breakdown of the control loop. There is no possibility to adapt the controller by simply adjusting its parameters. Instead, alternative actuators or sensors have to be found, which are not affected by the fault and which have similar interactions with the plant so that a reasonably selected controller is able to satisfy the performance specifications on the closed-loop system (see more in [Blanke, M. and Kinnaert, M. and Lunze, J. and Staroswiecki, M. (2003)]).

Although controller reconfiguration is more complicated than fault accommodation, it is still a hot topic in research and application for its strong adaptability. [Looze, D. and Weiss,

J. and Eterno, J. and Barrett, N. (2005)] focuses on the approach of automatic re-design of flight control system based on linear quadratic techniques, which maximizes the feedback system performance subject to a bandwidth constraint. [Elgersma, M. and Glavaški, S. (2001)] develops a distributed failure detection and isolation system for commuter and business aircraft. Nonlinear continuous/discrete systems are studied. The failure detection, isolation and recovery technique includes both discrete mode changes and continuous parameter values. [Liu, J. and Darabi, H. (2004)] develops a framework for reconfiguration of a discrete event system controller, which has a dynamic event observation set. Upon a change in the observation set, there is a mega-controller, which reconfigures the controller by a aggregation or disaggregation of the controller states.

1.3.2 Formal Verification

Starting from 1970's, formal methods for the design and analysis have been applied. In 1980's, they are firstly used in fault tolerance [Moitra, A. and Joseph, M. (1983)], [Schlichting, R. D. and Schneider, F. B. (1983)]. Formal verification is used in system design [Bernardeschi, C. and Fantechi, A. and Simoncini, L. (2000)], mostly in fault-tolerant computing systems, to confirm that fault tolerance is achieved. Examples of such systems include concurrent and real-time systems, communication networks and process control systems. For those systems, fault tolerance, as well as timing, has been thought to be implementing issues of them, different from the safety and liveness properties [Liu, Z. and Joseph, M. (1999)].

Currently, most researches of formal verification of fault tolerance are for fault tolerant computing. [Liu, Z. and Joseph, M. (1996)] shows how stepwise refinement [Abadi, M. and Lamport, L. (1988)], with the help of transformation, can be used for the development of fault-tolerant system, with or without timing constraints. The advantage of this approach is that the specification and verification techniques can be used for programs, as well as fault-tolerant systems. In [Liu, Z. and Joseph, M. (1999)], the authors study the relationship between fault tolerance and schedulability, because they affect each other and both affect the functionality and timing of the program. A framework is provided for such relation and for

formal development of safety-critical and/or timing critical computing systems. Two kinds of timers are used in the program, which makes the automated verification feasible. [Lincoln, P. and Rushby, J. (1993)] focuses on the algorithm for reliably distributing single-source data to multiple channels in the presence of faults. In [Ayache, S. and Conguet, E. and Humbert, P. and Rodriguez, C. and Sifakis, J. and Gerlich, R. (1996)], it tries to solve the problem of applicability of formal method to the world of space avionics computing. A general framework is proposed for the early validation of fault tolerance provisions in a complex computerized system.

1.4 Fault Detection and Fault Diagnosis

To maintain the correct performance, it is important to detect the faults as soon as possible, so that there will be enough time to apply corresponding measures before the faults turn into failures. Fault detection and diagnosis are important, since the faults in the components, such as sensors and actuators, are associated with potential damages and costs, which increase with time. Early and accurate detection of faults helps to avoid loss of human life and money. In fact, detection mechanisms alone can suffice to provide safety. Therefore, detection plays a crucial part in fault tolerance.

1.4.1 FMEA in System Design

Before thinking about the fault detection and diagnosis, there are some other things we need to consider first. What are the possible faults in the system? What are the effects of them? Which of them are critical for the dependability of the system? Which symptoms are essential for us to detect those faults? After those questions being answered, the step of detection can be developed without being aimless.

Other areas of system design employ standard patterns for design, and rules for the quality management of a development are well established. Fault-tolerant control development can benefit from similar procedures. The first step in the systematic design [Blanke, M. (1996)] is a component-based Failure Mode and Effects Analysis (FMEA). The FMEA analysis deals

with component faults and the propagation of fault effects. Components are such as sensors, controllers, and actuator motors. FMEA analysis is commonly required for safety-critical systems. The result of the FMEA analysis is thus a specification of which faults should be detected, and what reactions should be imposed on the system when certain patterns of fault effects are observed by the supervisor. The completeness of correctness properties of the design method are critical in daring to take this step. For more details, see [Blanke, M. and Izadi-Zamanabadi, R. and Bogh, S. A. and Lunau, C. P. (1997)].

1.4.2 Fault Detection

Fault detection, just as its name implies, is to indicate that something is wrong in the system. It is an old topic in design. A lot of fault detection researches have been done in dynamic systems. The first major survey is provided by Willsky in 1976 [Willsky, A. S. (1976)], which is later followed by some other good surveys [Isermann, R. (1984)], [Gertle, J. J. (1988)].

Fault detecting, simply speaking, tries to abstract useful information from the set of the system output. Usually, a fixed subset of the system output is used to detect some specific system faults. For example, an abrupt increase of current in one element is a potential indicator of the short circuit of that element.

One way to detect faults is to delicately choose a set of system output, such that, if a fault occurs, it can be detected by calculating the output, where some predefined system inputs are used. The difficulty in this method is to find an output set to detect a fault. In distributed systems, detection of predicates in faulty environment has studied [Garg, V. K. and Mitchell, J. R. (1998)], [Chase, C. M. and Garg, V. K. (1998)], [Gartner, F. C. and Kloppenburg, S. (2000)]. Another example is analytic redundancy. Till now, quite many practical examples have applied fault detection techniques, using analytical redundancy [Chow, E. Y. and Willsky, A. S. (1984)], [Cunningham, T. B. and Poyneer, R. D. (1977)], [Shapiro, E. Y. and Decarli, H. E. (1979)], [Stuckenberg, N. (1985)], [Merrill, M. C. (1985)]. Analytical redundancy (AR) [Chow, E. Y. and Willsky, A. S. (1984)] is a fault detection method that allows the explicit derivation of the maximum possible number of linearly independent system

model based consistency tests for a system [Leuschen, M. L. and Walker, I. D. and Cavallaro, J. R. (2005)]. Using a linear model of the system of interest, AR exploits the null-space of the state-space observability matrix to allow the creation of a set of test residuals [Chow, E. Y. and Willsky, A. S. (1984)]. Given a system model, residual is the difference between the system output and the model output. These residuals use sensor data histories and known control inputs to detect any deviation from the static or dynamic behaviors of the model in real time [Leuschen, M. L. and Walker, I. D. and Cavallaro, J. R. (2005)].

Another way is to artificially add some checks to the system, which will directly indicate the occurrence of faults. Some common techniques are:

- **Replication checks** Multiple replicas of a component are running simultaneously. The outputs of the replicas are compared and any discrepancy is an indication of a fault, supposing all the processes are deterministic. The particular form of this check, in hardware, is TMR, while, in software, it is N-version programming.

- **Timing checks** It is for timing faults. Timers are used to detect the termination of a process. If a timer times out, a timing faults happens. The choice of the time needs to be taken seriously, since setting the timer too tightly or too loosely may harm the accuracy of the detection.

- **Rum-time constraints checks** Certain constraints are set to some variables. For example, as long as the boundary values of variables are not being exceeded, there is no fault. But code and performance overhead will be introduced, since the variables and the boundaries are compared at run time.

1.4.3 Fault Diagnosis

Compared with fault detection, fault diagnosis is a more complex task, which consists of determining the type, size and location of the faults as well as its time of detection [Patton, R. J. and Frank, P. M. and Clark, R. N. (2000)]. For fault tolerant control, the location and the magnitude of the fault need to be determined in order to decide the appropriate way

of controller re-design or system reconfiguration. Fault diagnosis has been a research area for many years, and the theory is well established [Patton, R. J. and Frank, P. and Clarke, D. (1989)], [Gertler, J. (1995)]. [Patton, R. (1993)] gives an overview of available approaches, [Basseville, M. and Nikiforov, I. (1994)] treats the detection problem from a statistical point of view, [Sampath, M. and Sengupta, R. and Lafortune, S. and Sinnamohideen, K. and Teneketzis, D. C. (1996)] provides an analysis of fault-diagnostic control using discrete-event analysis.

Normally, diagnosis is composed of three steps:

- **Fault detection** Decide whether a fault has occurred. The time when the fault happens is determined in this step.
- **Fault isolation** Decide which component in the system has a fault. The location of the fault is determined.
- **Fault identification** Identify the fault and estimate its magnitude. This step is to determine what kind of fault has occurred.

In application, a dynamical system with input u and output y is subjected to some fault f. The system behavior depends on the fault $f \in \mathcal{F}$ where the element f_0 of the set $\mathcal{F}$ stands for the faultless case. The diagnostic system obtains the I/O pair (U,Y), which contains the sequences of input and output values sampled at discrete time points. The diagnostic problem to be solved is that, *for a given I/O pair (U,Y), find the fault f.* The problem concerns on-line diagnosis based on the available measurement data. No inspection on the process is possible. And the diagnostic problem has to meet real-time constraints accompanied with the system.

Most of the existing different diagnostic methods follow a common principle, called consistency-based diagnosis, which can be explained by using the notion of the system behavior. The idea of consistency-based diagnosis can be explained by means of Figure 1.3 [Blanke, M. and Kinnaert, M. and Lunze, J. and Staroswiecki, M. (2003)]. The behavior $\mathcal{B}$ is a subset of the space $\mathcal{U} \times \mathcal{Y}$ of all possible combinations of input and output signals. The dot A represents a single I/O pair, where $C = (u_C, y_C)$ represents a pair that is not consistent with the system

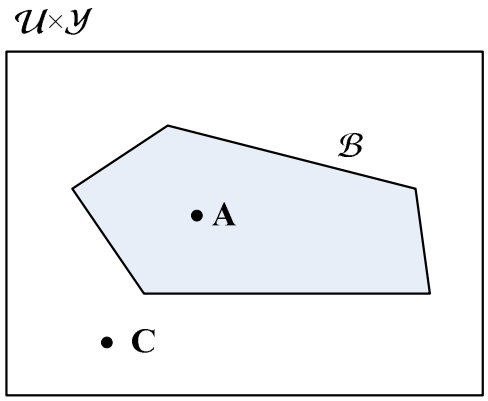

Figure 1.3 The graphical illustration of the system behavior

dynamics. That is, for the input u_C, the system produces an output $y \neq y_C$, then the fault is detectable. This is the principle of consistency-based diagnosis [Blanke, M. and Kinnaert, M. and Lunze, J. and Staroswiecki, M. (2003)].

The diagnostic method tests whether the measurement (U, Y) is consistent with the nominal system behavior. If the faulty system produces the I/O pair that is still in $\mathcal{B}$, no inconsistency occurs despite of the fault. Therefore the fault is not detectable. The question of whether a certain fault can be detected concerns the diagnosability of the system.

Fault diagnosis can be categorized into two methods: model based and non-model based. Model-based methods compare the observed behavior with model to detect the faults, while non-model based methods match the observed behavior to known faults. For discrete event systems, a certain model based approach for failure diagnosis is proposed in [Sampath, M. and Sengupta, R. and Lafortune, S. and Sinnamohideen, K. and Teneketzis, D. C. (1995)], and get extended in [Qiu, W. (2005)], [Debouk, R. and Lafortune, S. and Teneketzis, D. (2000)], [Jiang, S. and Kumar, R. (2002)], [Jiang, S. and Kumar, R. (2003)], [Jiang, S. and Kumar, R. and Garcia, H. E. (2003)], [Sampath, M. and Lafortune, S. (1998)], [Qiu, W. and Kumar, R. (2006)], [Zad, S. H. and Kwong, R. H. and Wonham, W. M. (2003)]. The application of DESs failure diagnosis includes HVAC systems [Sampath, M. and Sengupta, R. and Lafortune, S. and Sinnamohideen, K. and Teneketzis, D. C. (1996)], transportation systems [Lygeros, J. and Godbole, D. N. and Broucke, M. (2000)], [Godbole, D. N. and

Lygeros, J. and Singh, E. and Deshpande, A. and Lindsey, A. E. (2000)], communication networks [Beneveniste, A. and Fabre, E. and Haar, S. and Jard, C. (2003)], [Bouloutas, A. and Hart, G. W. and Schwartz, M. (1992)], [Miller, R. E. and Arisha, A. K. (2001)], manufacturing systems [Das, S. R. and Holloway, L. E. (2000)], [Pandalai, D. and Holloway, L. (2000)], digital circuits [Lin, F. (1994)], [Westerman, G. and Kumar, R. and Stroud, C. and Heath, J. R. (1998)], and power system [Hadjicostis, C. N. and Verghese, G. C. (2001)].

1.5 Organization of Dissertation

This dissertation is organized as follows:

In Chapter 2, necessary preliminaries and notations are introduced. We first introduce the definition of language, and the notations and operations about language. Then we give the definition of automaton, with a simple example. After that, we introduce the generated and marked language of an automaton. The definition of supervisor is also introduced. As the third part of the introduction, we give the definition of stability, including state stability and language stability.

In Chapter 3, we introduce a framework of fault-tolerant supervisory control of discrete event systems. We first formulate the definition of fault-tolerant supervisory control, and give a sufficient and necessary condition for the existence of a fault-tolerant supervisor. The condition involves the notion of controllability, observability and stability. An example of a simplified power system is provided. As an extension, we also develop a notion of weakly fault-tolerant supervisor control, which requires weaker conditions. A sufficient and necessary condition for the weakly fault-tolerant supervisor, and an example is provided. At the end of this chapter, we briefly introduce the notion of nonuniformly-bounded fault-tolerance as an extension.

In Chapter 4, we introduce the synthesizing algorithm to obtain a fault-tolerant supervisor. Since the previous chapter gives the existence condition, it is natural to formulate the algorithm to find a supervisor when the condition is satisfied. In this chapter, we synthesize an optimal fault-tolerant supervisor that maximizes the nonfaulty behavior of the controlled system and at the same time minimizes the faulty behavior. The complexity of this algorithm is quadratic

in the size of plant. A deliberately designed abstract example and an application example are provided to illustrate the algorithm.

In Chapter 5, we introduce the notion of safe-codiagnosability, as an extension of the notion of safe-diagnosability to decentralized setting, where multiple independent diagnosors can perform diagnosis without communication. Safe-codiagnosability requires that when a fault occurs, there exist at least one diagnosor that can detect the faulty event before the safety specification being violated. To verify safe-codiagnosability, we provide an algorithm with polynomial complexity.

In Chapter 6, we summarize this dissertation and discuss the possible research topics in the future.

CHAPTER 2. PRELIMINARIES AND NOTATIONS

In this section, we will introduce some prerequisite definitions, about automata and languages, and the notations that will be frequently used in the next chapters.

2.1 Languages

In the previous chapter, we introduced that discrete-event systems concern about the order of the states visited and the events that cause the state transitions. We assume that the behavior of the DES is described in terms of event sequences of the form $e_1 e_2 \cdots e_n$, where $e_i (i = 1, 2, \ldots, n)$ is the $\rangle$th event occurred. Such event sequence is called a *trace* or *string* of the systems. A collection of traces is called a *language*.

A string with no events is called an *empty string*, and is denoted by ϵ. The length of a string is the number of events contained in it. We use $|s|$ to denote the length of string s. Then, the length of an empty string is zero.

Language is defined over an event set. For example, given an event set $E = \{a, b, c\}$, a language can be defined as $L = \{\epsilon, a, abb, ca\}$. Let Σ^* denote the set of all finite length strings consisting of events from Σ, including empty string. Then, a language is a subset of Σ^*. For example, if $\Sigma = \{a, b, c\}$, then

$$\Sigma^* = \{\epsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, ca, aaa, \ldots\}$$

Given two languages $K_1, K_2 \subseteq \Sigma^*$, the following binary operations are available([Kumar, R. and Garg, V. K. (1995)]):

- The *intersection* of K_1 and K_2, denoted as $K_1 \cap K_2$, is the language

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

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

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

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

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

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

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

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

$$K_1/K_2 = \{s \in \Sigma^* | \exists t \in K_2 \text{ such that } st \in K_1\}.$$

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

$$K_1 \backslash K_2 = \{s \in \Sigma^* | \exists t \in K_2 \text{ such that } ts \in K_1\}.$$

If $uv = s$ with $u, v \in \Sigma^*$, then u is called a *prefix* of s, and v is called a *suffix* of s. u and v are also *substrings* of s. Note that ϵ and s are prefixes, suffixes and substrings of s, since $s = \epsilon s = s\epsilon$.

Consider a language $K \subseteq \Sigma^*$, the following unary operations are available[Kumar, R. and Garg, V. K. (1995)]:

- The *complement* of K, denoted as $K^c \subseteq \Sigma^*$, is the language

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

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

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

where $K^0 = \{\epsilon\}$, and for each $n \geq 0$, $K^{n+1} = K^n.K$.

- The *prefix closure*, denoted as $pr(K) \subseteq \Sigma^*$ (sometimes denoted also as $\bar{K}$), is the language

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

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

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

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

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

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

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

K is said to be *Kleene closed* if $K^* = K$; K is said to be *prefix closed* if $pr(K) = K$; K is said to be *extension closed* if $ext(K) = K$.

2.2 Automata

Automaton is also called state machine. An automaton is a device that is capable of representing a language according to well-defined rules [Cassandras, C. G. and Lafortune, S. (1999)]. The simplest and most understandable way to represent an automation is to use a directed graph, like Figure 1.1. Figure 1.1 gives an incomplete description of an automaton, where the set of the circles is the state set of the automaton, $X = \{1, 2, 3\}$. The set of the labels next to the arrows is the event set of the automaton. Let a be "move up", and b be "move down". Then the event set is $\Sigma = \{a, b\}$. The arrows represent the transition function of the automaton, which we denote as $\alpha : X \times E \to X$:

$$\alpha(1, a) = 2 \quad \alpha(2, b) = 1$$
$$\alpha(2, a) = 3 \quad \alpha(3, b) = 2$$

The notation $\alpha(1, a) = 2$ means that if the automaton is in state 1, the upon the occurrence of event a, the automaton will make an instantaneous transition to state 2.

There are two critical parts missing to complete the definition of an automaton, the initial state and the marked state(s). Suppose the building has an entrance at the first floor, and the office is located at the third floor. So people always starts from the first floor and goes to the third floor. Then, state 1 is the initial state, where the task of going to the office starts, and state 3 is the marked state, where the task completes. Note that the marked states may be multiple. Usually we use an arrow with no starting state to point to the initial state, and we use double cycles to represent the marked states. Figure 2.1 gives the complete representation of the automaton of the elevator.

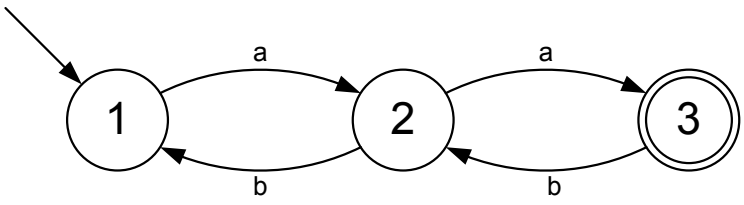

Figure 2.1 The complete automaton representing the elevator

Formally, an automaton consists of a state set, a finite set of events, a state transition function which describes the state(s) that are reached when a certain event occurs in a particular state, an initial state, and a set of marked states [Kumar, R. and Garg, V. K. (1995)]. So an automaton, denoted as G, is a 5-tuple:

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

where X denote the set of states of G, Σ is the (finite) event set of G, $\alpha : X \times (\Sigma \cup \{\epsilon\}) \rightarrow 2^X$ is the partial state transition function of G (it is a partial function since it is generally defined on a subset of $X \times (\Sigma \cup \{\epsilon\})$), $x_0 \in X$ is the initial state of G, and $X_m \subseteq X$ denotes the set of marked or accepting state of G [Kumar, R. and Garg, V. K. (1995)]. Here, a state transition on ϵ represents a hidden transition, also called an ϵ-move. Note that the state transition function does not uniquely determine the resulting state. In this case, G is called a *non-deterministic* automaton. And G is said to be *deterministic* if there is no ϵ-move and the transition function can uniquely determine the resulting state. Non-deterministic automata can be changed into deterministic automata. When Σ is an infinite set, the automaton is called an infinite automaton. My research is focused on deterministic finite automata. So, in the following, if not obviously mentioned, the automata are all finite and deterministic.

The *generated language* of $G = (X, \Sigma, \alpha, x_0, X_m)$ is

$$L(G) = \{s \in \Sigma^* | \alpha(x_0, s) \text{ is defined}\}.$$

The *marked language* of G is

$$L_m(G) = \{s \in L(G) | \alpha(x_0, s) \in X_m\}.$$

The language $L(G)$ contains all the traces that start from the initial state. A trace is in $L(G)$ if and only if it has a corresponding path in the transition graph, starting at x_0. The marked language $L_m(G)$ is a subset of $L(G)$. It contains all the traces that start from the initial

state and end in one of the marked state. Since the marked states are the desired destinations, the marked languages represent the completion of some certain tasks. The marked language is also called the *language recognized* by the automaton. And the automaton is therefore a *recognizer* of the given language.

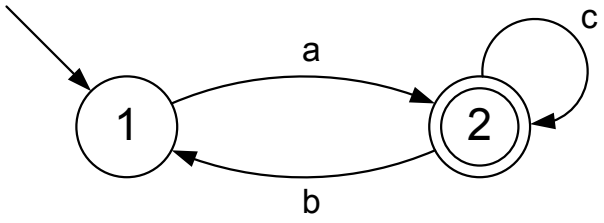

Figure 2.2 A simple example to show $L(G)$ and $L_m(G)$

Let's look at a simple example showing in Figure 2.2 above. There are two states, state 1 is the initial state, and state 2 is the marked state. And there are three transitions, a, b, and c. So, the generated language is

$$L = \{a, ac, acc, accc, \ldots, ab, aba, abac, abacc, abaccc, \ldots, acb, acba, acbac, acbacc, acbaccc, \ldots\},$$

and the marked language is

$$L_m = \{a, ac, acc, accc, \ldots, aba, abac, abacc, abaccc, \ldots, acba, acbac, acbacc, acbaccc, \ldots\}.$$

Given two automata $G_1 := (X_1, \Sigma, \alpha_1, x_{0_1}, X_{m_1})$ and $G_2 := (X_2, \Sigma, \alpha_2, x_{0_2}, X_{m_2})$, G_1 is said to be a subautomaton of G_2, denoted as $G_1 \sqsubseteq G_2$, if there exists an injective map $h : X_1 \to X_2$ such that $\forall s \in L(G_1) : h(\alpha_1(x_{0_1}, s)) = \alpha_2(x_{0_2}, s)$.

For traces s and t, we use $s \sqsubseteq_G t$ to denote that the sets of traces that occur in the generated and the marked languages of G after s are contained in those after t, i.e., $L(G)\backslash s \subseteq L(G)\backslash t$ and $L_m(G)\backslash s \subseteq L_m(G)\backslash t$. We write $s \cong_G t$ if $s \sqsubseteq_G t$ and $t \sqsubseteq_G s$. $s \cong_G t$ implies the equivalence of the behaviors following s and t, whereas $s \sqsubseteq_G t$ implies the behaviors following s are subsumed by the behaviors following t.

For control purposes, the event set of G is partitioned into the set of controllable events

$\Sigma_c \subseteq \Sigma$ and the set of uncontrollable events $\Sigma_u \subseteq \Sigma$. A language K is said to be controllable (with respect to G and Σ_u) if $pr(K)\Sigma_u \cap L(G) \subseteq pr(K)$. The events executed by the plant are filtered by an observation mask $M : \Sigma \rightarrow \Delta \cup \{\epsilon\}$ that maps the set of events to the set of "observed events" (Δ). A language K is said to be observable (with respect to G and mask M) if $\forall s, t \in pr(K), \sigma \in \Sigma : M(s) = M(t), s\sigma \in pr(K), t\sigma \in L(G) \Rightarrow t\sigma \in pr(K)$. A language K is said to be relative-closed with respect to G, if $pr(K) \cap L_m(G) = K \cap L_m(G)$.

A supervisor is another automaton $S := (Y, \Sigma, \beta, y_0, Y_m)$. The supervised plant is the synchronous composition of G and S, denoted $G||S := (X \times Y, \Sigma, \gamma, (x_0, y_0), X_m \times Y_m)$, where for $(x, y) \in X \times Y$ and $\sigma \in \Sigma$, $\gamma((x, y), \sigma)$ is defined if and only if both $\alpha(x, \sigma)$ and $\beta(y, \sigma)$ are defined and in which case, $\gamma((x, y), \sigma) = (\alpha(x, \sigma), \beta(y, \sigma))$. It can be concluded that the generated and the marked languages of the supervised plant are $L(G||S) = L(G) \cap L(S)$ and $L_m(G||S) = L_m(G) \cap L_m(S)$, respectively.

A supervisor S is said to be

1. *nonmarking* if $L_m(G||S) = L(G||S) \cap L_m(G)$,
2. *nonblocking* if $pr(L_m(G||S)) = L(G||S)$,
3. Σ_u-*compatible* if it does not disable any uncontrollable event (equivalently if $L(G||S)$ is controllable),
4. M-*compatible* if the controls following the indistinguishable traces are identical (equivalently if $L(G||S)$ is observable),
5. (Σ_u, M)-*compatible* if it is both Σ_u-compatible and M-compatible (See for example [Kumar, R. and Garg, V. K. (1995)]).

It is known that given a nonempty specification language $K \subseteq L_m(G)$, there exists a (Σ_u, M)-compatible, nonmarking and nonblocking supervisor if and only if K is relative-closed, controllable and observable [Lin, F. and Wonham, W. M. (1988)].

2.3 Stability

For discrete event systems, there are two forms of stability: state-stability, as introduced in [Brave, Y. and Heymann, M. (1990)], [Özveren, C. M. and Willsky, A. S. and Antsaklis, P. J. (1991)], and language-stability, as introduced in [Kumar, R. and Garg, V. K. and Marcus, S. I. (1993)], [Willner, Y. and Heymann, M. (1995)]. The notion of state-stability is first introduced below.

Given $\widehat{X} \subseteq X$, $x \in X$ is $\widehat{X}$-attractable in G if there exists a non-negative integer N such that for all traces t from x that are either deadlocking or have length greater than or equal to m, t visits $\widehat{X}$. $x \in X$ is controllably $\widehat{X}$-attractable in G if there exists a supervisor S such that x is $\widehat{X}$-attractable in $G \| S$. We use $\Omega_G(\widehat{X})$, called the region of attraction of $\widehat{X}$, to denote the set of all $\widehat{X}$-attractable states, and $\widehat{X}$ is called an attractor for the set $\Omega_G(\widehat{X})$. We use $\Omega^c_G(\widehat{X})$, called the region of controllable attraction of $\widehat{X}$, to denote the set of all controllably $\widehat{X}$-attractable states, and $\widehat{X}$ is called a controllable attractor for the set $\Omega^c_G(\widehat{X})$. A state set $\tilde{X} \subseteq X$ is said to be attractable to $\widehat{X}$ if $\tilde{X} \subseteq \Omega_G(\widehat{X})$ and controllably attractable to $\widehat{X}$ if $\tilde{X} \subseteq \Omega^c_G(\widehat{X})$. Clearly, $\widehat{X} \subseteq \Omega_G(\widehat{X}) \subseteq \Omega^c_G(\widehat{X})$.

