Università degli Studi di Cagliari

Xidian University

PHD DEGREE at the University of Cagliari

Electronic and Computer Engineering

PHD DEGREE at Xidian University

Control Theory and Control Engineering

Cycle XXXIV

TITLE OF THE PHD THESIS

Design of supervisors for active diagnosis in discrete event systems

Scientific Disciplinary Sector(s)

ING-INF/04

PhD Student:	Yihui Hu
Supervisor UniCa	Prof. Alessandro Giua
Supervisor XDU	Prof. Zhiwu Li

Final exam. Academic Year 2020 – 2021

Thesis defence: December 2021 Session

ABSTRACT

In recent years, with the development of computer science and electronic information technology, many large and complex systems have emerged. When a system runs online, it is inevitable to produce some faults. If a fault in a system cannot be detected and treated in time, some serious accidents may occur, resulting in economic and human losses. Therefore, in both academia and industry, fault diagnosis has received considerable attention.

A discrete event system is a discrete-state and event-driven system. In modern society, discrete event systems have wide practical application backgrounds. Many systems can be represented by discrete event systems, such as flexible manufacturing systems, smart urban transportation systems and computer communication networks. In recent years, the problem of fault diagnosis in discrete event systems has received much attention. The problem of fault diagnosis in discrete event systems is to determine if some faults have occurred by observing events generated by a plant. On the other hand, diagnosability is a property of a plant such that any fault can be detected within finite future steps after its occurrence. In practice, a plant should necessarily be diagnosable since any fault that has occurred must be detected and be repaired in time to guarantee its safety and reliability. This thesis considers two discrete event modeling formalisms and studies a problem of active diagnosis: it consists in modifying an undiagnosable plant's behavior by supervisory control, thus ensuring that the closed-loop system is diagnosable. The main results of this thesis are summarized as follows.

1. To reduce the complexity of designing an active diagnosis supervisor, we develop a method for active diagnosis based on the verifier structure of a plant. The existing methods for active diagnosis in the literature are based on the diagnoser of a plant whose computation may suffer from the state explosion problem, since the structure complexity of the diagnoser is exponential with respect to the number of states in the plant. Verifier is a structure which has been widely used to test the diagnosability of a plant. Its structure complexity is polynomial with respect to the number of states in the plant. This thesis proposes an active diagnosis supervisor based on the verifier of a plant. Compared with the conventional methods in the literature, the developed approach has the computational advantage.

2. The active diagnosis problem in Petri net model is formulated and studied. Petri net is a graphical and mathematical modeling tool with a higher modeling power than finite

state automata. However, there is no work to deal with the active diagnosis problem in Petri nets as far as we know. An alternative approach consists in computing the reachability graph of a plant and using the graph to design a supervisor by means of the automaton-based algorithms. However, such a method is rather inefficient since it requires a full enumeration of the reachability space of a net. This thesis uses an integer linear programming technique and develops a structure called quiescent basis reachability graph (QBRG) to describe the behavior of a plant without explicitly listing all reachable markings. Based on the QBRG structure, an algorithm is proposed to design an active diagnosis supervisor for a give plant net. Moreover, the supervisor designed guarantees that the closed-loop system is deadlock-free if no fault occurs.

3. The existing fault diagnosis techniques for discrete event systems requires to initialize and run a diagnoser synchronously with a plant. However, such a condition may not be easy to meet in some cases. Therefore, a notion called asynchronous diagnosis is proposed. Asynchronous diagnosis is to detect the occurrence of faults in a plant under the condition that a diagnostic agent is activated asynchronously with the plant. On the other hand, asynchronous diagnosability is the property of a plant such that the occurrence of any fault can be detected after a finite number of observations in the case of asynchronous activation of the diagnostic agent and the plant. This thesis develops a supervisor for an asynchronous undiagnosable plant such that the closed-loop system is asynchronous diagnosable.

