NOTE TO USERS

FAULT RECOVERY IN DISCRETE-EVENT SYSTEMS USING OBSERVER-BASED SUPERVISORS

Chenhuan Wang

A thesis

in

The Department

of

Electrical and Computer Egineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

June 2005

Your file Votre référence
ISBN: 0-494-10253-5
Our file Notre référence
ISBN: 0-494-10253-5

Abstract

Fault Recovery in Discrete-Event Systems using Observer-Based Supervisors

Chenhuan Wang

Fault recovery is one of the crucial tasks of supervisors of mission-critical and safety-critical systems. In this thesis, we study the synthesis of recovery procedures using discrete-event models.

It is assumed that the plant can be modeled as a finite-state automaton describing both normal and faulty behaviors. The faults are assumed permanent. Furthermore, it is assumed that a diagnosis system is available which can detect and isolate the faults with a bounded delay. No other assumptions are made about the diagnosis system and in fact, it may be designed based on any technique (continuous-variable or discrete-event). The combination of the plant and diagnosis system is the system to be controlled. This system has three modes: normal (when the plant operates with no faults), transient (when a fault has occurred but not detected by the diagnosis system) and recovery (when the fault is diagnosed and appropriate actions for recovery may be taken).

We solve the supervisory design problem using a state-based approach. It is assumed that design specifications are given for normal, transient and recovery modes

in terms of legal (safe) states. The system under supervision is also required to be nonblocking in normal and recovery modes.

Following a modular switching approach, we propose supervisory schemes in which separate supervisor modules are designed for normal, transient and recovery modes. We consider failure accommodation in cases where recovery to normal operation is not possible and also recovery in cases in which it is possible to resume normal operation. For each case, we provide two solutions, one in which the recovery supervisor is in the feedback loop when the system is started in its normal mode, and another solution in which the recovery supervisor is engaged only when a fault is detected and isolated. The latter approach is less computationally complex to implement. We investigate supervisor admissibility and nonblocking property of the system under supervision.

All of the supervisor modules are observer-based. In our opinion, the use of observer-based supervisors results in a more transparent solution and simplifies the analysis in our switching scheme when one supervisor replaces another in the feedback loop. In this thesis, in the process of our study of fault recovery, we also propose a systematic method for designing observer-based supervisors using normal languages.

Acknowledgments

I sincerely appreciate the help and financial support of my supervisor, Dr. Shahin Hashtrudi Zad, who has guided me very patiently along since the very beginning. I would also like to thank all the professors and staff members as well as my colleagues in Concordia University's Electrical and Computer Engineering, who taught or helped me during this research.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

In control systems, component failures such as sensor failures could degrade the system performance and reduce the controller's ability to perform effective control action. Therefore, performing fault recovery is very important for safety and performance considerations. A well designed supervisor should be able to take appropriate actions (such as control system reconfiguration) to accommodate failures. In this thesis, we study fault recovery in systems that can be modeled as discrete-event systems (DES).

The rapid evolution of computing and communication technologies has resulted in the design of highly complex control systems for which differential and difference equations have become inadequate and inappropriate to be employed in modeling and control. In order to better study the behavior of these complex, discrete-event models

are used. Discrete-event models are encountered in a variety of application domains such as manufacturing systems, communication networks, traffic control systems and information systems. A framework for the control of discrete-event systems was introduced by P.J.Ramadge and W.M.Wonham (RW) in 1982 ([15]). They developed a new set of modeling, analysis and design techniques suitable for the general logic principles of the operation of control systems at a higher level of abstraction.

In the RW theory, an uncontrolled plant is modeled as an automaton describing the structure and behavior of the plant. A control logic, supervisor, is designed to disable or enable certain (controllable) events in an appropriate way in order to prevent the system from exhibiting undesirable behavior. Moreover, "liveness" property (in the sense of avoiding deadlocks and livelocks) is also studied and taken into consideration.

Any malfunction of a component in a system is called fault or failure. We will not differentiate between fault and failure in this thesis. There are two kinds of failures: *permanent* and *nonpermanent*. We assume that faults are permanent in our framework. A failure is permanent if the system stays in this faulty condition unless some recovery procedures are involved, while nonpermanent failures do not persist and disappear after some time. For example, ground connection in the high-voltage transmission system because of storm may be permanent or nonpermanent. Failures can also be classified into two categories according to whether the system can recover to normal mode or not (in case of failure).

In literature, fault recovery has been studied in various types of control systems including DES-based [3],[5],[12]. In [3], a failure analysis technique is proposed using discrete-event models, and then a fault-tolerant supervisor is synthesized. Fault and failure are considered to be different kinds of events according to the degree of the degradation in the system performance. Fault represents the malfunction of a component and is tolerable, while a failure causes the breakdown of a system's component. An event sequence is said to be "tolerant" if it can lead to a marked state from the initial state upon the occurrence of faults. A fault-tolerant supervisor is then constructed based on all fault-tolerant event sequences in the system. The system under supervision is fault-tolerant if it is nonblocking. In this framework, the specification after a fault occurs does not change. Moreover, no alternative solution is provided if no tolerable event sequence can be found.

In [5], a learning and repair algorithm is proposed for dealing with a group of robots with modeling uncertainty. It is assumed that robots may switch offline due to failures or commands, that is, the number of units in a robot team may decrease. Therefore, the supervisor must be able to reconfigure itself and propose effective control in order to preserve the desired properties. When a unit switches offline, the learning method deletes the events which belong to the failed component alone from the previous supervisor, and then the repair algorithm is employed in order to restore the fractured automaton. This approach provides a practical design method for the

adaptive supervisory control problem. However, it can only handle the case in which some units switch offline and the solution seems to be suitable for the specific problem considered.

In [12], fault recovery problem is explored in discrete-event systems following linguistic approach. It is assumed that a diagnoser which can detect and isolate failures with bounded delays is available. They assume that the diagnosis system can be designed using any diagnosis technique, as long as the bounds for diagnosis delay are available. Thus, an abstract model of the diagnosis system is used. In this framework, the control problem and the diagnosis technique are almost separated, a feature which increases the flexibility of design. The specifications for the faulty mode are, in general, different from the specifications for the normal condition. A modular switching supervisory control mechanism is proposed. Both the nonblocking issue and admissibility property are studied. Moreover, the problem regarding multiple failures is also studied. In the case of multiple failures, the supervisory control problem may become computationally complex since all recovery supervisors must be in the feedback loop when the system starts even if no failure occurs.

In this thesis, we extend the results of [12] in two directions. First we propose a switching supervisory scheme in which the recovery supervisors are engaged only when a failure is diagnosed. This reduces the computational complexity (both time and space) of supervisor implementation during normal operation and the transient

mode when a failure has occurred but not diagnosed yet. Second, in this thesis, we study fault recovery problems in which recovery to normal operation may be possible. This issue was not addressed in [12].

In order to examine the above topic, similar to [12] we use the RW theory. However, instead of the linguistic approach (in which design specifications are given in terms of legal event sequences), we have decided to use the state-based approach (in which design specifications are given in terms of legal states). Both approaches are essentially equivalent. However, in our opinion, solutions based on state-based approach are more transparent and easier to interpret. This is particularly important in handling switching from one supervisor to another supervisor.

In the following, we briefly review the work on state-based supervisory control.

In [14],[9], the general control theory of Ramadge and Wonham is specialized to the vector discrete-event systems (VDES) in which the system state can be represented by a set of integer components. The concept of predicates and predicate transformers play an important role in the control of VDES. A static state-feedback controller, which implements control policy based on the current state, not based on the history (event sequence) of a plant, is proposed for VDES. Moreover, controllability, observability and modular control are investigated in the state-feedback control of VDES. Static state-feedback control can also be extended to dynamic state-feedback control

of VDES in which control specifications are given in terms of event sequences by adding memories. A systematic method on how to construct a memory is introduced. Further extension can be found in [8] and [18]. In [8], the design of "observer-based controllers" (supervisors) have been explored as a solution to the state-based supervisory control problems.

In [19], a model-based programming method is provided to track system state, diagnose fault, and perform reconfiguration. An earlier version of this work has been demonstrated in space on the NASA Deep Space One (DS-1) probe. Here the control system has a "model-based executive" (called Titan) which at any given time estimates the most likely state of the plant and generates a sequence of control actions to move the plant from the current state to the desired state (satisfying the design specifications). Another feature of this approach is a "Reactive Model-Based Programming Language" (RMPL) which allows the engineer to design and program the control system at a higher level, leaving tasks involving reasoning through system interactions to the language compiler and the executive (Titan).

Observer-based supervisors (OBS) used in DES problems are very similar to those used in modern control theory. In [6], OBS are used as solutions for supervisory control problems. Specifically, a standard procedure for supervisor design for problems of control under partial observation is proposed by modifying supervisors designed for

full observation. Using the proposed procedure, an optimal full-observation supervisor which generates the supremal controllable sublanguage according to RW theory can be converted to an observer-based supervisor which is at least as efficient (minimally restrictive) as the supervisor that generates the supremal controllable normal sublanguage. This procedure can be implemented on-line. Since no marking issue is considered, the modified supervisor is not guaranteed to be nonblocking.

In this thesis, we study the design of observer-based supervisors to solve fault recovery problems in discrete-event systems. An overview of the thesis is given in the next section.

1.2 Thesis Outline

In this thesis, we extend the work in [12] using a state-based approach to supervisory control.

In Chapter 2, we briefly review the automata theory and the RW supervisory control theory. The focus in this chapter will be on state-based supervisory control. However, the concepts of controllability, observability and normality used in linguistic approach are also discussed.

In Chapter 3, we present observer-based supervisory control. We assume that

an observer which can generate a state estimates based on the observed events is available, and the control action is based on the state estimate provided by the supervisor. The supervisor designed is thus called an Observer-Based Supervisor (OBS). A procedure for designing observer-based supervisors using normal languages is introduced. In the last section of this chapter, a merging scheme for multiple OBS supervisors, which will be used later in the supervisor design for fault recovery, is discussed. While OBS have been studied in literature in modern control and DES, the aforementioned design procedure for OBS using normal language and the merging operation are among the contributions of this thesis.

In Chapter 4, we formulate the fault recovery problem studied in this thesis. The plant is modeled as an automaton. Similar to [12], we assume that the diagnosis system can detect and isolate failures with a bounded delay, and we use an abstract model for the diagnosis system in the form of a finite-state automaton. The diagnoser may be constructed based on any technique, continuous-variable or discrete-event (e.g., [10],[16],[17],[22],[7],[13]) as long as the bounds for diagnosis are available. The benefit of separating the diagnosis and control problems is that the supervisor design problem becomes simplified.

The combined model of the plant and the diagnoser forms the system to be controlled. The system is considered to have three modes : normal, transient and recovery. In the normal mode, the system is working properly. Once a failure occurs in the

plant, the system is in the transient mode before the fault is diagnosed and isolated. Upon the failure detection, the system enters its recovery mode.

The specifications corresponding to each mode are given in terms of legal (safe) states, that is, the set of states for which a supervisor should enforce these specifications. In addition to these specifications, we would also like the supervised system to be nonblocking in both normal and recovery modes.

In Chapter 5, we present four design approaches to solve the supervisory control problem for fault recovery. We discuss the cases of failure accommodation (non-recovery to normal mode) and recovery to normal mode separately. The control problems including both cases are considered next. For fault recovery problem in each category, we provide two approaches to solve the control problem. The case in which the system has one failure mode is considered first, and then extension to multiple failures is discussed.

A modular switching scheme is adopted in the supervisory control for fault recovery problem. Using modular approach can simplify the design procedure, allow easy modification and upgrading of the supervisor when a component changes. It is also easy to synthesize. The supervisors designed are observer-based. Examples are provided to illustrate the proposed procedures.

Chapter 6 provides an illustrative example of the application of proposed methodology to a simplified manufacturing system. Finally, we will summarize our conclusions and discuss future work in Chapter 7.

Chapter 2

Background Review

In this chapter, we briefly review topics from the automata theory and the supervisory control theory of discrete-event systems which are relevant to our discussion in future chapters. The focus of our discussion of supervisory control will be on the "state-based approach".

2.1 Languages and Automata

Let Σ be an alphabet. For a language $L \subseteq \Sigma^*$, $\bar{L}$ denotes the *prefix-closure* (or simply closure) of L. L is *closed* if $L = \bar{L}$. For two languages $L, M \subseteq \Sigma^*$, L is called M-closed if $L = \bar{L} \cap M$.

The natural projection is an operation on two event sets where one set is a subset of the other, which erases all events in the larger event set that are not included in the

smaller set. Let $\Sigma_1 \subseteq \Sigma_2$. The natural projection, $P : \Sigma_2^* \rightarrow \Sigma_1^*$, is defined according to

$$P(\epsilon) = \epsilon$$

$$P(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_1; \\ \epsilon, & \text{if } \sigma \in \Sigma_2 - \Sigma_1. \end{cases}$$

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

Where ϵ is the empty string. The inverse function of P is a map $P^{-1} : \Sigma_1^* \rightarrow \Sigma_2^*$ defined as

$$P^{-1}(t) = \{s \in \Sigma_2^* \mid P(s) = t\}.$$

Consider a plant modeled as a deterministic finite-state automaton $\mathbf{G} = (Q, \Sigma, \delta, q_0, Q_m)$, where Q and Σ represent the state set and the event set, respectively. $\delta : Q \times \Sigma \rightarrow Q$ is the partial state transition function, and q_0 and Q_m denote the initial state and the set of marked states, respectively. For $q \in Q$, $\sigma \in \Sigma$, we write $\delta(q, \sigma)!$ if $\delta(q, \sigma)$ is defined.

$L(G)$ and $L_m(G)$ represent the *closed* and *marked* behavior of G, and are defined according to:

$$L(G) = \{s \in \Sigma^* \mid \exists\, q \in Q,\ q = \delta(q_0, s)\}$$

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

Namely, $L(G)$ contains the set of event sequences which can take G from the initial state q_0 to some state in Q. Note that $L(G)$ is closed (i.e., $\bar{L}(G) = L(G)$). $L_m(G)$

contains all the event sequences that can take G from the initial state q_0 to some marked state in Q_m. A plant G is said to be *nonblocking* if $\bar{L}_m(G) = L(G)$, that is, for any string $s \in L(G)$, there exists s' such that $ss' \in L_m(G)$.

Let $\mathcal{R}(G)$ be the set of states of G *reachable* from the initial state q_0. In other words,

$$\mathcal{R}(G) = \{q \in Q \mid \exists s \in L(G),\ q = \delta(q_0, s)\}.$$

The set of *coreachable states* of G is defined according to

$$\mathcal{CR}(G) = \{q \mid \exists s \in \Sigma^*,\ \delta(q, s) \in Q_m\}.$$

Thus G is nonblocking if and only if $\mathcal{R}(G) \subseteq \mathcal{CR}(G)$.

The following operations on automata are useful in the analysis of discrete-event systems [20],[2].

1. Product

   The product of two automata G_1 and G_2 with event sets Σ_1 and Σ_2 is an automaton that synchronizes on common events $(\Sigma_1 \cap \Sigma_2)$, that is, $\sigma \in \Sigma_1 \cap \Sigma_2$ occurs in the product if and only if it can occur in both G_1 and G_2. Specifically, let $G_1 = (Q_1, \Sigma_1, \delta_1, q_{0_1}, Q_{m_1})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, q_{0_2}, Q_{m_2})$. The product is an automaton $G_1 \times G_2 = (Q, \Sigma, \delta, q_0, Q_m)$ with $Q = Q_1 \times Q_2$, $Q_m = Q_{m_1} \times Q_{m_2}$, $q_0 = (q_{0_1}, q_{0_2})$, $\Sigma = \Sigma_1 \cap \Sigma_2$. The state transition function $\delta : Q \times \Sigma \rightarrow Q$ is

defined according to:

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

Let $G = \mathbf{meet}(G_1, G_2)$ denote the reachable part of $G_1 \times G_2$.

2. Synchronous Product (Parallel Composition)

In the synchronous product of $G_1 = (Q_1, \Sigma_1, \delta_1, q_{0_1}, Q_{m_1})$ and $G_2 = (Q_2, \Sigma_2, \delta_2, q_{0_2}, Q_{m_2})$, a transition with a common event $\sigma \in \Sigma_1 \cap \Sigma_2$ is defined if it is defined in both automata. Private events (events in $(\Sigma_1 - \Sigma_2)$ and $(\Sigma_2 - \Sigma_1)$) can be executed without synchronization. The parallel composition is denoted by $G = \mathbf{sync}(G_1, G_2)$. Thus G is the reachable part of the automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$ with $Q = Q_1 \times Q_2$, $Q_m = Q_{m_1} \times Q_{m_2}$, $q_0 = (q_{0_1}, q_{0_2})$, $\Sigma = \Sigma_1 \cup \Sigma_2$, and $\delta : Q \times \Sigma \to Q$ defined as:

$$\delta((q_1,q_2),\sigma) = \begin{cases} (\delta_1(q_1,\sigma),\delta_2(q_2,\sigma)), & \text{if } \delta_1(q_1,\sigma)! \text{ and } \delta_2(q_2,\sigma)!; \\ (\delta_1(q_1,\sigma),q_2), & \text{if } \sigma \in \Sigma_1 - \Sigma_2 \text{ and } \delta_1(q_1,\sigma)!; \\ (q_1,\delta_2(q_2,\sigma)), & \text{if } \sigma \in \Sigma_2 - \Sigma_1 \text{ and } \delta_2(q_2,\sigma)!; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

3. Trim

Trim is the reachable and coreachable part of an automaton. In other words, for $G = (Q, \Sigma, \delta, q_0, Q_m)$, we define $\mathbf{Trim}(G)=(Q_t, \Sigma, \delta_t, q_0, Q_{m_t})$ where $Q_t = \mathcal{CR}(\mathcal{R}(G))$, $Q_{m_t} = \mathcal{CR}(\mathcal{R}(G)) \cap Q_m$ and δ_t is the restriction of δ to $Q_t \times \Sigma$. Note that if $q_0 \notin \mathcal{CR}(\mathcal{R}(G))$, Q_t will be the empty automaton.

4. Project

Consider $G = (Q, \Sigma, \delta, q_0, Q_m)$. Let $\Sigma_o \subseteq \Sigma$. PG =**project**$(G, \Sigma - \Sigma_o)$ is the reachable deterministic finite-state automaton $PG = (Z, \Sigma_o, \xi, z_0, Z_m)$ with $Z \subseteq 2^Q$, $Z_m = \{z \in Z \mid z \cap Q_m \neq \emptyset\}$, $z_0 = \{q \mid \exists s \in (\Sigma - \Sigma_o)^* : \ q = \delta(q_0, s)\}$. Furthermore, $\xi : Z \times \Sigma_o \to Z$ is defined according to

$$z_{k+1} = \xi(z_k, \sigma_k) = \{q \mid \exists (s, s') \in (\Sigma - \Sigma_o)^*, q' \in z_k : \ q = \delta(q', s\sigma_k s')\}, \text{ for } k \geqslant 1.$$

Here the events in $\Sigma - \Sigma_o$ are treated as unobservable (or silent) and are removed from automaton model of the plant. Thus we have

$$L(PG) = PL(G),$$

$$L_m(PG) = PL_m(G),$$

where $P : \Sigma^* \to \Sigma_o^*$ is the natural projection. Note PG can also be regarded as observer with $z_k \subseteq Q$ as the estimate of the state of G after $\sigma_k \in \Sigma_o$ is observed.

5. Self-loop

Consider $G = (Q, \Sigma, \delta, q_0, Q_m)$. Let Σ' be a set of events with $\Sigma' \cap \Sigma = \emptyset$. The self-loop operation on G with event set Σ', denoted by G' =**selfloop**(G, Σ'), adds transitions $q \xrightarrow{\sigma} q$ for all $\sigma \in \Sigma'$ and $q \in Q$. Thus $L(G') = P^{-1}(L(G))$ and $L_m(G') = P^{-1}(L_m(G))$ where P is the natural projection $P : (\Sigma \cup \Sigma')^* \to \Sigma^*$.