A language $L \subseteq \Sigma^*$ is said to be *language-stable (ℓ-stable)* with respect to language $K \subseteq \Sigma^*$, or *converges* to K, if there exists $m \in \mathcal{N}$ such that for all $s \in L$ with $|s| \geq m$ or s deadlocks, exist $s' \leq s$ and $v \in K$ with $|s'| \leq m$ and $s = s'v$. In this case m is said to be the *delay-bound of convergence*. It follows from the definition that L is ℓ-stable with respect to K if for every trace $s \in L$ longer than m or is deadlocking, there exists a prefix of length at most m after which the corresponding suffix belongs to K. Further, a language $L \subseteq \Sigma^*$ is said to be *language-stabilizable (ℓ-stabilizable)* with respect to K, if there exists a supervisor S such that $L(G||S)$ is ℓ-stable with respect to K.

CHAPTER 3. A FRAMEWORK FOR FAULT-TOLERANT CONTROL OF DISCRETE EVENT SYSTEMS

In this chapter, we introduce our study on fault-tolerant supervisory control of discrete event systems. Given a plant, possessing both faulty and nonfaulty behavior, and a submodel for just the nonfaulty part, the goal of fault-tolerant supervisory control is to enforce a certain specification for the nonfaulty plant and another (perhaps more liberal) specification for the overall plant, and further to ensure that the plant recovers from any fault within a bounded delay so that following the recovery the system state is equivalent to a nonfaulty state (as if no fault ever happened). The specification for the overall plant is more liberal compared to the one for the nonfaulty part since a degraded performance may be allowed after a fault has occurred.

We formulate this notion of fault-tolerant supervisory control and provide a necessary and sufficient condition for the existence of such a supervisor. The condition involves the usual notions of controllability, observability and relative-closure, together with the notion of stability. An example of a power system is provided to illustrate the framework. We also propose a weaker notion of fault-tolerance where following the recovery, the system state is simulated by some nonfaulty state, i.e. behaviors following the recovery are also the behaviors from some faulty state.

Also, we formulate the corresponding notion of weakly fault-tolerant supervisory control and present a necessary and sufficient condition (involving the notion of language-stability) for the its existence. We also introduce the notion of nonuniformly-bounded fault-tolerance (and its weak version) where the delay-bound for recovery is not uniformly bounded over the set of faulty traces, and show that when the plant model has finitely many states, this more

general notion of fault-tolerance coincides with the one in which the delay-bound for recovery is uniformly bounded.

3.1 Introduction

In this chapter, we introduce a framework for *fault-tolerant supervisory control* of DESs. Given a plant G, possessing both faulty and nonfaulty behavior, and a submodel G^N for the nonfaulty part, the goal of fault-tolerant supervisory control is to enforce a certain specification K^N for the nonfaulty plant G^N and another (perhaps more liberal) specification $K \supseteq K^N$ for the overall plant G, and further to ensure that the plant recovers from any fault within a bounded delay, so that following the recovery the system state is equivalent to a nonfaulty state (as if no fault ever happened). A fault is modeled as an uncontrollable event, occurrence of which causes a transition from the nonfaulty part to the faulty part. The specifications K and K^N can be used to specify both the safety and the progress requirements. Since a degraded performance may be tolerable after the occurrence of a fault, the second specification is more liberal than the first one (and so it allows a larger set of traces).

In [Lafortune, S. and Lin, F. (1991)], authors considered a pair of specifications, representing the desired and the (more liberal) tolerable behavior for a plant G and proposed a general solution of their problem. In our setting, we also have two specifications: One is a desired behavior for the system without faults, and the other is a desired behavior for the system with faults. The control goal in our setting includes also the fault-tolerance: Other than meeting the respective specifications, the controller needs to ensure that a recovery takes following any fault within a bounded delay.

There has been some prior work on fault-tolerant control of DESs (see for example [Jensen, R. M. (2003)]). Some prior approaches involved controller switching upon the occurrence of a fault as in [Darabi, H. and Jafari, M. A. and Buczak, A. L. (2003)], or re-computation of a controller as in [Rohloff, K. R. (2005)]. The resulting controlled system can tolerate some faults but the system performance after faults can remain degraded since the notion of recovery from faults was not incorporated. Case studies involving synthesis of fault-tolerant

supervisors can also be found in [Cho, K. -H. and Lim, J. -T. (1996)], [Cho, K. -H. and Lim, J. -T. (1998)], [Zhou, M. C. and Dicesare, F. (1989)]. Design of certain coordination protocols for automated highway systems to achieve fault-tolerance under vehicle failures is reported in [Lygeros, J. and Godbole, D. N. and Broucke, M. (2000)], [Godbole, D. N. and Lygeros, J. and Singh, E. and Deshpande, A. and Lindsey, A. E. (2000)]. Takai et al. considered the problem of reliable decentralized supervisory control [Takai, S. and Ushio, T. (2000)], where they studied fault-tolerance with respect to the failures of the supervisors. Fault-tolerance in Petri Net is considered in [Iordache, M. V. and Antsaklis, P. J. (2004)], where liveness enforcing strategies are designed to deal with failures using system reconfigurations.

Here we consider the general problem of fault-tolerant supervisory control with fault recovery. A supervisor is used not only to enforce certain control specifications but also to ensure recovery following any fault. We formulate and study the above fault-tolerant control problem and provide a necessary and sufficient condition for the existence of such a supervisor. The condition involves the usual notions of controllability, observability and relative-closure, together with the notion of stability. The state-stability property is used to establish bounded delay recovery from a fault [Brave, Y. and Heymann, M. (1990)], [Özveren, C. M. and Willsky, A. S. and Antsaklis, P. J. (1991).

As mentioned above, by recovery we imply returning, within bounded delay, to a state that is equivalent to a nonfaulty state. In some applications, a weaker form of recovery may suffice where the behaviors following the recovery are also the behaviors from some nonfaulty state. Thus following the recovery, the system satisfies those properties that are also satisfied by the behaviors starting from some nonfaulty state. We study this weaker notion of fault-tolerant control, and give a necessary and sufficient condition for the existence of a weakly fault-tolerant supervisor. In contrast to the state-stability, the property of language-stability [Kumar, R. and Garg, V. K. and Marcus, S. I. (1993)], [Willner, Y. and Heymann, M. (1995)] is required.

We also introduce the notion of nonuniformly-bounded fault-tolerance (and its weak version) where the delay-bound for recovery is not uniformly bounded over the set of faulty traces, and show that when the plant model has finitely many states, this more general notion

of fault-tolerance coincides with the one in which the delay-bound for recovery is uniformly bounded.

The objective of our work is the synthesis of a fault-tolerant *controller* for a given plant, whereas the work in the computer science community (such as [Arora, A. and Gouda, M. (1993)], [Arora, A. and Kulkarni, S. S. (1998)], [Attie, P. C. and Arora, A. and Emerson, E. A. (2004)]) considers the design of a *system* (a *"plant"*) that is fault-tolerant. In [Arora, A. and Kulkarni, S. S. (1998)], [Attie, P. C. and Arora, A. and Emerson, E. A. (2004)], a system is said to be *nonmasking fault-tolerant* if after the occurrence of a fault the system specification is eventually satisfied. This is analogous to the notion of fault-tolerance we consider. [Arora, A. and Kulkarni, S. S. (1998)], [Attie, P. C. and Arora, A. and Emerson, E. A. (2004)] also considers the stronger notion of *masking fault-tolerance* which requires that the system specifications remain satisfied even after the occurrence of a fault. This stronger property can also be captured in our setting by requiring that the two specifications K and K^N represent the same property.

3.2 Fault-Tolerant Supervisory Control

In this section, we introduce a notion of fault-tolerant supervisory control. Consider a plant with model $G = (X, \Sigma, \alpha, x_0, X_m)$ which represents the behavior prior to as well as subsequent to faults, i.e., the overall behavior. Let the nonfaulty part of the plant G be modeled as $G^N = (X^N, \Sigma, \alpha^N, x_0, X_m^N)$. Without loss of generality, $G^N \sqsubseteq G$, i.e., G^N is a subautomaton of G. The pair (G, G^N) is said to be fault-tolerant if every post-fault behavior becomes equivalent to a nonfaulty behavior in a uniformly bounded delay. This property is captured as follows:

Definition 1 Given a plant G with its nonfaulty part G^N, (G, G^N) is said to be *fault-tolerant* if exists $m \in \mathcal{N}$ such that for $s \in L(G) - L(G^N), st \in L(G)$ with $|t| \geq m$ or st deadlocks, there exist $u \in L(G^N)$ and $t' \leq t$ with $|t'| \leq m$ and $st' \cong_G u$. In this case, m is called the *delay-bound of fault-tolerance.*

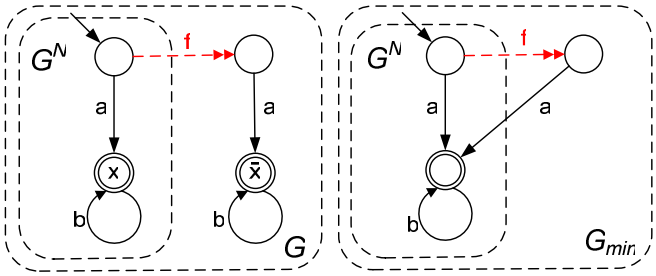

Figure 3.1 Automaton G and its corresponding G_{min}

The plant represented as the pair (G, G^N) is fault-tolerant if within a uniformly bounded delay of the occurrence of a fault, the plant state returns to a state equivalent to a nonfaulty state. Then the ensuing behavior is such that no fault ever happened. Therefore, after recovery, the system assumes full functionality. When the system model is minimal, i.e., possessing a minimal number of states, this means that following recovery, the system reaches a nonfaulty state. This, however, may not hold in general as shown in Figure 3.1, where G_{min} represents a minimal model of G. (The dashed edges represent uncontrollable transitions.) Following a fault, G_{min} recovers to a nonfaulty state in one transition and so clearly it is fault-tolerant. This is not the case for the model G but, being behaviorally equivalent to G_{min}, G is also fault-tolerant. (In Figure 3.1, state $\bar{x}$ is equivalent to nonfaulty state x.) Our definition is behavior-based and captures this situation.

The following example illustrates a fault-tolerant plant model.

Example 1 Consider the plant G and its nonfaulty part G^N shown in Figure 3.2 that models a machine. Initially the machine is idle and nonfaulty. The start event a transitions the machine to working and nonfaulty state, and the stop event b brings it back to the initial state. In the working and nonfaulty state, an occurrence of the fault event f causes the machine to transition to the working and faulty state. An execution of b at this state causes the machine to move to the idle and faulty state from where the repair event c brings the machine back to the initial state. An execution of the repair event in the idle and nonfaulty state does not change the machine state. It can be verified that G is minimal.

When there is an exit from the nonfaulty part due to the execution of f, a return within

a bounded delay is not guaranteed since there exists a cycle between the two faulty states. It follows that (G, G^N) is not fault-tolerant.

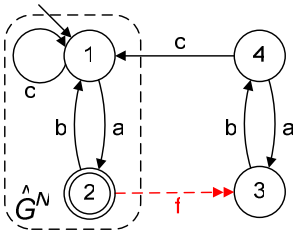

Figure 3.2 Plant G and its nonfaulty part G^N

The above notion of fault-tolerance is a type of state-stability property. This is established in the following theorem.

Theorem 1 Consider a plant $G = (X, \Sigma, \delta, x_0, X_m)$ and its nonfaulty part $G^N = (X^N, \Sigma, \delta^N, x_0, X_m^N)$, and suppose the corresponding minimal plant and its nonfaulty part are $G_{min} = (X_{min}, \Sigma, \delta_{min}, x_{0,min}, X_{m,min})$ and $G_{min}^N = (X_{min}^N, \Sigma, \delta_{min}^N, x_{0,min}, X_{m,min}^N)$ respectively. (G, G^N) is fault-tolerant if and only if X_{min} is attractable to X_{min}^N, i.e., $X_{min} \subseteq \Omega_{G_{min}}(X_{min}^N)$.

Proof: Since G_{min} and G_{min}^N are minimal models of G and G^N, $L(G_{min}) = L(G)$, $L(G_{min}^N) = L(G^N)$, and since for any $s, t \in L(G) = L(G_{min})$, $[s \cong_G t] \Leftrightarrow [s \cong_{G_{min}} t]$. Therefore, (G, G^N) is fault-tolerant if and only (G_{min}, G_{min}^N) is fault-tolerant.

Also note that $X_{min} \subseteq \Omega_{G_{min}}(X_{min}^N)$ is equivalent to $X_{min} - X_{min}^N \subseteq \Omega_{G_{min}}(X_{min}^N)$.

($\Rightarrow$) Pick $s \in L(G_{min}) - L(G_{min}^N)$. Since $s \in L(G_{min}) - L(G_{min}^N)$, exists $x \in X_{min} - X_{min}^N$ such that $\delta_{min}(x_0, s) = x$. Since $x \in X_{min} - X_{min}^N \subseteq \Omega_{G_{min}}(X_{min}^N)$, exists $m > 0$ such that, for all t, for which $\delta_{min}(x, t)$ is defined and either $|t| \geq m$ or $\delta_{min}(x, t)$ is deadlocking, exists $t' \leq t$ such that $\delta_{min}(x, t') \in X_{min}^N$. It shows that m is the desired delay bound for fault-tolerance. To see this, pick t such that $st \in L(G_{min})$ and either $|t| \geq m$ or st is deadlocking. Then $\delta_{min}(x, t') \in X_{min}^N$ for some $t' \leq t$. Let $u \in L(G_{min}^N)$ be such that $\delta_{min}(x_0, u) = \delta_{min}(x, t')$. Then u and st' reach the same state in G_{min}, so $u \cong_{G_{min}} st'$.

($\Leftarrow$) Now assuming (G_{min}, G^N_{min}) to be fault-tolerant, we establish that $X_{min} - X^N_{min} \subseteq \Omega_{G_{min}}(X^N_{min})$. Pick $x \in X_{min} - X^N_{min}$. Then there exists $s \in L(G_{min}) - L(G^N_{min})$ such that $\delta_{min}(x_0, s) = x$. From the fault-tolerance of (G_{min}, G^N_{min}), there exists $m > 0$ such that, for all t with $st \in L(G_{min})$ and either $|t| \geq m$ or st is deadlocking, exists $u \in L(G^N_{min})$ and $t' \leq t$ satisfying $u \cong_{G_{min}} st'$. From the minimality of G_{min}, equivalence of u and st' implies they reach the same state. Since $u \in L(G^N_{min})$, $\delta_{min}(x_0, st') = \delta_{min}(x_0, u) \in X^N_{min}$. It follows that, for each t such that $\delta_{min}(x, t)$ is defined, and $|t| \geq m$ or $\delta_{min}(x, t)$ is deadlocking, exist $t' \leq t$ such that $\delta_{min}(x, t') \in X^N_{min}$. This implies $X_{min} - X^N_{min} \subseteq \Omega_{G_{min}}(X^N_{min})$. ■

A given plant (G, G^N) may not be intrinsically fault-tolerant but could be made so through the use of control. This motivates us to formulate the notion of a fault-tolerant supervisor, which exercises appropriate control actions so that the controlled plant $(G||S, G^N||S)$ is fault-tolerant. The control actions of a fault-tolerant supervisor ensure that following any fault, a recovery takes place within a bounded number of steps, i.e., the controlled plant state returns to a state from where the future behaviors are such that as if no fault ever happened.

Definition 2 Given a plant G with its nonfaulty part G^N, a supervisor S is said to be *fault-tolerant* if $(G||S, G^N||S)$ is fault-tolerant.

The following example illustrates the notion of fault-tolerance of a supervisor.

Example 2 Consider the example of Figure 3.2. The start event a is controllable and disabled by a supervisor S at the idle and faulty state. The controlled systems $(G\|S, G^N\|S)$ is shown in Figure 3.3. It can be the seen that from any faulty state a return to some nonfaulty state is guaranteed within at most two transitions, i.e., $(G\|S, G^N\|S)$ is fault-tolerant, or equivalently, S is a fault-tolerant supervisor for (G, G^N).

The following corollary follows from Theorem 9.

Corollary 1 Given a plant G and its nonfaulty part G^N, and their corresponding minimal plant G_{min} and G^N_{min}, exists a fault-tolerant supervisor S if and only if $X_{min} \subseteq \Omega^c_{G_{min}}(X^N_{min})$.

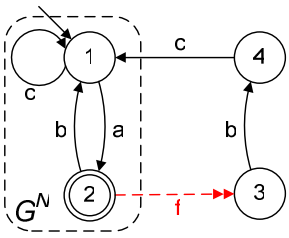

Figure 3.3 Controlled plant $(G\|S, G^N\|S)$

Proof: $X_{min} \subseteq \Omega^c_{G_{min}}(X^N_{min})$ if and only if exists supervisor S such that $X_{min} \subseteq \Omega_{G_{min}\|S}(X^N_{min})$, or equivalently exists supervisor S such that $(G_{min}\|S, G^N_{min}\|S)$ is fault-tolerant. (The last equivalence follows from Theorem 9.) ■

3.3 Existence of Fault-Tolerant Supervisor

The previous section formulated the notion of a fault-tolerant supervisor as one that ensures recovery from a fault within a bounded number of steps. In general, a supervisor needs to enforce certain other control specifications. For example in Figure 3.3, the plant possesses certain illegal and certain final states. The supervisor must also ensure that the illegal states are never visited while the final states are always reachable. To capture such control requirements, we use a pair of specification languages $K^N \subseteq L_m(G^N)$ and $K \subseteq L_m(G)$ satisfying $K^N \subseteq K$. Here K^N represents the control specification for the nonfaulty plant, and a different control specification, namely K, is used for the overall plant. This specification is taken to be "more liberal" $(K \supseteq K^N)$ since a downgraded performance may be tolerable after a fault has occurred.

Thus a fault-tolerant supervisory control problem is formulated as follows: Given a plant G, its nonfaulty part G^N, specifications K and K^N (with $K^N \subseteq K$), find a nonmarking, nonblocking, (Σ_u, M)-compatible and fault-tolerant supervisor S such that $L_m(G^N||S) = K^N$ and $L_m(G||S) = K$.

A necessary and sufficient condition for this is provided in the following theorem.

Theorem 2 Given a plant $G = (X, \Sigma, \alpha, x_0, X_m)$ with nonfaulty part $G^N = (X^N, \Sigma, \alpha^N, x_0, X^N_m)$, specification $\emptyset \neq K \subseteq L_m(G)$ for G and specification $\emptyset \neq K^N \subseteq$

$L_m(G^N)$ for G^N satisfying $K^N \subseteq K$, there exists a nonmarking, nonblocking (with respect to both G^N and G), (Σ_u, M)-compatible and fault-tolerant supervisor S such that

1. $L_m(G^N||S) = K^N$, $L(G^N||S) = pr(L_m(G^N||S))$, and
2. $L_m(G||S) = K$ and $L(G||S) = pr(L_m(G||S))$

if and only if

1. K is relative-closed, controllable and observable with respect to G,
2. In a minimal $R = (Q, \Sigma, \alpha, q_0, Q_m)$ and $R^N = (Q^N, \Sigma, \alpha^N, q_0, Q_m^N)$ with $R^N \sqsubseteq R$, $L_m(R^N) = K^N$, and $L_m(R) = K$, it holds that $Q \subseteq \Omega_R(Q^N)$,
3. $K^N = K \cap L_m(G^N)$, and $pr(K^N) = pr(K) \cap L(G^N)$.

Proof: From [Kumar, R. and Garg, V. K. (1995)], we know there exists a (Σ_u, M)-compatible, nonmarking and nonblocking supervisor S such that $L_m(G||S) = K$ and $L(G||S) = pr(K)$ if and only if K is relative-closed, controllable and observable with respect to G. R and R^N accept the same languages as $G||S$ and $G^N||S$, and they are minimal. From Theorem 9, we know $(G\|S, G^N\|S)$ is fault-tolerant if and only if in a minimal model $((G\|S)_{min}, (G^N\|S)_{min}) = (R, R^N)$ it holds that $Q \subseteq \Omega_R(Q^N)$.

So we only need to show that given $L_m(G\|S) = K$ and $L(G\|S) = pr(K)$, we have $L_m(G^N\|S) = K^N$ and $L(G^N\|S) = pr(K^N)$ if and only if K and K^N are constrained by $K^N = K \cap L_m(G^N)$ and $pr(K^N) = pr(K) \cap L(G^N)$. This follows from the following two series of equalities.

$$\begin{aligned} K^N &= L_m(G^N||S) \\ &= L_m(G^N) \cap L_m(S) \\ &= L_m(G^N) \cap L_m(S) \cap L_m(G) \\ &= L_m(G^N) \cap L_m(G||S) \\ &= L_m(G^N) \cap K, \text{ and} \end{aligned}$$

$$\begin{aligned} pr(K^N) &= L(G^N||S) \\ &= L(G^N) \cap L(S) \\ &= L(G^N) \cap L(S) \cap L(G) \\ &= L(G^N) \cap L(G||S) \\ &= L(G^N) \cap pr(K). \end{aligned}$$

■

Remark 1 In Condition 1 of Theorem 2, the relative-closure property can be checked in $O(|G||R|)$ [Kumar, R. and Garg, V. K. (1995)], the controllability property can be checked in $O(|G||R|)$, and observability property can be checked in $O(|G||R|^2)$. Condition 2 can be checked in $O(|Q|)$ [Brave, Y. and Heymann, M. (1990)]. Both parts of Condition 3 can be checked in $O(|G||R|)$. (The two parts of condition can be checked by checking in $G\|R$ whether $X_m^N \times Q_m \subseteq X_m^N \times Q_m^N$ and $X^N \times R \subseteq X^N \times R^N$, respectively.) Thus the overall complexity of verifying the condition of Theorem 2 is $O(|G||R|^2)$.

In Theorem 2 it is required that the supervisor be nonblocking with respect to both the nonfaulty plant and the overall plant. It turns out that due to the additional requirement of fault-tolerance, the requirement of nonblockingness with respect to the overall plant may be dropped, without altering the nature of the control problem. In other words, the following result holds.

Theorem 3 Given a plant $G = (X, \Sigma, \alpha, x_0, X_m)$ with nonfaulty part $G^N = (X^N, \Sigma, \alpha^N, x_0, X_m^N)$, there exists a nonblocking and fault-tolerant supervisor S such that $L(G^N||S) = pr(L_m(G^N||S))$ and $L(G||S) = pr(L_m(G||S))$ if and only if there exists a nonblocking and fault-tolerant supervisor S such that $L(G^N||S) = pr(L_m(G^N||S))$.

Proof: Clearly, the necessity ($\Rightarrow$) holds. We only need to prove the sufficiency ($\Leftarrow$) by showing $L(G||S) \subseteq pr(L_m(G||S))$.

To see this, pick any $s \in L(G||S)$. Then either $s \in L(G^N||S)$ in which case $s \in pr(L_m(G^N||S)) \subseteq pr(L_m(G||S))$, or $s \in L(G||S) - L(G^N||S)$. In the latter case, s is a faulty trace and let m be the delay-bound of fault-tolerance. Consider an extension t of s such that $st \in L(G||S)$, and $|t| \geq m$ or st deadlocks. In either case, there exists $u \in L(G^N||S) = pr(L_m(G^N||S))$ and $t = t'v$ such that $|t'| \leq m$ and $uv \in L(G||S)$. Since $u \in pr(L_m(G^N||S))$, there exists $v \in L_m(G^N||S)\backslash\{u\} \subseteq L_m(G||S)\backslash\{u\} = L_m(G||S)\backslash\{st'\}$. So it follows that $st' \in pr(L_m(G||S))$, which implies $s \in pr(L_m(G||S))$. ■

In Theorem 2, we allowed (K, K^N) to be an arbitrary pair of languages, and together they can capture both safety and liveness requirements. In some applications, the specification for the overall plant can simply be a safety specification, i.e., $K = pr(K)$. Theorem 2 can be specialized to address this situation resulting in the following corollary:

Corollary 2 Given a plant $G = (X, \Sigma, \alpha, x_0, X_m)$ with nonfaulty part $G^N = (X^N, \Sigma, \alpha^N, x_0, X_m^N)$, specification $\emptyset \neq K \subseteq L_m(G)$ for G and specification $\emptyset \neq K^N \subseteq L_m(G^N)$ for G^N satisfying $K^N \subseteq K$, there exists a nonmarking, nonblocking, (Σ_u, M)-compatible and fault-tolerant supervisor S such that

1. $L_m(G^N||S) = K^N$, $L(G^N||S) = pr(L_m(G^N||S))$, and
2. $L(G||S) = K$.

if and only if

1. K is prefix-closed, controllable and observable with respect to G,
2. In a minimal $R = (Q, \Sigma, \alpha, q_0, Q_m)$ and $R = (Q^N, \Sigma, \alpha^N, q_0, Q_m^N)$ with $R^N \sqsubseteq R$, $L_m(R^N) = K^N$ and $L(R) = K$, it holds that $Q - Q^N \subseteq \Omega_R(Q^N)$,
3. $K^N = K \cap L_m(G^N)$, and
4. $pr(K^N) = K \cap L(G^N)$.

The proof of Corollary 2 is similar to that of Theorem 2 and is omitted for brevity.

3.4 Application Example

We provide an application to illustrate our fault-tolerant control framework developed above by examining an abstracted model of a power system [Anderson, P. M. and Fouad, A. A. (1994)] shown in Figure 3.4. In this system, there are three generators (G_1, G_2 and G_3) and three loads (Load A, B and C), connected by nine buses (1 - 9).

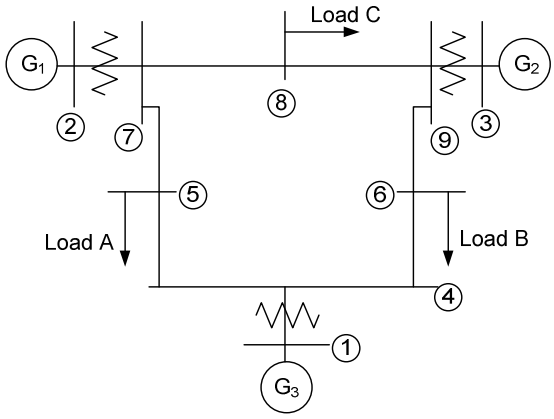

Figure 3.4 A 9-bus power system

The states of a power system are typically categorized into six different classes depending on the stability, safety, performance, and security margin properties they possess (as shown in the table below). *Stability* of the system refers to the stability of the individual generators. *Safety* refers to the line powers and bus voltages being within an acceptable threshold. The system is said to have *normal performance* when all load demands are met, and otherwise the performance is said to be *Degraded* (as some loads may be shed and some generators may be switched-off). *Security Margin* refers to the margin by which system load may be increased without violating stability or safety.