Keywords: Discrete event system, Fault diagnosis, Supervisory control theory, Active diagnosis, Asynchronous diagnosis

List of Figures

2.1 A deterministic finite-state automaton. 7

2.2 A Petri net system. 9

3.1 The active diagnosis scheme. 20

3.2 (a) Plant A for Example 3.1, (b) diagnoser of A, (c) diagnoser of the closed-loop system $(A, S')^{live}$, and (d) diagnoser of the closed-loop system $(A, S'')^{live}$. 22

3.3 (a) Automaton A for Example 3.2, (b) the closed-loop system (A, S_1), and (c) the closed-loop system (A, S_2). 23

3.4 (a) Automaton A for Example 3.3, (b) the diagnoser of A ($e'_k \neq e''_k$), (c) the diagnoser of A ($e'_k = e''_k$). 24

3.5 Automaton A for Example 3.4. 32

3.6 The verifier of automaton A in Figure 3.5. 33

3.7 Automaton A_{SC}. 35

3.8 The closed-loop system (A, S) for Example 3.6. 40

3.9 Automaton A for Example 3.7. 40

3.10 The verifier (partly) of automaton A in Figure 3.9. 40

4.1 The active diagnosis scheme. 44

4.2 (a) The LPN used in Example 4.1, and (b) its reachability graph after adding the quiescent event q. 47

4.3 (a) The LPN for Example 4.2, (b) its diagnostic BRG, (c) the underlying automaton of the BRG, and (d) the diagnoser of G_l. 50

4.4 (a) The structure G_l for Example 4.3, and (b) the corresponding QBRG G_q. 54

4.5 The Q-diagnoser A_d of the plant in Figure 4.2(a). 55

4.6 (a) The trimmed structure A'_d in Example 4.6, (b) the trimmed structure A''_d, and (c) the diagnosability enforcing supervisor S. 61

5.1 (a) Automaton A for Example 5.1, and (b) its quiescence automaton A_q.... 69

5.2 The asynchronous-quiescent diagnoser A_d. 72

5.3 Supervisor S for Example 5.4. 77

List of Tables

2.1 The transition function of the DFA in Fig. 2.1. 7

4.1 The markings of the net in Figure 4.2(b). 47
4.2 The basis markings of the BRG in Figure 4.3(b). 51
4.3 The minimal explanation vectors of the BRG in Figure 4.3(b). 51

List of Symbols

$\mathbb{N}$	The set of non-negative integers
A	An automaton $A = (X, \Sigma, \delta, x_0)$
X	The set of states of an automaton
E	The set of events of an automaton
E^*	The set of all finite sequences defined over E
E_o	The set of observable events
E_{uo}	The set of unobservable events
E_c	The set of controllable events
E_{uc}	The set of uncontrollable events
δ	The transition function of an automaton
x_0	The initial state of an automaton
$L(A)$	The generated language of automaton A
A_d	The diagnoser of an automaton
A_v	The verifier of an automaton
PN	A Petri net $PN = (P, T, Pre, Post)$
P	The set of places of a Petri net
T	The set of transitions of a Petri net
Pre	The $pre-$ incidence matrix of a Petri net
$Post$	The $post-$ incidence matrix of a Petri net
C	The incidence matrix of a Petri net
M	A marking of a Petri net
M_0	The initial marking of a Petri net system
$\langle PN, M_0 \rangle$	A Petri net system
$R(PN, M_0)$	The reachability set of a Petri net $\langle PN, M_0 \rangle$
$L(PN, M_0)$	The set of transition sequences enabled at M_0
G	A labeled Petri net $G = (PN, M_0, \Sigma, l)$
Σ	The set of labels of a labeled Petri net
l	The labeling function of a labeled Petri net
T^*	The set of all finite sequences defined over T
$\|\sigma\|$	The length of sequence σ

T_E	The set of explicit transitions
T_I	The set of implicit transitions
$R_I(M_b)$	The implicit reach of marking M_b
$\mathcal{B}$	The basis reachability graph of a Petri net
G_q	The quiescent basis reachability graph of a labeled Petri net
S	A supervisor

List of Abbreviations

DES	Discrete Event System
DFA	Deterministic Finite Automaton
PN	Petri Net
ILPP	Integer Linear Programming Problem
LPN	labeled Petri Net
RG	Reachability Graph
BRG	Basis Reachability Graph
QBRG	Quiescent Basis Reachability Graph
BRD	Basis Reachability Diagnoser