2.2 Supervisory Control

Consider a plant $G = (Q, \Sigma, \delta, q_0, Q_m)$. It is assumed that the event set Σ can be partitioned into controllable and uncontrollable events, i.e., $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$. Controllable events can be disabled or enabled. Not all events may be observable, and thus the event set is also partitioned into $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o and Σ_{uo} denote the set of observable and unobservable events, respectively.

Let $E \subseteq Q$ denote the set of "legal" (safe) states of the plant G. In supervisory control theory, we want to design a supervisor S to ensure that the plant never leaves E, and the plant under supervision is nonblocking. A supervisor monitors the sequence of observable events generated by the plant and restricts the behavior of the plant to the "legal" states by disabling and enabling of controllable events.

The assumption that the specification is given in terms of "legal" (safe) states is not limiting. Problems involving "legal event sequences" can be transformed into equivalent problems involving "legal" states as specifications by adding suitable automata to capture the "history" of event sequences [9].

Let $P : \Sigma^* \rightarrow \Sigma_o^*$ be the projection map that removes the unobservable events from each event sequence $s \in \Sigma^*$. For $s = \sigma_1 \cdots \sigma_k \in \Sigma^*$ generated by the plant (under supervision), Ps is the sequence observed by the supervisor. The supervisor

can be defined as a map $S : \Sigma_o^* \rightarrow 2^{\Sigma_c}$ with $S(Ps)$ being the set of controllable events disabled by the supervisor. S interacts with G to form the closed-loop system. The traditional control loop in DES is shown in Fig.2.1. Note that S only disables controllable events. A supervisor that never disables uncontrollable events is called *admissible* (controllable).

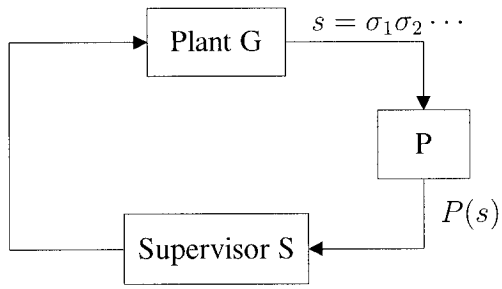


Figure 2.1: Traditional control loop

Let $L(S/G)$ be the language generated by the plant G under supervision of S (S/G), and $L_m(S/G)$ be the marked behavior. The closed behavior $L(S/G)$ is defined inductively as follows [20],[2]:

- $\epsilon \in L(S/G)$
- If $s \in L(S/G)$, $\sigma \notin S(Ps)$, and $s\sigma \in L(G)$, then $s\sigma \in L(S/G)$
- No other strings belong to $L(S/G)$.

We then have $L(S/G) \subseteq L(G)$, and according to the above definition, $L(S/G)$ is closed. The marked behavior of S/G is defined as

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

Thus $L_m(S/G)$ consists of sequences in the marked behavior that can be generated in system under supervision. Note that in S/G, marking is still determined by the plant G.

Abusing the notation, we let $\mathcal{R}(S/G)$ and $\mathcal{CR}(S/G)$ be the reachable and coreachable states of the plant under supervision:

$$\mathcal{R}(S/G) = \{q \in Q \mid \exists s \in L(S/G) : \ q = \delta(q_0, s)\}$$

$$\mathcal{CR}(S/G) = \{q \in Q \mid \exists s, s' : s \in L(S/G), ss' \in L(S/G), q = \delta(q_0, s) \text{ and } \delta(q, s') \in Q_m\}.$$

In the supervisory control problem, the objective is to find an admissible supervisor S such that

(i) $\mathcal{R}(S/G) \subseteq E$;

(ii) $\mathcal{R}(S/G) \subseteq CR(S/G)$.

The former condition confines G stay inside the set of desirable states, and the latter ensures that S/G is nonblocking.

Note that we have assumed that the plant has a single initial state. If a plant has multiple initial states (and that may be the case in the problems studied in this thesis), then we can convert the problem to an equivalent problem with single initial state as explained in the following. Suppose the plant is $G = (Q, \Sigma, \delta, Q_0, Q_m)$ where $Q_0 = \{q_{0_1}, \cdots, q_{0_p}\}$ is the set of initial states. Now consider $G' = (Q \cup \{q_0\}, \Sigma \cup \{\sigma_1, \cdots, \sigma_p\}, \delta', q_0, Q_m)$ which is obtained by adding a fictitious state q_0 to Q and

considering q_0 as the initial state. We also add unobservable, uncontrollable transitions $q_0 \xrightarrow{\sigma_i} q_{0_i}$. The rest of the transitions in G' are identical to those of G. We also replace the design specification E with $E \cup \{q_0\}$. Now any solution for supervisory control of G' is a solution for supervision of G and vice versa.

In the next chapter, we discuss a class of solutions to the above problem known as observer-based supervisors.

A supervisor S is *maximally permissive* (*optimal*) if it only disables an event when it has to. Note that in the case of full observation ($\Sigma = \Sigma_o$), we can always construct an optimal supervisor [9]. However, an optimal supervisor may not exist for the case of control under partial observation [11],[4].

The supervisory control problem discussed earlier is referred to as a "state-based" supervisory control problem since the design specification is given in terms of safe (legal) states (E). An alternative, equivalent formulation of the supervisory control problem follows a linguistic approach in which the design specification is given in terms of a legal language (i.e., legal event sequences).

The solutions to the linguistic supervisory control problem can be characterized in terms of controllable, observable and $L_m(G)$-closed languages.

Definition 2.2.1. ([15]) A language K is called *controllable* with respect to the plant G if

$$\bar{K}\Sigma_{uc} \cap L(G) \subseteq \bar{K}$$

Definition 2.2.2. ([11],[4]) A language $K \subseteq L(G)$ is said to be $(L(G), P)$-*observable* if

$$\forall (s, s') \in \Sigma^*, \sigma \in \Sigma;\ [s' \in \bar{K}\ \&\ s'\sigma \in L(G)\ \&\ s\sigma \in \bar{K}\ \&\ Ps = Ps'] \implies s'\sigma \in \bar{K}.$$

Let $E \subseteq L_m(G)$ be the legal language (design specification). The supervisory control under partial observation is to find an admissible supervisor such that

$$L_m(S/G) \subseteq E \qquad (S/G \text{ satisfies } E)$$

$$\bar{L}_m(S/G) = L(S/G) \qquad (\text{nonblocking condition}).$$

It can be shown that [11],[4] for $K \subseteq E$, there exists an admissible supervisor S such that

(i) $L_m(S/G) = K$

(ii) $\bar{L}_m(S/G) = L(S/G)$

if and only if

K is controllable, observable and $L_m(G)$-closed.

Thus the class of controllable, observable and $L_m(G)$-closed languages characterizes the set of solutions to the supervisory control problem.

Since the property of "observability" is not closed under union, instead of observable languages, one may use a subclass of observable languages, called *normal*

languages, that is more mathematically well-behaved (i.e., closed under the union operation).

Definition 2.2.3. ([11],[4]) A language $K \subseteq L(G)$ is $(L(G), P)$-*normal* if

$$\bar{K} = L(G) \cap P^{-1}P\bar{K}.$$

A normal language is observable but not vice versa. However, when all controllable events are observable ($\Sigma_c \subseteq \Sigma_o$), a controllable, observable language will be normal as well.

Finally, if S is a supervisor designed based on controllable, normal languages (i.e., $L(S/G)$ is normal and controllable), then S never disables controllable events that are unobservable [20].

Chapter 3

Observer-Based Supervisors

In this chapter, we are going to discuss supervisory control policies that are based on state estimates (of the plant) provided by an observer. A supervisor designed through an observer is thus called an observer-based supervisor (OBS). In Section 3.1, we discuss the structure of observer- based supervisors. A procedure for designing observer-based supervisor using normal languages is introduced in Section 3.2. Sections 3.3 and 3.4 discuss operations conjunction and merging of OBS supervisors, which will be used later in supervisor design for fault recovery.

Observer-based supervisors have been used in modern control theory. They have also been studied in the supervisory control of discrete-event systems (e.g. [6], [8]). Thus sections 3.1 and 3.3 are essentially background material. The material in sections 3.2 and 3.4, however, are among the contributions of this thesis.

3.1 Introduction to Observer-Based Supervisors

In a control problem with full event observation in which the objective is to keep the plant states out of forbidden (unsafe) states while ensuring the plant under supervision is nonblocking, an optimal (maximally permissive) control policy regarding the enabling and disablement of controllable events can be found that depends only on the current plant state, and not on the past event sequence generated by the plant [9].

Motivated by the above fact, in a control problem similar to the above-mentioned except with partial observation, we may seek to find supervisory control policies that rely on the current *state estimate* of the plant. We assume that the required state estimate is obtained using an observer. Thus the resulting supervisors will be "*observer-based supervisors*" (OBS). In this thesis, we focus our study on the design of observer-based supervisors. This is not very limiting. We can show that if S' is an admissible nonblocking supervisor that solves the control problem of avoiding forbidden states, and S' is designed based on normal languages (i.e., $L_m(S'/G)$ is normal), then we can find an admissible nonblocking OBS S such that $L_m(S'/G) \subseteq L_m(S/G)$ and G under supervision of S never enters forbidden states. The main advantage of using OBS supervisors is the simplicity of their control policy which is particularly useful in switching control schemes that we would like to develop in this thesis for fault recovery problems.

Given a plant G, an observer-based supervisor S can be implemented by a generator $S = (Z, \Sigma_o, \xi, z_0, f)$. The 4-tuple (Z, Σ_o, ξ, z_0) is an observer. $Z \subseteq 2^Q$ is the state set of the observer (supervisor) corresponding to the set of state estimates, z_0 is the initial state of the observer (supervisor), and $\xi : Z \times \Sigma_o \rightarrow Z$ represents the partial state transition function. $f : Z \rightarrow 2^{\Sigma_o}$ is an output map such that for each state estimate $z \in Z$, $f(z)$ is the set of observable events that are disabled by the supervisor at the current state z. Note that S is *admissible* if and only if $f(z) \subseteq \Sigma_c$ for $z \in Z$. We assume that all states of S are marked and therefore, do not include the marked states $Z_m = Z$ in the description of the supervisor S. The proposed supervisor S regulates the plant behavior based only on the current state estimate. Throughout this thesis, we make the following assumption:

> **Assumption**: *Only controllable events that are also observable may be disabled by the supervisor S, and thus unobservable controllable events are treated as uncontrollable.*

Note that this assumption applies to all supervisors that are designed based on controllable normal languages (i.e. cases in which the marked behavior of the plant under supervision is controllable and normal). For the purpose of our work, we do not make any other assumptions on the OBS designed. As mentioned earlier, an approach for designing OBS based on normal languages will be discussed in the next section.

z_0 is the initial state estimate with which the observer (and the supervisor) is

initialized. If the plant G and supervisor S are initialized together then

$$z_0 = \{q \mid \exists s \in \Sigma_{uo}^*,\ q = \delta(q_0, s)\}. \tag{1}$$

Note that in the switching supervisory scheme proposed in this thesis, a supervisor module may be initialized and put in the feedback loop while the plant is already in operation in which case z_0 may be different from that given in (1). We will discuss the issue of supervisor initialization later in Chapter 5 when we present our switching modular scheme. The state transition function ξ is defined as follows. For $z_k \in Z$, $\sigma_k \in \Sigma_o$, if $\xi(z_k, \sigma_k)$ is defined and $z_{k+1} = \xi(z_k, \sigma_k)$, then z_{k+1} can be calculated according to:

$$z_{k+1} = \{q \mid \exists\, q' \in z_k \,\&\, \exists\, s, s' \in \Sigma_{uo}^* :\ q = \delta(q', s\sigma_k s')\},\ k \geqslant 1. \tag{2}$$

It should be noted that by assumption, unobservable controllable events are not disabled by the supervisor, and therefore, in the above update law (2), the event sequences s and s' may include unobservable controllable events.

Of course a necessary condition for $\xi(z_k, \sigma_k)$ to be defined is that the set

$$\Phi(z_k, \sigma_k) = \{q \mid \exists\, q' \in z_k \,\&\, \exists\, s, s' \in \Sigma_{uo}^* :\ q = \delta(q', s\sigma_k s')\} \tag{3}$$

is nonempty. If $\Phi(z_k, \sigma_k)$ is nonempty and $\xi(z_k, \sigma_k)$ is not defined, then we consider event σ_k to be disabled by the supervisor. Therefore,

$$f(z_k) = \{\sigma \in \Sigma_o \mid \Phi(z_k, \sigma) \neq \emptyset \,\&\, \text{not}\xi(z_k, \sigma)!\}.$$

The OBS supervisor is admissible if and only if $f(z_k) \subseteq \Sigma_c$. The plant under supervision S/G can be represented by meet(G, selfloop(S, Σ_{uo})).

Note that the function f is defined in terms of the transition functions ξ and δ, and therefore it does not provide extra information to what is already available from the plant's model and the 4-tuple observer (Z, Σ, ξ, z_0). Thus, we may remove f from the 5-tuple S describing the supervisor. However, in some cases (for example, the study of the conjunction of supervisors), it is convenient to have a function to provide the list of disabled events and thus, we will keep f as part of the description of the supervisor and use the 5-tuple $S = (Z, \Sigma_o, \xi, z_0, f)$ to characterize OBS supervisors.

3.2 Procedure For Designing Observer-Based Supervisors Using Normal Languages

In this section, we introduce a procedure for designing OBS using normal languages. Let $G = (Q, \Sigma, \delta, q_0, Q_m)$ be the plant and $E \subseteq Q$ be the set of legal (safe) states. We would like to design an OBS S such that

(i) $\mathcal{R}(S/G) \subseteq E$;

(ii) $\mathcal{R}(S/G) \subseteq \mathcal{CR}(S/G)$.

We show that if an admissible nonblocking supervisor S' (not necessarily observer-based) exists such that $\mathcal{R}(S'/G) \subseteq E$ and $\mathcal{R}(S'/G) \subseteq \mathcal{CR}(S'/G)$, and $L_m(S'/G)$ is

controllable and normal (i.e., S' solves the control problem and S' can be characterized in terms of a controllable, normal language), then we can construct an admissible nonblocking OBS S such that

$$(\text{i})\mathcal{R}(S'/G) \subseteq \mathcal{R}(S/G) \subseteq E$$

$$(\text{ii})\mathcal{R}(S/G) \subseteq \mathcal{CR}(S/G).$$

Let G_E denote the subgenerator of G corresponding to states in $E(\subseteq Q)$. Furthermore, let TG_E =Trim(G_E). Thus, $L_m(TG_E) = L_m(G_E) = L(G_E) \cap L_m(G)$, which implies that $L_m(TG_E)$ is $L_m(G)$-closed ($L_m(TG_E) = \bar{L}_m(TG_E) \cap L_m(G)$). $L_m(TG_E)$ is the legal language for the supervisory control problem. Now let $S' = (X', \Sigma, \eta', x'_0, X')$ be a nonblocking admissible supervisor such that $L_m(S'/G)$ is an $L_m(G)$-closed, controllable, normal sublanguage of $L_m(TG_E)$. At any state $x' \in X'$, let $\Gamma_{S'}(x')$ denote the set of events enabled by supervisor S' at x':

$$\Gamma_{S'}(x') = \{\sigma \in \Sigma \mid \eta'(x', \sigma)!\}.$$

Since $L_m(S'/G)$ is controllable and normal, unobservable controllable events are not disabled by S'.

Supervisor S' enables and disables controllable events based on the observed event sequences. In other words, for strings $(s, s') \in \Sigma^*$, if $Ps = Ps'$, then the supervisory action must be the same, that is,

$$\Gamma_{S'}(\eta'(x'_0, s)) = \Gamma_{S'}(\eta'(x'_0, s')). \tag{4}$$

Here $P : \Sigma^* \rightarrow \Sigma_o^*$ is the natural projection.

Given the above discussion and Eq.(4), if we define PS' =Project(S', Σ_{uo}) = $(Y, \Sigma_o, \eta'_P, y_0, Y)$, then we can see that the two control loops in Fig.3.1 will have the same marked and closed behaviors.

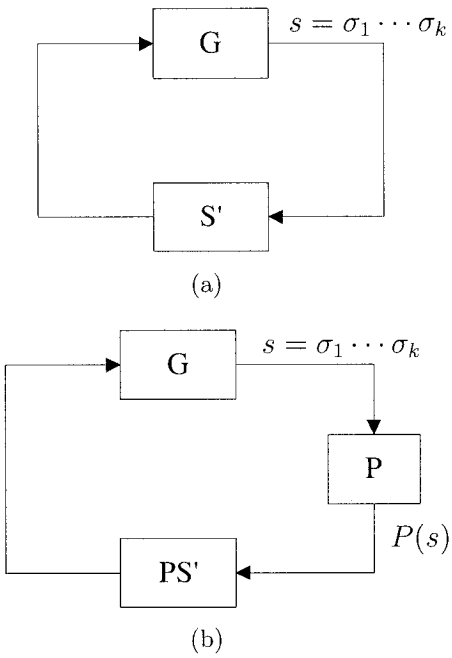


Figure 3.1: Control loop

Furthermore, for $t \in \Sigma_o^*$, we have $\Gamma_{PS'}(\eta'_P(y_0, t)) = \Gamma_{S'}(\eta'(x'_0, s)) \cap \Sigma_o$ for any $s \in P^{-1}t \cap L(S')$, where $P^{-1} : \Sigma_o^* \rightarrow \Sigma^*$ is the inverse projection map. Now let PG =Project(G, Σ_{uo}) = $(Z', \Sigma_o, \xi', z'_0, Z'_m)$ be the projection of G. Thus $Z' \subseteq 2^Q$, $z'_0 = \{q \mid \exists s \in \Sigma_{uo}^*, \delta(q_0, s) = q\}$, and $L_m(PG) = PL_m(G)$, $L(PG) = PL(G)$.

In Fig.3.1(b), PS' works on the observable event sequence generated by the plant. Now if PS' is replaced with S'' =meet(PS', PG), then the closed-loop behavior will remain the same.

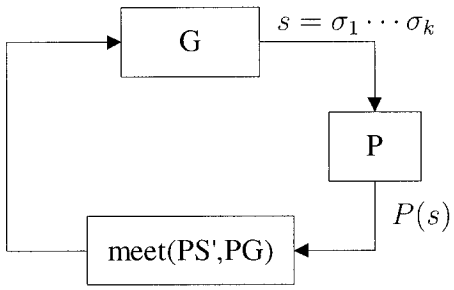


Figure 3.2: Control loop

For any state (y, z) $(y \in Y,\ z \in Z')$ of S'', $\Gamma_{S''}((y, z))$ is the set of controllable events enabled by the supervisor S''. Note that in general, for two states (y_1, z), (y_2, z) of S'' corresponding to the same state estimate z (provided by PG), the control decision $\Gamma_{S''}((y_1, z))$ and $\Gamma_{S''}((y_2, z))$ are not necessarily the same. However, we intuitively expect that all controllable events enabled at (y_1, z) can be allowed to happen at (y_2, z) too (and vice versa) since for the control problem considered here, we expect that the control decision can depend only on state estimate (which is identical for both (y_1, z) and (y_2, z)). This motivates the construction of the OBS: $S = (Z, \Sigma_o, \xi, z_0, f_S)$, where Z is the set of state estimates generated by S'', i.e.,

$$Z = \{z \mid \exists\, (y, z) \in \mathcal{R}(S'')\}.$$

Also,

$$z_0 = z_0' = \{q \mid \exists s \in \Sigma_{uo}^* :\ q = \delta(q_0, s)\}.$$

The transitions defined at $z \in Z$ are given by

$$\Gamma_S(z) = \cup\{\Gamma_{S''}((y,z)) \mid (y,z) \in \mathcal{R}(S'')\}.$$

In other words, $z' = \xi(z,\sigma)$ is defined if and only if for some $y, y' \in Y$, the transition $(y,z) \xrightarrow{\sigma} (y',z')$ is defined in S''. Clearly, at any state estimate z, a controllable event is disabled if it is disabled at all $(y,z) \in \mathcal{R}(S'')$. Obviously, $\mathcal{R}(S'/G) \subseteq \mathcal{R}(S/G)$ and $\mathcal{CR}(S'/G) \subseteq \mathcal{CR}(S/G)$.