A *normal* state is one where system behavior is acceptable with respect to all four properties, and security margin is large enough that the occurrence of a single fault keeps the system behavior acceptable. An *alert* state is one where the system behavior is acceptable but security margin is small so that the occurrence of a single fault causes the system behavior to become unacceptable (i.e., one of the four properties may be violated). An *emergency* state is one where

Table 3.1 State categories

Category	Stable	Safe	Performance	Margin
Normal (N)	yes	yes	normal	large
Alert (A)	yes	yes	normal	small
Emergency (E)	yes	no	-	-
In-extremis (I)	no	-	-	-
Recovery (R)	yes	yes	Degraded	-
Failed (F)	-	-	None	-

stability is not violated but safety is violated, whereas an *in-extremis* is state is one where the stability is violated. A *recovery* state is one where safety and stability are not violated but the system performance is downgraded. Finally a *failed* state is one where the system is out-of service.

Some features of the power system shown in Figure 3.4 are listed below:

1. The feasible uncontrollable events are the two line-faults:

Table 3.2 List of faults in power system example

f_1	Power line between buses 5 & 7 faulted
f_2	Power line between buses 5 & 4 faulted

2. The feasible controllable events are:

Table 3.3 List of controllable events in power system example

c	Switch-on capacitor at bus 5
$\bar{c}$	Switch-off capacitor at bus 5
p_0	Switch-on power control at generators/loads prior to the critical time
p_1	Switch-on power control at generators/loads after the critical time
$\bar{p}$	Switch-off power control at generators/loads
r	Repair a faulted line (when system is stable)

3. States are abstracted to represent the values of the 4 binary state-variables: $\{s_1 s_2 s_3 s_4\}$, the meaning of which is listed as follows:

Table 3.4 Meaning of state variables in power system example

Variable	Meaning	1	0
s_1	Line 5-9 disconnected?	yes	no
s_2	Line 5-4 disconnected?	yes	no
s_3	capacitor control on?	yes	no
s_4	power control on?	yes	no

There is an additional special state, namely the "Failed" state denoted as "F", in which the system is out-of-service.

4. No control is exercised that takes the system to a "worse" state (for example, the control action of "switching off the capacitor" is not allowed in the alert state 1010 as this will cause a transition to the emergency state 1000).

5. We assume that no fault occurs in a recovery state.

The abstracted model of the power system is shown in Figure 3.5. In Figure 3.5,

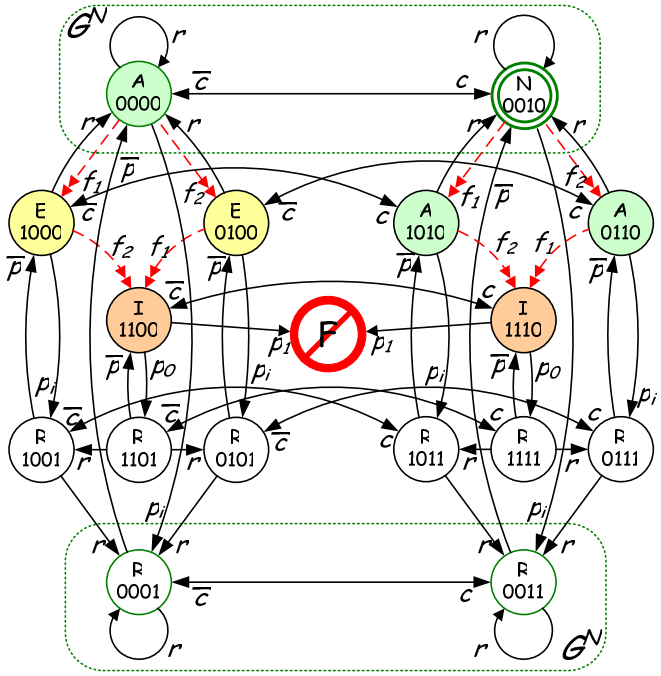

Figure 3.5 Model of power system of Figure 3.4

there are 16 "non-failed" states, and depending on their stability, safety, performance, and security margin properties are classified into one of five categories: normal (0010), alert (0000, 1010, 0110), recovery (---1), emergency (1000, 0100), and in-extremis (11-0). (Note that "-" represents a 0 or a 1.) In Figure 3.5 all solid edges are controllable and all dashed edges are uncontrollable. The only uncontrollable events are the two line-fault events. All events including the line-fault events are observable. (When a line-fault occurs, the circuit breakers at the ends of the line open-up to disconnect that line and that information can be used to determine the occurrence of a line-fault.)

The normal operation corresponds to no faults ($s_1 = s_2 = 0$), capacitive control on ($s_3 = 1$), and power control off ($s_4 = 0$). In this state the system behaves acceptable and has large security margin. Switching the capacitive control off or the occurrence of a single fault causes the security margin to become smaller and the system transitions to one of the alert states 0000 (when capacitive control is switched off), or 1010 (when fault f_1 occurs), or 0110 (when fault f_2 occurs). In state 0000 no fault has occurred but the capacitive control is off. Due to this, the occurrence of either fault f_1 or f_2 causes the system to violate safety (voltage at bus 5 dipping below a threshold) and transitions the system to an emergency state (1000 or 0100). On the other hand, in the normal state (0010), the occurrence of a single fault results in state (1010) or (0110), both of which offer acceptable behavior (no voltage dipping takes place due to the capacitive control being on). However, the occurrence of a second fault in one of these two alert states causes the system to transition to an in-extremis state (1110), where the system loses stability. Similarly, the occurrence of a second fault from one of the emergency states (1000 or 0100) causes the system to also lose stability, transiting it to an in-extremis state (1100). In an in-extremis state, system has lost stability and if the power control (generation/load shut-down) is not exercised in a timely fashion (before the "critical fault clearance" time), the system reaches the Failed state. Otherwise (if the power control is exercised in a timely fashion), the system acquires stability and safety but its performance is degraded (some generators/loads are tripped/shed). Thus a recovery state corresponds to a state where power control is on, i.e., $s_4 = 1$. A combination of repair of faulted lines and

re-energization of generators/loads causes the power system to recover to its normal operating mode.

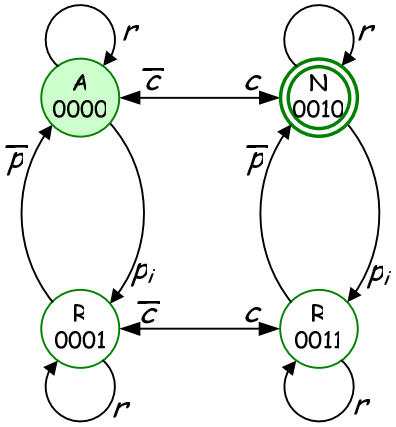

Figure 3.6 Nonfaulty part G^N of power system

In Figure 3.5, the overall plant model G has 17 states, and the nonfaulty part G^N consists of the states where no fault has occurred, i.e., the states with label (00–), which includes the normal state (0010), one of the alert states (0000), and two recovery states (0001, and 0011). The model for G^N is shown separately in Figure 3.6. The specification K for the overall plant excludes all traces that reach the Failed state, i.e., the Failed state is deemed forbidden for the overall plant. Among the four states of G^N, the normal state (0010) is deemed a final state, i.e., K^N excludes all traces of G^N that end at a non-final state. Clearly, $K^N \subseteq K$.

A fault-tolerant supervisor can be shown to exist and the controlled system under such a supervisor is shown in Figure 3.7. Each state is labeled with the maximal number of steps it takes to reach a state of G^N. As one can see, the system recovers in no more than 4 transitions, i.e, the delay-bound of recovery is 4 steps.

3.5 Weakly Fault-Tolerant Supervisory Control

In certain applications, a weaker form of recovery may suffice, namely, the recovery should cause the system to reach a state which is "simulated" by a nonfaulty state. That is, behaviors starting from a state after recovery be subsumed by those starting from a nonfaulty state. Then any safety and liveness property that the system satisfies starting from such a

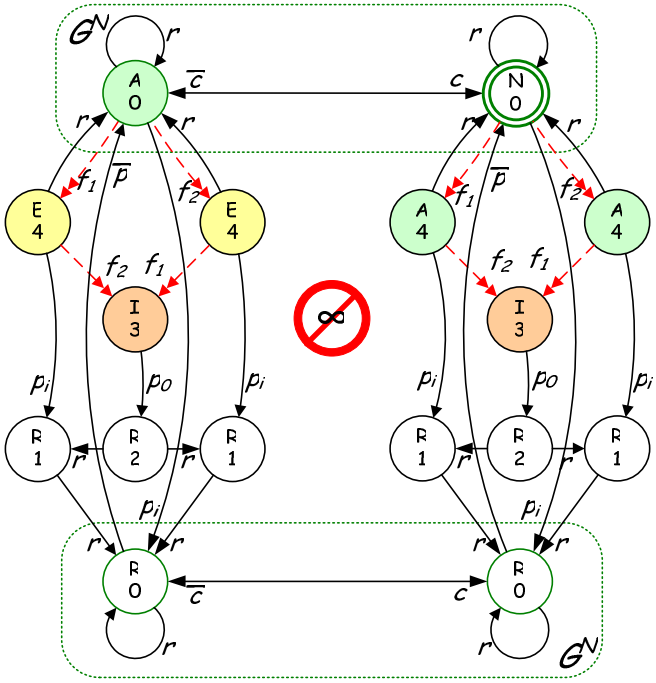

Figure 3.7 Supervised power system

nonfaulty state is also satisfied by the system starting from the state after recovery. This motivates us to introduce a weaker form of fault-tolerance. It turns out that this weaker form of fault-tolerance can be expressed as a second type of stability property, namely the language-stability property. The notion of weak fault-tolerance is captured as follows:

Definition 3 Given a plant G with its nonfaulty part G^N, (G, G^N) is said to be *weakly fault-tolerant* if exists $m \in \mathcal{N}$ such that for $s \in L(G) - L(G^N), st \in L(G)$ with $|t| \geq m$ or st deadlocks, exists $u \in L(G^N)$ and $t' \leq t$ with $|t'| \leq m$ and $st' \sqsubseteq_G u$. In this case, m is called the *delay-bound of weakly fault-tolerance*.

The following example illustrates a system that is weakly fault-tolerant but not fault-tolerant.

Example 3 Let us reconsider (G, G^N) shown in Figure 3.2, which as we discussed above is not fault-tolerant (due to the presence of the cycle in the faulty part). However after the transition on f, the state 3 is reached, from where all executable traces are also executable starting from

the nonfaulty state 2, i.e., the faulty state 3 is simulated by the nonfaulty state 2. To see this consider the behaviors following a faulty trace saf, where $s \in (c^* + (ab)^*)^*$. Then we claim that those behaviors are also possible following the nonfaulty trace sa, i.e., $saf \sqsubseteq_G sa$. The reason being that following the execution of saf, the system ultimately executes a trace in $(ba)^*bc$ which brings the system to state 2, whereas the execution of any trace in $(ba)^*bc$ following the trace sa also brings the system to state 2.

The above notion of weak fault-tolerance can be regarded as a type of language convergence property, as demonstrated by the next theorem. The basic idea is that the post-fault behavior, in a finite number of steps, matches a behavior that is a possible prior to the occurrence of a fault. In other words, the behaviors executable after a post-fault behavior are also executable after the nonfaulty behaviors. Since the matching of behavior following a finite execution is precisely the notion of language convergence, the weak fault-tolerance property can be described as a language convergence property. The faulty behavior is given by, $L(G) - L(G^N)$. The post-fault behavior has thus the following two parts, the generated part, $L(G)\backslash[L(G) - L(G^N)]$, and the marked part, $L_m(G)\backslash[L(G) - L(G^N)]$. Similarly the behavior after no fault has occurred also has two parts, namely $L(G)\backslash L(G^N)$ and $L_m(G)\backslash L(G^N)$. For the plant to be weakly fault-tolerant, traces in generated (resp., marked) post-fault behavior should converge to traces in generated (resp., marked) behavior prior to the occurrence of a fault. This is captured in the following theorem.

Theorem 4 Given a plant G and its nonfaulty part G^N, (G, G^N) is weakly fault-tolerant if and only if $L(G)\backslash[L(G) - L(G^N)]$ converges to $L(G)\backslash L(G^N)$ and $L_m(G)\backslash[L(G) - L(G^N)]$ converges to $L_m(G)\backslash L(G^N)$.

Proof: First we show that if (G, G^N) is weakly fault-tolerant, then there exists $m \in \mathcal{N}$ such that for each $t \in L(G)\backslash[L(G) - L(G^N)]$ with either $|t| \geq m$ or t deadlocks, there exists $t' \leq t$ and $v \in L(G)\backslash L(G^N)$ such that $t = t'v$ and $|t'| \leq m$. We claim that m can be chosen to be the delay-bound of weak fault-tolerance.

Since $t \in L(G)\backslash[L(G) - L(G^N)]$, there exists $s \in L(G) - L(G^N)$ such that $st \in L(G)$. Further since $|t| \geq m$ or t deadlocks, from fault-tolerance of (G, G^N), there exists $u \in L(G^N)$ and $t = t'v$: ($|t'| \leq m$ and $st' \sqsubseteq_G u$). Since $st'v \in L(G)$ and $u \in L(G^N)$, it follows that $uv \in L(G)$, therefore $v \in L(G)\backslash L(G^N)$, as desired.

Similarly, it can be shown that if (G, G^N) is weakly fault-tolerant with delay-bound of fault-tolerance m, then $t \in L_m(G)\backslash[L(G) - L(G^N)]$ with $|t| \geq m$ or t deadlocks, implies $t = t'v$ with $|t'| \leq m$ and $v \in L_m(G)\backslash L(G^N)$. As above there exists $s \in L(G) - L(G^N)$ such that $st \in L_m(G)$. Invoking the fault-tolerance property and noting that $st \in L_m(G)$, we can conclude that $t = t'v$ such that $|t'| \leq m$ and $uv \in L_m(G)$ for some $u \in L(G^N)$. Since $u \in L(G^N)$ and $uv \in L_m(G)$, it follows that $v \in L_m(G)\backslash L(G^N)$, as desired.

In order to prove the sufficiency, we show that the delay-bound m of fault-tolerance can be chosen to be the same as the delay-bound of convergence of $L(G)\backslash[L(G) - L(G^N)]$ to $L(G)\backslash L(G^N)$ and of $L_m(G)\backslash[L(G) - L(G^N)]$ to $L_m(G)\backslash L(G^N)$.

For $s \in L(G) - L(G^N)$ pick an extension t such that $st \in L(G)$ and either $|t| \geq m$ or st deadlocks. Then $t \in L(G)\backslash[L(G) - L(G^N)]$ and from the convergence of $L(G)\backslash[L(G) - L(G^N)]$ to $L(G)\backslash L(G^N)$, we have $t = t'v$ with $|t'| \leq m$ and $v \in L(G)\backslash L(G^N)$. The latter further implies the existence of $u \in L(G^N)$ such that $uv \in L(G)$. Therefore, $L(G)\backslash st' \subseteq L(G)\backslash u$, as desired.

It remains to show that if $st \in L_m(G)$, then $uv \in L_m(G)$. This requires the second convergence property. First note that $st \in L_m(G)$ and $s \in L(G) - L(G^N)$ implies $t \in L_m(G)\backslash[L(G) - L(G^N)]$. So from convergence of $L_m(G)\backslash[L(G) - L(G^N)]$ to $L_m(G)\backslash L(G^N)$, it follows that $t = t'v$ with $|t'| \leq m$ and $v \in L_m(G)\backslash L(G^N)$. $v \in L_m(G)\backslash L(G^N)$ further implies the existence of $u \in L(G^N)$ such that $uv \in L_m(G)$. Therefore, $L_m(G)\backslash st' \subseteq L_m(G)\backslash u$, as desired. ■

A given plant (G, G^N) may not be intrinsically weakly fault-tolerant but could be made so through the use of control. This motivates us to formulate the notion of a weakly fault-tolerant supervisor, which exercises appropriate control actions so that the controlled plant $(G||S, G^N||S)$ is weakly fault-tolerant. The control actions of a weakly fault-tolerant supervisor

ensure that following any fault, a recovery takes place within a bounded number of states, i.e., the controlled plant reaches a state, starting from where the executable behaviors are also executable starting from a nonfaulty state.

Definition 4 Given a plant G with its nonfaulty part G^N, a supervisor S is said to be *weakly fault-tolerant* if $(G||S, G^N||S)$ is weakly fault-tolerant.

The following corollary follows from Theorem 4.

Corollary 3 Given a plant G and its nonfaulty part G^N, a supervisor S is weakly fault-tolerant if and only if $L(G||S)\backslash[L(G||S) - L(G^N||S)]$ converges to $L(G||S)\backslash L(G^N||S)$ and $L_m(G||S)\backslash[L(G||S) - L(G^N||S)]$ converges to $L_m(G||S)\backslash L(G^N||S)$.

3.6 Existence of Weakly Fault-Tolerant Supervisor

Using the results of the previous section, we next present a condition for the existence of weakly fault-tolerant supervisor. As before, the supervisor is required to impose a specification K^N on the nonfaulty plant G^N and a possibly more liberal specification $K \supseteq K^N$ on the overall plant. We have the following existence result.

Theorem 5 Given a plant $G = (X, \Sigma, \alpha, x_0, X_m)$ with nonfaulty part $G^N = (X^N, \Sigma, \alpha^N, x_0, X_m^N)$, specification $\emptyset \neq K \subseteq L_m(G)$ for G and specification $\emptyset \neq K^N \subseteq L_m(G^N)$ for G^N satisfying $K^N \subseteq K$, there exists a nonmarking, nonblocking (with respect to both G^N and G), (Σ_u, M)-compatible and weakly fault-tolerant supervisor S such that

1. $L_m(G^N||S) = K^N$, $L(G^N||S) = pr(L_m(G^N||S))$, and
2. $L_m(G||S) = K$ and $L(G||S) = pr(L_m(G||S))$

if and only if

1. K is relative-closed, controllable and observable with respect to G,
2. $pr(K)\backslash[pr(K) - pr(K^N)]$ converges to $pr(K)\backslash pr(K^N)$, and $K\backslash[pr(K) - pr(K^N)]$ converges to $K\backslash pr(K^N)$,

3. $K^N = K \cap L_m(G^N)$, and $pr(K^N) = pr(K) \cap L(G^N)$.

Proof: From [Kumar, R. and Garg, V. K. (1995)], we know there exists a (Σ_u, M)-compatible, nonmarking and nonblocking supervisor S such that $L_m(G||S) = K$ and $L(G||S) = pr(K)$ if and only if K is relative-closed, controllable and observable with respect to G. Also from Theorem 9, we know S is weakly fault-tolerant such that $L_m(G^N||S) = K^N$, $L(G^N||S) = pr(K^N)$, $L_m(G||S) = K$ and $L(G||S) = pr(K)$ if and only if $pr(K)\backslash[pr(K) - pr(K^N)]$ converges to $pr(K)\backslash pr(K^N)$ and $K\backslash[pr(K) - pr(K^N)]$ converges to $K\backslash pr(K^N)$.

So we only need to show given $L_m(G||S) = K$ and $L(G||S) = pr(K)$, we have $L_m(G^N||S) = K^N$ and $L(G^N||S) = pr(K^N)$ if and only if K and K^N are constrained by $K^N = K \cap L_m(G^N)$ and $pr(K^N) = pr(K) \cap L(G^N)$. This is something we proved as part of the proof of Theorem 2. ■

Remark 2 The complexity of checking conditions 1 and 3 of Theorem 5 are discussed in Remark 1. Here we only discuss the complexity of checking the second condition. The languages appearing in the second condition can be recognized using automata of size $O(|Q|)$. This is because the automaton $(Q, \Sigma, \delta, Q - Q^N, Q_m)$ (i.e., R with its initial state replaced by the set $Q - Q^N$) generates the language $pr(K)\backslash[pr(K) - pr(K^N)]$ and marks the language $K\backslash[pr(K) - pr(K^N)]$, whereas the automaton $(Q, \Sigma, \delta, Q^N, Q_m)$ (i.e., R with its initial state replaced by the set Q^N) generates the language $pr(K)\backslash pr(K^N)$ and marks the language $K\backslash pr(K^N)$.

Note that the automata $(Q, \Sigma, \delta, Q - Q^N, Q_m)$ and $(Q, \Sigma, \delta, Q^N, Q_m)$ are nondeterministic due to the non-uniqueness of the initial state. In order to verify the language convergence properties of Condition 2, the algorithm given in [Willner, Y. and Heymann, M. (1995)] can be adapted to the nondeterministic setting to verify whether the language $K\backslash[pr(K) - pr(K^N)]$ marked by $(Q, \Sigma, \delta, Q - Q^N, Q_m)$ converges to the language $K\backslash pr(K^N)$ marked by $(Q, \Sigma, \delta, Q^N, Q_m)$ as follows: We construct the automaton $T := (Z, \Sigma, \gamma, Z_0, Z_m)$, where

$$Z = Q \times 2^Q,$$
$$Z_0 = \{(q, Q^N) | q \in Q - Q^N\},$$

$Z_m = \{(q, \widehat{Q}) | q \in Q_m, \widehat{Q} \cap Q_m \neq \emptyset\}$, and

$\forall q \in Q, \widehat{Q} \subseteq Q, \sigma \in \Sigma : \gamma((q, \widehat{Q}), \sigma) := (\delta(q, \sigma), \delta(\widehat{Q}, \sigma) \cup Q^N)$.

Let $\widehat{Z} := \{(q, \widehat{Q}) \mid L_m(R, q) \subseteq L_m(R, \widehat{Q})\}$, where $L_m(R, q)$ is the language marked by R starting from the state q, and $L_m(R, \widehat{Q}) := \cup_{\widehat{q} \in \widehat{Q}} L_m(R, \widehat{q})$. Note $L_m(T) = L_m((Q, \Sigma, \delta, Q - Q^N, Q_m)) = K \backslash [pr(K) - pr(K^N)]$, and a trace $t \in L_m(T)$ ends at a state $(q, \widehat{Q}) \in \widehat{Z}$ if and only if t possesses a suffix $v \in L_m((Q, \Sigma, \delta, Q^N, Q_m)) = K \backslash pr(K^N)$. Then following the results in [Willner, Y. and Heymann, M. (1995)] it can be shown that $K \backslash [pr(K) - pr(K^N)]$ converges to $K \backslash pr(K^N)$ if and only if $Z \subseteq \Omega(\widehat{Z})$. Next by simply replacing Q_m with Q in the convergence test just described, we can obtain a test for verifying whether $pr(K) \backslash [pr(K) - pr(K^N)]$ converges to $pr(K) \backslash pr(K^N)$. The complexity of checking the language convergence properties of Condition 2 can accordingly concluded to be $O(|Q|^2 (2^{|Q|})^2)$.

In Theorem 5 it is required that the supervisor be nonblocking with respect to both the nonfaulty plant and the overall plant. It turns out that due to the additional requirement of fault-tolerance, the requirement of nonblockingness with respect to the overall plant may be dropped, without altering the nature of the control problem. In other words, the following result holds.

Theorem 6 Given a plant $G = (X, \Sigma, \alpha, x_0, X_m)$ with nonfaulty part $G^N = (X^N, \Sigma, \alpha^N, x_0, X_m^N)$, there exists a nonblocking and fault-tolerant supervisor S such that $L(G^N||S) = pr(L_m(G^N||S))$ and $L(G||S) = pr(L_m(G||S))$ if and only if there exists a nonblocking and fault-tolerant supervisor S such that $L(G^N||S) = pr(L_m(G^N||S))$.

Proof: Clearly, the necessity ($\Rightarrow$) holds. We only need to prove the sufficiency ($\Leftarrow$) by showing $L(G||S) \subseteq pr(L_m(G||S))$.

To see this, pick any $s \in L(G||S)$. Then either $s \in L(G^N||S)$ in which case $s \in pr(L_m(G^N||S)) \subseteq pr(L_m(G||S))$, or $s \in L(G||S) - L(G^N||S)$. In the latter case, s is a faulty trace and let m be the delay-bound of fault-tolerance. Consider an extension t of s such that $st \in L(G||S)$. Without loss of generality, $|t| \geq m$ or st deadlocks. In either case, there exists $u \in L(G^N||S) = pr(L_m(G^N||S))$ and $t = t'v$ such that $|t'| \leq m$ and $uv \in L(G||S)$. Since

$u \in pr(L_m(G^N||S))$, there exists $v \in L_m(G^N||S)\backslash\{u\} \subseteq L_m(G||S)\backslash\{u\} = L_m(G||S)\backslash\{st'\}$. So it follows that $st' \in pr(L_m(G||S))$, which implies $s \in pr(L_m(G||S))$. ■

The following corollary is a specialization of Theorem 5, where the specification K for the overall plant is simply a safety specification (so that $K = pr(K)$).

Corollary 4 Given a plant $G = (X, \Sigma, \alpha, x_0, X_m)$ with nonfaulty part $G^N = (X^N, \Sigma, \alpha^N, x_0, X_m^N)$, specification $\emptyset \neq K \subseteq L_m(G)$ for G and specification $\emptyset \neq K^N \subseteq L_m(G^N)$ for G^N satisfying $K^N \subseteq K$, there exists a nonmarking, nonblocking, (Σ_u, M)-compatible and weakly fault-tolerant supervisor S such that

1. $L_m(G^N||S) = K^N$, $L(G^N||S) = pr(L_m(G^N||S))$, and
2. $L(G||S) = K$.

if and only if

1. K is prefix-closed, controllable and observable with respect to G,
2. $K\backslash[K - pr(K^N)]$ converges to $K\backslash pr(K^N)$,
3. $K^N = K \cap L_m(G^N)$, and $pr(K^N) = K \cap L(G^N)$.

3.7 Application Example (Continued)

In order to illustrate weakly fault-tolerant control we revisit the power system considered in Section 4.6. We relax one of the assumptions that a fault cannot occur in one of the recovery states. The corresponding revised model of the power system is shown in Figure 3.8. Note the revised model has extra fault transitions.

It can be verified that reaching the recovery state 1101 or 1111 cannot be avoided starting from a nonfaulty state. If the initial nonfaulty state is the normal state or the alert state, then an in-extremis state 1110 or 1100 is reached uncontrollably (through a sequence of two faults), from where the control action p_0 must be executed to ensure the unreachability of the failed state causing the system to reach either 1101 or 1111. On the other hand if the initial

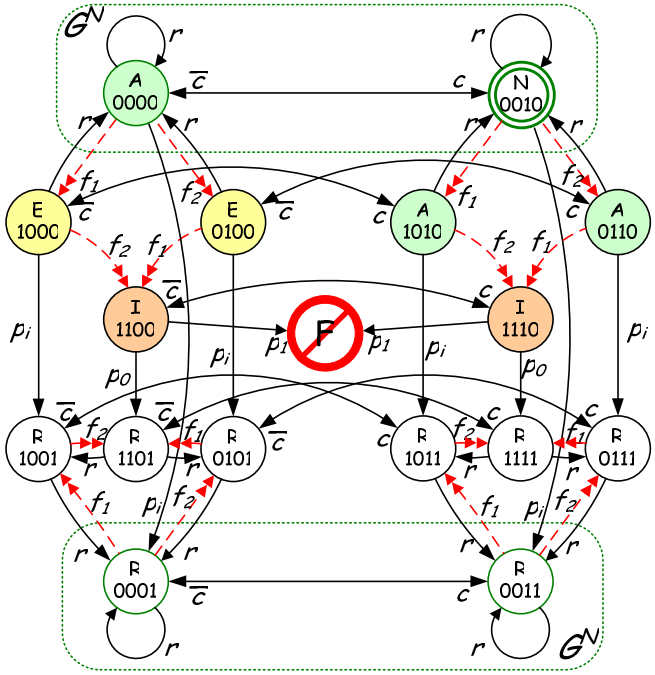

Figure 3.8 Revised model of power system of Figure 3.4

nonfaulty state is a recovery state (0001 or 0011), the recovery state 1101 or 1111 is reached uncontrollably (through a sequence of two faults).