Content

摘要 I

ABSTRACT III

List of Figures V

List of Tables VII

List of Symbols IX

List of Abbreviations XI

Chapter 1 Introduction 1

1.1 Fault Diagnosis and Diagnosability Verification in DESs 1

1.2 Active Diagnosis Problem 3

1.3 Organization and Contributions 4

1.3.1 Organization 4

1.3.2 Contributions 5

Chapter 2 Preliminaries 7

2.1 Automaton 7

2.2 Petri Net 9

2.3 Supervisory Control 12

Chapter 3 Active Diagnosis Problem in the Framework of Automata 15

3.1 Diagnosability Verification in Automata 15

3.1.1 Diagnoser 16

3.1.2 Verifier 18

3.2 Active Diagnosis Formulation 19

3.2.1 Diagnoser-based Approach for Active Diagnosis in Automata 21

3.2.2 Properties of Supervisor-induced Deadlocks 23

3.3 $Stop$-free Control Policy and Properties 25

3.3.1 $Stop$-free Event Set 25

3.3.2 Computing a Feasible $Stop$-free Event Set 28

3.3.3 Complexity Analysis 34

3.4 Online Active Diagnosis Method 35

3.5 Conclusion 41

Chapter 4 Active Diagnosis Problem in Labeled Petri Nets 43

4.1 Active Diagnosis Problem Formulation in Labeled Petri Nets 43

4.1.1 Diagnosability of LPNs with Deadlocks 45
4.1.2 Quiescent Behavior and Quiescent Event 46
4.2 Basis Reachability Graph with Quiescence and Q-diagnosers 47
4.2.1 Diagnosability of Deadlock-free LPNs and Basis Diagnosers 47
4.2.2 Quiescent Basis Reachability Graph and Q-diagnoser 49
4.3 Diagnosability Enforcing Supervisor 55
4.4 Conclusion 63
Chapter 5 Asynchronous Diagnosability Enforcement in Discrete Event Systems Based on Supervisory Control Theory 65
5.1 Motivation 65
5.2 Problem Formulation 66
5.2.1 Asynchronous Diagnosis and Diagnosability 66
5.2.2 Asynchronous Diagnosability of Nonlive Plants 67
5.2.3 Observable Quiescence 68
5.3 Asynchronous Diagnosability Verification of Nonlive Plants 70
5.4 Asynchronous Diagnosability Enforcement Supervisor 74
5.5 Conclusion 78
Chapter 6 Conclusions and Future Works 79
6.1 Conclusions 79
6.2 Future Works 80
References 81
Acknowledgement 89
Biography 91

Chapter 1 Introduction

A discrete event system (DES) is a dynamic system whose state evolution is driven by discrete events. Such systems have become prevalent in the past three decades, primarily due to the proliferation of digital technologies, interconnectivity, and sensor technology. These developments have led to the emergence of many highly complex systems. Examples include automated manufacturing systems, communication networks, traffic control and transportation systems, autonomous vehicles, and others. In practice, almost all large and complex systems are inevitably subject to some physical faults, which can deviate the performance of the system or even cause a breakdown. Therefore, fault diagnosis techniques in discrete event systems have received much attention in recent years.

In discrete event systems, a fault is represented as an unobservable event. The problem of fault diagnosis is to determine if some fault has occurred by observing the events generated by a plant. On the other hand, diagnosability is an important property of a plant such that any fault can be detected within finite future steps. In practice, a plant is expected to be diagnosable to guarantee its safety and reliability. Therefore, for an undiagnosable plant, some corrective actions should be done to guarantee its diagnosability, which is the so-called diagnosability enforcement.

It is well known that there are two common mathematical tools to model discrete event systems such as automata and Petri nets. In this thesis, for an undiagnosable plant modeled by an automaton or a Petri net, we design a supervisor for the plant such that the closed-loop system is diagnosable. This chapter first presents an overview about the fault diagnosis and diagnosability verification in discrete event systems. Then the motivation of our work is provided. Finally, the structure of the thesis and the contributions are summarized.

1.1 Fault Diagnosis and Diagnosability Verification in DESs

Fault diagnosis for discrete event systems (DESs) [1] has been extensively studied in recent years. The problem of fault diagnosis [2–5] is to determine if some faults have occurred by observing events generated by a plant. In [3], a diagnostic agent called diagnoser is developed for online diagnosis. Each diagnoser state provides a possible diagnosis information of the plant and updating this diagnosis information only requires the occurrence of an observable event. Later, the fault diagnosis technique is also studied in decentralized structure where two or more observation sites (with distinct observation capabilities) monitor activity that occurs in a given system, and their goals is to perform fault diagnosis by exchanging information among a coordinator [6–10].