The state estimate function is of course defined according to

$$z_{k+1} = \xi(z_k, \sigma_k) = \{q \mid \exists q' \in z_k \;\&\; \exists s, s' \in \Sigma_{uo}^* :\; q = \delta(q', s\sigma_k s')\}, \text{ for } k \geqslant 1,$$

where $z_k, z_{k+1} \in Z$ and $\sigma_k \in \Sigma_o$.

Proposition 3.2.1. *With S as defined above, we have*

$$\mathcal{R}(S/G) \subseteq E \text{ and } \mathcal{R}(S/G) \subseteq \mathcal{CR}(S/G).$$

Proof. In order to prove the above proposition, we show that for all state estimates z_k generated by S, we have $z_k \subseteq E$ and all states in z_k are coreachable. Since z_k is the set of states where the state of G can possibly be, we can conclude $\mathcal{R}(S/G) \subseteq E$ and $\mathcal{R}(S/G) \subseteq \mathcal{CR}(S/G)$.

We show the above mentioned statement by induction on k. For $k = 0$, obviously, z_0 satisfies $z_0 \subseteq E$ and all states in z_0 are coreachable. Otherwise, the supervisory

control problem would not have a solution based on controllable normal languages. Now if $z_{k+1} = \xi(z_k, \sigma_k)$, then $\sigma_k \in \Gamma_S(z_k)$, and thus $\sigma_k \in \Gamma_{S''}((y, z_k))$ for some y with $(y, z_k) \in \mathcal{R}(S'')$. Also, in S'', we have a transition $(y, z_k) \xrightarrow{\sigma_k} (y', z_{k+1})$ for some y' with $(y', z_{k+1}) \in \mathcal{R}(S'')$. Since S' (thus S'') solves the supervisory control problem, then $z_{k+1} \subseteq E$ and z_{k+1} is coreachable in S''/G. Coreachability of z_{k+1} in S''/G implies that for any $q \in z_k$, there exists a sequence "s" in S''/G and $q' \in Q_m$ such that $q \xrightarrow{s} q'$. Since $L(S''/G) \subseteq L(S/G)$, the sequence "$s$" causing transition $q \xrightarrow{s} q'$ is also possible in S/G. Therefore, all states in z_{k+1} are coreachable in S/G.

□

Example 3.1:

We illustrate the above design procedure using a simple example. The plant is shown in Fig.3.3 with $\Sigma_c = \{\alpha, \theta, \rho\}$, $\Sigma_o = \{\alpha, \theta, \rho, \beta, \tau\}$. In the figure, transitions corresponding to controllable events are shown with $\twoheadrightarrow$. Also, unobservable transitions are shown with dashed lines. State 5 is the forbidden state in this example, i.e., $E = Q - \{5\}$. The projected plant PG and TG_E (representing the legal language) are displayed in Fig.3.4 and Fig.3.5, respectively. Observe that $L_m(TG_E)$ is not normal. To show this, we note that string $\alpha\alpha\gamma\beta\alpha \in \bar{L}_m(TG_E)$ and $\alpha\alpha\beta\gamma\alpha \notin \bar{L}_m(TG_E)$, and $P(\alpha\alpha\gamma\beta\alpha) = P(\alpha\alpha\beta\gamma\alpha) = \alpha\alpha\beta\alpha$, but $\alpha\alpha\beta\gamma\alpha \in L(G)$; therefore, $L_m(TG_E)$ is not normal.

Now consider supervisor S' shown in Fig.3.6(a). The plant under supervision (formed by computing meet(G, S')) is depicted in Fig.3.7. We can verify that S' is admissible, S'/G is nonblocking, $L_m(S'/G)$ is normal language , and $\mathcal{R}(S'/G) \subseteq E$.

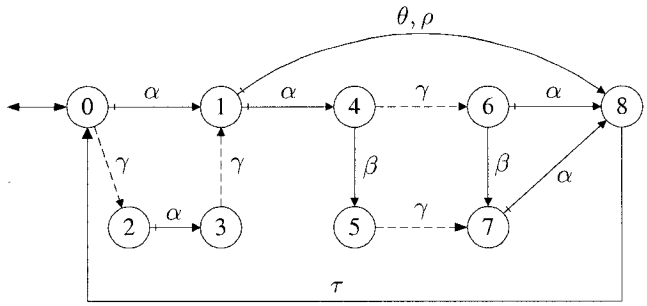


Figure 3.3: Example 3.1: G

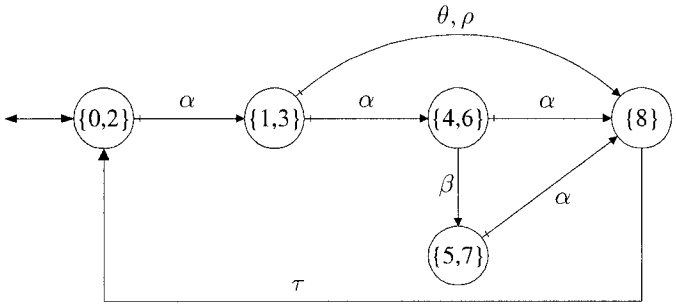


Figure 3.4: Example 3.1: PG

Next, we compute the product of PS' and PG (i.e., meet(PS',PG)). The result is shown in Fig.3.8. Finally, the OBS S shown in Fig.3.9 is computed. The plant under supervision S/G which can be obtained from meet$(G, \text{selfloop}(S, \Sigma_{uo}))$ is depicted in Fig.3.10.

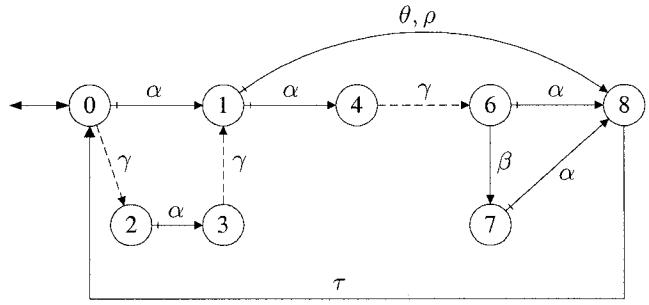
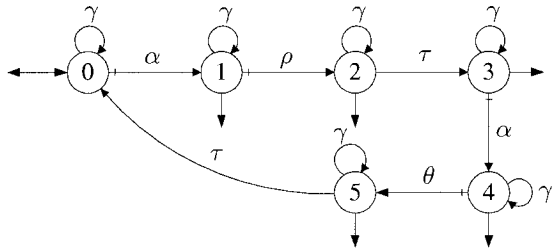
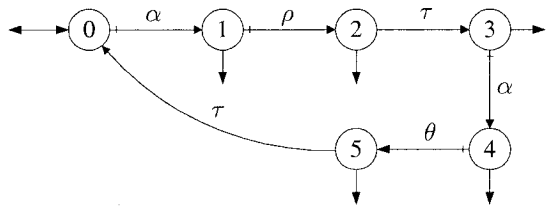


Figure 3.5: Example 3.1: TG_E



(a) S'



(b) PS'

Figure 3.6: Example 3.1: supervisor S' and its projection PS'

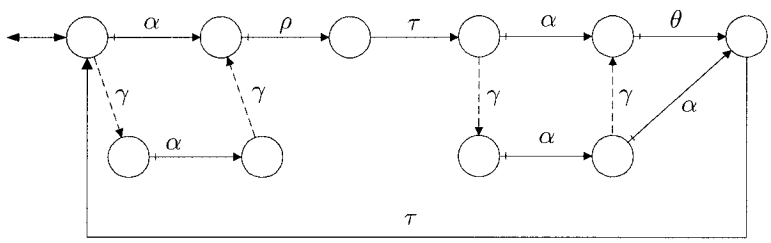


Figure 3.7: Example 3.1: S'/G

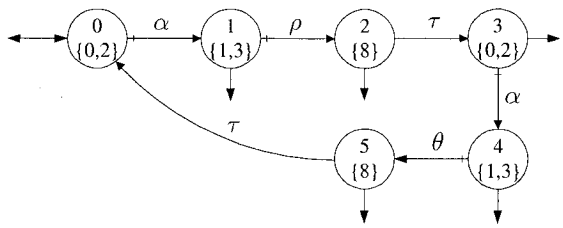


Figure 3.8: Example 3.1: meet(PS',PG)

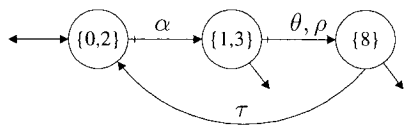


Figure 3.9: Example 3.1: OBS S

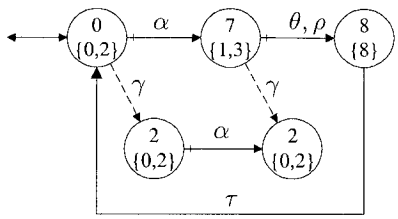


Figure 3.10: Example 3.1: S/G

3.3 Conjunction of Observer-Based Supervisors

In this thesis, we propose a modular switching supervisory scheme for fault recovery. The conjunction of supervisor modules are defined as in [21],[20],[2]. In this section, we discuss the mathematical representation of the conjunction of observer-based supervisors.

Suppose $S_1 = (Z_1, \Sigma_o, \xi_1, z_0, f_1)$ and $S_2 = (Z_2, \Sigma_o, \xi_2, z_0, f_2)$ are two supervisors designed for two partial specifications E_1 and E_2 of a plant G, and thus $\mathcal{R}(S_1/G) \subseteq E_1$ and $\mathcal{R}(S_2/G) \subseteq E_2$. Fig.3.11 shows the *conjunction* of the two supervisors, denoted by $S = S_1 \wedge S_2$, in the feedback loop. The supervisors monitor the sequence of observable events generated by the plant G and each computes a state estimate ($z^{(1)}$ and $z^{(2)}$ respectively). We define the set of controllable events disabled by S to be $f_1(z^{(1)}) \cup f_2(z^{(2)})$. In other words, S disables the events that are disabled by either S_1 or S_2. Since S_1 and S_2 are designed for the same plant model and have the same initial state z_0, for a common observed sequence $\sigma_1\sigma_2\cdots\sigma_k \in \Sigma_o^*$, we have $z_k^{(1)} = z_k^{(2)}$ (Note that the observer update law (2) does not depend on the map f). The conjunction of more than two supervisors is defined similarly.

Note that we can think of the conjunction $S = S_1 \wedge S_2$ as an OBS $S = (Z, \Sigma_o, \xi,$

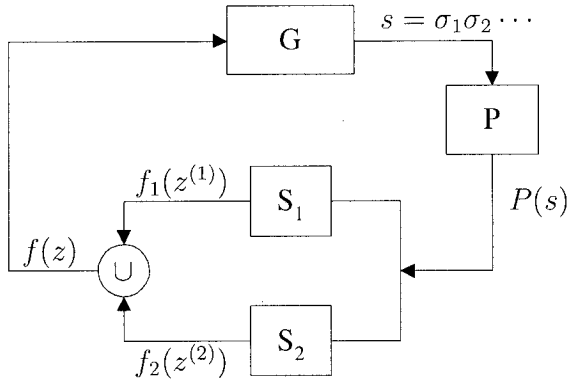


Figure 3.11: Conjunction of two supervisors

z_0, f) with $Z \subseteq Z_1 \cap Z_2$, ξ defined according to

$$\xi(z,\sigma) = \begin{cases} \xi_1(z,\sigma) = \xi_2(z,\sigma), & \text{if } \xi_1(z,\sigma)! \text{ and } \xi_2(z,\sigma)!; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

and $f : Z \to 2^{\Sigma_o}$ with $f(z) = f_1(z) \cup f_2(z)$.

In the fault recovery problem considered in this thesis, in some cases, the plant models used for designing supervisor modules are different for each supervisor module. Therefore, for the same observable event sequence, the state estimates obtained by the supervisor modules may be different, i.e, $z_k^{(1)} \neq z_k^{(2)}$. In these cases, some adjustments to the above definition are required. We will elaborate on this issue in Chapter 5.

3.4 Merging Observer-Based Supervisors

In Chapter 5, we will propose a modular switching supervisory control scheme for recovery from faults. In this approach, various modules in the form of observer-based supervisors for meeting specific design objectives are designed and put into

the feedback loop. For example, one OBS module may be designed to ensure safety requirements during recovery from a particular fault. In this case, the initial condition (z_0) with which the OBS is initialized may vary and in general, will depend on the time the fault occurs and the time the fault is diagnosed. Therefore, we have to design several versions of an OBS module for the same plant and design specification. Obviously, the number of the various versions of the OBS (corresponding to different initial conditions) could be large. In order to efficiently store the OBS in computer memory, we can *merge* the various versions into a single supervisor with multiple initial conditions. For brevity, we discuss the merging procedure for two supervisors. Extension to larger numbers of supervisors is straightforward.

Suppose $S_1 = (Z_1, \Sigma_o, \xi_1, z_{1,0}, f_1)$, $S_2 = (Z_2, \Sigma_o, \xi_2, z_{2,0}, f_2)$ are two admissible nonblocking OBS supervisors designed for a plant G to enforce a specification E. The merger of S_1 and S_2, which we denote by S =merge(S_1, S_2), is a 5-tuple $S = (Z, \Sigma_o, \xi, z_0, f)$. Here $Z = Z_1 \dot{\cup} Z_2$, and $z_0 = \{z_{1,0}, z_{2,0}\}$. The transition function $\xi : Z \times \Sigma_o \rightarrow Z$ is defined as follows:

$$z_{k+1} = \xi(z_k, \sigma_k) = \begin{cases} \xi_i(z_k, \sigma_k), & \text{if } \xi_i(z_k, \sigma_k) \text{ is defined either for } i = 1 \text{ or for } i = 2; \\ \text{not defined}, & \text{otherwise.} \end{cases}$$

Note that the above definition is well defined, since if both $\xi_1(z_k, \sigma_k)$ and $\xi_2(z_k, \sigma_k)$ are defined then we must have $\xi_1(z_k, \sigma_k) = \xi_2(z_k, \sigma_k)$. As a result of the merger, any

transition that is defined in one OBS, say S_1, and not defined in S_2, will be defined in the merger. In other words, if $z_1 \xrightarrow{\sigma} z_2$ is defined in S_1 not in S_2, in the merged supervisor S, $z_1 \xrightarrow{\sigma} z_2$ is allowed. Thus we have

$$f(z) = \begin{cases} f_1(z), & \text{if } z \in Z_1 - Z_2; \\ f_2(z), & \text{if } z \in Z_2 - Z_1; \\ f_1(z) \cap f_2(z), & \text{if } z \in Z_1 \cap Z_2; \end{cases}$$

The idea behind this is that since both S_1 and S_2 are observer-based, if at state z_1 of S_1, event σ is enabled and thus acceptable, it must be acceptable (not violating the specifications) even if the supervisor S was initialized with $z_{2,0}$. Let $S(z_{i,0})$ denote S initialized with $z_{i,0}$. Then we have the following proposition.

Proposition 3.4.1. *$S(z_{i,0})$ ($i = 1, 2$) is a nonblocking supervisor and*

$$\mathcal{R}(S_i/G) \subseteq \mathcal{R}(S(z_{i,0})/G) \subseteq E.$$

Proof. In order to prove the above proposition, we show that for all state estimates z_k generated by S_i, we have $z_k \in \mathcal{R}(S(z_{i,0}))$; and for all states $z_{i,k} \in \mathcal{R}(S(z_{i,0}))$, we have $z_{i,k} \subseteq E$. Since z_k and $z_{i,k}$ are the set of states where the state of G can possibly be, we can conclude $\mathcal{R}(S_i/G) \subseteq \mathcal{R}(S(z_{i,0})/G) \subseteq E$.

The first set inclusion, i.e., $z_k \in \mathcal{R}(S(z_{i,0}))$, is straightforward by the definition of merger and definition of the transition function ξ. If an event is not disabled at a state of S_i, it will not be disabled by $S(z_{i,0})$ either.

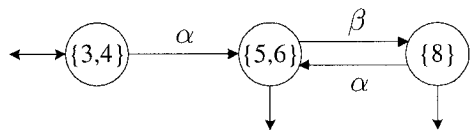
We show the second set inclusion by induction on k. For $i = 1$ and $k = 0$, obviously, $z_{1,0}$ satisfies $z_{1,0} \subseteq E$. Otherwise, the supervisory control problem would

not have a solution. Now if $z_{1,k+1} = \xi(z_{1,k}, \sigma_k)$, then $\sigma_k \notin f_1(z_{1,k}) \cap f_2(z_{1,k})$, and not disabled by both S_1 and S_2, thus $z_{1,k+1} \in \mathcal{R}(S_1)$ or $z_{1,k+1} \in \mathcal{R}(S_2)$. Since S_1 and S_2 solve the supervisory control problem, then $z_{1,k+1} \subseteq E$. For $i = 2$, the proof is similar.

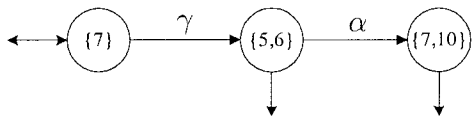
□

Example 3.2:

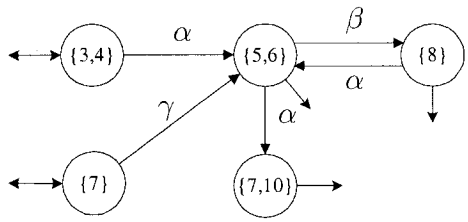
Fig.3.12(a),3.12(b) shows two OBS supervisors S_1 and S_2. S =merge(S_1, S_2) is depicted in Fig.3.12(c). Note that transition $\{5,6\} \xrightarrow{\alpha} \{7,10\}$ is not defined in S_1 whereas in S, if S is initialized with $z_{1,0} = \{3,4\}$, the transition $\{5,6\} \xrightarrow{\alpha} \{7,10\}$ is defined and possible.



(a) S_1



(b) S_2



(c) S =merge(S_1, S_2)

Figure 3.12: Example 3.2: merging two OBS supervisors

Chapter 4

Problem Formulation

In this chapter, we formulate the fault recovery problem examined in this thesis. The models of the plant and the diagnosis system (assumed available) will be discussed in Sections 4.2 and 4.3. The combined model of the plant and the diagnoser is formed in Section 4.4 as the system to be controlled. We classify failures into two categories based on whether recovery to normal mode is possible or not, and discuss the control objectives (design specifications) in Section 4.5. An example of modeling is provided in Section 4.6.

4.1 Introduction

In control systems, component failures could occur unexpectedly and reduce the supervisor's capability to observe the system's behavior and perform effective control action. Therefore, a reliable supervisor should be able to perform fault recovery if a

failure occurs in the system.

We assume that the plant G can be modeled as a finite-state automaton. The plant model includes both the normal and the faulty behaviors. The event set of the plant contains normal operation events, failure or fault events and possibly recovery events. The fault events are assumed to be uncontrollable and unobservable. In order to detect and isolate a failure, a diagnoser D with bounded delays is assumed to be available. The diagnosis system could be constructed based on any diagnosis technique as long as bounds for diagnosis are available. The system to be controlled is the combined model of the plant and the diagnosis system. The modified control loop is depicted in Fig.4.1. Here S is the supervisor.