Consider any of the two recovery states, say 1101. If a controller disables both the repair events at this state, then the system deadlocks and recovery to the nonfaulty part does not occur. Similarly if one of the recovery events is not disabled, the state 1101 becomes part of a cycle of faulty states and so a bounded-delay recovery to nonfaulty states is not guaranteed. It follows that there does not exists a control so that the controlled system is fault-tolerant.

It turns out that there exists a control so that the controlled system is weakly fault-tolerant. Such a controlled system is shown in Figure 3.9. The controller disables the c and $\bar{c}$ transitions between the emergency and the alert states, and also all the $\bar{p}$ transitions. Due to the presence of the cycles between the recovery states of the faulty part, the controlled system is not fault-tolerant. However the controlled system is weakly fault-tolerant since the traces executable from a recovery state in the faulty part (state 1001 or 1101 or 0101 or 1011 or 1111 or 0111) are also executable from one of the recovery states of the nonfaulty part (state 0001 or 0011), and from any faulty state a recovery state of the faulty part is reached within a bounded-delay.

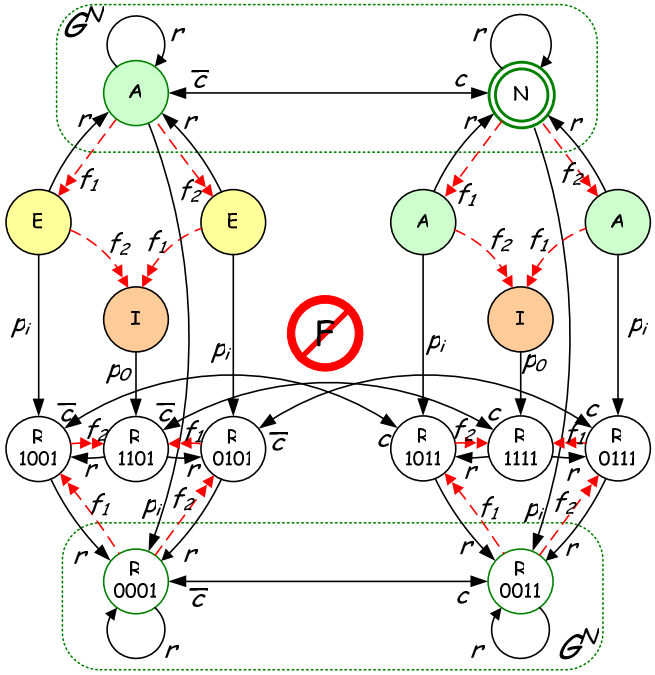

Figure 3.9 Supervised power system that is weakly fault-tolerant

3.8 Nonuniformly Bounded Fault-Tolerance

The notion of fault-tolerance we proposed requires the recovery to occur within a uniformly bounded delay. A relaxed notion of fault-tolerance is one where the delay bound for recovery is finite but not necessarily uniformly bounded over all faulty traces. The following definition formalizes the notions of nonuniformly-bounded fault-tolerance.

Definition 5 Given a plant G with its nonfaulty part G^N, (G, G^N) is said to be *nonuniformly-bounded fault-tolerant* if for $s \in L(G) - L(G^N)$, exists $m \in \mathcal{N}$ such that for $st \in L(G)$ with $|t| \geq m$ or st deadlocks, exist $u \in L(G^N)$ and $t' \leq t$ with $|t'| \leq m$ and $st' \cong_G u$. (G, G^N) is said to be *nonuniformly-bounded weakly fault-tolerant* if for $s \in L(G) - L(G^N)$, exists $m \in \mathcal{N}$ such that for $st \in L(G)$ with $|t| \geq m$ or st deadlocks, exist $u \in L(G^N)$ and $t' \leq t$ with $t' \leq t$ and $st \sqsubseteq_G u$.

Note in the above definition the delay bound m of fault-tolerance is a function of the faulty trace $s \in L(G) - L(G^N)$. While this is more general than the uniformly-bounded case

considered earlier, we show below that when G and G^N are finite-automata, the two notions are equivalent.

Theorem 7 Suppose the plant G and its nonfaulty part G^N are finite automata. Then

1. (G, G^N) is nonuniformly-bounded fault-tolerant if and only if (G, G^N) is fault-tolerant.
2. (G, G^N) is nonuniformly-bounded weakly fault-tolerant if and only if (G, G^N) is weakly fault-tolerant.

Proof: We begin by proving the first part. Without loss of generality G is assumed to be minimal. We show that when (G, G^N) is nonuniformly-bounded fault-tolerant, it holds that $X - X^N \subseteq \Omega(X^N)$ (which from Theorem 9 is equivalent to (G, G^N) being fault-tolerant). Suppose for contradiction that this is not true. Then either there exists a cycle of states in $X - X^N$ or some state in $X - X^N$ is a deadlocking state. Let $s \in L(G) - L(G^N)$ be a trace that ends on such a cycle or at such a deadlocking state. In the former case (when s ends on a cycle), for every $m \in \mathcal{N}$ exists an extension $st \in L(G)$ along the cycle such that all states visited beyond the trace s belong to $X - X^N$. From minimality of G, exists no $t' \leq t$ such that $st' \cong_G u$ for some $u \in L(G^N)$, contradicting the fact that (G, G^N) is nonuniformly-bounded fault-tolerant. The same conclusion is obtained in the latter case (when s ends at a deadlocking state). This completes the proof of the first part.

To prove the second part, we consider the automaton $T := (Z, \Sigma, \gamma, Z_0, Z_m)$ constructed in Remark 2 and show that if (G, G^N) is nonuniformly-bounded weakly fault-tolerant, then $Z \subseteq \Omega(\widehat{Z})$, which as mentioned in Remark 2 is equivalent to the fact that $L_m(G)\backslash[L(G) - L(G^N)]$ converges to $L_m(G)\backslash L(G^N)$. (The convergence of $L(G)\backslash[L(G) - L(G^N)]$ to $L(G)\backslash L(G^N)$ can be proved similarly by setting $Q_m = Q$ in the definition of the automaton $T = (Z, \Sigma, \gamma, Z_0, Z_m)$ and establishing that $Z \subseteq \Omega(\widehat{Z})$.) Note together these two language convergence properties are equivalent to (G, G^N) being weakly fault-tolerant (see Theorem 4).

Suppose for contradiction that $Z \not\subseteq \Omega(\widehat{Z})$. Then either there exists a cycle of states belonging to $Z - \widehat{Z}$ or some state in $Z - \widehat{Z}$ is a deadlocking state. In the former case, for any $m \in \mathcal{N}$, exists trace $t \in L_m(T) = L_m((Q, \Sigma, \delta, Q - Q^N, Q_m)) = L_m(G)\backslash[L(G) - L(G^N)]$ with

$|t| \geq m$ such that no suffix of t belongs to $L_m((Q, \Sigma, \delta, Q^N, Q_m)) = L_m(G) \backslash L(G^N)$. So exists $s \in L(G) - L(G^N)$ such that $st \in L_m(G)$ and t is arbitrarily long, yet there do not exist suffix v of t and trace $u \in L(G^N)$ such that $uv \in L_m(G)$. This is a contradiction to the fact that (G, G^N) is nonuniformly-bounded weakly fault-tolerant. The same conclusion can be arrived at even in the latter case when $Z - \widehat{Z}$ possesses a deadlocking state. This completes the proof. ■

The following example shows that in general nonuniformly-bounded fault-tolerance is weaker than the uniformly-bounded case.

Example 4 Consider a plant G and its nonfaulty part G^N shown in Figure 3.10 with $L(G) = \cup_{n \geq 1} pr(a^n f b^n)$, $L_m(G) = \{\epsilon\} \cup_{n \geq 1} a^n f b^n$, $L(G^N) = a^*$, and $L_m(G^N) = \emptyset$, where f represents the faulty event. Then $L(G) - L(G^N) = \cup_{n \geq 1, m \leq n} a^n f b^m$. Pick $s_n = a^n f \in L(G) - L(G^N)$. Then $L_m(G) \backslash \{s_n\} = \{b^n\} \subseteq L_m(G) \backslash [L(G) - L(G^N)]$. The only suffix of b^n that is equivalent to a trace in $L(G^N) = a^*$ is the ϵ trace. So the delay-bound of fault-tolerance for the trace s_n is given by n (and is bounded). However this delay-bound grows unboundedly as the index n of s_n grows. We conclude that (G, G^N) is nonuniformly-bounded fault-tolerant, but it is not fault-tolerant.

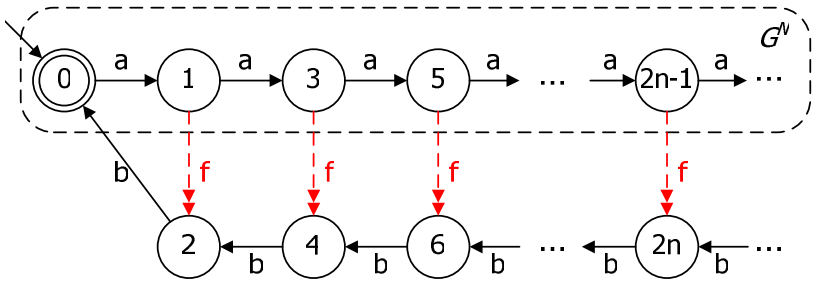

Figure 3.10 (G, G^N) that is only nonuniformly bounded fault-tolerant

3.9 Extension

Our framework can be modified to accommodate such a relaxation. In the initial part of this chapter, where we required that post-recovery behavior be equivalent to some pre-fault behavior, the condition of state-convergence will get replaced by the nonblockingness of the

post-fault states with respect to the pre-fault states. In the later part of this chapter, where we required that the post-recovery behavior be "simulated" by some pre-fault behavior, the condition of language-convergence will get replaced by a weaker form, introduced as weak language-stability in [Kumar, R. and Garg, V. K. and Marcus, S. I. (1993)] and as non-finite language convergence in [Willner, Y. and Heymann, M. (1995)].

The following result characterizes the notions of nonuniformly-bounded fault-tolerance introduced above.

Theorem 8 Consider a plant $G = (X, \Sigma, \delta, x_0, X_m)$ and its nonfaulty part $G^N = (X^N, \Sigma, \delta^N, x_0, X_m^N)$, and suppose the corresponding minimal plant and its nonfaulty part are $G_{min} = (X_{min}, \Sigma, \delta_{min}, x_{0,min}, X_{m,min})$ and $G_{min}^N = (X_{min}^N, \Sigma, \delta_{min}^N, x_{0,min}, X_{m,min}^N)$ respectively.

1. (G, G^N) is nonuniformly-bounded fault-tolerant if and only if X_{min} is nonblocking with respect to X_{min}^N, i.e.,

$$x \in X_{min} \Rightarrow \exists t \in \Sigma^* : \delta_{min}(x, t) \in X_{min}^N.$$

2. (G, G^N) is nonuniformly-bounded weakly fault-tolerant if and only if

$$L(G) \backslash [L(G) - L(G^N)] \subseteq \Sigma^* [L(G) \backslash L(G^N)],$$

$$L_m(G) \backslash [L(G) - L(G^N)] \subseteq \Sigma^* [(L_m(G) \backslash L(G^N)) \cup \{\epsilon\}].$$

Proof: We begin by proving the first part. For the necessity suppose (G, G^N) is nonuniformly-bounded fault-tolerant, and pick $x \in X_{min}$. If $x \in X_{min}^N$, we can set $t = \epsilon$; else $x \in X_{min} - X_{min}^N$ and so exists $s \in \Sigma^*$ such that $\delta_{min}(x_{0,min}, s) = x$, which implies $s \in L(G_{min}) - L(G_{min}^N) = L(G) - L(G^N)$. Then exist $st \in L(G) = L(G_{min})$ and $u \in L(G^N) = L(G_{min}^N)$ such that $st \cong_G u$. From minimality of G_{min}, st and u reach the same state in G_{min}. Since $u \in L(G_{min})$, this implies the state reached by u (and so also by st) belongs to X_{min}^N. To prove the sufficiency, we pick $s \in L(G) - L(G^N) = L(G_{min}) - L(G_{min}^N)$ and let $x \in X_{min} - X_{min}^N$ be the state reached by

s. From hypothesis, exists $t \in \Sigma^*$ such that $\delta_{min}(x,t) \in X^N_{min}$. Let $u \in L(G^N_{min}) = L(G^N)$ be a trace such that $\delta^N_{min}(x_0,u) = \delta_{min}(x,t)$. Then it is clear that $st \cong_{G_{min}} u$, which is equivalent to $st \cong_G u$.

For the necessity of the second part suppose (G, G^N) is nonuniformly-bounded weakly fault-tolerant, and pick traces $t \in L(G)\backslash[L(G) - L(G^N)]$, and $t' \in L_m(G)\backslash[L(G) - L(G^N)]$. Then since $\epsilon \in L(G)\backslash L(G^N)$, $t \in \Sigma^*[L(G)\backslash L(G^N)]$. Similarly since $\epsilon \in (L_m(G)\backslash L(G^N)) \cup \{\epsilon\}$, $t' \in \Sigma^*[(L_m(G)\backslash L(G^N)) \cup \{\epsilon\}]$.

Remark 3 Algorithms for verifying both nonblockingness (equivalently backward-reachability) and weak language-stability are well-understood. It turns out that both these properties can be polynomially verified. So by relaxing the notion of fault-tolerance to allow nonuniform delay bound for recovery, one gains on the computational complexity of verifying the existence of a fault-tolerant control.

3.10 Conclusion

We presented a framework for fault-tolerant supervisory control. Notations of fault-tolerance and weakly fault-tolerance have been proposed. Given a plant along with its non-faulty part, the goal of a fault-tolerant supervisory control is to enforce a specification for the nonfaulty plant and another (perhaps more liberal) specification for the overall plant, and also to ensure a bounded delay recovery up on the occurrence of a fault. Recovery implies that the ensuing behaviors are equivalent to those starting from a nonfaulty state. In case of weak fault-tolerance, recovery implies that the ensuing behaviors are subsumed by those starting from a nonfaulty state. Necessary and sufficient conditions for the existence of fault-tolerant as well as weakly fault-tolerant supervisor are provided. The condition involves the usual notions of controllability, observability and relative-closure, together with the notion of stability. The notion of state-stability is needed for fault-tolerance, whereas the weak fault-tolerance requires the notion of language-stability. Algorithms to verify state-stability are presented in [Brave, Y. and Heymann, M. (1990)], [Özveren, C. M. and Willsky, A. S. and Antsaklis, P. J. (1991)] and are of linear complexity. Algorithms to verify language-stability are presented in [Kumar,

R. and Garg, V. K. and Marcus, S. I. (1993)], [Willner, Y. and Heymann, M. (1995)]; the complexity is polynomial in the plant language (the language which needs to converge) and quadratic-exponential in the specification language (the language to which the convergence occurs). We also introduced the notion of nonuniformly-bounded fault-tolerance (and its weak version) where the delay-bound for recovery is not uniformly bounded over the set of faulty traces, and showed that this notion is equivalent to the notion of "uniformly-bounded fault-tolerance" considered earlier when the underlying system is one of finitely many states. Future work will explore the synthesis of maximally-permissive fault-tolerant supervisors.

CHAPTER 4. Synthesis of Optimal Fault-Tolerant Supervisor for Discrete Event Systems

In an earlier work [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007a)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007b)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2008)], we introduced a framework for fault-tolerant supervisory control of discrete event systems and presented a necessary and sufficient condition for its existence. In this paper, we introduce the synthesis of an optimal fault-tolerant supervisory controller. Given a discrete event plant with both faulty and nonfaulty behaviors, an optimal fault-tolerant supervisor we synthesize enforces a set of behaviors in which (i) a recovery is guaranteed within a bounded delay following any fault, (ii) the enforced set of nonfaulty behaviors are maximized, and (iii) the enforced set of faulty behaviors prior to the recovery are minimized. The computation has complexity *quadratic* in the size of plant. The optimal fault-tolerant supervisor possesses another useful property: It minimizes the recovery-delay for any faulty state. A practical example is given to illustrate the approach.

4.1 Introduction

Discrete Event Systems (DESs) are systems with discrete states that evolve in response to events [Ramadge, P. J. and Wonham, W. M. (1987)], [Kumar, R. and Garg, V. K. (1995)]. Examples include manufacturing systems, communication protocols, reactive software, and asynchronous hardware. A goal of supervisory control [Ramadge, P. J. and Wonham, W. M. (1987)], [Kumar, R. and Garg, V. K. (1995)] of such systems is to enforce a given specification by restricting the behavior of a given system (called plant). The supervisory role is characterized by

the fact that at any given plant state, the supervisor determines a set of controllable events to be enabled, so that the plant evolves over enabled events (including the uncontrollable events) without violating a given specification.

In an earlier work [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007a)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007b)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2008)], we introduced a framework for fault-tolerant supervisory control of discrete event systems and presented a necessary and sufficient condition for its existence. Fault-tolerance is a property requiring that a system continues to function, possibly with a degraded performance, even when some of its components fail. Given a plant G, possessing both faulty and nonfaulty behaviors, and a submodel G^N for the nonfaulty part, the goal of fault-tolerant control is to enforce a certain specification K^N for the nonfaulty plant G^N and another (perhaps more liberal) specification $K \supseteq K^N$ for the overall plant G, and further to ensure that the plant recovers from any fault within a bounded delay, so that following the recovery the system state is equivalent to a nonfaulty state (as if no fault ever happened). A fault is modeled as an uncontrollable event, occurrence of which may cause a transition from the nonfaulty part to the faulty part. Either of the specifications K^N and K can be used to specify both the safety and the progress requirements. Since a degraded performance may be tolerable after the occurrence of a fault, the second specification is more liberal than the first one (and so it allows a larger set of traces). The condition for the existence of a fault-tolerant controller involves the usual notions of controllability, observability and relative-closure, together with the notion of stability [Brave, Y. and Heymann, M. (1990)], [Özveren, C. M. and Willsky, A. S. and Antsaklis, P. J. (1991)], which is used to establish bounded delay recovery from a fault.

Fault-tolerance is a property requiring that a system continues to function, perhaps with a degraded performance, even when some of its components fail. In applications, fault-tolerance is achieved by using redundancy. Following the occurrences of failures, a fault-tolerant system can continue its proper operation, although the operation may be degraded. Fault-tolerance requires the ability of recovering from faulty behaviors. That is, the system should resume

normal functionality, fully or partially, in finite time. The notion of fault-tolerance is a type of state-stability [Brave, Y. and Heymann, M. (1990)], [Özveren, C. M. and Willsky, A. S. and Antsaklis, P. J. (1991)] property. If a system is not originally fault-tolerant, it could be made so using appropriate control. The purpose of fault-tolerant control is to have the controlled system achieve fault-tolerance.

There has been some prior work on fault-tolerant control of DESs (see for example [Jensen, R. M. (2003)]). Some involved controller switching upon the occurrence of a fault as in [Darabi, H. and Jafari, M. A. and Buczak, A. L. (2003)], or re-computation of a controller as in [Rohloff, K. R. (2005)]. The resulting controlled system can tolerate some faults but the system performance after faults will remain degraded since the notion of recovery from faults was not incorporated. Case studies involving synthesis of fault-tolerant supervisors can also be found in [Cho, K. -H. and Lim, J. -T. (1996)], [Cho, K. -H. and Lim, J. -T. (1998)], [Zhou, M. C. and Dicesare, F. (1989)]. Design of certain coordination protocols for automated highway systems to achieve fault-tolerance under vehicle failures is reported in [Lygeros, J. and Godbole, D. N. and Broucke, M. (2000)], [Godbole, D. N. and Lygeros, J. and Singh, E. and Deshpande, A. and Lindsey, A. E. (2000)]. Takai et al. considered the problem of reliable decentralized supervisory control [Takai, S. and Ushio, T. (2000)], where they studied fault-tolerance with respect to the failures of the supervisors. Fault-tolerance in Petri Net is considered in [Iordache, M. V. and Antsaklis, P. J. (2004)], where liveness enforcing strategies are designed to deal with failures using system reconfigurations. In [Lafortune, S. and Lin, F. (1991)], authors considered a pair of specifications, representing the desired and the (more liberal) tolerable behavior for a plant.

In this chapter we study the synthesis of an optimal fault-tolerant supervisory controller when the required existence conditions (as reported in [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007a)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007b)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2008)]) are not satisfied. An optimal fault-tolerant supervisor we synthesize enforces a set of behaviors in which (i) a recovery is guaranteed within a bounded delay following any fault, (ii) the enforced set of nonfaulty

behaviors are maximized, and (iii) the enforced set of faulty behaviors prior to the recovery are minimized. Given (G, G^N), where G is a plant and G^N is its nonfaulty part, and a state-based specification (X^g, X^g_m) representing legal states and legal final states respectively, we compute a subplant $(\tilde{G}, \tilde{G}^N)$ such that (i) $\tilde{G}^N$ $(\sqsubseteq G^N)$ is a maximal controllable subplant of G^N for which there exists G' with $\tilde{G}^N \sqsubseteq G' \sqsubseteq G$ and $(G', \tilde{G}^N)$ is fault-tolerant, (ii) $\tilde{G}$ is a minimal such G', and (iii) safety and nonblockingness properties are satisfied. The above is guided by the goal to maximize the achievable nonfaulty behaviors and at the same time minimize the faulty behaviors that must be tolerated, without having to sacrifice safety, nonblockingness, and recovery.

We show that $\tilde{G}^N$ can be uniquely chosen (since the corresponding property is closed under union), whereas nonunique minimal choices exist for $\tilde{G}$ (the corresponding property is closed under the intersection over decreasing chains). We present an algorithm, of complexity *quadratic* in the size of G, for computing $(\tilde{G}, \tilde{G}^N)$ and illustrate the algorithm through an example. It utilizes another new algorithm presented in the chapter for computing a minimal subplant in which the attraction of one set of states is guaranteed to another set of states.

The remainder of this paper is organized as following. Section 4.2 gives the basic notation and preliminaries. Section 4.3 formulates the synthesis of an optimal fault-tolerant supervisor. Section 4.4 provides an algorithm for the computation of such a supervisor and an illustrative example. Section 4.5 proves the optimality of the recovery delay in the controlled plant computed by the algorithm of Section 4.4. Section 4.6 gives a practical application example. Section 4.7 concludes the paper. The paper is based on a prior conference version [Wen, Q. and Kumar, R. and Huang, J. (2008)], extended to include new results and new application.

4.2 Preliminaries and Notations

A DES to be controlled, called plant, is modeled as an automaton, denoted by a five tuple $G := (X, \Sigma, \alpha, x_0, X_m)$, where X denotes the set of states, Σ denotes the finite set of events, $\alpha : X \times \Sigma \to X$ denotes the partial deterministic state transition function, $x_0 \in X$

denotes the initial state, and $X_m \subseteq X$ denotes the set of marked states. For $x \in X$, we use $\Sigma(x) \subseteq \Sigma$ to denote the set of events defined at x, i.e., $\Sigma(x) := \{\sigma \in \Sigma \mid \alpha(x, \sigma) \text{ is defined}\}$. For $x \in X, \sigma \in \Sigma(x)$, $(x, \sigma, \alpha(x, \sigma))$ is called a transition of G.

Σ^* is used to denote the set of all finite-length sequences of events, called traces, which includes the zero-length trace ϵ. The length of a trace s, denoted as $|s|$, is defined to be the number of events in the trace. A subset of Σ^* is called a language. The generated language of G, denoted as $L(G) \subseteq \Sigma^*$, contains all traces s for which $\alpha(x_0, s)$ is defined. The marked language of G, denoted as $L_m(G)$, contains all generated traces that reach a marked state.

Given two automata $G_1 := (X_1, \Sigma, \alpha_1, x_{0_1}, X_{m_1})$ and $G_2 := (X_2, \Sigma, \alpha_2, x_{0_2}, X_{m_2})$, G_1 is said to be a subautomaton of G_2, denoted as $G_1 \sqsubseteq G_2$, if there exists an injective map $h : X_1 \to X_2$ such that $\forall s \in L(G_1) : h(\alpha_1(x_{0_1}, s)) = \alpha_2(x_{0_2}, s)$. Given a plant $G = (X, \sigma.\alpha, x_0.X_m)$, a subset of states $\widehat{X}$, and a subset of transitions Δ, the subplant of G restricted to $(\widehat{X}, \Delta)$, is given by, $G\,|_{(\widehat{X},\Delta)} := (\widehat{X}, \Sigma, \alpha|_{\widehat{X},\Delta}, x_0, X_m \cap \widehat{X})$, where for $x \in \widehat{X}, \sigma \in \Sigma$,

$$\alpha\,|_{(\widehat{X},\Delta)}\,(x, \sigma) := \begin{cases} \alpha(x, \sigma) & \text{if } \alpha(x, \sigma) \in \widehat{X}, (x, \sigma, \alpha(x, \sigma)) \in \Delta \\ \text{undefined} & \text{otherwise} \end{cases}$$

It is possible that Δ is not specified, in which case $G\,|_{\widehat{X}}$ is same as $G\,|_{(\widehat{X}, X\times\Sigma\times X)}$.

For traces s and t, we use $s \leq t$ to denote that s is a prefix of t and $s < t$ to denote that s is a proper prefix of t. For a language $K \subseteq \Sigma^*$, $pr(K)$, called the prefix-closure of K, denotes the set of all prefixes of traces in K, i.e., $pr(K) = \{s \in \Sigma^* \mid \exists t \in K : s \leq t\}$. It is clear that $K \subseteq pr(K)$, and K is said to be prefix-closed if $K = pr(K)$. A language K is said to be relative-closed with respect to G, if $pr(K) \cap L_m(G) = K \cap L_m(G)$.

We use $K\backslash s$ to denote the set of traces that occur in the language K after the trace s has occurred, i.e., $L\backslash s := \{t \in \Sigma^* \mid st \in L\}$. For traces s and t, we use $s \sqsubseteq_G t$ to denote that the sets of traces that occur in the generated and the marked languages of G after s are contained in those after t, i.e., $L(G)\backslash s \subseteq L(G)\backslash t$ and $L_m(G)\backslash s \subseteq L_m(G)\backslash t$. We write $s \cong_G t$ if $s \sqsubseteq_G t$ and $t \sqsubseteq_G s$. $s \cong_G t$ implies the equivalence of the behaviors following s and t, whereas $s \sqsubseteq_G t$ implies the behaviors following s are subsumed by the behaviors following t.

Definition 6 [Brave, Y. and Heymann, M. (1990)] Given a plant $G = (X, \Sigma, \alpha, x_0, X_m)$ and a state set $\widehat{X} \subseteq X$, $x \in X$ is said to be $\widehat{X} - attractable$ in G if there exists $m \in \mathcal{N}$ such that for all t for which $\alpha(x,t)$ is defined and either $|t| \geq m$ or t deadlocks, exists $t' \leq t$ with $|t'| \leq m$ such that $\alpha(x,t') \in \widehat{X}$. m is called the *delay bound of convergence*. $x \in X$ is said to be controllably $\widehat{X} - attractable$ in G if there exists a supervisor S such that x is $\widehat{X} - attractable$ in $G \| S$.