On the other hand, diagnosability [3] is a property of a plant such that any fault can be detected within finite future steps after its occurrence. In the literature, there are many results on the verification of diagnosability in the framework of automata. Sampath *et al.* [3] verify the diagnosability by using the diagnoser structure and show that a plant is diagnosable if and only if there does not exist any indeterminate cycle in its diagnoser. Although a diagnoser can be used to both verify the diagnosability and perform the online fault diagnosis, its structure complexity is exponential with respect to the number of states of a plant. Therefore, a structure called verifier [11, 12] is proposed to test diagnosability. The number of states in a verifier is polynomial with respect to the number of states of the plant. Based on the verifier structure and its variants, various notions of diagnosability under different settings, such as decentralized diagnosability [13], robust diagnosability subject to intermittent loss of observations [14], and transition-based codiagnosability [15], can be verified.

Petri net is a graphical and mathematical tool with a higher modeling power than finite state automata. In Petri nets, structural analysis and abstraction techniques can be used to reduce the computation complexity of analysis and control [16–18]. Therefore, many researchers focus on the diagnosis of Petri nets [19–26]. In [19], the authors develop a basis-marking-based approach for diagnosis. By means of basis markings and the corresponding justifications, the enumeration of paths in a reachability graph can be avoided. The authors in [21–23] extend this approach to the case of labeled Petri nets, i.e., nets where two or more transitions can share the same label. They develop a basis reachability graph (BRG), which can be built off-line and provides an efficient method for online diagnosis. The study in [27] defines a new type of marking with negative elements, called g-marking, and based on which proposes an online diagnosis algorithm. In [24], an online diagnosis approach based on integer linear programming technique is proposed. The study in [28] addresses the issue of fault diagnosis using a partially observed Petri net (POPN), where a POPN is first converted into a labeled net and then an algorithm is reported based on the reachability graph.

Moreover, many works have been done on the verification of diagnosability in the framework of Petri nets [29–36]. Basile *et al.* [29, 30] present a necessary and sufficient condition for diagnosability by solving an integer linear programming problem. In [32], Cabasino *et al.* introduce modified basis reachability graph and basis reachability diagnoser to verify diagnosability of a net without enumerating all reachable markings. Similarly, Jiroveanu *et al.* [33] propose an automaton called ROF-automaton, which provides a compact representation of the state space. Cabasino *et al.* [31] show that the diagnosability of an unbounded Petri net can be determined by analyzing the coverability graph of the so-called verifier net. Recently, Ran *et al.* [37] also explore diagnosability verification in decentralized Petri net models. A different model is considered by Ramírez-Treviño *et al.* in [35, 38], where the diagnosability problem in interpreted Petri nets (IPNs) is studied. In their framework, the considered plant net has an output function that associates an output vector to each marking. In [38], the authors introduce a notion called input-output diagnosability and provide sufficient structural conditions to verify it on the premise that any T-semiflow must contain all risky transitions. This work is further extended in [35] where a concept called relative distance is used to present a new characterization providing sufficient conditions for diagnosability, and polynomial algorithms are proposed to determine the diagnosability.

1.2 Active Diagnosis Problem

In practice, a plant is usually expected to be diagnosable since any fault that has occurred must be detected and be repaired in time to guarantee its safety and reliability. If a plant is diagnosable, a diagnostic agent can be designed to perform the online diagnosis. Otherwise the plant must be particularly treated to guarantee its diagnosability, which is the diagnosability enforcement.

For an undiagnosable plant, one approach for enforcing diagnosability is to modify its physical structure or to add sensors to change its observation structure. In the framework of automata, Cassez *et al.* [39] present a dynamic strategy for sensor activation to guarantee diagnosability. Moreover, Wang *et al.* [40] propose a sensor activation policy to enforce diagnosability with a minimal cost. This problem is also studied in stochastic automata [41]. In Petri nets, Basile *et al.* [42] develop an integer linear programming to find a minimal set of sensors that makes a net system k-diagnosable. Cabasino *et al.* [43] introduce a new labeling function making a system diagnosable using verifier net. To circumvent the state explosion problem that the method in [43] may potentially encounter, Ran *et al.* [44] develop a new structure called the unfolded verifier, and determine a new labeling function to enforce the diagnosability with a minimal cost.