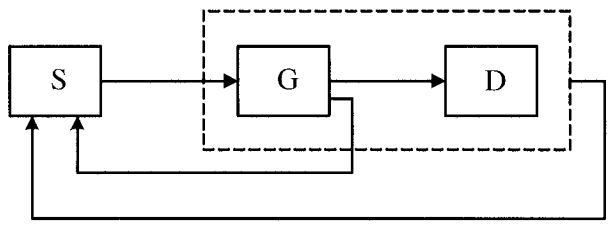


Figure 4.1: Modified control loop (G: plant, D: diagnosis system, S: supervisor)

The system therefore operates in three modes: normal, transient and recovery. Initially, the system operates in the normal condition. Once a fault or failure occurs, the system enters its transient mode through unobservable transitions. Upon failure detection, the system enters its recovery mode. The system's state transition graph is shown below.

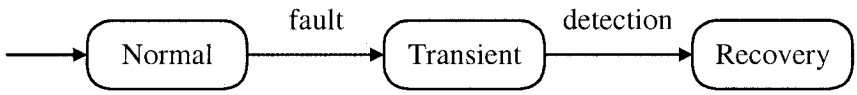


Figure 4.2: State transition diagram of the system to be controlled

The set of specifications are given in terms of the safe (legal) states in each mode. For example, the specifications for temperature control in a large room could be: (a) Normal: the states in which the temperature is between 20 and 24; (b) Transient: the temperature must be between 17 and 27; (c) Recovery: the temperature must be between 15 and 30. In general, the transient and recovery specifications are less restrictive than the normal specification.

4.2 Plant Model

We assume that the plant can be modeled as a finite-state automaton: $G = (Q, \Sigma_G, \delta_G, q_0, Q_m)$. Both the normal and faulty behaviors are included in this model. The event set Σ_G can be partitioned into controllable and uncontrollable events, i.e. $\Sigma_G = \Sigma_{G,c} \dot{\cup} \Sigma_{G,uc}$. The event set Σ_G can also be expressed as the disjoint union of three subsets, $\Sigma_G = \Sigma_P \dot{\cup} \Sigma_f \dot{\cup} \Sigma_r$, where $\Sigma_f = \{f_1, \cdots, f_p\}$ $(p \geqslant 1)$ represents the set of failure events which are assumed to be uncontrollable and unobservable in our framework, Σ_r denotes the set of recovery events that can lead the plant back to one of its normal states, and Σ_P includes the rest of plant events. If recovery to normal

operation is not possible, then $\Sigma_r = \emptyset$.

We assume that the failures are *permanent.* Therefore, if a failure event f_i $(i = 1, \cdots, p)$ occurs, the plant will enter a faulty mode F_i and will remain in this mode unless a recovery event occurs. Failures can be divided into two categories in terms of system's behavior during fault recovery. One category represents failures from which recovery to normal mode is not possible. Another kind of failure is the one for which recovery to normal operation is possible.

The set of states can be divided into two blocks: states that correspond to normal mode Q_N, and states that correspond to faulty mode Q_F. As mentioned, the plant has p failure modes $F_1, \cdots, F_p$. We assume that no more than one failure mode occurs at any time in the plant. This is referred to as "*single failure scenario*". Accordingly, the set of states Q_F can further be partitioned into p blocks: $Q_{F_1}, \cdots, Q_{F_p}$. An abstraction of the plant model with $p = 2$ failure modes is shown in Fig.4.3. Note that in this example, recovery to normal mode is possible in F_1 mode, but not in F_2 mode.

4.3 Diagnoser

We assume that a diagnosis system is available to detect and isolate failures and to notify the supervisor. Furthermore, it is assumed that the diagnoser can detect and

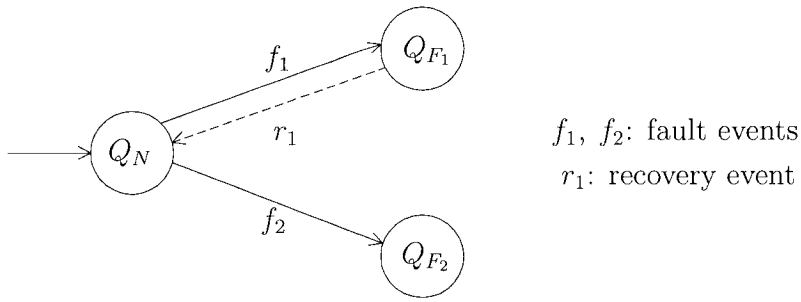


Figure 4.3: Plant with 2 failure modes

isolate each failure within a bounded delay (expressed in terms of events). Note that the diagnoser can be designed using any technique based on continuous-variable or discrete-event models, as long as the bounds for diagnosis are available. These bounds depend on the type of fault, plant dynamics, and the diagnosis technique used. We use an abstraction of the diagnosis system in our framework.

We model the diagnoser by a finite-state automaton $D = (X, \Sigma_D, \delta_D, x_0, X_m)$, where $\Sigma_D = \Sigma_G \dot{\cup} \Sigma_d$ with Σ_d denoting the set of *detection* events. Detection events are observable and uncontrollable and notify the supervisor that a failure has occurred in the plant and has been diagnosed. Instead of giving a general formula for the diagnoser model, we use an example to illustrate the structure of the diagnoser model.

Fig.4.4 depicts a diagnosis system that can diagnose two failures f_1 and f_2 in a plant. The diagnosis delay is 1 to 2 events for f_1, and 0 to 2 events for f_2. Here r denotes the recovery event, Σ_G represents the set of the plant events. The detection

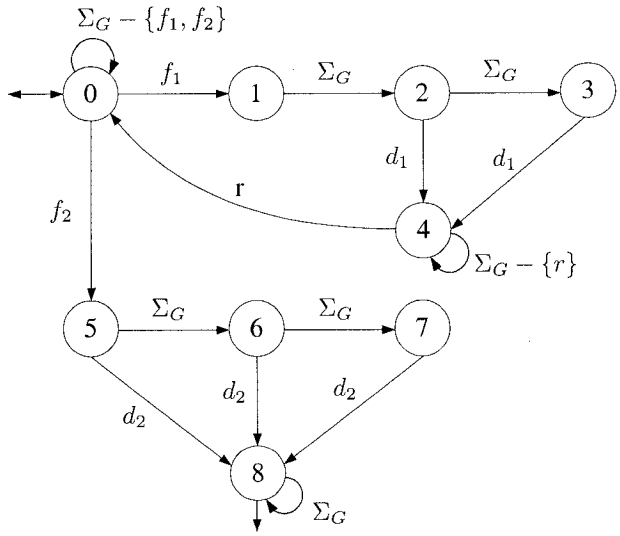


Figure 4.4: Diagnoser model

events d_1 and d_2 correspond to f_1 and f_2, respectively. The diagnoser and the plant are initialized together, and thus the initial state 0 corresponds to the set of states of the plant in normal operation. Once a failure f_1 (resp., f_2) occurs in the system, the diagnoser enters its intermediate states, state 1, 2, 3 (resp., state 5, 6, 7) before the fault is diagnosed and the detection event d_1 (resp., d_2) is generated. The number of intermediate states depends on the diagnosis delay. The plant will continue to generate events in Σ_G before the detection of failure. If a detection event "d_1" (resp., "d_2") is generated, the diagnoser enters state 4 (resp., state 8) announcing the diagnosis of failure f_1 (resp., f_2). In the case of failure f_1, the system can recover to the normal condition, and the diagnoser can reinitialize itself after recovery (transition from state 4 back to the initial state 0). When recovery to normal mode is possible, the state after detection is not marked, and only the initial state is marked. When recovery to normal mode is not possible (e.g., for f_2), we mark the state after failure detection

(state 8 in Fig.4.4), resulting in a set of marked states in the recovery mode of GD. The intermediate states (1,2,3,5,6,7) correspond to the transient mode of GD and are not marked. Diagnoser models for any other number of failures and diagnosis delays can be constructed similarly.

The diagnoser model is an abstraction of the adopted diagnosis system. Using an abstraction leads to the separation of the diagnosis problem and the control problem, which provides us with a simplified design procedure. Of course, the bounds for diagnosis delay have to be computed based on the characteristics of the plant and the diagnosis system. Since we use an abstract model of the diagnosis system, we may end up with a more conservative control policy . For instance, in a certain cycle of operation, the diagnosis delay may be smaller than the maximum delay considered for the diagnoser. This could in turn result in a conservative control.

We assume that the diagnoser can diagnose a failure (with bounded delay). This implies that the system generates the minimum number of events corresponding to minimum diagnosis delay. This assumption may not always hold. For example, if the system can only generate 2 events after a failure event and no more events can be further generated while the minimum diagnosis delay is 3 events. In this case, the fault may never be detected. This is a modeling issue which must be considered when diagnosis delays are computed. Our assumption on the generation of minimum events holds if the lower bound of diagnosis delay is zero or in the case of "live" systems, that

is, the systems will never reach a state where no events could be generated. In timed DES, this assumption is always true since the clock keeps generating tick events.

4.4 System to be Controlled

The synchronous product of the plant and the diagnoser GD =sync(G, D) is considered as the system to be controlled. The new feedback loop is shown in Fig.4.1. The system has three operation modes: normal, transient and recovery. When the system is working properly, it is considered to be in the normal mode. Once a failure occurs in the plant and not diagnosed, the system enters its transient mode. Upon the occurrence of the detection event, the system enters its recovery mode. Accordingly, the set of states of the controlled system can be partitioned as $Q^{GD} = Q_N^{GD} \dot{\cup} Q_T^{GD} \dot{\cup} Q_R^{GD}$. Q_T^{GD} and Q_R^{GD} can further be partitioned into p blocks related to the p failure modes (see Fig.4.5).

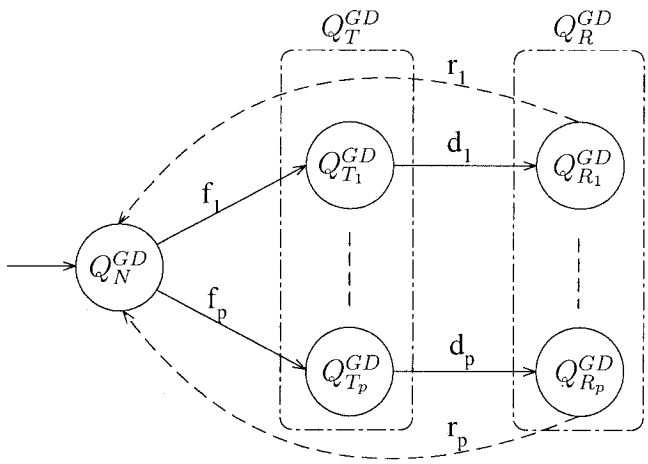


Figure 4.5: State partition

4.5 Fault Recovery Problem

As mentioned, failures are classified into two categories according to whether recovery to normal mode is possible or not. The fault recovery problem is discussed separately for these two categories. We first consider cases when all faults belong to one category. Extension to cases when faults from both categories are present in a system will be discussed later.

The set of specifications are given in terms of "legal" subset of the state set Q^{GD}. Let E_N, E_T and E_{R_i} $(i = 1, \cdots, p)$ denote the specifications in normal, transient and recovery modes respectively. Thus we have $E_N \subseteq Q_N^{GD}$, $E_T \subseteq Q_T^{GD}$, and $E_{R_i} \subseteq Q_{R_i}^{GD}$. Note that for each failure mode, different recovery specification may be adopted.

We would like to design an admissible supervisor S to prevent the system from entering the illegal states. Assuming that S is the proposed supervisor, we would like the system under supervision (S/GD) to meet the specifications of normal, transient, recovery modes, and to be nonblocking in both normal and recovery modes. That is, we want the following condition to be satisfied. The design specifications are the same for both aforementioned categories of failures:

(1a) $\mathcal{R}(S/GD_N) \subseteq E_N$;

(1b) $\mathcal{R}(S/GD_N) \subseteq \mathcal{CR}(S/GD_N)$;

(2) $\mathcal{R}(S/GD_{NT}) \cap Q_T^{GD} \subseteq E_T$;

(3a) $\mathcal{R}(S/GD_{NT_iR_i}) \cap Q_{R_i}^{GD} \subseteq E_{R_i}$;

(3b) $\mathcal{R}(S/GD_{NT_iR_i}) \subseteq \mathcal{CR}(S/GD_{NTR})$.

4.6 Modeling Example: Manufacturing Cell

In order to demonstrate our modeling procedure, we give an example of a manufacturing cell adapted from [1]. The manufacturing cell is described in [1] using timed discrete-event models. Our discussion here is limited to an untimed model.

The manufacturing system considered consists of two machines (MACH1 and MACH2) and two conveyors (CONV1 and CONV2) as shown in Fig.4.6. CONV1 works as an input conveyor with infinite source, and CONV2 is used as an output device with infinite sink. Two types of workpieces, part1 and part2, are picked up from CONV1, processed by the two machines, and then dropped in CONV2. For simplicity, the transfer of workpieces within the machines is assumed to be part of the working process of the machines. The machines can be repaired after break down.

The automata model of the machines and conveyors are displayed in Fig.4.7. Both conveyors are turned on and off simultaneously; thus one model is provided.

The list of events in the above figure are explained in the following:

α_{ij}: $MACH_i$ starts working on $part_j$;

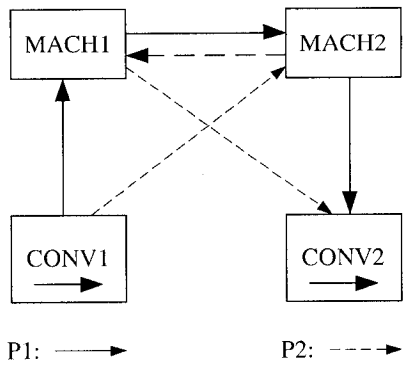


Figure 4.6: The manufacturing cell

β_{ij}: $MACH_i$ finishes working on $part_j$;

f_i: $MACH_i$ breaks down;

r_i: $MACH_i$ is repaired;

on: Conveyors are turned on;

off: Conveyors are turned off.

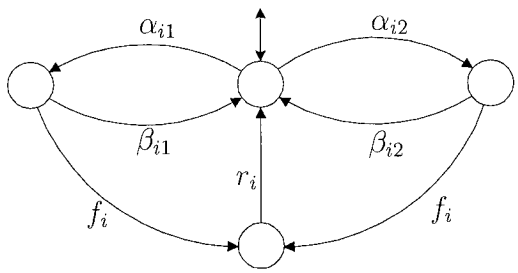
The set of controllable and observable events are: $\Sigma_c = \{\alpha_{ij}, r_i, on, off\}$, $\Sigma_o = \{\alpha_{ij}, \beta_{ij}, r_i, on, off\}$.

In the manufacturing cell, "single failure scenario" is assumed. We capture this assumption in our model using the automaton SFS shown in Fig.4.8.

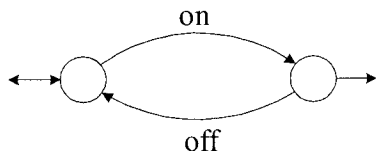
The closed-loop behavior of the manufacturing cell should satisfy the following specifications in the normal mode:

(N1) A part can only be processed by one machine at a time;

(N2) Part1 must be processed by MACH1 first, and then by MACH2;



(a) $MACH_i$, $i = 1, 2$



(b) CONV

Figure 4.7: The automata models of machines and conveyors

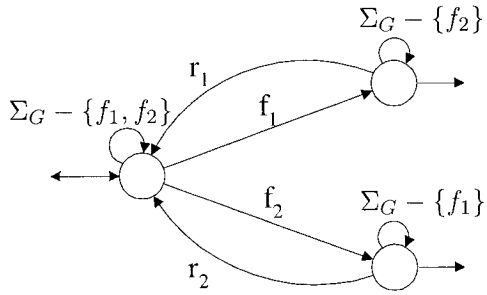


Figure 4.8: SFS

(N3) Part2 must be processed by MACH2 first, and then by MACH1;

(N4) In each production cycle, one part1 and one part2 must be processed;

(N5) The conveyors must be turned on before the machines start.

The transient specifications (which in this case are the same as these of the normal mode) are:

(T1) A part can only be processed by one machine at a time;

(T2) Part1 must be processed by MACH1 first, and then by MACH2;

(T3) Part2 must be processed by MACH2 first, and then by MACH1;

(T4) In each production cycle, one part1 and one part2 must be processed;

(T5) The conveyors must be turned on before the machines start.

The recovery specifications are:

(R1) A part can only be processed by one machine at a time;

(R2) Part1 must be processed by MACH1 first, and then by MACH2;

(R3) Part2 must be processed by MACH2 first, and then by MACH1;

(R4) In each production cycle, one part1 and one part2 must be processed;

(R5) The conveyors must be turned on before the machines start;

(R6) After a failure is detected and isolated in a machine, first the conveyors must be turned off, next the machine must be repaired, and then the production cycle must be resumed and completed.

We use specification (N1) as an example to explain how a specification can be represented in the state-based framework in terms of safe (legal) states. First, an automaton model is constructed based on the natural language description of the specification to represent the specification (Fig.4.9). This model describes the admissible event sequences for the process of part1. Next, we convert this automaton to a model that includes a forbidden state reachable only through the illegal event sequences. Fig.4.10 is the specification model used for our supervisor design. According to the design specification, if MACH1 starts working on part1 (α_{11}), then MACH2 is not allowed to work on part1 until MACH1 finishes working on it. Therefore, an event α_{21} following α_{11} leads to the forbidden state B_1. Similarly, the event sequence

$\alpha_{21}\alpha_{11}$ leads to B_1.

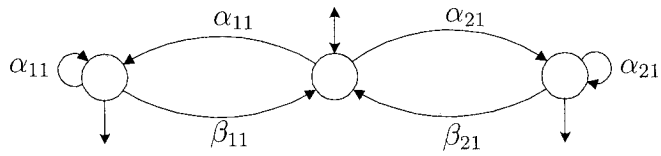


Figure 4.9: $SPEC_{N1,1}$

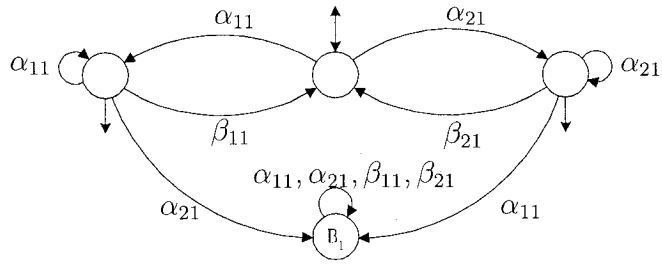


Figure 4.10: $SPEC'_{N1,1}$

Models similar to $SPEC'_{N1,1}$ can be constructed for the rest of design specifications. The plant G can be formed by the synchronous product of the machine and conveyor models, SFS, and all of the modified specification models:

$$\text{G=sync}(MACH1, MACH2, CONV, SFS, SPEC'_{N1,1}, \cdots, SPEC'_{R6}).$$

We assume that the diagnosis system can detect failures with a delay of 0 to 2 events. We do not give the state transition graph of the diagnoser for brevity. The combined model of the plant G and the diagnoser D forms the system to be controlled, GD =sync(G, D).

The states of GD can be represented by the tuples $(x_{m_1}, x_{m_2}, x_c, x_s, x_{N1,1}, \cdots, x_{R6}, x_D)$. Thus any state of GD for which one of components $x_{N1,1}, \cdots, x_{R6}$ is a forbidden state

(for example, B_1) is considered forbidden (unsafe) for GD.

In the following chapter, we will discuss procedures for designing modular switching supervisors to solve the fault recovery problem.