We use $\Omega_G(\widehat{X})$, called the region of attraction of $\widehat{X}$, to denote the set of all $\widehat{X}$-attractable states, and $\widehat{X}$ is called an attractor for the set $\Omega_G(\widehat{X})$. We use $\Omega^c_G(\widehat{X})$, called the region of controllable-attraction of $\widehat{X}$, to denote the set of all controllably $\widehat{X}$-attractable states, and $\widehat{X}$ is called a controlled-attractor for the set $\Omega^c_G(\widehat{X})$. A state set $\tilde{X} \subseteq X$ is said to be attractable to $\widehat{X}$ if $\tilde{X} \subseteq \Omega_G(\widehat{X})$ and controllably-attractable to $\widehat{X}$ if $\tilde{X} \subseteq \Omega^c_G(\widehat{X})$. Clearly, $\widehat{X} \subseteq \Omega_G(\widehat{X}) \subseteq \Omega^c_G(\widehat{X})$.

For control purposes, the event set of G is partitioned into the set of controllable events $\Sigma_c \subseteq \Sigma$ and the set of uncontrollable events $\Sigma_u \subseteq \Sigma$. A language K is said to be controllable (with respect to G and Σ_u) if $pr(K)\Sigma_u \cap L(G) \subseteq pr(K)$.

A supervisor is another automaton $S := (Y, \Sigma, \beta, y_0, Y_m)$. The supervised plant is the synchronous composition of G and S, denoted $G||S := (X \times Y, \Sigma, \gamma, (x_0, y_0), X_m \times Y_m)$, where for $(x,y) \in X \times Y$ and $\sigma \in \Sigma$, $\gamma((x,y),\sigma)$ is defined if and only if both $\alpha(x,\sigma)$ and $\beta(y,\sigma)$ are defined and in which case, $\gamma((x,y),\sigma) = (\alpha(x,\sigma), \beta(y,\sigma))$. It can be concluded that the generated and the marked languages of the supervised plant satisfy: $L(G||S) = L(G) \cap L(S)$ and $L_m(G||S) = L_m(G) \cap L_m(S)$, respectively. A supervisor S is said to be (i) nonmarking if $L_m(G||S) = L(G||S) \cap L_m(G)$, (ii) nonblocking if $pr(L_m(G||S)) = L(G||S)$, and (iii) Σ_u-compatible if it does not disable any uncontrollable event (equivalently if $L(G||S)$ is controllable). It is known that given a nonempty specification language $K \subseteq L_m(G)$, there exists a Σ_u-compatible, nonmarking and nonblocking supervisor if and only if K is relative-closed and controllable [Ramadge, P. J. and Wonham, W. M. (1987)], [Kumar, R. and Garg, V. K. (1995)].

The following notion of fault-tolerance was introduced in [Wen, Q. and Kumar, R. and

Huang, J. and Liu, H. (2007a)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007b)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2008)].

Definition 7 Consider a pair of languages (H, H^N) with $H^N \subseteq H$. The pair (H, H^N) is said to be fault-tolerant if exists $m \in \mathcal{N}$ such that for $s \in pr(H) - pr(H^N)$, $st \in pr(H)$ with $|t| \geq m$ or st deadlocks, there exist $u \in pr(H^N)$ and $t' \leq t$ with $|t'| \leq m$ and $st' \cong_G u$. In this case, m is called the *delay-bound of fault-tolerance*. Given a plant G with its nonfaulty part G^N, (G, G^N) is said to be fault-tolerant if $(L(G), L(G^N))$ is fault-tolerant. A supervisor S is said to be fault-tolerant if $(G\|S, G^N\|S)$ is fault-tolerant.

The following theorems were obtained in [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007a)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007b)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2008)].

Theorem 9 ([Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007a)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007b)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2008)]) Consider a plant $G = (X, \Sigma, \delta, x_0, X_m)$ and its nonfaulty part $G^N = (X^N, \Sigma, \delta^N, x_0, X_m^N)$, and suppose the corresponding minimal plant and its nonfaulty part are $G_{min} = (X_{min}, \Sigma, \delta_{min}, x_{0,min}, X_{m,min})$ and $G_{min}^N = (X_{min}^N, \Sigma, \delta_{min}^N, x_{0,min}, X_{m,min}^N)$ respectively. (G, G^N) is fault-tolerant if and only if X_{min} is attractable to X_{min}^N, i.e., $X_{min} \subseteq \Omega_{G_{min}}(X_{min}^N)$.

Theorem 10 *([Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007a)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007b)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2008)]) Given a plant $G = (X, \Sigma, \alpha, x_0, X_m)$ with nonfaulty part $G^N = (X^N, \Sigma, \alpha^N, x_0, X_m^N)$, specification $\emptyset \neq K \subseteq L_m(G)$ for G and specification $\emptyset \neq K^N \subseteq L_m(G^N)$ for G^N satisfying $K^N \subseteq K$, there exists a nonmarking, nonblocking (with respect to both G^N and G), Σ_u-compatible and fault-tolerant supervisor S such that*

1. *$L_m(G^N||S) = K^N$, $L(G^N||S) = pr(L_m(G^N||S))$, and*
2. *$L_m(G||S) = K$ and $L(G||S) = pr(L_m(G||S))$*

if and only if

1. *K is relative-closed and controllable with respect to G,*

2. *(K, K^N) is fault-tolerant, and*

3. *$K^N = K \cap L_m(G^N)$ and $pr(K^N) = pr(K) \cap L(G^N)$.*

4.3 Formulation of Optimal Fault-Tolerant Control Synthesis Problem

Theorem 10 provides a condition under which a desired fault-tolerant supervisor exists. When this condition is satisfied, a trim recognizer of K can be chosen as a supervisor. Here we study the problem of synthesizing a fault-tolerant supervisor when the condition of Theorem 10 is not satisfied. A desirable goal is to maximize the achievable nonfaulty behaviors and at the same time minimize the faulty behaviors that must be tolerated, without sacrificing safety, nonblockingness and recovery. The motivation being, we allow maximal functionality of the system in the absence of faults, and at the same time limit the system's faulty behavior within a minimal range without sacrificing recovery.

It turns out that the supremal nonfaulty fault-tolerant behavior does not exist in general. That is, given a language pair (K, K^N), we cannot always find a fault-tolerant sublanguage pair $(\tilde{K}, \tilde{K}^N)$, where $\tilde{K} \subseteq K$ and $\tilde{K}^N \subseteq K^N$, such that any other fault-tolerant sublanguage pair $(\widehat{K}, \widehat{K}^N)$ satisfies $\widehat{K} \subseteq \tilde{K}$ and $\widehat{K}^N \subseteq \tilde{K}^N$.

Consider the following example for illustration. Figure 4.1 shows a plant G and its nonfaulty part G^N, where f is a faulty event and is the only uncontrollable event. From Figure 4.2, we can see that there are two subplant pairs (G_1, G_1^N) and (G_2, G_2^N), which are fault-tolerant. Note in (G_1, G_1^N) the faulty state 6 is equivalent to the nonfaulty state 3, whereas in (G_2, G_2^N) the faulty state 6 is equivalent to the nonfaulty state 2. Thus in both cases the system reaches a state that is equivalent to a nonfaulty state within one transition of the occurrence of the fault (i.e., the delay bound for recovery is one in both cases). However the language $(L(G_1) \cup L(G_2), L(G_1^N) \cup L(G_2^N))$ which is realized by (G, G^N) shown in Figure 4.1 is not

fault-tolerant. This is because there exists no nonfaulty state that is equivalent to the faulty state 6 where the system can stay with recovery.

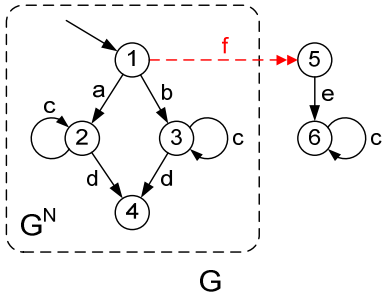

Figure 4.1 Plant G with its nonfaulty part G^N

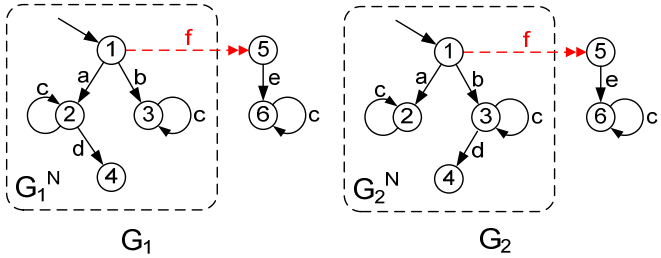

Figure 4.2 Two fault-tolerant subplants

It happens that maximal nonfaulty behaviors that are fault-tolerant also do not exist, for the limit of a class of monotonically increasing nonfaulty fault-tolerant behaviors may not be fault-tolerant. To see this, consider the monotonically increasing sequence of plant behaviors shown in Figure 4.3, where the nth behavior in the sequence is generated by the plant (G_n, G_n^N). The nonfaulty part G_n^N contains n a's, whereas the overall plant G_n contains a faulty trace with same number of b's, i.e., the delay bound for recovery in (G_n, G_n^N) is n. The limiting plant behavior $(L(G_\infty) := \cup_n L(G_n), L(G_\infty^N) := \cup_n L(G_n^N))$ (see Figure 4.4) is not fault-tolerant since the faulty plant can execute an unbounded number of b's before a recovery to the nonfaulty part occurs.

The examples above show that neither the supremal nor a maximal nonfaulty fault-tolerant sublanguage exists in general. Lack of supremal and even maximal nonfaulty fault-tolerant sublanguages motivate us to restrict our attention to state-feedback based control, under which

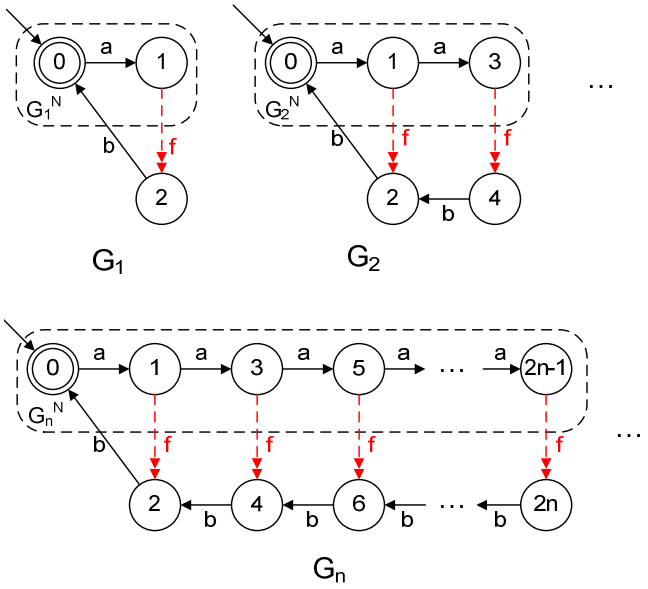

Figure 4.3 Plant $(G_n, G_n^N), n \geq 1$

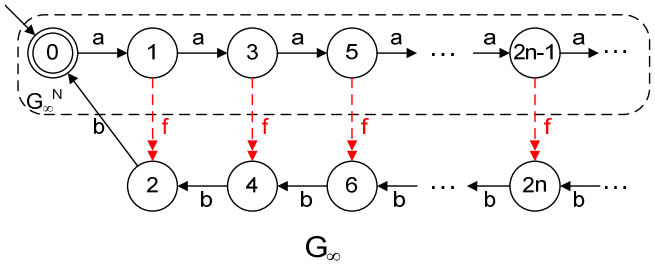

Figure 4.4 Plant (G_∞, G_∞^N)

the controlled plant is always a subplant of the uncontrolled plant. In this setting, we are able to show the existence of fault-tolerant control that maximizes the nonfaulty behavior while minimizing the faulty behavior without sacrificing safety, nonblockingness, and recovery.

Without loss of generality, the specification is given as a state set pair (X^g, X^g_m), where $X^g \subseteq X$ is the set of legal states and $X^g_m \subseteq X^g \cap X_m$ is the set of legal final states. Since under a state-feedback control the controlled plant is a subplant of the uncontrolled plant, examining various state-feedback controllers is equivalent to examining various subplants of a given plant.

We first define the set of all fault-tolerant subplants, denoted $F(G, G^N)$ as follows:

Definition 8 Given a plant model (G, G^N) with $G^N \sqsubseteq G$, the class of *fault-tolerant subplants*, denoted $F(G, G^N)$, is the set of all subplants $(\tilde{G} \sqsubseteq G, \tilde{G}^N \sqsubseteq G^N)$ with state set pair $(\tilde{X}, \tilde{X}^N)$ and marked state set pair $(\tilde{X}_m, \tilde{X}^N_m)$, such that

- $\tilde{X} \subseteq X^g$, $\tilde{X}_m \subseteq X^g_m$;
- $L_m(\tilde{G})$ is relatively closed and controllable with respect to G;
- $\tilde{X} \subseteq \Omega_{\tilde{G}}(\tilde{X}^N)$.

Remark 4 Since $\tilde{G}^N$ represents the nonfaulty subplant and $\tilde{G}$ represents the overall subplant, our goal is to maximize $\tilde{G}^N$ (which will maximize nonfaulty behavior), and at the same time minimize $\tilde{G}$ (which will minimize the fault behavior). In doing so, we want to ensure that $(\tilde{G}, \tilde{G}^N) \in F(G, G^N)$.

We next introduce the class of fault-tolerant nonfaulty subplants and show that this class is closed under union.

Definition 9 The class of *fault-tolerant nonfaulty subplants* is defined as:

$$F^N_G(G^N) := \{\tilde{G}^N \sqsubseteq G^N \mid \exists \tilde{G} \sqsubseteq G : (\tilde{G}, \tilde{G}^N) \in F(G, G^N)\}.$$

The following theorem shows that the above class of fault-tolerant nonfaulty subplants is closed under union.

Theorem 11 Let Λ be an index set such that $\forall\lambda \in \Lambda$, $G_\lambda^N \in F_G^N(G^N)$. Then $\bigcup_{\lambda\in\Lambda} G_\lambda^N \in F_G^N(G^N)$.

Proof: We show the existence of $G' \sqsubseteq \cup_\lambda G_\lambda$ such that $(G', \cup_\lambda G_\lambda^N) \in F(G, G^N)$.

Suppose $(\cup_\lambda G_\lambda, \cup_\lambda G_\lambda^N)$ is fault-tolerant. Then we claim that we can choose $G' = \cup_\lambda G_\lambda$. Since for each λ, $X_\lambda \subseteq X^g$, we have $\cup_\lambda X_\lambda \subseteq X^g$. Since for each λ, $L_m(G_\lambda)$ is controllable which means no uncontrollable event is disabled in G to obtain each G_λ, it is the case that no uncontrollable event is disabled to obtain $\cup_\lambda G_\lambda$, i.e., $L_m(\cup_\lambda G_\lambda)$ is controllable. Since for each λ, $L_m(G_\lambda)$ is relatively-closed which implies that $X_\lambda \cap X_m \subseteq X_\lambda \cap X_m^g$, it is the case that $\cup_\lambda X_\lambda \cap X_m \subseteq \cup_\lambda X_\lambda \cap X_m^g$, i.e., $L_m(\cup_\lambda G_\lambda)$ is relatively-closed.

On the other hand suppose $(\cup_\lambda G_\lambda, \cup_\lambda G_\lambda^N)$ is not fault-tolerant. Then exists a cycle in the faulty part. Since for each λ, (G_λ, G_λ^N) is fault-tolerant which implies the faulty part of G_λ does not contain any cycle, i.e., a certain edge of each faulty-part cycle of $\cup_\lambda G_\lambda$ is missing in G_λ. Then each such edge must be labeled with a controllable event (since $L_m(G_\lambda)$ is controllable). Let G' be obtained from by removing each edge that contributes to a cycle in the faulty-part of $\cup_\lambda G_\lambda$ and is missing in $G_{\bar\lambda}$ for a $\bar\lambda \in \Lambda$. Then the faulty-part of G' is acyclic. Also since only controllable edges are removed to obtain G' from $\cup_\lambda G_\lambda$, the controllability property is preserved. Since $L_m(\cup_\lambda G_\lambda)$ is controllable (see above), we can claim that $L_m(G')$ is controllable. Further since G' is obtained from $\cup_\lambda G_\lambda$ by removing certain edges that appear as part of certain cycles, the state set is preserved upon the removal of such edges (i.e., $X' = \cup_\lambda X_\lambda$). It can then be concluded that relative-closure property is also preserved, and so $L_m(G')$ is also relatively-closed.

It remains to show that $(G', \cup_\lambda G_\lambda^N)$ is fault-tolerant. Since the faulty-part of G' is acyclic, it suffices to show that for any faulty state $x \in X' = \cup_\lambda X_\lambda$ exists a path in G' to the nonfaulty part $\cup_\lambda X_\lambda^N$. Pick any such state x. Then exists λ such that x is a faulty state of G_λ. From the fault-tolerance of (G_λ, G_λ^N), exists a path in G_λ from x to $G_\lambda^N \sqsubseteq \cup_\lambda G_\lambda^N$. If this path does not contain any of the edges that were removed to obtain G', then we are done. Otherwise this path visits a state $\bar x$ from where an edge that is present in $\cup_\lambda G_\lambda$ but is missing in $G_{\bar\lambda}$ has been removed. From the fault-tolerance of $(G_{\bar\lambda}, G_{\bar\lambda}^N)$ exists a path in $G_{\bar\lambda} \sqsubseteq G'$ from $\bar x$ to $G_{\bar\lambda}^N \sqsubseteq G'$.

This concludes the proof. ■

Since $F_G^N(G^N)$ is closed under union, it possesses a supremal element, $\sup F_G^N(G^N) \in F_G^N(G^N)$, with the property that if $\tilde{G}^N \in F_G^N(G^N)$, then $\tilde{G}^N \sqsubseteq \sup F_G^N(G^N)$.

In the above, we defined a class of fault-tolerant nonfaulty subplants. Next, given a nonfaulty subplant, we define a class of overall subplant that is fault-tolerant.

Definition 10 Given $\tilde{G}^N \sqsubseteq G^N$, the class of *overall subplants that are fault-tolerant wrt* $\tilde{G}^N$ is defined as:

$$F_{\tilde{G}^N}(G) := \{\tilde{G} \sqsubseteq G | (\tilde{G}, \tilde{G}^N) \in F(G, G^N)\}.$$

$\tilde{G}$ is an infimal element of $F_{\tilde{G}^N}(G)$ if $\tilde{G} \in F_{\tilde{G}^N}(G)$, and $G' \in F_{\tilde{G}^N}(G)$ implies $\tilde{G} \sqsubseteq G'$. $\tilde{G}$ is a minimal element of $F_{\tilde{G}^N}(G)$ if $\tilde{G} \in F_{\tilde{G}^N}(G)$ and $G' \sqsubseteq \tilde{G}$ implies $G' \notin F_{\tilde{G}^N}(G)$.

The following result establishes certain closure properties of $F_{\tilde{G}^N}(G)$ under intersection.

Theorem 12 $F_{\tilde{G}^N}(G)$ does not possess infimal element but possess minimal elements whenever it is nonempty.

Proof: For the first part, we show that $F_{\tilde{G}^N}(G)$ is not closed under intersection. As seen from Figure 4.5, $G_1, G_2 \in F_{G^N}(G_1 \cup G_2)$ (since for each $i = 1, 2$, $(G_i, G^N) \in F(G_1 \cup G_2, G^N)$). However it is clear from Figure 4.5 that $(G_1 \cap G_2, G^N) \notin F(G_1 \cup G_2, G^N)$, i.e., $G_1 \cap G_2 \notin F_{G^N}(G_1 \cup G_2)$.

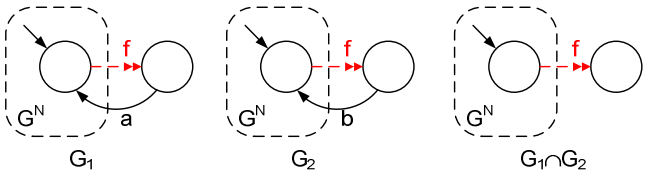

Figure 4.5 $(G_1, \tilde{G}^N)$ and $(G_2, \tilde{G}^N)$ are fault-tolerant, but $(G_1 \cap G_2, \tilde{G}^N)$ is not

Next we consider the second part. Since G has finite number of states and transitions, the number of subplants of G is finite. So, whenever $F_{\tilde{G}^N}(G)$ is nonempty, there exist at least one subplant of G in $F_{\tilde{G}^N}(G)$ for which there exists no subplant in the set $F_{\tilde{G}^N}(G)$, and so the existence of a minimal overall subplant that is fault-tolerant with respect to $\tilde{G}^N$ follows. ■

The set of all the minimal elements of $F_{\tilde{G}^N}(G)$ is denoted as $\mathrm{MIN}F_{\tilde{G}^N}(G)$, and a minimal element is denoted as $\min F_{\tilde{G}^N}(G)$.

4.4 Computation of Optimal Fault-Tolerant Control

We set out the goal of synthesizing an optimal fault-tolerant control that maximizes the achievable nonfaulty behavior, minimizes the achievable faulty behavior and ensures safety, nonblockingness and bounded-delay recovery. The computation of such a fault-tolerant state-feedback control for a given plant (G, G^N) with state set (X, X^N) and specification (X^g, X^g_m), where $X^g \subseteq X, X^g_m \subseteq X^g \cap X_m$, requires the computation of a subplant pair $(\tilde{G}, \tilde{G}^N)$ with state set $(\tilde{X}, \tilde{X}^N)$, where $\tilde{G}^N = \sup F^N_G(G^N)$, and $\tilde{G} \in \mathrm{MIN}F_{\tilde{G}^N}(G)$.

The computation of an optimal fault-tolerant control discussed above requires the computation of the region of controllable-attraction and a minimal set of transition for achieving the controllable-attractability. The following algorithm computes the region of controllable-attraction $\Omega^c_G(\widehat{X})$ of states in $\widehat{X} \subseteq X$ for a plant G, and is obtained by extending the one given in [Kumar, R. and Garg, V. K. and Marcus, S. I. (1993)] to keep track of a minimal set of transitions that must be enabled to achieve the controllable-attractability. Also, whenever $\alpha(\widehat{X}, \Sigma^*_u) \subseteq \Omega^c_G(\widehat{X})$, the algorithm computes a *minimal* fault-tolerant subplant $\Upsilon_G(\widehat{X})$ with the nonfaulty states $\widehat{X}$.

Algorithm 1 Consider a plant $G = (X, \Sigma, \alpha, x_0, X_m)$ and a state set $\widehat{X} \subseteq X$.

1. Initialization step:

   $k = 0$, $\Omega_{-1} = \emptyset$, $\Omega_0 = \widehat{X}$, $\Delta_0 = \emptyset$.

2. Iteration step:

   - $\Omega_{k+1} = \Omega_k \cup \{x \in X - \Omega_k | \alpha(x, \Sigma) \cap \Omega_k - \Omega_{k-1} \neq \emptyset, \alpha(x, \Sigma_u) \subseteq \Omega_k\}$.
   - $\Delta_{k+1} = \Delta_k \cup \{(x, \sigma, x') \mid \alpha(x, \sigma) = x', x \in \Omega_{k+1} - \Omega_k, \sigma \in \Sigma_u \cup \Sigma_x, x' \in \Omega_k\}$, where $\Sigma_x = \emptyset$ if $\alpha(x, \Sigma_u) \neq \emptyset$ and otherwise $\Sigma_x = \{\sigma_x\}$ such that $\alpha(x, \sigma_x) \in \Omega_k$.

3. Termination step:

   If $\Omega_{k+1} \neq \Omega_k$, then increment k by 1 and iterate; else $\Omega^c_G(\widehat{X}) := \Omega_k$, $\Delta_G(\widehat{X}) := \Delta_k$, and $\Upsilon_G(\widehat{X}) := G|_{\bar{X},\bar{\Delta}}$, where $\bar{X} := \{x \in \bar{X} | \exists x_1 \in \alpha(\widehat{X}, \Sigma^*_u), \{(x_i, \sigma_i, x_{i+1}) \in \Delta_G(\widehat{X})\}_{i=1}^n : x_{n+1} = x\}$, $\bar{\Delta} := \{(x, \sigma, x') \in \Delta_G(\widehat{X}) \mid x \in \bar{X}\}$.

Remark 5 The complexity of Algorithm 1 can be seen to be *linear* in the size of plant G. This is because at most $|X|$ iterations are being performed, and in each iteration a constant amount of computation is being done.

Algorithm 1 computes the region of controllable-attraction of $\widehat{X}$, by iteratively adding controllably attractable states with increasing delay bound of convergence, and also keeps a record of a minimal set of transitions that must remain enabled at the states included in the region of controllable attraction. Note a (single) controllable transition is included (this is the transition used for recovery) only when no uncontrollable transitions are defined at that state. This ensures the minimality of the enabled transitions.

A desired minimal subplant is obtained by including only the minimally required states of the region of controllable attraction, and only the transitions originating at those states that belong to the minimal set of transitions required for controllable attraction. The minimality of the states included follows from the fact that only those faulty states that are reachable from a nonfaulty state by executing a sequence of uncontrollable transitions, followed by a sequence of transitions belonging to the minimal set of transitions required for controllable attraction, are included in the minimal subplant. It then follows that the computed subplant is minimal. This is stated in the following theorem, the proof of which follows from the preceding discussion.

Theorem 13 Given a plant $G = (X, \Sigma, \alpha, x_0, X_m)$ and state sets $\tilde{X}, \widehat{X} \subseteq X$ such that $\tilde{X} \subseteq \Omega^c_G(\widehat{X})$, $\Upsilon_G(\widehat{X})$ computed by Algorithm 1 is a minimal fault-tolerant subplant with nonfaulty part $\widehat{X}$.

The following algorithm computes a fault-tolerant subplant $(\tilde{G}^N, \tilde{G})$ such that $\tilde{G}^N = \sup F^N_G(G^N)$ and $\tilde{G} \in \mathrm{MIN} F_{\tilde{G}^N}(G)$.

Algorithm 2 Consider plant $G = (X, \Sigma, \alpha, x_0, X_m)$ with nonfaulty part $G^N = (X^N, \Sigma, \alpha^N, x_0, X_m^N)$, and specification (X^g, X_m^g).

Uncontrollable/blocking states removal:

1. Initialization step:

   $k = 0$, $X_k^g := X^g$.

2. Iteration step:

   If $x_0 \notin X_k^g$, then terminate (no solution exists); else

   - $X_{k+1}^g := \{x \in X_k^g \mid \alpha(x, \Sigma_u^*) \subseteq X_k^g, \alpha|_{X_k^g}(x, \Sigma^*) \cap X_m^g \neq \emptyset\}$.

3. Termination step: If $X_{k+1}^g \neq X_k^g$, increment k by 1 and iterate; else

   $G|_{X_k^g} =: \bar{G} = (\bar{X}, \Sigma, \bar{\alpha}, x_0, \bar{X}_m)$, and $G^N|_{X_k^g} =: \bar{G}^N = (\bar{X}^N, \Sigma, \bar{\alpha}^N, x_0, \bar{X}_m^N)$.