Chapter 5

Fault Recovery Problem and Synthesis Procedures

In this chapter, we present our solutions for the fault recovery problem. We first discuss the cases where recovery to normal mode is not possible. We refer to this case as "failure accommodation". Next we consider the cases in which recovery to normal mode is possible. Following this, we examine problems involving both of the above scenarios. For fault recovery problem related to each category, we provide two approaches. All solutions provided follow a modular switching scheme to the supervisory control problem. Using a modular approach can simplify the design procedure, facilitate modification and upgrading of the supervisor when a component or a specification changes.

5.1 Failure Accommodation Problems (No Recovery to Normal Mode)

We first discuss the case of one failure mode. The control problem dealing with multiple failure modes will be discussed next. The system to be controlled is shown in Fig.5.1. Initially, the system starts in normal condition. Once a failure event f occurs, the system enters its transient mode and will remain in this mode until the fault is detected. It is assumed that the failure can be detected with a finite delay. Once the detection event d is generated, the system is in the recovery mode and will stay in this mode afterwards. During the recovery period, the system may operate under less restrictive (compared to normal operation) requirements. We will solve this type of problem using two strategies introduced below. In both approaches, a modular switching supervisory mechanism is used.

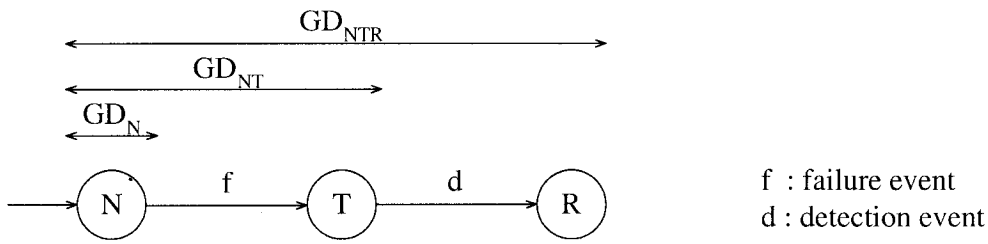


Figure 5.1: Plant model for failure accommodation problems

5.1.1 First Approach

5.1.1.1 Supervisor Design

We assume that three nonblocking OBS supervisors S_N, S_T, S_R have been designed to enforce E_N, E_T and E_R ,respectively, based on the plant models GD_N, GD_{NT} and GD_{NTR}. Therefore,

(1a) $\mathcal{R}(S_N/GD_N) \subseteq E_N$;

(1b) $\mathcal{R}(S_N/GD_N) \subseteq \mathcal{CR}(S_N/GD_N)$;

(2) $\mathcal{R}(S_T/GD_{NT}) \cap Q_T^{GD} \subseteq E_T$;

(3a) $\mathcal{R}(S_R/GD_{NTR}) \cap Q_R^{GD} \subseteq E_R$;

(3b) $\mathcal{R}(S_R/GD_{NTR}) \subseteq \mathcal{CR}(S_R/GD_{NTR})$.

As indicated, S_N and S_R are nonblocking supervisors for GD_N and GD_{NTR}, respectively. Let $S_N = (Z_N, \Sigma_o, \xi_N, z_0^{(N)}, f_N)$, $S_T = (Z_{NT}, \Sigma_o, \xi_T, z_0^{(T)}, f_T)$, and $S_R = (Z_{NTR}, \Sigma_o, \xi_R, z_0^{(R)}, f_R)$. Fig.5.2 shows how the three controllers are used from the normal to recovery mode.

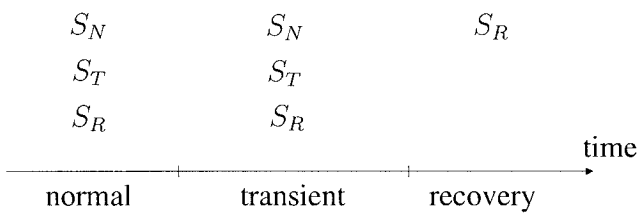


Figure 5.2: Supervisors involved in the first approach

During the normal mode, the conjunction of S_N, S_T and S_R controls the plant initially. Once a failure occurs, the system will be in the transient mode. Since fault

events are unobservable, a supervisor can not differentiate normal mode from transient mode. Therefore, the conjunction of S_N, S_T and S_R will continue to control the system until a detection event is generated. Upon the detection of the failure, the system enters the recovery mode and starts to perform recovery action. During the recovery period, S_R will control the system alone. Here S_R is used from the beginning in order to ensure that the recovery requirements can be met later during recovery. In most of the cases, S_R only follows the control action performed by S_N and S_T during the normal and transient modes. While in other cases, S_R may have to restrict the behavior of the plant during the normal and transient modes so that later the recovery specifications are not violated.

Since S_N, S_T and S_R are designed to restrict the plant behavior GD_N, GD_{NT} and GD_{NTR} respectively, the plant models for these three controllers are not the same. Therefore, we have to make some adjustment to the definition of the conjunction of supervisors to be able to represent the joint operation of S_N, S_T and S_R.

In the following, we describe the operation of the conjunction of S_N, S_T and S_R. Three different observers are used by the OBS supervisors S_N, S_T and S_R. Let us discuss the relations among the state estimates of S_N, S_T and S_R. Initially, by assumption, we have $z_0^{(N)} = z_0^{(T)} \cap Q_N^{GD}$ and $z_0^{(T)} = z_0^{(R)}$. Let $z_k^{(N)}$, $z_k^{(T)}$ and $z_k^{(R)}$ be the state estimates by three supervisors following the observation of the sequence $s = \sigma_1 \cdots \sigma_k \in \Sigma_o^*$ (before the detection event d occurs). We can easily see that

$z_k^{(T)} = z_k^{(R)}$ and $z_k^{(N)} = z_k^{(T)} \cap Q_N^{GD}$. At this point, the set of events disabled by the conjunction of S_N, S_T and S_R will be $f_N(z_k^{(N)}) \cup f_T(z_k^{(T)}) \cup f_R(z_k^{(R)})$.

Note that we can think of $S_N \wedge S_T \wedge S_R$ as an OBS $S = (Z, \Sigma_o, \xi, z_0, f)$ with $Z \subseteq Pwr(Q_{NT}^{GD})$, ξ defined as:

$$\xi(z,\sigma) = \begin{cases} \xi_T(z,\sigma), & \text{if } \xi_N(z \cap Q_N^{GD},\sigma)! \text{ and } \xi_T(z,\sigma)! \text{ and } \xi_R(z,\sigma)!; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

and $f(z) = f_N(z \cap Q_N^{GD}) \cup f_T(z) \cup f_R(z)$.

After "d" occurs, S_N and S_T are taken out of the control loop and S_R takes sole control of the system. The events disabled will be $f_R(z^{(R)})$. Note that after the detection event "d" is generated, $z^{(R)} \subseteq Q_R^{GD}$.

The design procedure is easy to extend to the case of multiple failure modes. For $p \geqslant 2$, S_N and S_T as before, are assumed to be OBS supervisors to enforce E_N and E_T. Moreover, p recovery supervisors $S_{R_1}, \cdots, S_{R_p}$ are designed to enforce the recovery specifications $E_{R_1}, \cdots, E_{R_p}$, respectively, and to satisfy the nonblocking requirements. We define $S_R = \wedge S_{R_i}$ $(i = 1, \cdots, p)$. Accordingly, in the normal and transient modes, the conjunction of S_N, S_T and S_R controls the plant. Once a failure, say f_i, is detected, the corresponding recovery supervisor S_{R_i} takes sole control of the plant during recovery.

We illustrate our design procedure by an example. Fig.5.3 is the system to be controlled (GD) with $\Sigma = \{\alpha, \beta, \gamma, f, d\}$, $\Sigma_c = \{\alpha, \beta, \gamma\}$, and $\Sigma_o = \{\alpha, \beta, \gamma, d\}$. The plant states in each mode are $Q_N^{GD} = \{0, 1, 2, 3\}$, $Q_T^{GD} = \{4, 5, 6, 10, 11\}$, $Q_R^{GD} = \{7, 8, 9, 12\}$, and $Q_m = \{0, 1, 8, 9\}$. The set of forbidden (unsafe) states are state 1, 6, 9, 12. Unsafe states are depicted in bold circles in Fig.5.3. Thus we have $E_N = \{0, 2, 3\}$, $E_T = \{4, 5, 10, 11\}$, $E_R = \{7, 8\}$.

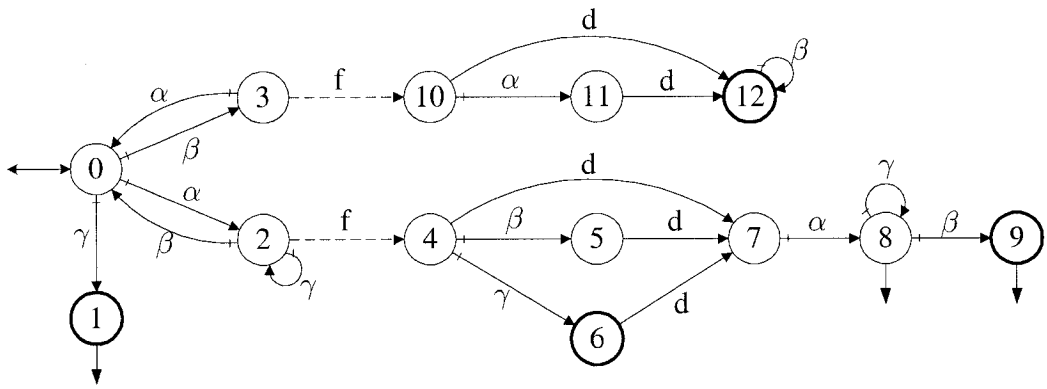
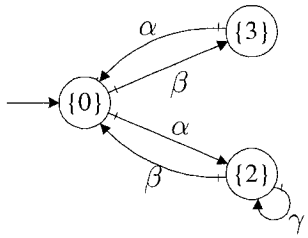
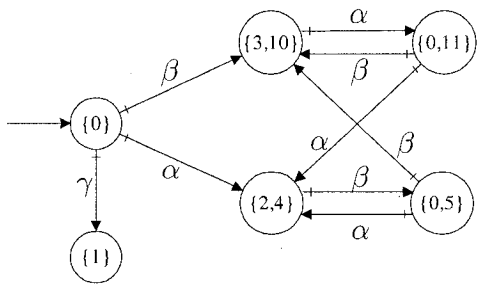


Figure 5.3: GD

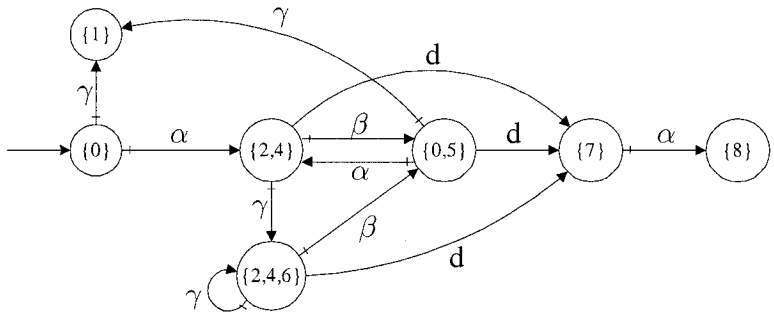
In order to enforce the set of specifications, three OBS supervisors S_N, S_T and S_R shown in Fig.5.4 are proposed. Note that S_N and S_R are nonblocking supervisors for GD_N and GD_{NTR}, respectively. S_N disables γ at state $\{0\}$ to prevent GD_N from entering state 1. Event γ is disabled by S_T at state $\{2, 4\}$, and supervisor S_R disables β at state $\{0\}$ and $\{0, 5\}$ to prevent GD from entering state 12 (using the uncontrollable sequence $\beta f d$ from state 0), and also disables β at $\{8\}$ to prevent the system from entering state 9.



(a) S_N



(b) S_T



(c) S_R

Figure 5.4: S_N,S_T and S_R

The system under supervision of $S_N \wedge S_T \wedge S_R$ (i.e., meet(GD,selfloop($S_N \wedge S_T \wedge S_R$,{$f$}))) is displayed in Fig.5.5. We can observe that the conjunction of S_N, S_T and S_R disables event γ and β at state $\{0\}$ since S_R has to disable β at state $\{0\}$ to ensure that the system will not enter the forbidden state 12 during recovery (even though S_N and S_T allow β at state $\{0\}$). Moreover, the system under supervision is nonblocking in the normal and recovery modes. No uncontrollable events are disabled in any mode.

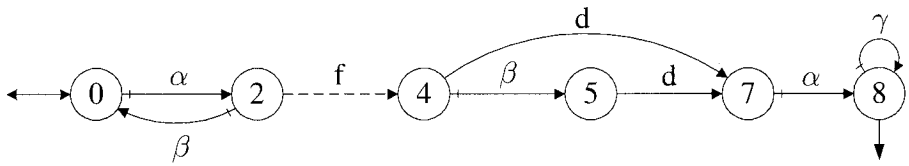


Figure 5.5: GD under supervision of $S_N \wedge S_T \wedge S_R$

In the following, we discuss the issues of nonblocking property of the system under supervision and supervisor admissibility.

5.1.1.2 Supervisor Admissibility

The OBS supervisors S_N, S_T and S_R are admissible with respect to their respective plant models GD_N, GD_{NT} and GD_{NTR}. S_N is designed for GD_N. However, it is also in the feedback loop during the transient mode, and it may attempt to disable some uncontrollable transitions in this mode. Being unaware of the occurrence of a failure in the plant, the observer for normal mode keeps generating state estimates while the plant is in transient mode. S_N may attempt to disable uncontrollable transitions which can take place in the transient mode only. Since S_T is designed for GD_{NT}, it never disables any uncontrollable event in the normal and transient modes.

In order to solve the above-mentioned admissibility issue, we can employ a similar procedure to that in [12]: (1) Add a marked state z_d to S_N; (2) For any state $z_k^{(N)}$ ($z_k^{(N)} \neq z_d$) of S_N, attach a transition $z_k^{(N)} \xrightarrow{\sigma} z_d$, where $\sigma \in \Sigma_p$ is an observable but uncontrollable event, and σ is not defined at $z_k^{(N)}$ in S_N (i.e., not$\xi_N(z_k^{(N)}, \sigma)$!); (3) Add

selfloops $z_d \xrightarrow{\sigma} z_d$ for all $\sigma \in \Sigma_{GD,o}$, where $\Sigma_{GD,o}$ denotes the set of observable events of GD. Obviously, the modified supervisor, which we call S_{NM}, will never disable an uncontrollable event during the transient mode. The transitions added in the above procedure never happen in the normal mode, and may occur only in the transient mode. When S_{NM} enters z_d, the supervisor knows that a failure has occurred and the system is in the transient mode. After this, S_{NM} is effectively out of the control loop. During the normal operation, S_N and S_{NM} behave similarly. Therefore, the system under supervision of S_N (S_N/GD_N) and S_{NM} (S_{NM}/GD_N) are the same.

For the case of multiple failures, since each recovery supervisor S_{R_i} is designed for $GD_{NT_iR_i}$, we employ the same procedure to S_{R_i} to make it admissible during other transient modes (i.e.,T_j, $j \neq i$).

5.1.1.3 Nonblocking Property

In addition to the design specifications, we would also like the system under supervision to satisfy the nonblocking property during the normal and recovery modes. No nonblocking requirement is set for the transient mode, since by assumption, the system enters the recovery mode with a bounded delay. It is assumed that S_N and S_R are nonblocking supervisors for GD_N and GD_{NTR}, respectively. However, the system under supervision of the conjunction of S_N, S_T and S_R may not be nonblocking in the normal and the recovery modes.

To formulate the nonblocking property, we construct a supervisor which in terms of supervisory action is identical to our modular switching supervisor. Note that this procedure will not be implemented in practice, it is only employed here in order to formally define the nonblocking property. First, $\tilde{S}_N$ is constructed by modifying S_{NM}. We construct $\tilde{S}_N$ by adding a new marked state z'_d to S_{NM}. Then we attach a transition $z_k^{(N)} \xrightarrow{d} z'_d$ at every state $z_k^{(N)}$ of S_N. Moreover, we add selfloops at state z'_d for all $\sigma \in \Sigma_{GD,o}$. After applying the modifications, $\tilde{S}_N$ behaves similarly to S_{NM} in the normal and transient modes, and once "d" occurs, $\tilde{S}_N$ stops disabling events and is out of the feedback loop. Therefore, the system under supervision of $\tilde{S}_N$ in the normal mode is the same as the system under supervision of S_N. Similarly, we apply the same procedure to S_T to get $\tilde{S}_T$. $\tilde{S}_T$ has the same effect as S_T during the normal and transient modes, and is taken out of the control loop upon failure detection. Let $S = \text{meet}(\tilde{S}_N, \tilde{S}_T, S_R)$. Then S can be thought of as the supervisor which controls the system from normal to recovery mode. Therefore, the nonblocking condition for the recovery mode is that

$$\mathcal{R}(S/GD_{NTR}) \subseteq \mathcal{CR}(S/GD_{NTR})$$

The following proposition states that if the supervisor modules do not conflict during normal mode (i.e., the conjunction is a nonblocking supervisor), then the plant under supervision will be nonblocking during recovery as well.

Proposition 5.1.1. *Assume S_N and S_R are nonblocking supervisors for GD_N and*

GD_{NTR}, respectively, and $\mathcal{R}(S/GD_N) \neq \emptyset$. If the system under supervision of $S_N \wedge S_T \wedge S_R$ (or S) is nonblocking in normal mode, i.e., $\mathcal{R}(S/GD_N) \subseteq \mathcal{CR}(S/GD_N)$, then the system GD_{NTR} under supervision of S will be nonblocking during recovery, that is, $\mathcal{R}(S/GD_{NTR}) \subseteq \mathcal{CR}(S/GD_{NTR})$.

Proof. We have to show that in the system under supervision, the set of marked states are reachable from every reachable state.

I. States in normal mode: Since we assume that the system under supervision in normal mode (S/GD_N) is nonblocking, it is obvious that from every state $q \in \mathcal{R}(S/GD_N)$, there exist some marked states which can be reached from q.

II. In transient mode, a failure event has occurred in the system, and the plant is in some transient state. Since it is assumed that a failure can be detected within finite delays, from every $q \in \mathcal{R}(S/GD_{NT})$, there is a sequence of events leading to some state $q' \in Q_R^{GD}$. Under the assumption that S_R is a nonblocking controller wrt GD_{NTR}, there exist a marked state $q'' \in Q_R^{GD}$ which can be reached from q'. Thus q is coreachable.

III: Let q be a state in recovery mode reached under the supervision of S. Therefore, $q \in \mathcal{R}(S/GD_{NTR})$ and $q \in Q_R^{GD}$. Since $\mathcal{R}(S/GD_{NTR}) \subseteq \mathcal{R}(S_R/GD_{NTR})$ implies $q \in \mathcal{R}(S_R/GD_{NTR})$, and by assumption, S_R is a nonblocking supervisor for GD_{NTR}, the controlled system could eventually reach some marked state $q' \in Q_R^{GD}$ from q.

□

For the case of multiple failures, specifically for the jth failure mode, we apply a similar procedure to S_N, S_T and S_{R_i} $(i = 1 \cdots p, i \neq j)$ by attaching d_i transitions to new added states and construct $\tilde{S}_N$, $\tilde{S}_T$ and $\tilde{S}_{R_i}$'s $(i \neq j)$. Let S_{NTR_j} =meet$(\tilde{S}_N, \tilde{S}_T, \tilde{S}_{R_1}, \cdots, \tilde{S}_{R_{j-1}}, \tilde{S}_{R_{j+1}}, \cdots, \tilde{S}_{R_p}, S_{R_j})$. The nonblocking condition for the recovery mode R_j is that

$$\mathcal{R}(S_{NTR_j}/GD_{NT_jR_j}) \subseteq \mathcal{CR}(S_{NTR_j}/GD_{NT_jR_j}) \tag{5}$$

Then we have the following proposition.