Optimal fault-tolerant subplant computation:

1. Initialization step:

   $k = 0, X_k^N := \bar{X}^N$.

2. Iteration step:

   If $x_0 \notin X_k^N$, then terminate (no solution exists); else

   - $X_{k+1}^N := \{x \in X_k^N \mid \bar{\alpha}(x, \Sigma_u^*) \subseteq \Omega_{\bar{G}}^c(X_k^N), \bar{\alpha}|_{\Omega_{\bar{G}}^c(X_k^N)}(x, \Sigma^*) \cap (X_k^N \cap X_m^g) \neq \emptyset\}$.

3. Termination step:

   If $X_{k+1}^N \neq X_k^N$, increment k by 1 and iterate; else

   - $\tilde{X}^N = X_k^N$, $\tilde{G}^N := \bar{G}^N|_{\tilde{X}^N}$;
   - $\tilde{G} := \Upsilon_{\bar{G}}(\tilde{X}^N)$.

The steps of Algorithm 2 can be understood as follows. The "uncontrollable/blocking states removal" step computes the subplant that recognizes the supremal relative-closed and controllable sublanguage by keeping only those legal states $\bar{X} \subseteq X^g$ that are controllable (invariant under the executions of uncontrollable transitions) and nonblocking (can reach a legal final state in X_m^g while staying inside $\bar{X}$). This provides a supremal safe and nonblocking

subplant $(\bar{G}, \bar{G}^N)$. Additional states in $\bar{X}^N$ may need to be removed to satisfy fault-tolerance since the faulty states in the uncontrollable reach of $\bar{X}^N$ may not be controllably-attractable to $\bar{X}^N$. This is accomplished in the "optimal fault-tolerant subplant computation" step. Starting from the initial iteration in which $k = 0$ and $X_k^N = \bar{X}^N$, the kth iteration obtains a subset X_{k+1}^N by retaining only those states in X_k^N whose uncontrollable reach is within the region of controllable-attraction of X_k^N and that are nonblocking with respect to the marked states in $X_k^N \cap X_m^g$ while staying inside the region of controllable-attraction of X_k^N. This maximizes the set of nonfaulty behaviors. To minimize the faulty behavior, we should only keep those states in X_k as part of X_{k+1} that are uncontrollably reachable from the nonfaulty part, i.e., the states in $\alpha_k(X_{k+1}^N, \Sigma_u^*)$. However not all such states may be controllably-attractable to X_{k+1}^N. X_{k+1} is hence chosen to be a minimal controlled-attractor for those states of $\alpha_k(X_{k+1}^N, \Sigma_u^*)$ that are controllably-attractable to X_{k+1}^N. The iteration continues if $X_{k+1}^N \neq X_k^N$ and otherwise it terminates yielding the nonfaulty states $\tilde{X}^N$ of a desired optimal fault-tolerant subplant. Further from Algorithm 1, the plant $\tilde{G} := \Upsilon_{\bar{G}}(\tilde{X}^N)$ is the minimal fault-tolerant subplant of $\bar{G}$ with nonfaulty states $\tilde{X}^N$.

Remark 6 It can be verified that the number of iterations of the steps "uncontrollable/blocking states removal" as well as "optimal fault-tolerant subplant computation" is bounded by the number of states in G, and each such iteration has a complexity that is linear in the size of plant G. It follows that the complexity of Algorithm 2 is *quadratic* in the size of plant G.

The following theorem establishes the correctness of Algorithm 2.

Theorem 14 Given overall plant G, nonfaulty plant G^N and specification (X^g, X_m^g), Algorithm 2 computes $(\tilde{G}^N, \tilde{G})$ with $\tilde{G}^N = \sup F_G^N(G^N)$ and $\tilde{G} \in \text{MIN}F_{\tilde{G}^N}(G)$.

Proof: We first prove that $\tilde{G}^N$ is the supremal element of $F_G^N(G^N)$. First note that the "uncontrollable/nonblocking removal step" computes the supremal subplant $(\bar{G}, \bar{G}^N)$ that is safe (does not reach an illegal state), controllable (is invariant with respect to the execution of feasible uncontrollable transitions), and nonblocking (a legal final state can always be reached within $\bar{G}$). Thus if $(\bar{G}, \bar{G}^N)$ also happens to be fault-tolerant, then $\bar{G}^N$ will be the supremal

element of $F_G^N(G^N)$, and otherwise the supremal element of $F_G^N(G^N)$ must be a subplant of $\bar{G}^N$. The "optimal fault-tolerant subplant computation" step iteratively computes the state set of such a subplant. The iteration starts with $\bar{X}^N$ of $\bar{G}^N$ and iteratively removes states to obtain $\tilde{X}^N$. It is clear from the "optimal fault-tolerant computation" step that the uncontrollable reach of $\tilde{X}^N$ is controllably-attractable to $\tilde{X}^N$ and so for $\tilde{G}^N$ exists $G' = \bar{G}|_{\Omega^c_{\bar{G}}(\tilde{X}^N)}$ such that $(G', \tilde{G}^N)$ is fault-tolerant. Further since illegal states are not reached in $\bar{G}$, they are also not reached in G'. Next since the uncontrollable transitions of $\tilde{X}^N$ reach states that are controllably-attractable to $\tilde{X}^N$, we can conclude that G' is invariant with respect to the execution of feasible uncontrollable transitions, i.e., it is controllable. Also since the states in $\tilde{X}^N$ can always reach the states in $\tilde{X}^N \cap X_m^g$ without having to leave the states of G', $\tilde{X}^N$ is nonblocking with respect to $\tilde{X}^N \cap X_m^g$. Then owing to the fault-tolerance property the states of G' are also nonblocking with respect to $\tilde{X}^N \cap X_m^g$. Thus $(G', \tilde{G}^N)$ is safe, controllable, nonblocking, and fault-tolerant. The supremality of $\tilde{X}^N$ follows from the fact that any state in $\bar{X}^N - \tilde{X}^N$ that gets removed in the "optimal fault-tolerant subplant computation" violates either the fault-tolerance or the nonblocking property, and so cannot be present in the supremal solution.

Next since G' is a subplant of $\bar{G}$, $\Upsilon_{G'}(\tilde{X}^N) = \Upsilon_{\bar{G}}(\tilde{X}^N)$. It follows that, $\tilde{G} = \Upsilon_{G'}(\tilde{X}^N) = \Upsilon_{\bar{G}}(\tilde{X}^N)$ is the minimal subplant of G' with the nonfaulty states $\tilde{X}^N$. It follows that $\tilde{G} \in \text{MIN}F_{\tilde{G}^N}(G)$. ■

Example 5 *The following example illustrates the Algorithm 2. Consider the plant (G, G^N) given in Figure 4.6. Encircled states denote the final states. There is a single illegal state labeled dump; the remaining states form the state set X^g. The specification for the marked states is given as $X_m^g = X^g \cap X_m = X_m$. The dotted double arrowed transitions are uncontrollable and the remaining ones are controllable. All transitions from a state in X^N to a state in X are considered faulty (and also uncontrollable). Note each transition has a distinct event label and so it can be identified by the event labeling the transition.*

The subplant (G_0, G_0^N) obtained after the removal of uncontrollable and blocking states is shown in Figure 4.7. Note the states 4 and 8 get removed since they can reach the illegal state

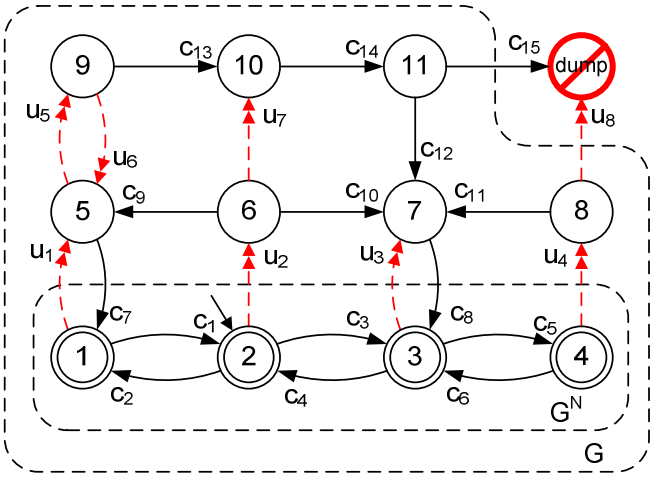

Figure 4.6 Plant (G, G^N)

uncontrollably. Also note that $X_0^N = \{1, 2, 3\}$*, and* $X_0 = \{1, 2, 3, 5, 6, 7, 9, 10, 11\}$.

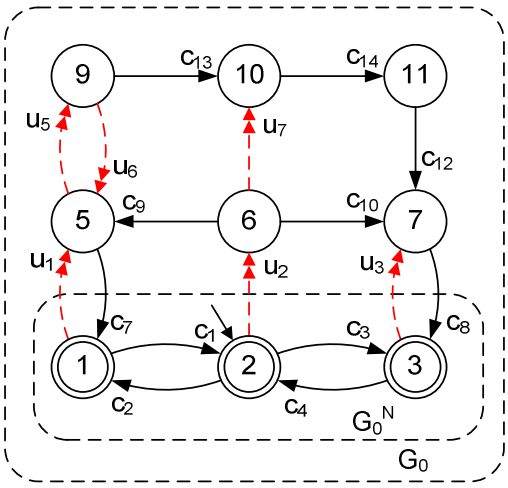

Figure 4.7 Controllable and nonblocking subplant (G_0, G_0^N)

Since $x_0 = \{2\} \in X_0$*, the computation of the fault-tolerant subplant proceeds as follows.*

1. *Iteration no. 1:*

   - $X_1^N = \{1, 2, 3\}$;
   - $\Omega_{G_0}^c(X_1^N) = \{1, 2, 3, 6, 7, 10, 11\}$,
     $\Delta_{G_0}(X_1^N) = \{c_8, c_{12}, c_{14}, u_7\}$;
   - $X_1 = \{1, 2, 3, 6, 7, 10, 11\}$. *The resulting* (G_1, G_1^N) *is shown in Figure 4.8.*

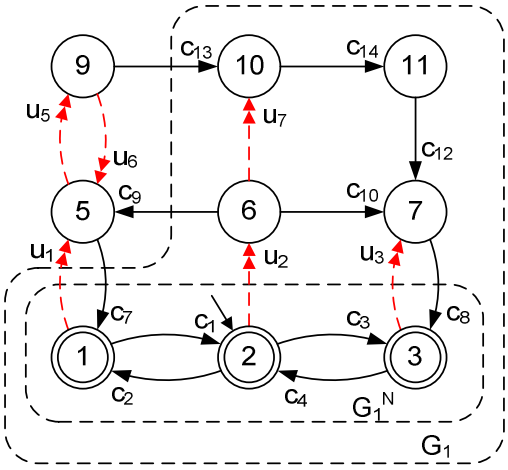

Figure 4.8 (G_1, G_1^N) *obtained after iteration no. 1*

2. Iteration no. 2:

- $X_2^N = \{2, 3\}$;
- $\Omega^c_{G_1}(X_2^N) = \{2, 3, 6, 7, 10, 11\}$,
  $\Delta_{G_1}(X_2^N) = \{c_8, c_{12}, c_{14}, u_7\}$;
- $X_2 = \{2, 3, 6, 7, 10, 11\}$. *The resulting* (G_2, G_2^N) *is shown in Figure 4.9.*

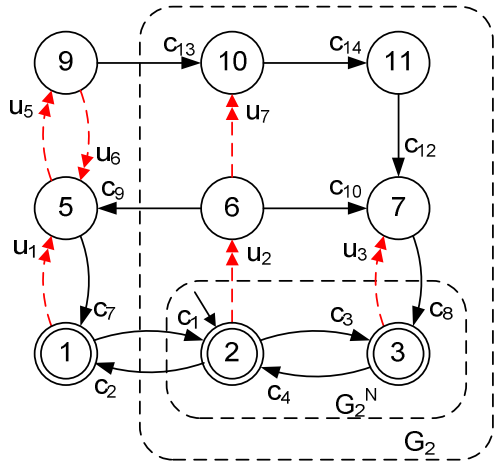

Figure 4.9 (G_2, G_2^N) *obtained after iteration no. 2*

3. Iteration no. 3:

- $X_3^N = \{2, 3\}$. *Since* $X_3 = X_2$, *the iteration stops.*

After removing the controllable transitions $\{c_2, c_9\}$ *that leave the state set* X_2 *and also all controllable transitions in the faulty part of* G_2 *that are not included in* $\Delta_{G_2}(X_3^N) = \{c_8, c_{12}, c_{14}, u_7\}$*, we get the desired fault-tolerant subplant shown in Figure 4.10.*

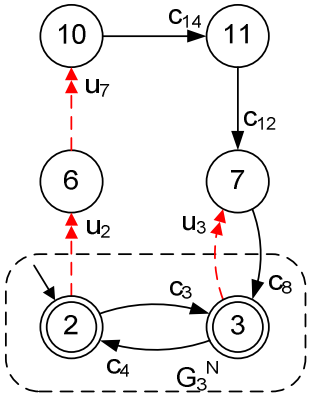

Figure 4.10 Optimal fault-tolerant subplant

We can see that state 2 and 3 are the only nonfaulty states from where after the occurrence of a fault it is possible to recover within a bounded delay. State 1 does not have this property since it is possible to uncontrollably reach state 5 from where a bounded delay recovery is not possible (state 5 is contained in a cycle of uncontrollable transitions). On the other hand state 4 does not have this property since it is possible to uncontrollably reach the illegal state from state 4. It can be seen then that the computed nonfaulty part is supremal. The faulty states 6, 7, and 10 must be present in the overall subplant since those states are uncontrollably reached from the nonfaulty states 2 and 3. Since the only way to recover from the faulty state 10 is through state 11, state 11 must also be included in the overall subplant. Finally removing any controllable transition in the faulty part renders the overall subplant "fault-intolerant". It follows that the computed faulty part is minimal.

4.5 Optimality of Recovery Delay

The algorithm we proposed above computes an optimal fault-tolerant supervisor with the properties that it

- maximizes the achievable nonfaulty behavior,
- minimizes the achievable faulty behavior,
- ensures safety and nonblockingness, and
- ensures bounded-delay recovery.

In this section we show that the synthesized optimal fault-tolerant supervisor has the following additional property that it

- minimizes the recovery-delay.

In the following we formally define the recovery-delay of a state, which is the maximum over the length of the paths starting from the said state and ending at the first nonfaulty state.

Definition 11 Given a plant pair (G, G^N) and a state $x \in X$, the recovery-delay of x, $\rho(x, (G, G^N))$, is defined as: $\rho(x, (G, G^N)) := \max\{|s| \mid \alpha(x, s) \in X^N, \forall t < s : \alpha(x, t) \notin X^N\}$.

For a nonfaulty state $x \in X^N$, we let $\rho(x, (G, G^N)) = 0$, and for a faulty state $x \in X - X^N$ that is not attractable to the nonfaulty part, we let $\rho(x, (G, G^N)) = \infty$.

Letting $\tilde{G}^N := \sup F_G^N(G^N)$ and $\tilde{G} \in \text{MIN}F_{\tilde{G}^N}(G)$ with state set $(\tilde{X}^N, \tilde{X})$, in the following we prove that given any fault-tolerant subplant pair (G', G'^N) of (G, G^N), for any state $x \in \tilde{X} - \tilde{X}^N$, $\rho(x, (G', G'^N)) \geq \rho(x, (\tilde{G}^N, \tilde{G}))$. First, we prove in the following lemma that the recovery-delay of a faulty state equals to the order at which it is added to the region of controllable attraction.

Lemma 1 *Consider a plant pair (G, G^N), an optimal fault-tolerant subplant $(\tilde{G}, \tilde{G}^N)$, and a faulty state $x \in X - X^N$. $\forall k \geq 1$, $x \in \Omega_k - \Omega_{k-1}$ if and only if $\rho(x, (\tilde{G}, \tilde{G}^N)) = k$, where Ω_k is the states included in the k-th iteration of the computation of the region of controllable attraction $\Omega_G^c(\tilde{X}^N)$ (see Algorithm 1).*

Proof: We prove by induction on k. For the base step, $k = 1$. For the forward implication, suppose, $x \in \Omega_1 - \Omega_0 = \Omega_1 - \tilde{X}^N$. Then by definition, $\tilde{\alpha}(x, \Sigma) \subseteq \tilde{X}^N$, and so it follows that $\rho(x, (\tilde{G}, \tilde{G}^N)) = 1$. On the other hand for the backward implication suppose $\rho(x, (\tilde{G}, \tilde{G}^N)) = 1$.

This implies the longest path from x to $\tilde{X}^N$ in $\tilde{G}$ is of length 1, implying that $\tilde{\alpha}(x,\Sigma) \subseteq \tilde{X}^N$, which in turn implies that $x \in \Omega_1$. Further since x is faulty, it follows that $x \in \Omega_1 - X^N \subseteq \Omega_1 - \tilde{X}^N = \Omega_1 - \Omega_0$.

For the inductive step, suppose $\rho(x,(\tilde{G},\tilde{G}^N)) = n$ if and only if $x \in \Omega_n - \Omega_{n-1}$, and consider the case when $k = n+1$. For the forward implication, since $x \in \Omega_{n+1} - \Omega_n$, then by definition, $\tilde{\alpha}(x,\Sigma) \subseteq \Omega_n$ and there exists $\sigma \in \Sigma$ such that $\tilde{\alpha}(x,\sigma) \in \Omega_n - \Omega_{n-1}$. Therefore, $\rho(x,(\tilde{G},\tilde{G}^N)) = 1 + \rho(\tilde{\alpha}(x,\sigma),(\tilde{G},\tilde{G}^N)) = 1+n$. On the other hand, for the backward implication, suppose $\rho(x,(\tilde{G},\tilde{G}^N)) = 1+n$. This implies that there exists $\sigma \in \Sigma$ such that $\rho(\tilde{\alpha}(x,\sigma),(\tilde{G},\tilde{G}^N)) = n$. From induction hypothesis, $\tilde{\alpha}(x,\sigma) \in \Omega_n - \Omega_{n-1}$. Since $x \in \Omega_G^c(\tilde{X}^N)$, the fact that $\tilde{\alpha}(x,\sigma) \in \Omega_n - \Omega_{n-1}$ implies $x \in \Omega_{n+1}$. On the other hand since $\rho(x,(\tilde{G},\tilde{G}^N)) = 1+n \neq n$, it follows from induction hypothesis that $x \notin \Omega_n$. Therefore, $x \in \Omega_{n+1} - \Omega_n$. This completes the proof.■

Above lemma states that when a state is added in the region of controllable attraction in the k-th step, the recovery-delay for this state in an optimal fault-tolerant control plant is k, and vise versa. To establish the minimality of the recovery-delay, the following theorem shows that any other controller has a larger recovery-delay.

Theorem 15 *Consider a plant pair (G,G^N), an optimal fault-tolerant subplant $(\tilde{G},\tilde{G}^N)$, a fault-tolerant subplant (G',G'^N) with $X'^N = \tilde{X}^N$, and a faulty state $x \in \tilde{X} - \tilde{X}^N$. Then $\rho(x,(G',G'^N) \geq \rho(x,(\tilde{G},\tilde{G}^N))$.*

Proof:

First note that if $x \notin X'$, then $\rho(x',(G',G'^N)) = \infty > \rho(x,(\tilde{G},\tilde{G}^N))$. So it suffices to consider $x \in X'$. Then since $x \in \tilde{X} - \tilde{X}^N$, $x \in X' - \tilde{X}^N = X' - X'^N$. Suppose $\rho(x,(\tilde{G},\tilde{G}^N)) = k \geq 1$. Then from Lemma 1, $x \in \Omega_k - \Omega_{k-1}$, where Ω_k is as defined in the statement of Lemma 1.

Suppose for contradiction that $\rho(x,(G',G'^N)) < k$. Then following the computation of $\Omega_G^c(\tilde{X}^N)$ as given in Algorithm 1, it must be the case that $x \in \Omega_j$ with $j < k$. This is a contradiction to the fact that $x \in \Omega_k - \Omega_{k-1}$. ■

4.6 Application Example

In this section, we provide an application example consisting of a simplified cooling-water system for gas turbine, shown in Figure 4.11.

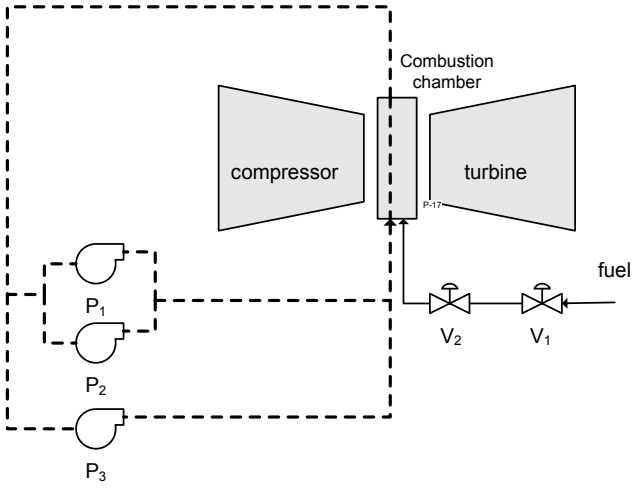

Figure 4.11 A simplified cooling water system for gas turbine

In Figure 4.11, there are three cooling-water pumps (P_1, P_2, and P_3), two fuel gas control valves (V_1 and V_2), a compressor, a gas chamber, and a turbine. The fuel gas valves control the fuel to the combustion chamber: higher the fuel supply, higher the turbine spinning speed, and higher the combustion chamber temperature. V_1 is the valve that operates under normal conditions, whereas V_2 is the emergency valve that operates when V_1 is stuck. The cooling-water is used to cool down the lubrication oil that lubricates the blade and shaft in the combustion chamber. For simplicity, we suppose that the cooling-water is directly supplied to the combustion chamber. Among the three pumps, P_1 is the one that operates under the normal conditions, and is called the leading pump. P_2 is a standby pump, called the lagging pump, and P_3 is the emergency pump which is used when an emergency action is needed.

When the system is idling, the pumps are off and valves are closed. When the system is turned on, emergency valve V_2 is kept fully open, whereas valve V_1 is controlled by a computer. When the turbine is working, pump P_1 circulates the cooling-water to take away the excess

heat from the combustion. Under normal situation, P_1 is adequate enough to keep the combustion chamber temperature in an acceptable range. When more fuel is injected, the chamber temperature increases. When a certain temperature limit T_1 is surpassed, P_2 is turned on to bring the chamber temperature to normal. If the temperature continues to rise and surpasses a higher limit T_2, emergency pump P_3 is turned on to further help cool down the chamber. In our example, P_2 and P_3 together are assumed powerful enough to lower the temperature in any situation provided they are turned on in a timely fashion.

In this example, the fuel control valve V_1 or the compressor may incur fault. The valve may get stuck open (so when less fuel is needed, a lot more is still sent to the combustion chamber). Compressor may fault to supply air at a higher pressure, causing generation of excessive heat in the combustion process. In this case the chamber temperature exceeds T_1, and the correct action after the occurrence of this fault is to turn on the lag pump P_2. If the temperature continues to rise, another temperature fault, which is the violation of limit T_2, may occur. In this case, the emergency pump P_3 is also turned on. When both P_2 and P_3 are running, special actions are taken to prevent the deterioration of the situation. Such special actions include the control of the emergency valve V_2 to limit the fuel, and the lowering of the air pressure of the compressor to limit the air flow to the combustion chamber. These emergency actions are canceled only when the temperature returns below T_1. During the time the temperature is rising, if both pumps are not turned on within a certain time, say $\tilde{t}$, the system may fail when a certain temperature limit T_3 is surpassed.

So, in this application, the two uncontrollable faulty events are the two temperature faults:

f_1	Temperature limit T_1 is violated
f_2	Temperature limit T_2 is violated

Also, there are two other uncontrollable events, corresponding to the decrease of the temperature due to the starting of the pumps:

d_1	Temperature decreases below T_1
d_2	Temperature decreases below T_2

The feasible controllable events are:

p_2	Switch-on pump P_2
$\overline{p_2}$	Switch-off pump P_2
p_3	Switch-on pump P_3
$\overline{p_3}$	Switch-off pump P_3
t_2	Switch-on pump P_2 in $\tilde{t}$ time
t_3	Switch-on pump P_3 in $\tilde{t}$ time
$\bar{t}$	Switch-on pump P_2 or P_3 after $\tilde{t}$ time

The states X are abstracted to represent the values of the 5 binary state variables, $s_1, \ldots, s_5$, the meanings of which are listed as follows:

Variable	Meaning	1	0
s_1	Pump P_2 is on?	yes	no
s_2	Pump P_3 is on?	yes	no
s_3	Temperature is over T_1?	yes	no
s_4	Temperature is over T_2?	yes	no
s_5	Temperature is over T_3?	yes	no

Note since pump P_1 remains on when the system is running, it's not necessary to track the state of P_1.

The states $--000$ ($-$ represents 1 or 0) are the normal states X^N. Since when s_4 is 1, s_3 is also 1, there is no states of the form $--01-$. When s_5 is 1, the system fails, and so such states are called *failed* states, and are represented as the same state denoted F.

The abstracted model of the cooling-water system is shown in Figure 4.12. In Figure 4.12, there are 12 "non-failed" states and one failed state. The initial state is also the final state, which is shown as the double-circle state. All solid edges represent controllable events and all dashed edges represent uncontrollable events. The dashed edges with double arrows are faulty events. All events are observable.

As shown in Figure 4.12, the overall plant model G has 13 states. and the nonfaulty part G^N consists of the states where no fault has occurred, i.e., the states with label $(--000)$. The

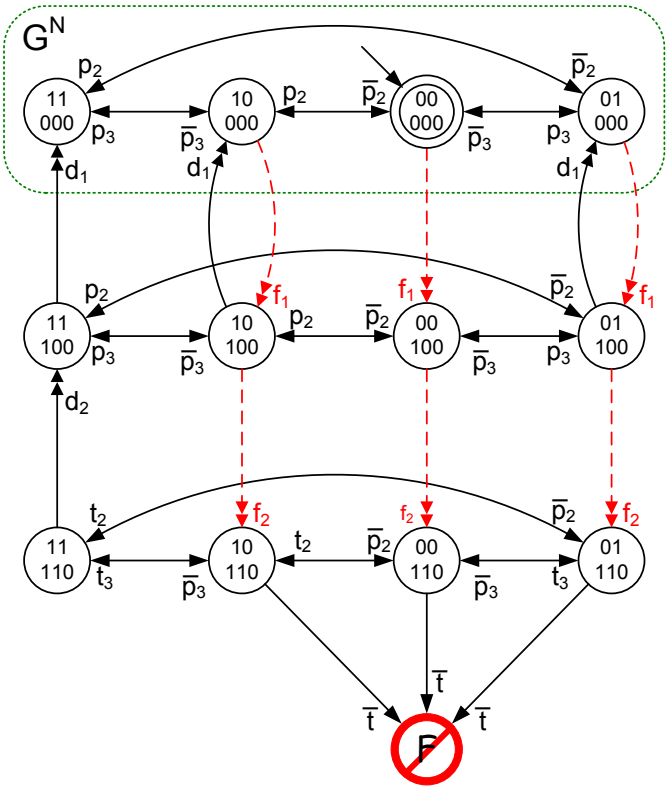

Figure 4.12 Model of cooling-water system of Figure 4.11

specification for the overall plant excludes all traces that reach the failed state, i.e., the failed state is deem forbidden for the overall plant. State 00000 is the initial and the final state, and the specification K^N for the nonfaulty plant excludes all traces of G^N that end at a non-final state.

In the following we apply Algorithm 2 to compute the optimal fault-tolerant subplant. The computation is accomplished in two steps. The result after the "Uncontrollable/blocking states removal" step is shown in Figure 4.13. In this step, only the failed state is removed, since all the transitions that reach the failed state are controllable.

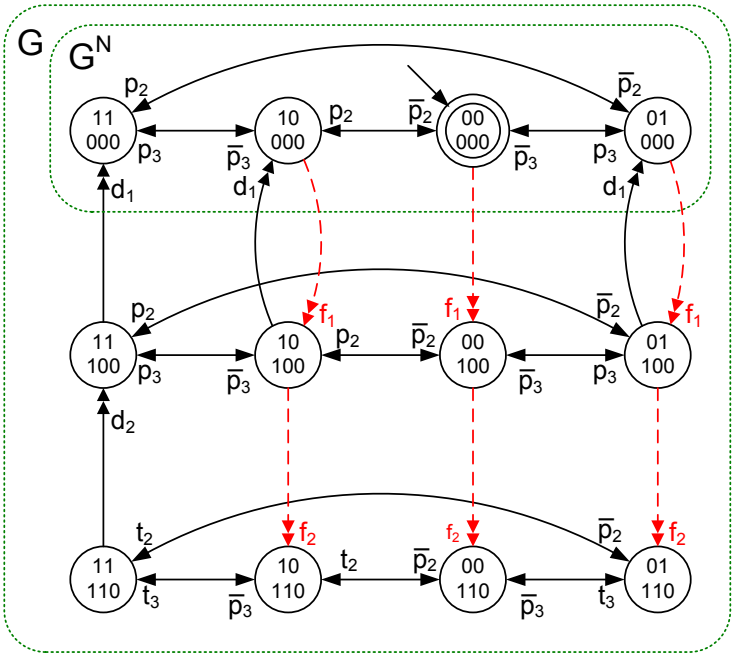

Figure 4.13 The controllable and nonblocking subplant of (G, G^N)

For the "optimal fault-tolerant subplant computation" step, we use Algorithm 1 to compute the region of controllable attraction of the nonfaulty states, as well as the optimal fault-tolerant subplant. Note that in this example all faulty states are controllably attractable except the failed state. Such attractable states get added into the region of controllable attraction in five iterations. During this computation, we also obtain the minimal set of transitions required for achieving controllable attractability, $\Delta_G(X^N) = \{(11100, d_1, 11000), (11110, d_2, 11100),$ $(10110, t_3, 11110), (01110, t_2, 11110), (10100, f_2, 10110), (01100, f_2, 01110), (00110, t_2, 10110),$

$(00100, f_2, 00110), (10100, d1, 10000), (01100, d1, 01000)\}$. It turns out that every state in the region of controllable attraction of the nonfaulty states is reachable from a nonfaulty state by executing a sequence of uncontrollable transitions, followed by a sequence of transitions belonging to the minimal set of transitions required for controllable attraction. Thus the minimal fault-tolerant subplant includes all states of the region of controllable attraction of the nonfaulty states, and the "optimal fault-tolerant subplant computation" terminates in one iteration. The resulting optimal fault-tolerant subplant is shown in Figure 4.14.

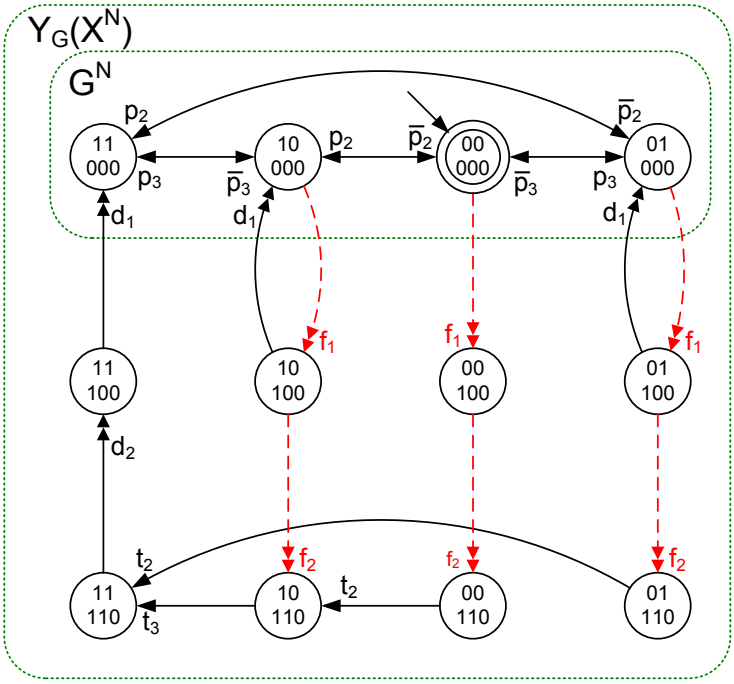

Figure 4.14 The optimal fault-tolerant subplant $\Upsilon_G(X^N)$

In Figure 4.14, it is clear that all faulty states are able to recover within a bounded delay. It can be seen that the nonfaulty part is supremal, since no nonfaulty state or transition of the nonfaulty part is removed, whereas the faulty part is minimal, since if any state or any transition in the faulty part is removed, the plant will no longer remain fault-tolerant. Finally it can be seen that safety and nonblockingness are also satisfied.

4.7 Conclusion

A notion of fault-tolerant supervisory control was introduced in our prior work [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007a)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007b)], [Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2008)] where the controlled system must not only satisfy the desired safety and progress properties but must also be fault-tolerant, i.e., following the occurrence of any fault a recovery to a nonfaulty or nonfaulty-equivalent state must occur within a bounded delay. Here we formulated the notion of an optimal fault-tolerant supervisor to be one that maximizes the nonfaulty behavior and at the same time minimizes the faulty behavior that must be tolerated, and also ensures safety, nonblockingness, and bounded-delay recovery. We showed that while the problem in general does not admit an optimal solution, an optimal solution does exist over the class of state-feedback control policies. We presented an algorithm to find such an optimal solution. The complexity of the algorithm is *quadratic* in the size of a given plant. The optimal fault-tolerant supervisor is also shown to minimize the recovery-delay for any faulty state.

CHAPTER 5. Decentralized Diagnosis of Event-Driven Systems for Safely Reacting to Failures

For the fault-tolerant control framework that we presented in earlier chapters, the diagnosis of a failure that may have occurred is not explicitly required. In this chapter we present a method for the explicit diagnosis of a failure after it occurs. The failure must be diagnosed prior to any safety specification is violated so that a recovery action can be taken.

In this chapter, we introduce the notion of *safe-codiagnosability*, extending the notion of safe-diagnosability [Paoli, A. and Lafortune, S. (2005)] to the decentralized setting, where there exist multiple diagnosers performing diagnosis using their own observations without communicating to each other. 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 [Qiu, W. and Kumar, R. (2004)], where the latter did not have any safety requirement. 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 as those proposed in [Qiu, W. and Kumar, R. (2004)] can be applied for off-line synthesis of individual diagnosers, as well as for on-line diagnosis using them.

[0]This research was a cooperative achievement with Wenbin Qiu (ISU 2006 graduate).

5.1 Introduction

Failure diagnosis is an active area of research, and has received considerable attention in the literature. A failure is a deviation from an expected or desired behavior. Various approaches have been proposed for failure diagnosis, including fault-trees, expert systems, neural networks, fuzzy logic, Bayesian networks, and analytical redundancy [Pouliezos, A. D. and Stavrakakis, G. S. (1994)]. 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 (DESs) – systems with discrete states that change when certain events occur, a certain model based approach for failure diagnosis is proposed in [Sampath, M. and Sengupta, R. and Lafortune, S. and Sinaamohideen, K. and Teneketzis, D. (1995)]. The property of *diagnosability* requires that once a failure has occurred, it be detected and diagnosed within bounded "delay" (within bounded number of transitions). The diagnosability can be tested polynomially as shown later in [Jiang, S. and Huang, Z. and Chandra, V. and Kumar, R. (2001)], [Yoo, T. S. and Lafortune, S. (2002)]. In [Sampath, M. and Lafortune, S. (1998)], the notion of active failure diagnosis was introduced where control is exercised to meet given specifications while satisfying diagnosability. In [Das, S. R. and Holloway, L. E. (2000)], [Pandalai, D. and Holloway, L. (2000)], a template based approach was developed for failure diagnosis in timed discrete event system. 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 [Lin, F. (1994)], [Zad, S. H. and Kwong, R. H. and Wonham, W. M. (2003)], where the occurrence of a failure is modeled as reaching of certain "faulty states". To facilitate generalization of failure specifications, linear-time temporal logic (LTL) based specification and diagnosis of its failure was proposed in [Jiang, S. and Kumar, R. (2004)]. A theory for failure diagnosis of repeatedly-occurring/intermittent failures was introduced in [Jiang, S. and Kumar, R. and Garcia, H. E. (2003)].

The above mentioned work dealt with *centralized* failure diagnosis, where a central diagnoser is responsible for failure detection and diagnosis in the system. [Debouk, R. and

Lafortune, S. and Teneketzis, D. (2000)] 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. [Sengupta, R. and Tripakis, S. (2002)] 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. The decentralized diagnosis problem with asymmetric communication was discussed in [Boel, R. K. and van Schuppen, J. H. (2002)], where communication is one-way and without delays. In a prior work [Qiu, W. and Kumar, R. (2004)], we studied the problem of decentralized failure diagnosis, where the system failure is diagnosed by multiple local diagnosers. A notion of *codiagnosability* was introduced to capture the fact that the occurrence of any failure must be diagnosed within bounded delay by at least one local diagnoser using its own observations of the system execution. Polynomial algorithms were provided for (i) testing codiagnosability, (ii) computing the delay bound of diagnosis, (iii) off-line synthesis of diagnosers, and (iv) on-line diagnosis using them.

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 [Paoli, A. and Lafortune, S. (2005)]. We extend this notion to the decentralized setting, where there exist multiple diagnosers performing diagnosis using their own observations without communicating to each other, by formulating the notion of *safe-codiagnosability*. 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*). 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. The above notion of safe-

codiagnosability may also be viewed as an extension of the notion of codiagnosability [Qiu, W. and Kumar, R. (2004)], where the latter did not have any safety requirement. 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 algorithms of polynomial complexity for verifying safe-codiagnosability. (The verification algorithm presented in [Paoli, A. and Lafortune, S. (2005)] was based upon the structural property of a deterministic diagnoser, and had an exponential complexity owing to the exponential size of the diagnoser.) For a safe-codiagnosable system, the same methods as those proposed in [Qiu, W. and Kumar, R. (2004)] can be applied for off-line synthesis of individual diagnosers, as well as for on-line diagnosis using them.

5.2 Notions and Preliminaries

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 $K^c := \Sigma^* - K$. If trace t is a *prefix* of trace s, it is denoted as $t \leq s$. Given a language $K \subseteq \Sigma^*$, its prefix-closure, denoted $pr(K)$, is defined as, $pr(K) := \{s \in \Sigma^* | \exists t \in K \text{ s.t. } s \leq t\}$, and K is said to be *prefix-closed* if $K = pr(K)$. 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 $K_1/K_2 := \{s \in \Sigma^* | \exists t \in K_2 \text{ s.t. } st \in K_1\}$. The set of *deadlocking traces* of a language K are those traces from which no further extensions exist in K, i.e., $s \in K$ is deadlocking trace if $\{s\}\Sigma^* \cap K = \{s\}$.

A DES is modeled as a *finite state machine* (FSM)/*finite automaton* (FA) G and is denoted by $G(X, \Sigma, \alpha, x_0)$, where X is the set of states, Σ is the finite set of events, $x_0 \in X$ is the initial state, and $\alpha : X \times \bar{\Sigma} \to 2^X$ is the transition function, where $\bar{\Sigma} := \Sigma \cup \{\epsilon\}$. G is said to be *deterministic* if $|\alpha(\cdot, \cdot)| \leq 1$ and $|\alpha(\cdot, \epsilon)| = 0$; otherwise, it is called *nondeterministic*. $(x, \sigma, x') \in X \times \bar{\Sigma} \times X$ is a transition of G if $x' \in \alpha(x, \sigma)$; it is an ϵ-transition if $\sigma = \epsilon$. 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 $X \times \bar{\Sigma}$ to domain $X \times \Sigma^*$ recursively as follows: $\forall x \in X, s \in \Sigma^*, \sigma \in \Sigma, \alpha(x, \epsilon) = \epsilon^*(x)$, and $\alpha(x, s\sigma) = \epsilon^*(\alpha(\alpha(x, s), \sigma)))$. The *generated language* by G is defined as $L(G) := \{s \in \Sigma^* | \alpha(x_0, s) \neq \emptyset\}$, i.e., it includes all traces that can be executed from the initial state of G. 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 $\sigma_i \in \bar{\Sigma}$ 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 $\bar{G} = \{\bar{X}, \Sigma, \bar{\alpha}, x_0\}$, where $\bar{X} := X \cup \{F\}$, and $\bar{\alpha}$ is defined as follows.

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

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

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

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

If the system execution is observed through a single global observer, we can define a *global observation mask* as $M : \bar{\Sigma} \to \bar{\Delta}$ with $M(\epsilon) = \epsilon$, where $\bar{\Delta} := \Delta \cup \{\epsilon\}$ and Δ is the set of observed symbols. The definition of M can be extended from events to event sequences inductively as follows:

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

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 $M_i : \bar{\Sigma} \to \bar{\Delta}_i$ $(i \in I = \{1, \cdots, m\})$, where m is the number of local observers, $\bar{\Delta}_i := \Delta_i \cup \{\epsilon\}$ and Δ_i is the set of locally observed symbols.

5.3 Safe-Codiagnosability

In this section, we present the definition of safe-codiagnosability and the "separation property" of safe-codiagnosability. As described in [Qiu, W. and Kumar, R. (2004)], *for the purpose of diagnosis, a system with deadlocking states can be converted to a deadlock free system by adding a self-loop labeled ϵ at each of its deadlocking state without affecting the diagnosis analysis.* So without loss of generality, we assume a system to be diagnosed, a "plant", to be deadlock free.

Definition 12 [Qiu, W. and Kumar, R. (2004)] Let L be the prefix-closed language generated by a plant, and K be a prefix-closed sublanguage specifying the non-faulty plant behavior $(K = pr(K) \subseteq L)$. Assume there are m local sites with observation masks $M_i : \bar{\Sigma} \to \bar{\Delta}_i$ $(i \in I = \{1, \cdots, m\})$. (L, K) is said to be *codiagnosable* with respect to $\{M_i\}$ if

$$\begin{aligned} &(\exists n \in \mathcal{N})(\forall s \in L - K)(\forall st \in L - K, |t| \geq n) \Rightarrow \\ &(\exists i \in I)(\forall u \in M_i^{-1} M_i(st) \cap L, u \in L - K) \end{aligned} \tag{5.1}$$

In the following lemma we provide an alternative definition of codiagnosability.

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 (5.1) in definition of codiagnosability requires that 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.,

$$\neg(\forall i \in I)(\exists u_i \in M_i^{-1}M_i(st) \cap L, u_i \in K) \tag{5.2}$$

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 $\underset{i\in I}{\cap} M_i^{-1}M_i(K)$. Thus the condition (5.2) can be equivalently written as, $st \notin \underset{i\in I}{\cap} M_i^{-1}M_i(K)$.

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,

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

■

Remark 7 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

$$(L - K) \underset{i\in I}{\cap} M_i^{-1}M_i(K) = \emptyset. \tag{5.3}$$

We say a faulty sublanguage $H \subseteq L - K$ is zero-delay codiagnosable with respect to $\{M_i\}$ if $H \underset{i\in I}{\cap} M_i^{-1}M_i(K) = \emptyset$.

Note that (5.3) is equivalent to,

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

i.e., (L, K) is zero-delay codiagnosable if and only if the non-faulty behavior K is *decomposable* [Rudie, K. and Wonham, W. M. (1992)] with respect to the non-faulty+faulty (plant) behavior L.

Definition 12 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 13 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 $M_i : \bar{\Sigma} \to \bar{\Delta}_i$ ($i \in I = \{1, \cdots, m\}$). (L, K, K_S) is said to be *safe-codiagnosable* with respect to $\{M_i\}$ if

$$\begin{aligned} &(\exists n \in \mathcal{N})(\forall s \in L - K)(\forall st \in L - K, |t| \geq n) \Rightarrow \\ &(\exists i \in I)(\exists v \in pr(st) \cap K_S) \\ &(\forall u \in M_i^{-1} M_i(v) \cap L, u \in L - K) \end{aligned} \tag{5.4}$$

Definition 13 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 13 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

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

Proof: The condition (5.4) in definition of 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.,

$$\neg(\forall i \in I)(\forall v \in pr(st) \cap K_S)$$
$$(\exists u_i \in M_i^{-1}M_i(v) \cap L, u_i \in K) \tag{5.5}$$

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,

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

which is the same set of traces,

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

which is the set

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

Note that a trace w belongs to this last set if and only if $pr(w) \subseteq [\underset{i\in I}{\cap} M_i^{-1}M_i(K) \cup K_S^c]$, 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 (5.5) can be equivalently written as, $st \notin supP[\underset{i\in I}{\cap} M_i^{-1}M_i(K) \cup K_S^c]$. 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,

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

■

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 14 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 $\underset{i\in I}{\cap} M_i^{-1}M_i(K)$ 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 16 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: ($\Leftarrow$) 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: $K_S^{\partial} \bigcap_{i \in I} M_i^{-1} M_i(K) = \emptyset$.

($\Rightarrow$) From Lemma 3 we have,

$$\exists n \in \mathcal{N} : [(L-K)\Sigma^{\geq n} \cap L]$$

$$\cap supP[\underset{i\in I}{\cap} M_i^{-1}M_i(K) \cup K_S^c] = \emptyset.$$

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. ■

5.4 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 16, 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

$$T = (G \| \bar{R}) \times R \times R$$

for verifying the codiagnosability of (L, K). This automaton is defined as $T = (Z, \Sigma_T, \gamma, z_0)$, where

- $Z = (X \times \bar{Y}) \times Y \times Y$.
- $\Sigma_T = \bar{\Sigma}^3$, where $\bar{\Sigma} = \Sigma \cup \{\epsilon\}$.
- $z_0 = ((x_0, y_0), y_0, y_0)$.
- $\gamma : Z \times \bar{\Sigma}^3 \to Z$ is defined as: $\forall z = ((x, y), y_1, y_2) \in Z, \sigma_T = (\sigma, \sigma_1, \sigma_2) \in \Sigma_T - \{(\epsilon, \epsilon, \epsilon)\}$, $\gamma(z, \sigma_T) := ((\alpha(x, \sigma), \bar{\beta}(y, \sigma)),\ \beta(y_1, \sigma_1), \beta(y_2, \sigma_2))$ 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

$$\exists i \in [k, l] \text{ s.t. } (\bar{y}^i = F) \wedge (\sigma^i \neq \epsilon), \tag{5.6}$$

where $z^i = ((x^i, \bar{y}^i), y_1^i, y_2^i) \in Z$, 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 $\bar{R}$ 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 $\bar{R}$ 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 [(Qiu, W. and Kumar, R. , 2004, Theorem 1)], in the following theorem we show the correctness of the test for checking zero-delay codiagnosability of K_S^{∂}.

Theorem 17 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 $s \in K_S^{\partial} \bigcap\limits_{i \in I} M_i^{-1} M_i(K)$, i.e., $K_S^{\partial} \bigcap\limits_{i \in I} M_i^{-1} M_i(K) \neq \emptyset$. 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 $s \in \bigcap\limits_{i \in I} M_i^{-1} M_i(K)$, 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 Lemma 17 and [Qiu, W. and Kumar, R. (2004)], we get the following corollary showing the correctness of Algorithm 3.

Corollary 5 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 (5.6) 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 8 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 [Qiu, W. and Kumar, R. (2004)] that the complexity for constructing the testing automaton T and checking codiagnosability of (L, K) is $O(|X| \times |Y|^{m+1} \times |\Sigma|^{m+1})$. 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 $O(|X| \times |Y_S| \times |Y|^m \times |\Sigma|^{m+1})$. It follows that overall complexity of checking safe-codiagnosability of (L, K, K_S) is, $O(|X| \times (|Y| + |Y_S|) \times |Y|^m \times |\Sigma|^{m+1})$.

Remark 9 In Algorithm 3, we use two testing automata T and T_S to verify safe-codiagnosability of (L, K, K_S). These two testing automata can be combined into a testing automaton $T' = (G \| R_S \| \bar{R}) \times R \times R$ by replacing $\bar{R}$ by $R_S \| \bar{R}$ 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 $O(|X| \times |Y_S| \times |Y|^{m+1} \times |\Sigma|^{m+1})$, which is an order higher. Thus the "separation" result obtained in Theorem 16 provides an order reduction in the complexity of testing safe-codiagnosability.

Once a system is deemed safe-codiagnosable, the same methods as those presented in [Qiu, W. and Kumar, R. (2004)] 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. Details are omitted here.

The following example illustrates how to verify the safe-codiagnosability using Algorithm 3.

Example 6 Figure 5.1 (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 $T = (G\|\bar{R}) \times R \times R$, 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 5.1 (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 5.2 (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 5.2 (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.

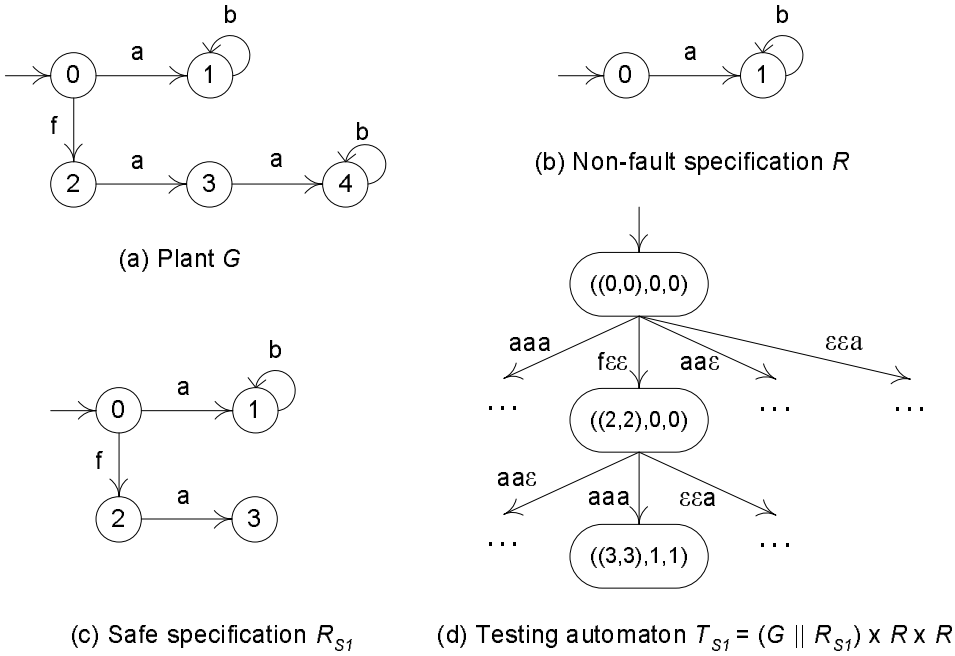

Figure 5.1 Models G, R and R_{S1}, and testing automaton T_{S_1} (right)

5.5 Conclusion

This paper studies 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 [Paoli, A. and Lafortune, S. (2005)] 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-

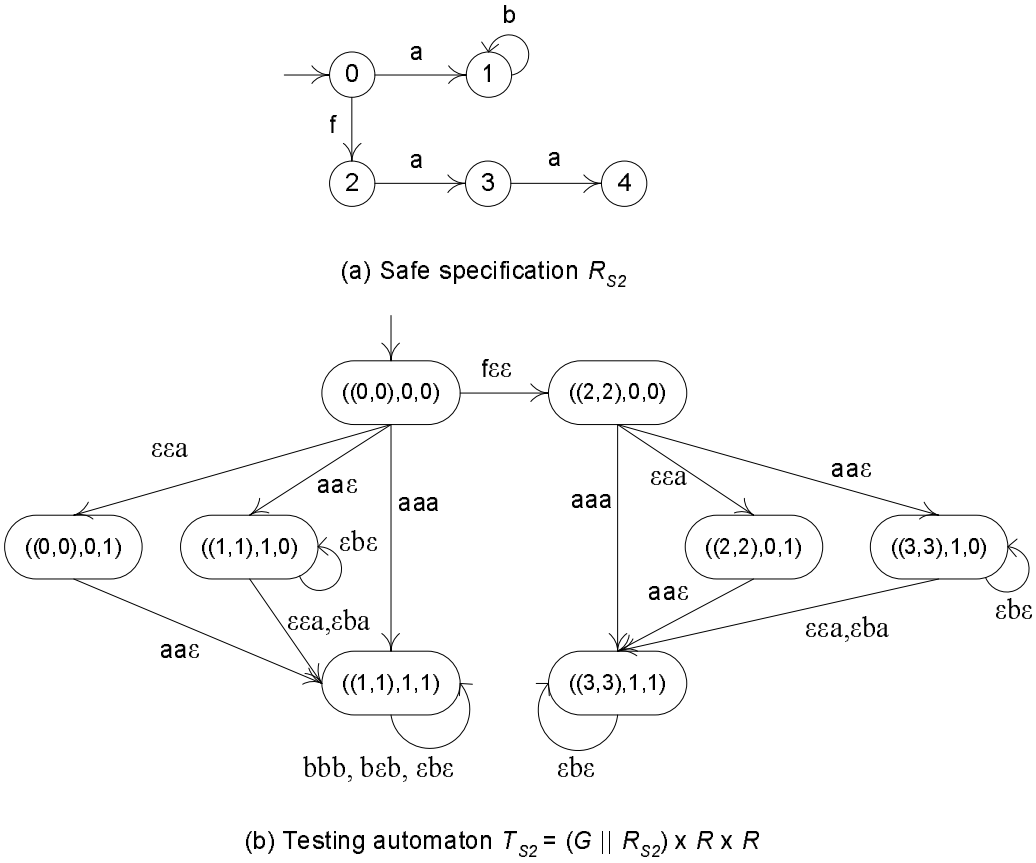

Figure 5.2 Safe specification model R_{S2} and testing automaton T_{S_2}

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 polynomial 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.

CHAPTER 6. Conclusion

6.1 Summarization of Dissertation

The main contributions of this dissertation on fault-tolerant supervisory control of discrete event systems include:

1. In this dissertation, we propose the problem of fault-tolerant supervisory control. Given a plant, possessing both faulty and nonfaulty behaviors, and a submodel for just the nonfaulty part, the goal of fault-tolerant supervisory control is to enforce a certain specification for the nonfaulty plant and another (perhaps more liberal) specification for the overall plant, and further to ensure that the plant recovers from any fault within a bounded delay so that following the recovery the system state is equivalent to a nonfaulty state (as if no fault ever happened). Formalizing this notion is the basis of the further study of existence condition and synthesizing algorithms.