Proposition 5.1.2. *Assume that S_{R_j} is a nonblocking supervisor for $GD_{NT_jR_j}$ $(1 \leqslant j \leqslant p)$. If $S_N \wedge S_T \wedge S_{R_1} \wedge \cdots \wedge S_{R_p}$ is a nonblocking supervisor for GD_N, then S_{NTR_j} is a nonblocking supervisor for $GD_{NT_jR_j}$.*

If the nonblocking condition (5) holds for all failure modes $(j = 1, \cdots, p)$, then the fault recovery supervisor is a nonblocking supervisor.

5.1.2 Second Approach

In the first approach, S_R (and $S_N \wedge S_T$) is initialized with the plant, and the conjunction of $S_N \wedge S_T$ and S_R controls the plant during the normal and transient modes. In some cases, S_R may not disable any event in addition to those disabled by $S_N \wedge S_T$. In these situations, S_R simply tracks the plant under supervision of $S_N \wedge S_T$ during normal and transient modes, and only it is in the recovery mode where S_R really affects the plant behavior. This motivates the second approach in this section in which S_R is designed to be used in the feedback loop after the fault is diagnosed. In other words,

during the normal and transient modes, the system is supervised by $S_N \wedge S_T$. Once a fault is diagnosed, S_R is initialized and put into the feedback loop, and $S_N \wedge S_T$ is removed (Fig.5.6).

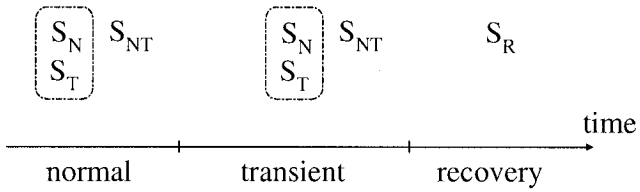


Figure 5.6: Supervisors used in the second approach

This second approach is easier to implement (requiring less computations) since during the normal and transient modes, S_R is not active. This advantage becomes more important in system with large number faults (and thus the recovery supervisors S_{R_i}).

The main drawback is that we wait till detection event occurs before we engage the recovery supervisor and in some cases, this may be too late. In other words, in some cases, to ensure recovery specification E_R can be met, we have to limit the plant behavior in normal and transient modes. Note that the solutions provided by the second approach in which the set of events disabled by S_R is a subset of events disabled by $S_N \wedge S_T$ during the normal and transient modes can be considered as a subset of the solutions of the first approach.

In the following, we discuss the design procedure following the second approach. As in the first approach, we first discuss the single failure case ($p = 1$).

We assume that S_N and S_T are OBS supervisors designed for GD_N and GD_{NT} to enforce E_N and E_T, respectively, and the following conditions are satisfied:

(1a) $\mathcal{R}(S_N/GD_N) \subseteq E_N$;

(1b) $\mathcal{R}(S_N/GD_N) \subseteq \mathcal{CR}(S_N/GD_N)$;

(2) $\mathcal{R}(S_T/GD_{NT}) \cap Q_T^{GD} \subseteq E_T$.

Suppose S_N and S_T are represented by $S_N = (Z_N, \Sigma_o, \xi_N, z_0^{(N)}, f_N)$, and $S_T = (Z_{NT}, \Sigma_o, \xi_T, z_0^{(T)}, f_T)$. The conjunction of S_N and S_T, denoted by S_{NT}, will control the system during the normal and transient modes.

After failure detection, we need to switch to the recovery controller S_R. Let $z_{fin}^{(T)}$ be the last state estimate provided by $S_N \wedge S_T$ right before the detection event is generated. Then the state estimate supplied to S_R as initial state estimate will be:

$$z_0^{(R)} = \{q \mid \exists\, q' \in z_{fin}^{(T)} \;\&\; \exists s, s' \in \Sigma_{uo}^* :\; q = \delta(q', sds')\}.$$

Obviously the state of the system GD immediately after "d" occurs must belong to

$$Q_{R,0}^{GD} = \{q \mid \exists\, q' \in z_{fin}^{(T)}, s \in \Sigma_{uo}^* :\; q = \delta(q', sd)\}.$$

Suppose $x_{R_0} \in Q_{R,0}^{GD}$. Let $Q_{x_{R_0}}^{GD}$ denote the set of states in Q_R^{GD} reachable from x_{R_0}. We define $GD_R(x_{R_0}) = (Q_{x_{R_0}}^{GD}, \Sigma_{GD}, \delta_{x_{R_0}}, x_{R_0}, Q_m \cap Q_{x_{R_0}}^{GD})$ as the automaton with

initial state x_{R_0}, and containing states in Q_R^{GD} reachable from x_{R_0} and the transitions among them. Therefore, $\delta_{x_{R_0}}$ is the restriction of δ to $Q_{x_{R_0}}^{GD}$. Let S_R be an OBS supervisor designed such that for all $x_{R_0} \in Q_{R,0}^{GD}$,

$$\mathcal{R}(S_R/GD_R(x_{R_0})) \subseteq E_R;$$

$$\mathcal{R}(S_R/GD_R(x_{R_0})) \subseteq \mathcal{CR}(S_R/GD_R(x_{R_0})).$$

If an S_R as defined above exists, then after detection event "d" occurs, the supervisory control can switch from $S_N \wedge S_T$ to S_R, and S_R will enforce specification E_R and guarantee nonblocking property during the recovery mode.

Of course, $z_{fin}^{(T)}$ depends on the time that the fault occurs and the time that the fault is diagnosed (event "d" is generated). Therefore, the recovery supervisor may be initialized with different initial states. In order to find the different values of $z_{fin}^{(T)}$ and thus the initial states $z_0^{(R)}$, we can use the following procedure.

Let S_{NM} be the controllable, normal supervisor obtained by modifying S_N as described in Section 5.1.1.2 ("Supervisor Admissibility"). Let S_{NMT} =meet(S_{NM}, S_T) and Z_{NMT} be the set of state estimates provided by the observer in S_T:

$$Z_{NMT} = \{z \mid \exists z' : (z', z) \text{ is a state of } S_{NMT}\}.$$

Finally, the set of all final states $Z_{fin}^{(T)}$ can be obtained as those state estimates containing states in Q_T^{GD} from which Q_R^{GD} is reachable through event sequences sd with

s being a sequence of unobservable events:

$$Z_{fin}^{(T)} = \{z \in Z_{NMT} \mid \exists q \in z,\ q' \in Q_R^{GD},\ s \in \Sigma_{uo}^* :\ q' = \delta(q, sd)\}.$$

Therefore, based on each initial state estimate $z_0^{(R)}$, a recovery supervisor is designed. Assuming that for all possible initial state estimates, recovery supervisors can be designed, let $S_R^1, \cdots, S_R^k$ denote the different versions of recovery supervisors. Now following the merging procedure described in Chapter 3, we can merge all of these supervisors into a single OBS with multiple initial states. We shall denote the resulting supervisor as S_R.

We demonstrate the design procedure using an example. The system to be controlled GD is shown in Fig.5.7. The set of events are $\Sigma = \{\alpha, \beta, \gamma, \tau, f, d\}$, $\Sigma_c = \{\alpha, \beta, \gamma\}$, and $\Sigma_o = \{\alpha, \beta, \gamma, d\}$. The system states in each mode are $Q_N^{GD} = \{0, 1, 2, 3\}$, $Q_T^{GD} = \{4, 5, 6, 10, 11\}$, $Q_R^{GD} = \{7, 8, 9, 12, 13\}$, and $Q_m^{GD} = \{0, 1, 8, 9\}$. The set of forbidden (unsafe) states are state 1, 6, 9, and are depicted in bold circles in Fig.5.7. Thus we have $E_N = \{0, 2, 3\}$, $E_T = \{4, 5, 10, 11\}$, $E_R = \{7, 8, 12, 13\}$.

In order to enforce the set of specifications, three OBS supervisors S_N, S_T and S_R are designed and shown in Fig.5.8. S_N disables γ at state $\{0\}$ to prevent GD_N from entering state 1. Event γ is disabled by S_T at state $\{2, 4\}$ to prevent GD from entering state 6. The possible initial state estimates of S_R, $z_0^{(R)}$, are $\{12\}$ and $\{7, 8\}$ as

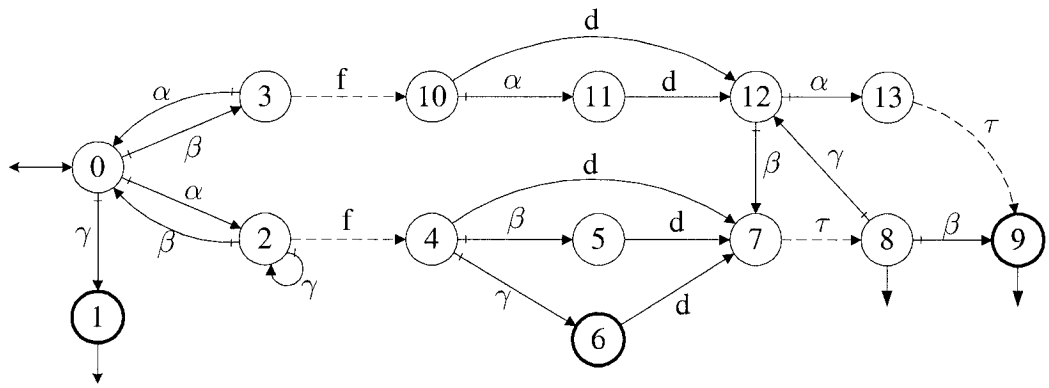


Figure 5.7: GD

seen in Fig.5.8(b). States $\{12\}$ and $\{7,8\}$ are reached through a d transition that are shown with double headed arrow. Two OBS supervisors S_R^1 and S_R^2 are designed and initialized with $\{12\}$ and $\{7,8\}$, respectively. After merging S_R^1 and S_R^2, we obtain S_R (Fig.5.8(e)). To prevent GD from entering state 9 during recovery, supervisor S_R disables α at state $\{12\}$ (since τ is unobservable, thus uncontrollable), and disables β at state $\{7,8\}$.

The system under supervision of $S_N \wedge S_T$ and S_R is displayed in Fig.5.9. We can observe that the conjunction of S_N, S_T will disable event γ at state 0, 2 and 4, and S_R disables α at state 12 and disables β at state 8. Moreover, the system under supervision is nonblocking in the normal and recovery modes. No uncontrollable events are disabled in each mode.

The system under supervision should satisfy the nonblocking property during the normal and recovery modes. Let us assume that the recovery supervisor has been

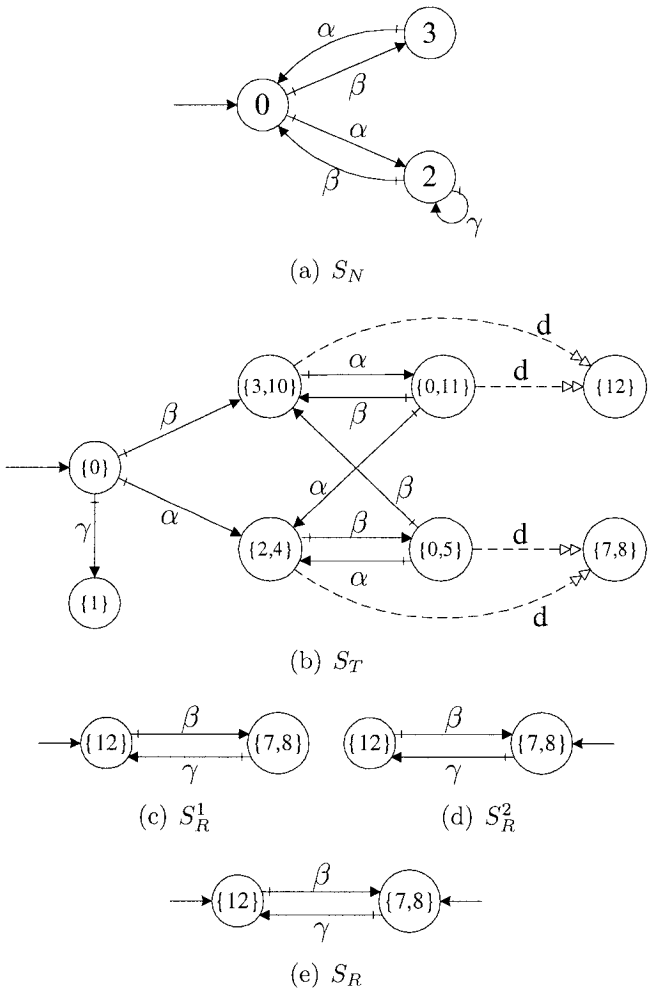


Figure 5.8: OBS supervisors S_N,S_T and S_R

designed following the procedure outlined in this subsection. This would guarantee nonblocking property during the recovery mode. Thus in this case, we only have to be concerned with nonblocking property in the normal mode as explained in the following proposition. The proof is similar to the proof of Proposition 5.1.1 and is not provided for brevity.

Proposition 5.1.3. *Assume S_N is a nonblocking supervisor for GD_N, and S_R can be designed as outlined in this subsection. Thus if $S_N \wedge S_T$ is a nonblocking supervisor*

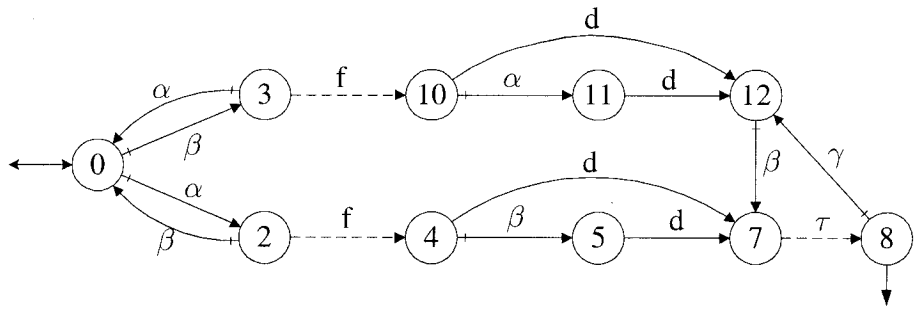


Figure 5.9: GD under supervision of $S_N \wedge S_T$ and S_R

for GD_N, then the system under supervision will be nonblocking in both normal and recovery modes.

For the case of multiple failure modes ($p \geqslant 2$), S_N and S_T are designed as before to enforce E_N and E_T, respectively. Moreover, p OBS recovery supervisors $S_{R_1}, \cdots, S_{R_p}$ are designed to enforce the recovery specifications $E_{R_1}, \cdots, E_{R_p}$, respectively, and satisfy the nonblocking requirement. Accordingly, in the normal and transient mode, the conjunction of S_N and S_T controls the plant. Once a failure, say f_i, is detected (d_i generated), S_T passes the initial state estimates to the appropriate recovery supervisor S_{R_i}, and S_{R_i} takes sole control of the plant during recovery.

5.2 Problems Involving Recovery to Normal Mode

In this section, we consider problems in which recovery to normal is possible. In other words, after fault diagnosis, the faulty component (or system) is repaired and the system resumes its normal operation. Fig. 5.10 shows the system to be controlled in the case of single failure ($p = 1$).

Following a discussion parallel to that of section 5.1, we propose two approaches: one in which the recovery supervisor is engaged when the system starts and is kept active afterwards (third approach); and another approach in which the recovery supervisor is only engaged during the recovery mode (fourth approach). Compared with the former approach (i.e., third approach), the latter (fourth approach) has the advantage that it is computationally less complex to implement since during the normal and transient modes, the recovery supervisor is not engaged in the feedback loop.

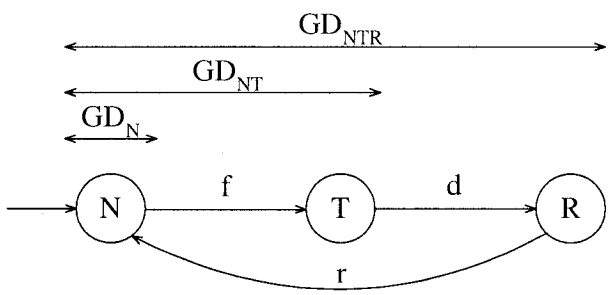


Figure 5.10: Plant model in the case of recovery to normal mode

5.2.1 Third Approach

Fig.5.11 shows the supervisors involved in each mode. As in the first approach, the conjunction of S_N, S_T and S_R controls the system in the normal and transient modes, and S_R supervises the system during recovery. S_R is put into the feedback loop initially when the system starts so that during recovery, the system's behavior does not violate the design specifications. After the failure is detected and isolated, $S_N \wedge S_T$

will be disabled, and S_R alone controls the system. After recovery, the normal and transient supervisors are once again put into the feedback loop. The recovery supervisor also remains active. After recovery, the system resumes its normal operation, and the supervisors S_N and S_T are initialized with the state estimates obtained through the observer of the recovery supervisor (to be explained in the following). Therefore, S_N and S_T are initialized with new state estimates each time the system recovers to normal condition (following a failure) which in turn may lead to new states for S_N and S_T. As a result, S_N and S_T may not be designed in one step as in the first approach and in fact, the design will be iterative. But as we will see, the design procedure will terminate in a finite number of steps. If the system always goes back to its initial state after recovery, the design procedure converges in one step and thus is the same as in the first approach.

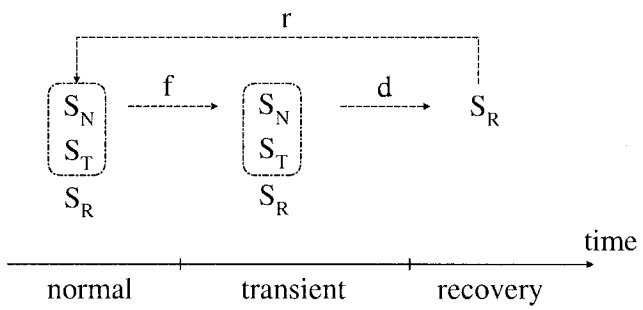


Figure 5.11: Supervisors Involved

The supervisor S_N, S_T and S_R are designed following the steps in Table 5.1. The details are given in the following.

Table 5.1: Third approach: OBS design steps

Step	S_N	S_T	S_R
0	$S_{N,0}$	$S_{T,0}$	S_R
1	$S_{N,1}$ =merge($S_{N,0}$, $S^1_{N,1}$, $\cdots$, $S^{k_1}_{N,1}$)	$S_{T,1}$ =merge($S_{T,0}$, $S^1_{T,1}$, $\cdots$, $S^{k_1}_{T,1}$)	S_R
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n^*	S_{N,n^*} =merge(S_{N,n^*-1}, S^1_{N,n^*} ,$\cdots$, $S^{k_1}_{N,n^*}$)	S_{T,n^*} =merge(S_{T,n^*-1}, S^1_{T,n^*}, $\cdots$, $S^{k_1}_{T,n^*}$)	S_R

Step 0: $S_{N,0}$, $S_{T,0}$ and S_R are designed to enforce E_N, E_T and E_R respectively as in the first approach, that is,

(1a) $\mathcal{R}(S_{N,0}/GD_N) \subseteq E_N$;

(1b) $\mathcal{R}(S_{N,0}/GD_N) \subseteq \mathcal{CR}(S_{N,0}/GD_N)$;

(2) $\mathcal{R}(S_{T,0}/GD_{NT}) \cap Q_T^{GD} \subseteq E_T$;

(3a) $\mathcal{R}(S_R/GD_{NTR}) \cap Q_R^{GD} \subseteq E_R$;

(3b) $\mathcal{R}(S_R/GD_{NTR}) \subseteq \mathcal{CR}(S_R/GD_{NTR})$.