2. We formulate the notion of fault-tolerant supervisory control and provide a necessary and sufficient condition for the existence of such a supervisor. This condition involves the usual notions of controllability, observability and relative-closure, together with the notion of state stability. Before putting effort on looking for a fault-tolerant supervisor, the test of the existence of such supervisor is necessary. This condition provides us a tool to check if it worths to exert ourselves to find a controlled subplant with fault tolerance.

3. We propose the problem of weakly fault-tolerant supervisory control. The conditions for fault-tolerant supervisor are strong such that, in real life, most systems are not able to achieve fault tolerance. Instead of recovering full functionality of the original system, a weakly fault-tolerant supervisor will allow the controlled system to achieve partial

functionality without violating safety specification. This problem is more realistic since most real systems cannot recover full functionality after repair.

4. We formulate the notion of weakly fault-tolerant supervisory control and provide a necessary and sufficient condition for the existence of such a supervisor. Failure of finding a fault-tolerant supervisory may urge us to find a supervisor achieving weak fault tolerance. With a weakly fault-tolerant supervisor, the controlled system will have a safe behavior, and the controlled behaviors will always be a subset of the original nonfaulty behaviors, such that all controlled behaviors are legal and no unsafe behavior will be performed.

5. As an extension, we introduce the notion of nonuniformly bounded fault tolerance. In the previous problems, all recoveries are achieved in an uniformly bounded delay. The nonuniformly bounded fault tolerance is an extreme situation of the previous problems. The condition for nonuniformly bounded fault tolerance is even weaker than weakly fault tolerance, but recovery is still guaranteed. This notion completes the formulation of fault tolerance.

6. We observe the nonexistence of some faulty/nonfaulty fault-tolerant behaviors. We find that neither the supremal nonfaulty fault-tolerant behavior nor the maximal nonfaulty fault-tolerant behavior does not exist in general. This observation forces us to restrict our attention to state-feedback based control, which gives us a direction in the design of synthesizing algorithms.

7. We propose the formulation of optimal fault-tolerant control synthesis problem. In this formulation, we set our goal in synthesizing the fault-tolerant supervisor. Here, "optimal" means the maximal nonfaulty behavior and minimal faulty behavior, which is reasonable since after control we do not want to limit the normal nonfaulty behavior, but the faulty recovery behavior should be as little as possible to prevent any possible violation of safety specification.

8. We give the algorithm to synthesize an optimal fault-tolerant supervisor. When the existence condition mentioned above is not satisfied, this algorithm will result in a supervisor

that maximizes the achievable nonfaulty behavior, minimizes the achievable faulty behavior and ensures safety, nonblockingness and bounded-delay recovery. The complexity of this algorithm is quadratic in the size of the plant to be controlled.

The main contributions of this dissertation on decentralized/distributed failure diagnosis of discrete event systems include:

1. We introduce the notion of safe-codiagnosability. The proposal of this notion is based on the purpose to study the system property of being able to react safely to failure in a decentralized setting. It is an extension of safe-diagnosability 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 fault within bounded delay and also before the system behavior becomes unsafe.

2. We establish the necessary and sufficient conditions for safe-codiagnosability. It shows that the property of safe-codiagnosability can be seperated into the properties of codiagnosability with "zero-delay codiagnosability" of "boundary safe traces".

3. We give an algorithm with polynomial complexity to verify safe-codiagnosability. This algorithm is based on checking whether there exists a situation that violates the condition of safe-codiagnosability. From the separation of safe-codiagnosability, the algorithm can also be seperated into three parts that first checks the codiagnosability, then checks the "boundary safe states", and finally checks zero-delay codiagnosability. The complexity of the algorithm is polynomial with respect to the size of the plant and specifications.

6.2 Future Research Topics

For the fault-tolerant supervisor control of discrete event systems, we provide an algorithm to synthesize an optimal fault-tolerant supervisor. A future direction is the algorithm for an optimal weakly fault-tolerant supervisor. First, a formalization of optimal fault-tolerant supervisor control synthesis problem needs to be proposed. Since the weak fault tolerance is defined on the basis of language stability, the definition of optimality will be based on supremal

nonfaulty sublanguage and minimal faulty sublanguage. The design of the algorithm may use the idea of our existed algorithm, if we can take the recovered faulty states (after which the behaviors are subsumed by the behaviors after some nonfaulty states) as nonfaulty states and apply the existed algorithm.

For some systems, it is hard to recover to normal system, but some degraded behaviors are acceptable. If returning to the nonfaulty part is not achievable, the supervisor should direct the system to those states with acceptable degraded faulty behaviors. Generally, preference should be given to the states, a extended algorithm should be able to control the system to reach the states with highest preference.

For the problems we study in this dissertation, all events are observable. That is, the supervisor knows every transition in the system, therefore knows the current state of the system. A possible future direction is to consider partial observability. In this case, the occurrence of some events is not detectable. Since in real system, detecting sensors are not able to detect all events, which makes this direction more reasonable. In this extension, the supervisor will not be able to know the current state, which will make the achievement of fault tolerance and safety more difficult.

All problems in this dissertation assume only one supervisor. Modern large systems always require the cooperation of multiple subsystems. Since ability of each single supervisor is limited by hardware or software conditions, the large systems may need several controllers, each of which is in charge of one part of the whole system. This motivates us to consider the possibility of decentralized/distributed fault-tolerant control. In this decentralized/distributed fault-tolerant control, each controller is responsible only to detect a portion of the whole observable events and can only control a portion of the whole controllable events. The fault tolerance of the whole system is achieved by the cooperation of all the controllers.

Another possible extension of this dissertation is to consider time in the problem. In real-time system, not only the sequence of the execution of the actions but also the timing is critical. In this dissertation, we consider only untimed discrete event systems. To study real-time system, we need to change our attention to timed discrete event systems. In this extension,

the fault-tolerant supervisor will not only achieve fault tolerance but also meet certain real-time deadlines by considering the timing properties in timed discrete event systems. That is, all tasks should be finished before certain deadline is reached, which implies more limited selection of enabled events.

The research in this dissertation may also be extended to hybrid system, which is currently a popular research area. In a hybrid system, the system dynamics is described by both continuous and discrete-event models. A simple example of the hybrid system is the embedded system where both analog and digital inputs/outputs exist. Analog signals are usually described by continuous model, while digital signals are usually described by discrete-event models. To apply the theories developed in this dissertation, some abstraction techniques may be needed for the continuous part. With a correct model, these discrete-event theories can be used to control hybrid systems.

BIBLIOGRAPHY

Cassandras, C. G. and Lafortune, S. (1999). *Introduction to Discrete Event Systems* Boston: Kluwer Academic Publishers.

Kumar, Ratnesh and Garg, Vijay K. (1995). *Modeling and Control of Logical Discrete Event Systems* Boston: Kluwer Academic Publishers.

Avižienis, A. and Laprie, J.-C. and Randell, B. (2000). Fundamental Concepts of Dependability. *Proceedings of the 3rd Information Survivability Workshop (ISW'2000)*, 3–18.

Pierce, W. H. (1965). *Failure-Tolerant Computer Design*. Academic Press.

Voas, J. (2001). Fault Tolerance. *IEEE Software*, *18*(4), 54–57.

Jalote P. (1998). *Fault Tolearnce in Distributed Systems*. Printice Hall.

Weber, D. G. (1989). Formal specification of fault-tolerance and its relation to computer security. *SIGSOFT Softw. Eng. Notes*, *14*(3), 273–277. ACM Press.

Lamport, L. and Shostak, R. and Pease, M. (1980). Reaching agreement in the Presence of Faults. *Journal of the Association of Computing Machinery*, *27*(2), 228-234.

Lamport, L. and Shostak, R. and Pease, M. (1982). The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, *4*(3), 382-401.

Berman, P. and Garay, J. A. and Kerry, K. J. (1989). Towars Optimal Distributed Consensus. *Proceedings of the 30th FOCS*, 410-415.

Berman, P. and Garay, J. A. (1991). Efficient Distributed Consensus with n=(3+ϵ)t Processors. *Proceedings of the 5th International Workshop on Distributed Algorithms*, Lecture Notes in Computer Science. *579*

Garay, J. A. and Moses, Y. (1998). Fully Polynomial Byzantine Agreement for n>3t Processors in t+1 Rounds. *SIAM Journal on Computing*, *27*(1), 247-290.

Gartner, F. C. (1999). Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, *31*(1), 1–26.

Laprie, J.-C. and Arlat, J. and Beounes, C. and Kanoun, K. (1990). Definition and analysis of hardware- and software-fault-tolerant architectures. *IEEE Computer*, *23*(7), 39–51.

Selic, B. (2002). *Fault Tolerance Techniques for Distributed Systems. http://www-106.ibm.com/developerworks/rational/library/114.html.*

Schlichting, R. D. and Schneider, F. B. (1983). Fail-Stop Processors: An Approach to Designing Fault-Tolerant Computing Systems. *IEEE Transactions on Computing Systems*, *1*(3), 222-238.

Schneider, F. B. (1983). *Fail-stop processors.* New York: IEEE Computer Society.

Blanke, M. and Staroswiecki, M. and Wu, N. E. (2001). Concepts and Methods in Fault-Tolerant Control. *Proceedings of the American Control Conference*, 410-415.

Blanke, M. and Izadi-Zamanabadi, R. and Bogh, S. A. and Lunau, C. P. (1997). *Fault Tolerant Control Systems - A Holistic View. PDept. of Control Engineering Report (R-1997-4175), Aalborg University.*

Dega, J.-L. (1996). *The Rudundancy mechanisms of the Ariane 5 operational control center.* New York: IEEE Computer Society.

Blanke, M. (1996). *A component based approach to industrial fault detection and accommodation. Proceedings of the IFAC World Congress* San Frasisco, Vol. N, 97-102

Willsky, A. S. (1976). A survey of design method for failure detection in dynamic systems. *Automatica*, *12*(6), 601-611.

Isermann, R. (1984). Process Fault Detection Based on Modelling and Estimation Methods. *Automatica*, *20*, 387-404.

Gertle, J. J. (1988). Survey of model-based failure detection and isolation in complex plants. *IEEE Control System Magazine*, *8*(6), 3-11.

Garg, V. K. and Mitchell, J. R. (1998). Distributed predicate detection in a faulty environment. *Proceedings of the 18th International Conference on Distributed Computing Systems*, 416-423.

Chase, C. M. and Garg, V. K. (1998). Detection of Global Predicates: Techniques and Their Limitations. *Distributed Computing*, *11*(4), 191-201.

Gartner, F. C. and Kloppenburg, S. (2000). Consistent detection of global predicates under a weak fault assumption. *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems*, 94-108.

Chow, E. Y. and Willsky, A. S. (1984). Analytical Redundancy and the Design of Robust Failure Detecting Systems. *IEEE Transactions on Automatic Control*, *29*(7), 603-614.

Cunningham, T. B. and Poyneer, R. D. (1977). Sensor Failure Detection using Analytical Redundancy. *Proceedings of Joint Automatic Control Conference*, 278-287.

Shapiro, E. Y. and Decarli, H. E. (1979). Analytical Redundancy for Flight Control Sensors on the Lockheed L-1011 Aircraf. *Proceedings of the 18th IEEE CDC Conference*, 449-454.

Stuckenberg, N. (1985). Sensor Failure Detection in Flight Control Systems Using Deterministic Observers. *Proceedings of the 7th IFAC Symposium on Identification and System Parameter Estimation*, *1*, 705-710.

Merrill, M. C. (1985). Sensor Failure Detection for Jet Engines Using Analytical Redundancy. *Journal of Guidance, Control, and Dynamics*, *8*(6), 673-682.

Leuschen, M. L. and Walker, I. D. and Cavallaro, J. R. (2005). Fault Residual Generation via Nonlinear Analytical Redundancy. *IEEE Transactions on Control Systems Technology*, *13*(3), 452-458.

Patton, R. J. and Frank, P. M. and Clark, R. N. (2000). *Issues of Fault Diagnosis for Dynamic Systems*. London: Springer-Verlag.

Patton, R. J. and Frank, P. and Clarke, D. (1989). *Fault Diagnosis in Dynamic Systems: Theory and Applications*. Prentice Hall.

Gertler, J. (1995). Towards a theory of dynamic consistency relations. *Proceedings of the IFAC Workshop On-line Fault Detection and Supervision in the Chemical Process Industries*, 143-156.

Patton, R. (1993). Robustness issues in fault-tolerant control. *IEE Colloquium on Fault Diagnosis and Control System Reconfiguration*, *1*, 1-25.

Basseville, M. and Nikiforov, I. (1994). *Statistical Change Detection*. Prentice Hall.

Sampath, M. and Sengupta, R. and Lafortune, S. and Sinnamohideen, K. and Teneketzis, D. C. (1996). Fault Diagnosis Using Discrete-Event Models. *IEEE Transactions on Control Systems Technology*, *4*(2), 105-124.

Blanke, M. and Kinnaert, M. and Lunze, J. and Staroswiecki, M. (2003). *Diagnosis and Fault Tolerant Control*. Berli: Springer-Verlag.

Sampath, M. and Sengupta, R. and Lafortune, S. and Sinnamohideen, K. and Teneketzis, D. C. (1995). Diagnosability of Discrete Event Systems. *IEEE Transactions on Automatic Control*, *40*(9), 1555-1575.

Qiu, W. (2005). *Decentralized/distributed failure diagnosis and supervisory control of discrete event systems*. Department of Electrical and Computer Engineering, Iowa State University.

Debouk, R. and Lafortune, S. and Teneketzis, D. (2000). Coordinated Decentralized Protocols for Failure Diagnosis of Discrete Event Systems. *Discrete Event Dynamic Systems: Theory and Applications*, *10*(1-2), 33–86.

Jiang, S. and Kumar, R. (2002). Failure Diagnosis of Discrete Event Systems with Linear-Time Temporal Logic Fault Specifications. *Proceedings of 2002 American Control Conference*, 128-133.

Jiang, S. and Kumar, R. (2003). Diagnosis of Repeated Failures for Discrete Event Systems with Linear-Time Temporal Logic Specifications. *Proceedings of IEEE Conference on Decision and Control*, 3221-3226.

Jiang, S. and Kumar, R. and Garcia, H. E. (2003). Diagnosis of Repeated/intermittent Failures in Discrete Event Systems. *IEEE Transactions on Automatic Control*, *19*(2), 310-323.

Sampath, M. and Lafortune, S. (1998). Active Diagnosis of Discrete Event Systems. *IEEE Transactions on Automatic Control*, *43*(7), 908-929.

Qiu, W. and Kumar, R. (2006). Decentralized Failure Diagnosis of Discrete Event Systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, *36*(2), 384-395.

Zad, S. H. and Kwong, R. H. and Wonham, W. M. (2003). Fault Diagnosis in Discrete Event Systems: Framework and Model Reduction. *IEEE Transactions on Automatic Control*, *48*(7), 1199-1212.

Lygeros, J. and Godbole, D. N. and Broucke, M. (2000). A fault tolerant control architecture for automated highway system. *IEEE Transactions on Control Systems Technology*, *8*(2), 205-219.

Godbole, D. N. and Lygeros, J. and Singh, E. and Deshpande, A. and Lindsey, A. E. (2000). Communication protocols for a fault-tolerant automated highway system. *IEEE Transactions on Control Systems Technology*, *8*(5), 787-800.

Beneveniste, A. and Fabre, E. and Haar, S. and Jard, C. (2003). Diagnosis of asynchronous discrete event systems: a net unfolding approach. *IEEE Transactions on Automatic Control*, *48*(5), 714-727.

Bouloutas, A. and Hart, G. W. and Schwartz, M. (1992). Simple finite-state fault detectors for communication networks. *IEEE Transactions on Communications*, *40*(3), 477-479.

Miller, R. E. and Arisha, A. K. (2001). Fault identification in networks by passive testing. *Proceedings of IEEE Annual Simulation Symposium*, 277-284.

Das, S. R. and Holloway, L. E. (2000). Characterizing a confidence space for discrete event timing 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.

Pandalai, D. and Holloway, L. (2000). Template languages for fault monitoring of timed discrete event processes. *IEEE Transactions on Automatic Control*, *45*(5), 868-882.

Lin, F. (1994). Diagnosability of discrete event systems and its applications. *Discrete Event Dynamic Systems: Theory and Applications*, *4*(1).

Westerman, G. and Kumar, R. and Stroud, C. and Heath, J. R. (1998). Discrete event systems approach for delay fault analysis in digital circuits. *roceedings of 1998 American Control Conference*, *1*, 239-243.

Hadjicostis, C. N. and Verghese, G. C. (2001). Power system monitoring based on relay and circuit breaker information. *Proceedings of the 2001 IEEE International Symposium on Circuits and Systems*, *2*, 197-200.

Abdelwahed, S. and Wu, J. and Biswas, G. and Ramirez, J. and Manders, E. (2005). Online Fault Adaptive Control for Efficient Resource Management in Advanced Life Support Systems. *Habitation - International Hournal for Human Support Research*, *10*(2), 105-115.

Ji, M. and Zhang, Z. and Biswas, G. and Sarkar, N. (2003). Hybrid Fault Adaptive Control of a Wheeled Mobile Robot. *IEEE/ASME Transactions on Mechatronics*, *8*(2), 226-233.

Karsai, G. and Biswas, G. and Pasternak, T. and Narasimhan, S. and Pecili, G. and Simon, G. and Kovacshazy, T. (2001). Fault Adaptive Control: a CBS Application. *Proceedings of the 8th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, 205-211.

Simon, G. and Karsai, G. and Biswas, G. and Abdelwahed, S. and Mahadevan, N. and Szemethy, T. and Pecili, G. and Kovacshazy, T. (2003). Model-Based Fault Adaptive Control of Complex Dynamic Systems. *Proceedings of the 20th IEEE Instrumentation and Measurement Technology Conference*, *1*, 176-181.

Simon, G. and Kovacshazy, T. and Pecili, G. and Szemethy, T. and Karsai, G. and Ledeczi, A. (2002). Implementation of reconfiguration management in fault-adaptive control systems. *Proceedings of the 19th IEEE Instrumentation and Measurement Technology Conference*, *1*, 123-127.

Looze, D. and Weiss, J. and Eterno, J. and Barrett, N. (2005). An automatic redesign approach for restructurable control systems. *IEEE Control Systems Magazine*, *5*(2), 16-22.

Elgersma, M. and Glavaški, S. (2001). Reconfigurable control for active management of aircraft system failures. *Proceedings of the American Control Conferenc*, 2627-2639.

Liu, J. and Darabi, H. (2004). Control reconfiguration of discrete event systems controllers with partial observation. *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, *34*(6), 2262-2272.

Moitra, A. and Joseph, M. (1983). Cooperative recovery from faults in distributed programs. *Proceedings of IFIP 9th World Congress*, 481-486.

Bernardeschi, C. and Fantechi, A. and Simoncini, L. (2000). Formally Verifying fault tolerant system designs. *The Computer Journal*, *43*(3), 191–205.

Liu, Z. and Joseph, M. (1999). Specification and Verification of Fault-Tolerance, Timing, and Scheduling. *IEEE Transactions on Programming Languages and Systems*, *21*(1), 46-89.

Liu, Z. and Joseph, M. (1996). Verification of Fault Tolerance and Real Time. *Proceedings of the 26th IEEE Symposium on Fault Tolerant Computing Systems (FTCS-26)*, 220-229.

Abadi, M. and Lamport, L. (1988). The Existence of Refinement Mapping. *Proceedings of 3rd IEEE Symposium on Logi and Computer Science*, 165-175.

Lincoln, P. and Rushby, J. (1993). a formally verified algorithm for interactive consistency under a hybrid fault model. *Digest of papers of the 23th IEEE Symposium on Fault Tolerant Computing Systems (FTCS-23)*, 402-411.

Ayache, S. and Conguet, E. and Humbert, P. and Rodriguez, C. and Sifakis, J. and Gerlich, R. (1996). Formal Methods for the Validation of Fault Tolerance in Autonomous Spacecraft. *Proceedings of the 26th IEEE Symposium on Fault Tolerant Computing Systems (FTCS-26)*, 353-357.

Lin, F and Wonham, W. M. (1988). On observability of discrete-event systems. *Information Sciences*, *44*(3), 173-198.

Brave, Y. and Heymann, M. (1990). On stabilization of discrete event processes. *International Journal of Control*, *51*(5), 1101-1117.

Özveren, C. M. and Willsky, A. S. and Antsaklis, P. J. (1991). Stability and Stabilizability of Discrete event dynamical systems. *Journal of ACM*, *38*(3), 730-752.

Kumar, R. and Garg, V. K. and Marcus, S. I. (1993). Language stability and stabilizability of discrete event dynamical systems. *SIAM Journal of Control and Optimization*, *31*(5), 1294-1320.

Willner, Y. and Heymann, M. (1995). Language convergence in controlled discrete-event systems. *IEEE Transactions on Automatic Control*, *40*(4), 616-627.

Lafortune, S. and Lin, F. (1991). On tolerable and desirable behaviors in supervisory control of discrete event systems. *Discrete Event Dynamical System: Theory and Application*, *1*(1), 61-92.

Jensen, R. M. (2003). *DES Controller Synthesis and Fault Tolerant Control: A Survey of Recent Advances*. Technical Report TR-2003-40. IT University of Copenhagen.

Darabi, H. and Jafari, M. A. and Buczak, A. L. (2003). A Control Switching theory for Supervisory Control of Discrete Event SystemsI. *IEEE Transactions on Robotics and Automation*, *19*(1), 131-137.

Rohloff, K. R. (2005). Sensor Failure Tolerant Supervisory Control. *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005*, 3493 - 3498.

Cho, K. -H. and Lim, J. -T. (1996). Failure Diagnosis and Fault Tolerant Supervisory Control Systems. *IEICE Transactions on Information and System*, *E79-D*(9), 1223 - 1231.

Cho, K. -H. and Lim, J. -T. (1998). Synthesis of Fault Tolerant Supervisor for Automated Manufacturing Systems: A Case Study on Photolithographic Process. *IEEE Transaction on Robotics and Automation*, 348 - 351.

Zhou, M. C. and Dicesare, F. (1989). Adaptive Design Of Petri Net Controllers For Error Recovery In Automated Manufacturing Systems. *IEEE Transactions on Systems, Man and Cybernetics*, *19*(5), 963 - 973.

Takai, S. and Ushio, T. (2000. Reliable decentralized supervisory control of discrete event systems. *IEEE Transations on System, Man, and Cybernetics—Part B*, *30*(5), 661-667.

Iordache, M. V. and Antsaklis, P. J. (2004). Resilience to Failure and Reconfigurations in the Supervision Based on Place Invariants. *Proceedings of the 2004 American Control Conference*, 4477 - 4482.

Arora, A. and Gouda, M. (1993). Closure and Convergence: A Foundation of Fault-Tolerant Computing. *IEEE Transactions on Software Engineering, Special issue on software reliability*, *19*(11), 1015 - 1027.

Arora, A. and Kulkarni, S. S. (1998). Component based design of multitolerant systems. *IEEE Transactions on Software Engineering*, *24*(1), 63-78.

Attie, P. C. and Arora, A. and Emerson, E. A. (2004). Synthesis of Fault-Tolerant Concurrent Programs. *ACM Transactions on Programming Languages and Systems*, *26*(1), 125-18.

Anderson, P. M. and Fouad, A. A. (1994). *Power System Control and Stability*. New York: IEEE Press.

Ramadge, P. J. and Wonham, W. M. (1987). Supervisory Control of a class of Discrete Event Processes. *SIAM Journal of Control and Optimization*, *25*(1), 206-230.

Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2008). A framework for fault-tolerant supervisory control of discrete event systems. *IEEE Transaction on Automatic Control*, *53*(8), 1839-1849

Paoli, A. and Lafortune, S. (2005). Safe diagnosability for Fault-Tolerant Supervision of discrete event systems. *Automatic*, *41*(8), 1335-1347.

Qiu, W. and Kumar, R. (2004). Decentralized Failure Diagnosis of Discrete Event Systems. *Proceedings of 2004 International Workshop on Discrete Event Systems*, 145-150.

Pouliezos, A. D. and Stavrakakis, G. S. (1994). *Real time fault monitoring of industrial processes*. Boston, MA: Kluwer Academic Publishers.

Sampath, M. and Sengupta, R. and Lafortune, S. and Sinaamohideen, K. and Teneketzis, D. (1995). Diagnosability of Discrete Event Systems. *IEEE Transactions on Automatic Control*, *40*(9), 1555-1575.

Jiang, S. and Huang, Z. and Chandra, V. and Kumar, R. (2001). A Polynomial Time Algorithm for Diagnosability of Discrete Event Systems. *IEEE Transactions on Automatic Control*, *46*(8), 1318-1321.

Yoo, T. S. and Lafortune, S. (2002). Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, *47*(9), 1491-1495.

Sampath, M. and Lafortune, S. (1998). Active diagnosis of discrete event systems. *IEEE Transactions on Automatic Control*, *43*(7), 908-929.

Das, S. R. and Holloway, L. E. (2000). 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.

Pandalai, D. and Holloway, L. (2000). Template languages for fault monitoring of timed discrete event processes. *IEEE Transactions on Automatic Control*, *45*(5), 868-882.

Lin, F. (1994). Diagnosability of discrete event systems and its applications. *Discrete Event Dynamic Systems: Theory and Applications*, *4*(1), 197-212.

Zad, S. H. and Kwong, R. H. and Wonham, W. M. (2003). System-assigned learning strategies and CBI. *IEEE Transactions on Automatic Control*, *48*(7), 1199-1212.

Jiang, S. and Kumar, R. (2004). Failure Diagnosis of Discrete Event Systems with Linear-time Temporal Logic Fault Specifications. *IEEE Transactions on Automatic Control*, *49*(6), 934-945.

Jiang, S. and Kumar, R. and Garcia, H. E. (2003). Diagnosis of repeated/intermittent failures in discrete event systems. *IEEE Transactions on Robotics and Automation*, *19*(2), 310-323.

Debouk, R. and Lafortune, S. and Teneketzis, D. (2000). Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamical Systems: Theory and Applications*, *10*, 33-79.

Sengupta, R. and Tripakis, S. (2002). Decentralized diagnosis of regular language is undecidable. *Proceedings of IEEE Conference on Decision and Control*, 423-428.

Boel, R. K. and van Schuppen, J. H. (2002). Decentralized failure diagnosis for discrete-event systems with constrained communication between diagnosers. *Proceedings of International Workshop on Discrete Event Systems*,

Rudie, K. and Wonham, W. M. (1992). Think globally, act locally: decentralized supervisory control. *IEEE Transactions on Automatic Control*, *37*(11), 1692-1708.

Wen, Q. and Kumar, R. and Huang, J. (2008). Synthesis of Fault-Tolerant Supervisory Control for Discrete Event Systems. *Proceedings of 2008 American Control Conference*, Seattle, WA.

Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007b). Weakly Fault-Tolerant Supervisory Control of Discrete Event Systems. *Proceedings of 2007 American Control Conference*, New York, NY.

Wen, Q. and Kumar, R. and Huang, J. and Liu, H. (2007a). Fault-tolerant supervisory control of discrete event systems : Formulation and existence results. *Proceedings of Dependable Control of Discrete Systems*, Paris, France.