Step 1: Next, $S_{N,1}$ and $S_{T,1}$ are designed to enforce normal and transient specifications after a failure is detected and the system recovers for the first time. The design procedure is as follows. Let $z_{fin}^{(R)}$ be the last state estimate provided by S_R right before the recovery event "r" is generated the first time. Then the state estimate supplied to the normal and transient supervisors as initial state estimates will be:

$$z_{0,1}^{(T,1)} = \{q \mid \exists\, q' \in z_{fin}^{(R)} \;\&\; \exists\, s, s' \in \Sigma_{uo}^* :\; q = \delta(q', srs')\}.$$

and

$$z_{0,1}^{(N,1)} = z_{0,1}^{(T,1)} \cap Q_N^{GD}.$$

Note that $z_{0,1}^{(T,1)}$ is identical to the state estimate of S_R after recovery.

Depending on the time a fault occurs, the time it is diagnosed and the time recovery event occurs, the state estimates supplied to the normal and transient supervisors could vary. Let us denote these initial state estimates by $z_{0,1}^{(N,1)}, \cdots, z_{0,k_1}^{(N,1)}$ and $z_{0,1}^{(T,1)}, \cdots, z_{0,k_1}^{(T,1)}$. Based on these initial state estimates, a set of OBS supervisors $S_{N,1}^1, \cdots, S_{N,1}^{k_1}$ and $S_{T,1}^1, \cdots, S_{T,1}^{k_1}$ are designed to enforce E_N and E_T, respectively. Next we merge these supervisors with $S_{N,0}$ and $S_{T,0}$, that is, $S_{N,1}$ =merge$(S_{N,0}, S_{N,1}^1, \cdots, S_{N,1}^{k_1})$ and $S_{T,1}$ =merge$(S_{T,0}, S_{T,1}^1, \cdots, S_{T,1}^{k_1})$. Note that S_R is not changed because it is designed based on the states of the entire plant GD_{NTR} and as explained before, the state estimate after recovery is one of the states of S_R. Thus no states or transitions need to be added to S_R.

Step n: Repeat the above procedure until no new states or transitions are added to the normal and transient supervisors, i.e., $S_{N,n} = S_{N,n-1}$, $S_{T,n} = S_{T,n-1}$. This procedure will converge in a finite number of steps, say n^*. The reason is that the number of states and transitions of $S_{N,i}$ and $S_{T,i}$ form non-decreasing sequences and these sequences are bounded from above by $2^{|Q^{GD}|}$ and $2^{|Q^{GD}|} \times |\Sigma_o|$.

Finally, we have $S_N = S_{N,n^*}$, $S_T = S_{T,n^*}$.

For $p \geqslant 2$, the design procedure is similar. $S_{N,0}$ and $S_{T,0}$ are designed as above.

Moreover, $S_{R_1}, \cdots, S_{R_p}$ are designed for the p recovery modes. After the recovery events $r_1, \cdots, r_p$ occurs, a set of new supervisors are obtained and merged with $S_{N,0}$ and $S_{T,0}$, respectively. The process is repeated till it terminates.

The discussion of nonblocking property is similar to the fourth approach (to be discussed in the next subsection). The design procedure for the fourth approach is also iterative. Here for brevity, we will only provide an example for the fourth approach.

5.2.2 Fourth Approach

As seen in Fig.5.12, the supervisor design would be the combination of the second and the third approach. The conjunction of S_N and S_T, denoted by $S_{NT} = S_N \wedge S_T$, is used to supervise the system during the normal and transient modes, and S_R is active while the system performs recovery. S_N , S_T and S_R are supervisors with multiple initial states. As in the second approach, $S_N \wedge S_T$ supplies the initial states for S_R. Similarly, upon recovery, S_R provides the initial states for S_N and S_T. The details are given in the following.

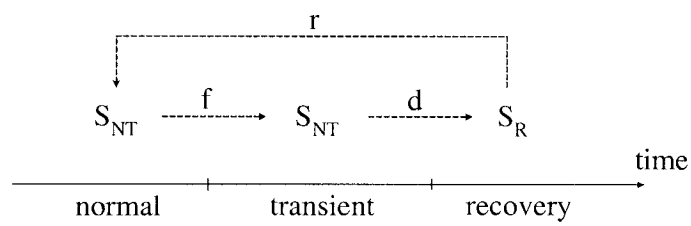


Figure 5.12: Supervisors Involved

The design procedures of S_{NT} and S_R are summarized in Table 5.2. The procedures are as follows.

Table 5.2: Supervisor design steps

Step	S_N	S_T	S_R
0	$S_{N,0}$	$S_{T,0}$	$S_{R,0}$ =merge($S^1_{R,0}$,$\cdots$,$S^{p_0}_{R,0}$)
1	$S_{N,1}$ =merge($S_{N,0}$,$S^1_{N,1}$, $\cdots$, $S^{k_1}_{N,1}$)	$S_{T,1}$ =merge($S_{T,0}$,$S^1_{T,1}$, $\cdots$, $S^{k_1}_{T,1}$)	$S_{R,1}$ =merge($S_{R,0}$,$S^1_{R,1}$, $\cdots$, $S^{p_1}_{R,1}$)
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n^*	S_{N,n^*} =merge(S_{N,n^*-1}, S^1_{N,n^*}, $\cdots$,$S^{k_1}_{N,n^*}$)	S_{T,n^*} =merge(S_{T,n^*-1}, S^1_{T,n^*}, $\cdots$,$S^{k_1}_{T,n^*}$)	S_{R,n^*} =merge(S_{R,n^*-1}, S^1_{R,n^*}, $\cdots$,$S^{p^*_n}_{R,n^*}$)

Step 0: $S_{N,0}$, $S_{T,0}$ and $S_{R,0}$ are designed to enforce E_N, E_T and E_R, respectively, as in the second approach. Note that $S_{R,0}$ is obtained by merging $S^1_{R,0}$, $\cdots$, $S^{p_0}_{R,0}$ which are a set of OBS designed for different initial states of the recovery supervisor. $S_{N,0}$ also satisfies the nonblocking property, i.e. $\mathcal{R}(S_{N,0}/GD_N) \subseteq \mathcal{CR}(S_{N,0}/GD_N)$. Note that in general, recovery ("r" transition) may not be possible from all states $q \in Q^{GD}_R$. Let $Q^{GD}_{R,r} \subseteq Q^{GD}_R$ denote the set of states at which the recovery event r is defined ($\delta(q,r)!$ for $q \in Q^{GD}_{R,r}$). $S_{R,0}$ must be designed such that during recovery, the set $Q^{GD}_{R,r}$ is reachable, in other words, the plant under supervision is nonblocking during recovery and here states in $Q^{GD}_{R,r}$ are considered as the marked states.

Step 1: $S^i_{N,1}$ and $S^i_{T,1}$ are designed following the same procedure explained in step 1 of the third approach based on the initial states supplied by $S_{N,0} \wedge S_{T,0} \wedge S_{R,0}$. Then let $S_{N,1}$ =merge($S_{N,0}, S^1_{N,1}, \cdots, S^{k_1}_{N,1}$), $S_{T,1}$ =merge($S_{T,0}, S^1_{T,1}, \cdots, S^{k_1}_{T,1}$). Once

$S_{N,1}$ and $S_{T,1}$ are obtained, a new set of initial state estimates following the failure detection are determined and used to design $S_{R,1}$; and $S_{R,1}$ is designed based on the same procedure used for $S_{R,0}$ in step 0.

Step n: Repeat the above procedure until no new supervisors can be obtained, i.e. $S_{N,n} = S_{N,n-1}$, $S_{T,n} = S_{T,n-1}$ and $S_{R,n} = S_{R,n-1}$. Termination occurs in a finite number of steps, say n^*, since the number of states and transitions of $S_{N,i}$, $S_{T,i}$ and $S_{R,i}$ in each step form non-decreasing sequences and these sequences are bounded by $2^{|Q^{GD}|}$ (for the number of states) and $2^{|Q^{GD}|} \times |\Sigma_o|$ (for the number of transitions).

Finally $S_N = S_{N,n^*}$, $S_T = S_{T,n^*}$, $S_R = S_{R,n^*}$.

We provide an example to illustrate our design method. The system to be controlled GD is shown in Fig.5.13. The set of events are $\Sigma = \{\alpha, \beta, \gamma, \tau, f, d, r\}$, $\Sigma_c = \{\alpha, \beta, \gamma, r\}$, and $\Sigma_o = \{\alpha, \beta, \gamma, d, r\}$. The plant states in each mode are $Q_N^{GD} = \{0, 1, 2\}$, $Q_T^{GD} = \{4, 5, 6\}$, $Q_R^{GD} = \{7, 8, 9\}$, and $Q_m = \{0, 1, 8, 9\}$. The set of forbidden (unsafe) states are state 1, 6, 9. Then we have $E_N = \{0, 2\}$, $E_T = \{4, 5\}$, $E_R = \{7, 8\}$.

The three OBS supervisors $S_{N,0}$, $S_{T,0}$ and $S_{R,0}$ are shown in Fig.5.14. As shown in Fig.5.14(b), the state estimate supplied to $S_{R,0}$ by $S_{N,0} \wedge S_{T,0}$ is $\{7, 8\}$. Also, $S_{R,0}$ provides state estimate $\{0, 2, 4\}$ for $S_{N,1}$ and $S_{T,1}$. Thus the initial conditions of $S_{N,1}$ and $S_{T,1}$ are $\{0, 2\}$ $(= \{0, 2, 4\} \cap Q_N^{GD})$ and $\{0, 2, 4\}$, respectively. Supervisors $S_{N,1}^1$ and

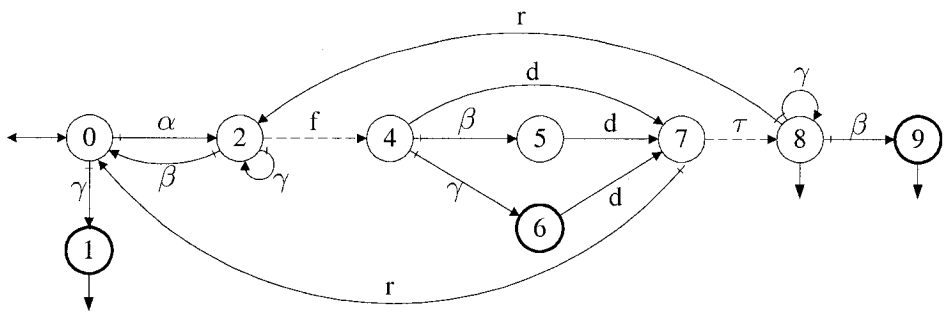
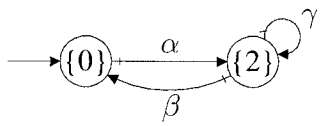
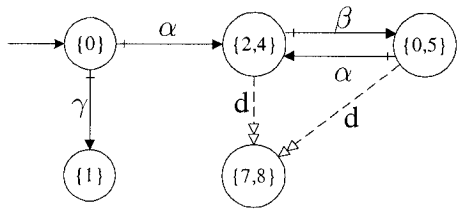


Figure 5.13: GD

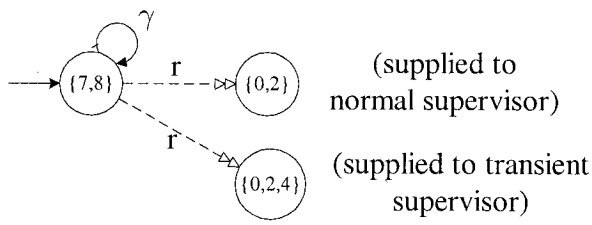
$S^1_{T,1}$ after recovery are displayed in Fig.5.15. After merging, S_N and S_T are obtained as shown in Fig.5.16. Next, $S_{R,1}$ is designed and it turns out that $S_{R,1}$ and $S_{R,0}$ are identical, and therefore the design procedure terminates. Note that S_N, S_T and S_R can have multiple initial states (corresponding to different initial state estimates).



(a) $S_{N,0}$



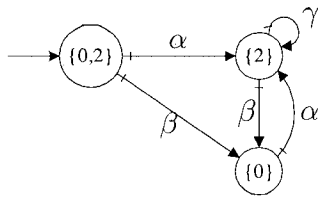
(b) $S_{T,0}$



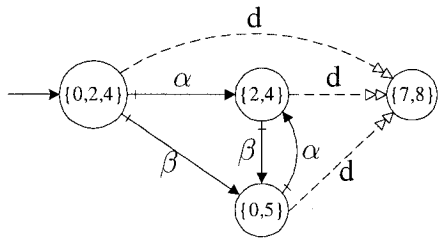
(c) $S_{R,0}$

Figure 5.14: Supervisors $S_{N,0}$,$S_{T,0}$ and $S_{R,0}$

To study the nonblocking property of the system under supervision, similar to



(a) $S^1_{N,1}$

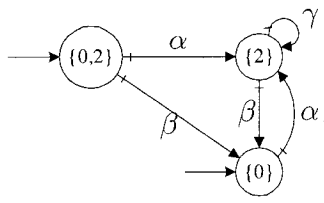


(b) $S^1_{T,1}$

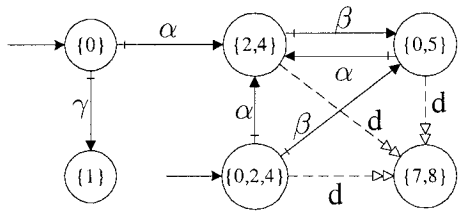
Figure 5.15: OBS supervisors $S^1_{N,1}$ and $S^1_{T,1}$

the method used for the first approach, we find a nonswitching supervisor which in terms of supervisory action is identical to the modular switching supervisor designed in this subsection. Let S_{NM} be obtained from S_N following a procedure similar to that given in the first approach to ensure S_N does not disable uncontrollable events in the transient mode. Let $S_{NMT} = S_{NM} \times S_T$ [1]. We intend to construct a single generator for the equivalent supervisor and this generator is obtained by connecting the states of S_{NMT} and S_R, thus combining their state transition graphs. For a state pair $(\tilde{z}_1, z_1)$ of S_{NMT} and z_1' of S_R, we attach a transition $(\tilde{z}_1, z_1) \xrightarrow{d} z_1'$ if $z_1' = \{q \mid \exists q' \in z_1 \ \& \ \exists\, s, s' \in \Sigma^*_{uo} : \ q = \delta(q', sds')\}$. For a state z_2' of S_R and state pair $(\tilde{z}_2, z_2)$ of S_{NMT}, we add a transition $z_2' \xrightarrow{\tau} (\tilde{z}_2, z_2)$ if $z_2 = \{q \mid \exists q' \in z_2', \ (s, s') \in \Sigma^*_{uo} : \ q = \delta(q', srs')\}$. Thus S_{NMT} and S_R are connected

[1]The product of S_{NM} and S_T has multiple initial states. In our work, it is the state transitions in the product that is our main concern.



(a) S_N = $S_{N,1}$ =merge($S_{N,0}, S^1_{N,1}$)



(b) $S_T = S_{T,1}$ =merge($S_{T,0}, S^1_{T,1}$)

Figure 5.16: Supervisors S_N and S_T after merging

using d and r transitions. We denote the resulting supervisor as S. We take the initial state of S to be $(\tilde{z}_0, z_0)$ where $\tilde{z}_0$, z_0 are the initial states of $S_{N,0}$ and $S_{T,0}$. S can be thought of as the supervisor that controls the system the entire time. S_{NMT} controls the system initially. Once a failure is detected, S_{NMT} is disabled and we switch to S_R. If r is generated, S_{NMT} will be in control. The nonblocking property of GD_N and GD_{NTR} under S can now be easily verified.

The design method can be easily extend to the cases of multiple failure modes. We will not provide the details for brevity. In the following , we discuss the cases when recovery to normal mode and failure accommodation (non recovery to normal) are combined together.

We consider the case when $p = 2$, and recovery to normal mode is possible in the

failure mode F_1 but not in F_2. We assume that the third approach for the design of S_{R_1} and the first approach for the design of S_{R_2} are used. Design based on other combinations of approaches is very similar. If recovery supervisors are in the feedback loop during the normal and transient modes, we assume that as mentioned, S_N and S_T are designed for GD_N and GD_{NT}, respectively, to enforce E_N and E_T. Two recovery supervisors S_{R_1} and S_{R_2} are designed for $GD_{NT_1R_1}$ and $GD_{NT_2R_2}$, respectively. If r_1 occurs, the system can resume its normal operation and S_{R_1} may supply new initial states for S_N and S_T. As in the third approach, new supervisors which are designed to enforce E_N and E_T after recovery are obtained and merged with S_N and S_T, respectively. This process is repeated till S_N and S_T do not change. The conjunction of S_{R_1}, S_{R_2}, and S_N and S_T will be in the feedback loop in the normal and the transient modes, and S_{R_1}, or S_{R_2}, will take sole control in the recovery mode.

5.3 Conclusions

In this chapter, we discussed two scenarios. In the first case, recovery to normal mode is not possible. This includes cases in which the recovery actions are either to shut down the system, or to reconfigure it to function with limited functionality. It may always cover cases, where the faulty component or system has been replaced with a spare part (but the faulty component is still considered part of the system, albeit not active). Two design procedures were provided. The first method covers a larger set of solutions. Thus, in some problems, the second approach may not yield

a solution while the first approach may be applicable. However, the first approach is computationally more complex (both time and space) to implement. The application of these approaches to a small factory will be discussed in the next chapter.

The second category of problems studied were those in which recovery to normal operation is possible. This could be the case where repair or replacement with a spare component is possible. For those problems, two design procedures were provided with the first representing a larger set of solutions, but with the drawback of being more computationally complex (both time and space) to implement.

Chapter 6

Example: A Small Factory

In this chapter, we illustrate our design procedures using a simplified industrial example. Here, all design computations are done by hand. Therefore, we have chosen a simplified problem as opposed to a more complex problem (such as the example of manufacturing cell given in Chapter 4).

6.1 Plant Model of The Small Factory

The small factory consists of a machine and a buffer. An abstraction of the manufacturing process is shown in Fig.6.1.

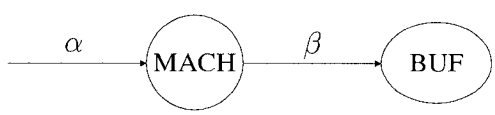


Figure 6.1: Small factory

The machine in the above figure takes a workpiece (event α) from some input device, processes it and then drops it (event β) in the buffer. We assume that there is a sensor which can signal the deposit of a workpiece in the buffer. It is also assumed that as a result of failure, the machine does not drop the finished workpiece in the buffer and thus no deposit is registered by the sensor (i.e., γ is unobservable). The sensor may also fail in which case even if the workpiece is dropped in the buffer, it will not be registered (γ unobservable). Event "f" models the above-mentioned failures. If "f" occurs, the machine may still work, but the exact number of workpieces in the buffer will be unknown. Fig.6.2 shows the finite state machine model of MACH. States I and W correspond to the state of the machine in normal idle and working states respectively, and states I' and W' denote the idle and working state of the machine in the presence of failure. Event r represents the replacement of the failed sensor or corrective measure to ensure proper deposit of workpieces in buffer. In our plant model, $\Sigma_c = \{\alpha, r\}$, and $\Sigma_o = \{\alpha, \beta, r\}$.

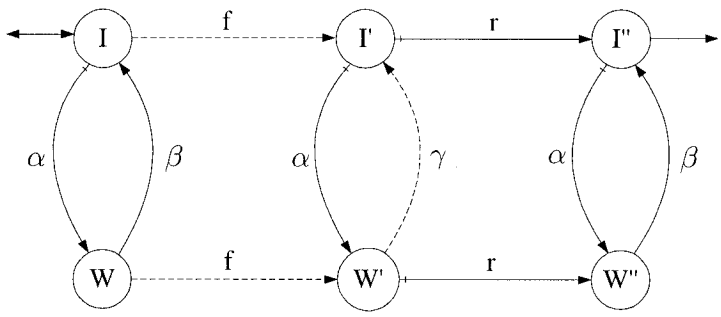


Figure 6.2: MACH: automaton model of the machine

It is assumed that the buffer in our problem has a capacity of 3. Fig.6.3 shows an

automaton to capture the requirement that the number of workpieces in the buffer must not exceed 3. Whenever an event β (deposit) or γ (possible deposit) occurs in the plant, the state of BUF will change. State B represents the forbidden state that BUF will enter when it overflows.

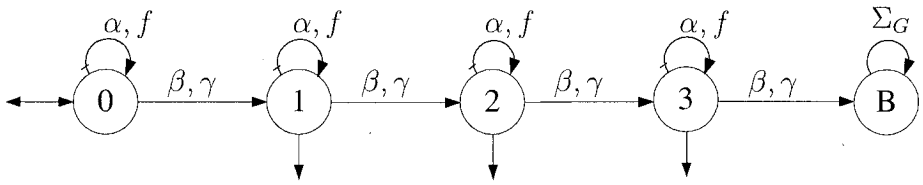


Figure 6.3: BUF: FSM Model of the Buffer

The synchronous product of $MACH$ and BUF, given by G =sync($MACH, BUF$), represents the plant.

6.2 Diagnoser

In this example, we assume that our diagnosis system can detect the failure with a delay of 0 to 3 events. We model the diagnoser using an automaton $D = (Y, \Sigma_D, \delta_D, Y_0, Y_m)$ shown in Fig.6.4.

6.3 System to be Controlled

The system to be controlled in normal, transient and recovery modes is the combined model of G and D, i.e. GD =sync(G, D) (Fig.6.5).

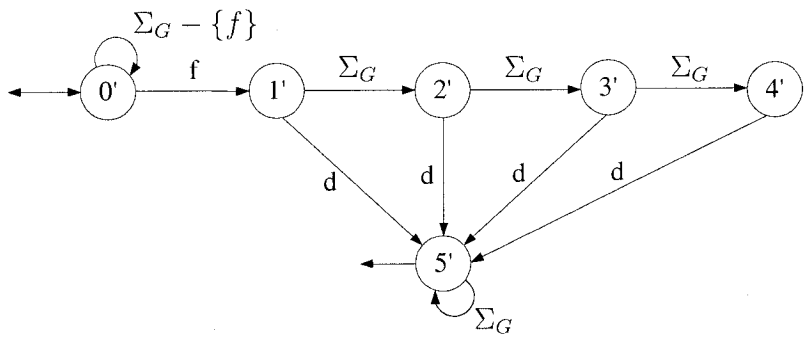


Figure 6.4: Diagnoser model

We allow 2 workpieces in the buffer during the normal mode, and 3 workpieces in the transient and recovery modes. The set of safe states (the specifications in each mode) E_N, E_T and E_R are also shown in the diagram. We want to design an admissible supervisor to enforce the specifications. We would also like the system under supervision to be nonblocking in both the normal and the recovery modes. Solutions based on the first and second approach are given in the next section.

6.4 Controller Design

I. First Approach

According to the design procedure, three OBS supervisors S_N, S_T and S_R are computed. Since the example is simple, the supervisors have been designed intuitively. In cases involving large plants, a computer programme should be developed to implement the systematic design procedures such as the one proposed in section 3.2 based on normal languages. S_N, S_T and S_R are shown in Fig.6.6. To simplify the graphs, the state estimates (associated with supervisor states) are not shown in the

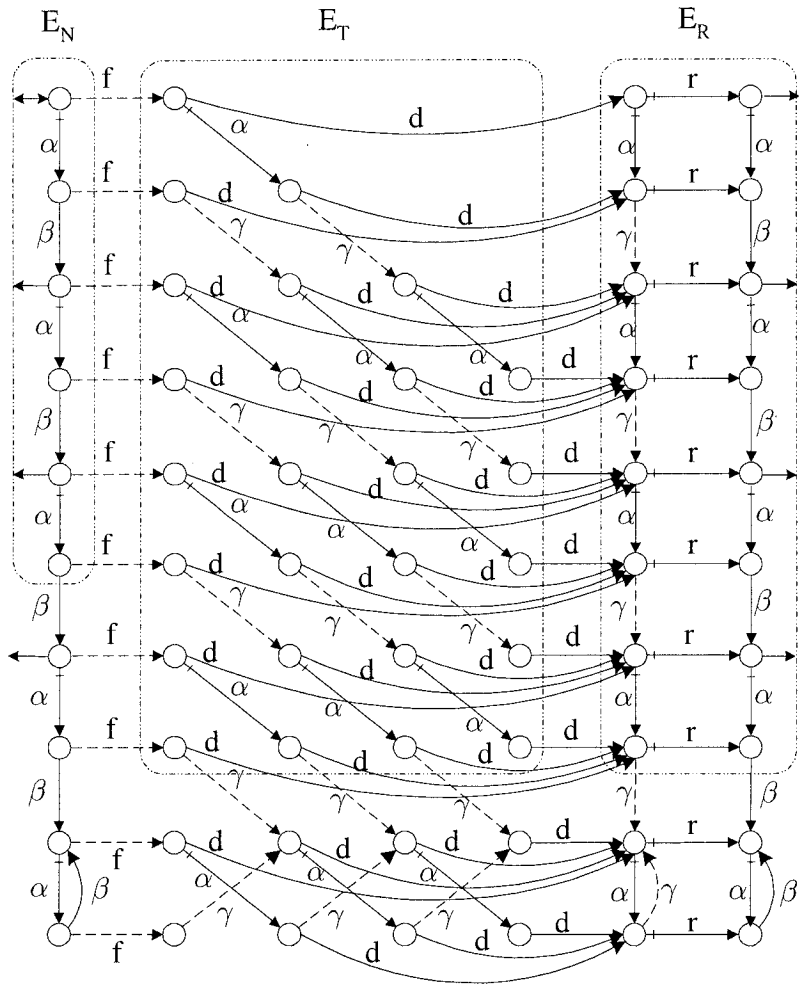


Figure 6.5: The system GD where recovery to normal is impossible

graphs.

Accordingly, the conjunction of S_N, S_T and S_R is used to control the system in both the normal and the transient modes. It can be seen that S_N limits the number of the workpieces in the buffer during the normal operation to two (two pairs of α's and β's are permitted). S_T limits the number of α's to three. This can be justified in the following way. The number of workpieces in the buffer is less than or equal to

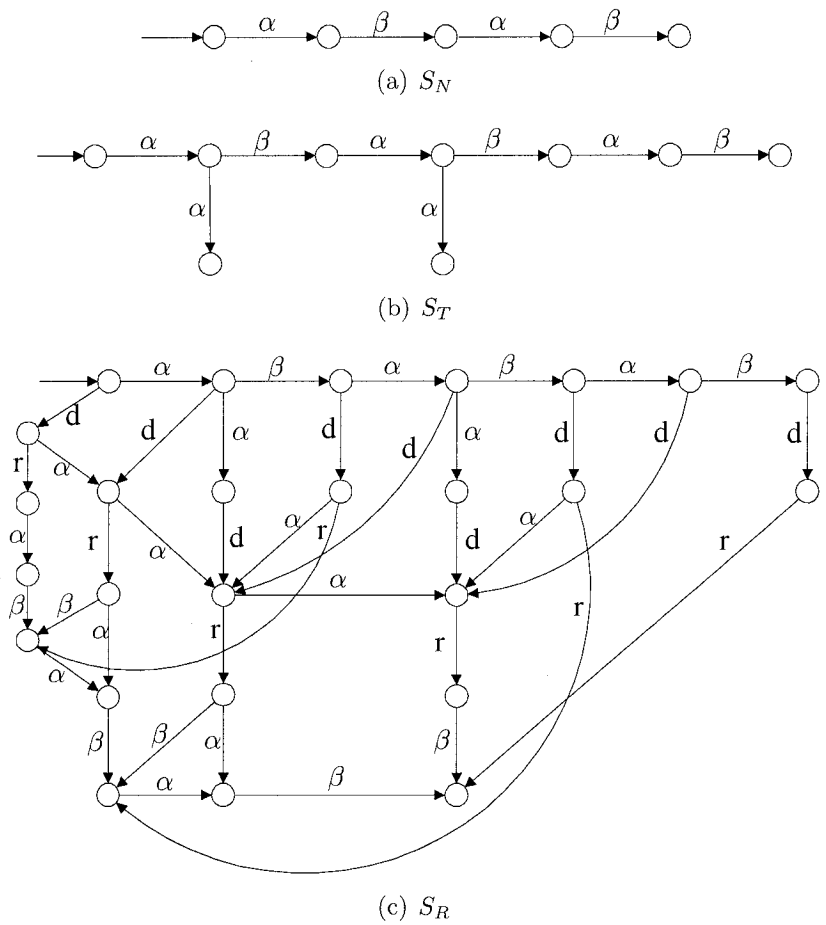


Figure 6.6: Controller S_N, S_T, S_{NT} and S_R

the number of β's plus the number of γ's and thus in turn is less than or equal to the number of α's. In the worst case, the number of workpieces in the buffer could be exactly equal to the number of α's observed. Thus S_T limits the number of α's to three to prevent overflow. S_R behaves similarly by always limiting the number of α's to three. S_R controls the system during recovery. In this case, S_R, during normal and transient modes, does not disable any controllable event that has not been disabled by S_N or S_T, and thus follows the control actions of S_N and S_T. The system under supervision is shown in Fig.6.7. We can check that the system under supervision is

nonblocking in normal and recovery modes.

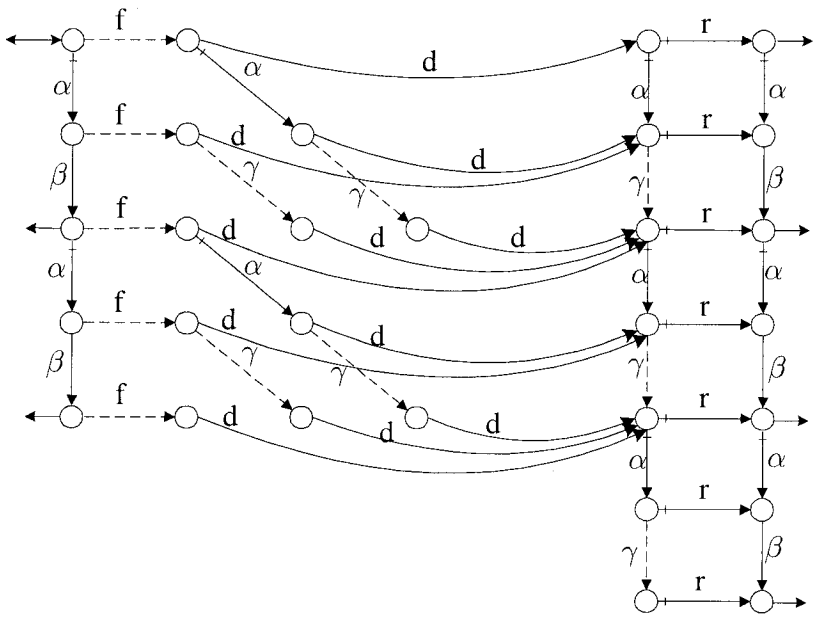


Figure 6.7: The system under supervision

II. Second Approach

In this set up, the supervisor S_N and S_T are the same as the ones we designed in the first approach. $S_N \wedge S_T$ is the supervisor that controls the system during normal and transient modes. The conjunction of S_N and S_T is shown in Fig.6.8 with the initial state estimates supplied to S_R being included.

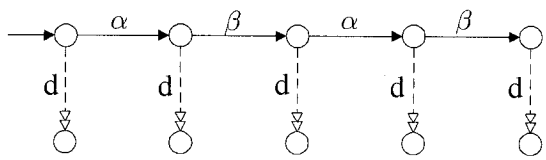


Figure 6.8: meet(S_N, S_T)

S_R is designed according to the procedure explained. The diagram of S_R is shown in Figure6.9. We can see in Fig.6.9 that S_R contains 5 initial states $\{(I', 0, 5')\}$, $\{(W', 0, 5'), (I', 1, 5')\}$, $\{(I', 1, 5')\}$, $\{(W', 1, 5'), (I', 2, 5')\}$ and $\{(I', 2, 5')\}$.

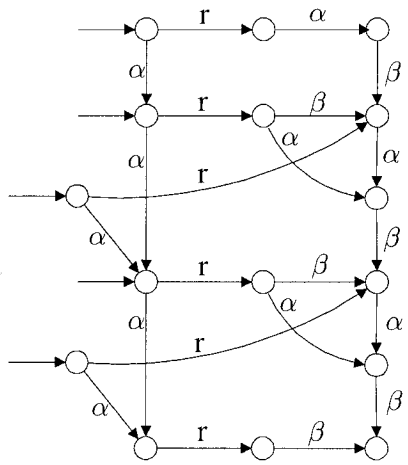


Figure 6.9: S_R

As far as the control action after failure detection is concerned, S_R designed in the second approach performs the same control as S_R designed in the first approach. The only difference is that the first S_R tracks the system's behavior from the beginning, while the second one does not; the second S_R is initialized after failure detection. We can easily verify that the normal plant under the supervision of $S_N \wedge S_T$ is nonblocking, and S_R is also a nonblocking supervisor during recovery.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, fault recovery problem is studied using discrete-event models. We first discuss the design of observer-based supervisors in which the control decisions are made based on the current state estimate rather than the history of events generated by the plant. Then four approaches using observer-based supervisors are proposed to solve the supervisory control problem associated with fault recovery.

We assume that the plant can be modeled as an automaton describing the plant behavior in both normal and faulty modes. The faults are assumed permanent. It is also assumed that the plant is equipped with a diagnoser which can detect and isolate failures with bounded delays. After the fault is detected, a *detection* event is generated to signal the occurrence of a failure to the supervisor so that the supervisor

can implement proper control action to recover from fault. The diagnoser is modeled as an automaton which is an abstract model of the diagnosis system. We assume that the diagnoser can be constructed based on any diagnosis technique, continuous-variable or discrete-event model. Thus the diagnosis and control problems are almost separated and as a result the supervisor design is simplified.

The combined model of the plant and the diagnosis system forms the system to be controlled, which is slightly different from the plant in the traditional control feedback loop. The system to be controlled has three modes: normal, transient and recovery. The set of specifications corresponding to each mode are given in terms of legal states. Following a modular switching supervisory control approach, (observer-based) supervisors are designed to enforce the specifications in each mode. These observer-based supervisors are designed and coordinated in different ways according to the four design procedures. Moreover, the issues of nonblocking property and supervisor admissibility are studied.

The faults can be classified into two categories according to whether recovery to normal mode is possible or not. We develop two approaches in each category. In one approach which covers a larger set of solutions, the recovery supervisor is put in the feedback loop when the system is initialized in its normal mode. In the second approach, however, the recovery supervisor is engaged only when a failure is diagnosed.

Obviously, the second approach has less computational complexity for implementation. The fault recovery problem is first solved in the case of single failure mode, then the result is extended to multiple failure modes. First, we consider systems with all faults belonging to one category, and then systems with faults from both categories are discussed.

The benefit of using modular switching approach is that when a subtask of the overall system is changed, we do not need to modify the whole supervisor. Therefore, easy modification and upgrading are allowed in large systems. Furthermore, modular designs are easier (less computationally complex) to implement.

The supervisor studied in this thesis are observer-based. In this thesis, we have proposed a new method for designing OBS based on normal languages.

7.2 Future Work

The current research can be continued in the following areas.

- In this thesis, we solve the supervisory control problem for fault recovery using untimed discrete-event model. We can extend our recovery framework to timed discrete-event systems.

- The worst-case size of the state set of OBS supervisors is exponential in the

number of open-loop system states. It would be useful to develop a method for minimizing the number of the states of OBS supervisors without changing the supervisory action.

- In this thesis, "single failure scenario" is assumed throughout the design. However, in general, simultaneous failures can occur in DES. Using our design recovery framework, we may solve the control problem involving simultaneous failures. However, the number of supervisor modules increase rapidly with the number of simultaneous faults in a system and the control solution could become very complex. Another systematic way has to be developed to deal with simultaneous failures.

- It would be useful to develop a software to implement the design and analysis procedures for the OBS supervisors so that the recovery procedure developed in this thesis can be applied to large DES.

- In this thesis, we assume that only observable controllable events can be disabled, and unobservable events are all considered to be uncontrollable. However, in general, some unobservable events can be controllable, thus can be disabled by the supervisor. It would be interesting to extend our recovery framework to the cases involving unobservable controllable events.

- In our framework, we assume that faults are permanent. Dealing with transient faults is another challenge.

Bibliography

[1] B. A .Brandin, W. M. Wonham, and B. Benhabib. Manufacturing cell supervisory control - a modular timed discrete-event system approach. *IEEE Trans. Robotics and Automation*, 1:846–851, 1993.

[2] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, 1999.

[3] K. H. Cho and J. T. Lim. Synthesis of fault-tolerant supervisor for automated manufacturing systems: A case study on photolithographic process. *IEEE Trans. on Robotics and Automation*, 14(2):348–351, April 1998.

[4] R. Cieslak, C. Desclaux, A. S. Fawaz, and P. Varaiya. Supervisory control of discrete-event processes with partial observation. *IEEE Trans. on Automatic Control*, 33:249–260, 1988.

[5] D. Gordon and K. Kiriakidis. Reconfigurable robot teams: Modeling and supervisory control. *IEEE Trans. on Control Systems Technology*, 12(5):763–769, Sept. 2004.

[6] M. Heymann and F. Lin. On-line control of partially observed discrete event systems. *Discrete Event Dynamic Systems*, 4:221–236, 1994.

[7] R. Isermann. Supervision, fault-detection and fault-diagnosis methods- an introduction. *Control Engineering Practice*, 5:639–652, 1997.

[8] R. Kumar, V. Garg, and S. I. Marcus. Predicates and predicate transformers for supervisory control of discrete event dynamical systems. *IEEE Trans. on Automatic Control*, 38(2):232–247, Feb. 1993.

[9] Y. Li and W. M. Wonham. Control of vector discrete-event systems I - the base model. *IEEE Trans. on Automatic Control*, 38(8):1214–1227, 1993.

[10] F. Lin. Diagnosability of discrete event systems and its application. *Discrete Event Dynamic Systems*, 4:197–212, 1994.

[11] F. Lin and W. M. Wonham. On observability of discrete event systems. *Information Sciences*, 44:173–198, 1988.

[12] M. Moosaei and S. Hashtrudi Zad. Fault recovery in control systems: A modular discrete-event approach. *Intern. Conf. on Electrical and Electronics Engineering (CINVESTAV / IEEE)*, pages 445–450, 2004.

[13] G. Provan and Y-L Chen. Model-based fault-tolerant control reconfiguration for general network topologies. *IEEE Conf. Control Applications*, pages 473–478, Sept. 2000.

[14] P. J. Ramadge and W. M. Wonham. Modular feedback logic for discrete-event systmes. *SIAM Journal on Control and Optimization*, 25:1202–1218, 1987.

[15] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25:206–230, 1987.

[16] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete event systems. *IEEE Trans. on Automatic Control*, 40:1555–1575, 1995.

[17] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete-event models. *IEEE Trans. Control System Technology*, 4:105–124, 1996.

[18] S. Takai, T. Ushio, and S. Kodama. Static-state feedback control of discrete-event systems under partial observation. *IEEE Trans. on Automatic Control*, 40:1950–1954, 1995.

[19] B. C. Williams, M. D. Ingham, S. H. Chung, and P. H. Elliott. Model-based progamming of intelligent embedded systems and robotic space explorers. *Proc. of the IEEE*, 91:212–237, 2003.

[20] W. M. Wonham. Supervisory control of discrete-event systems. [Online] Available: http://www.control.utoronto.ca/DES.

[21] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete-event systems. *Maths. of Control, Signals and System*, 1:13–30, 1988.

[22] S. Hashtrudi Zad, R. H. Kwong, and W. M. Wonham. Fault diagnosis in discrete-event systems: Framework and model reduction. *IEEE Trans. on Automatic Control*, 48:1199–1212, 2003.