Although the aforementioned methods in the literature can be used to enhance diagnosability, they are practically intrusive since new sensors have to be physically deployed, which may not be possible due to technical and/or financial difficulties. Therefore, many researchers focus on another kind of approach called active diagnosis [45–53]. In an active diagnosis scheme, a supervisor is designed to forbid all undiagnosable evolutions of a plant, thus ensuring that the closed-loop system remains diagnosable. In [45] the diagnosability of a plant is guaranteed by preventing it from cycling in the indeterminate cycles of the diagnoser. In this method the diagnoser is first computed and then iteratively trimmed such that any indeterminate cycle can only be consecutively visited finite times. Extended from [45] several other works are reported. The work in [46] computes an admissible sequence of actions that refine the diagnosis without radically changing the mission of the plant. In [47] active diagnosis is formulated as a Büchi game, where a winning strategy yields an active diagnoser. The authors in [48] study the problem under the assumption that the system can observe quiescence and formulate the problem as a Büchi game. The work in [54] and [55] deals with active fault diagnosis in deterministic input/output automata and probabilistic systems, respectively. In [49], the authors propose a supervisor based on all-enforcement-structure (AES), an automaton similar to the observer automaton, in which all possible control decisions are enumerated. The size of the AES is also exponential with respect to the size of a plant. Although the above contributions can be used to enhance diagnosability, they are all based on some observer-like structures whose computation suffers from state explosion. Therefore, it is necessary to propose an active diagnosis approach with lower structure complexity.

1.3 Organization and Contributions

1.3.1 Organization

The remainder of the thesis is organized as follows.

Chapter 2 recalls some basic notions of automata and Petri nets.

Chapter 3 studies the active diagnosis in the framework of automata. A verifier-based approach for active diagnosis is presented in this chapter. Note that although the verifier can be used to efficiently test diagnosability, it is not straightforward in designing a supervisor such that the closed-loop system is diagnosable. As far as we know, this is the first work on the active diagnosis whose structural complexity of the control structure is polynomial with respect to the number of states of the plant. Therefore, the proposed approach has the computational advantage than the diagnoser-based ones.

Chapter 4 formulates the active diagnosis in labeled Petri nets. As far as we know, it is the first work in the literature to study the active diagnosis in labeled Petri nets (LPNs).

Although the active diagnosis problem in LPNs can be solved by computing the reachability graph of a plant and using the graph to design a supervisor by means of the automaton-based algorithms, such a method is rather inefficient since it requires a full enumeration of the reachability space of a net. This chapter proposes a state abstraction technique to design an active diagnosis supervisor such that the enumeration of the reachability space is avoided.

Chapter 5 studies the problem of enforcing the asynchronous diagnosability. Asynchronous diagnosability is the property of a plant such that the occurrence of any fault can be detected after a finite number of observations under the condition that a diagnostic agent may be asynchronously activated with the plant. As far as we know, the approach presented in this chapter is the first work to deal with this problem in the literature.

Finally, conclusions and some possible future works are presented in Chapter 6.

1.3.2 Contributions

In this thesis, three contributions are presented in Chapters 3-5. The contributions are summarized as follows:

In Chapter 3, An active diagnosis supervisor based on the verifier structure of a plant is developed. The existing methods for active diagnosis in the literature are based on the diagnoser of a plant whose computation may suffer from the state explosion problem, since the structure complexity of the diagnoser is exponential with respect to the number of states in the plant. Verifier is a structure which has been widely used to test the diagnosability of a plant. Its structure complexity is polynomial with respect to the number of states in the plant. This chapter proposes an active diagnosis supervisor based on the verifier of a plant. Compared with the conventional methods in the literature, the developed approach has the computational advantage.

In Chapter 4, the active diagnosis problem in labeled Petri nets is formulated. Petri net is a graphical and mathematical modeling tool with a higher modeling power than finite state automata. However, there is no work to deal with the active diagnosis problem in Petri nets as far as we know. This chapter uses an integer linear programming technique and develops a structure called quiescent basis reachability graph (QBRG) to describe the behavior of a plant without explicitly listing all reachable markings. Based on the QBRG structure, an algorithm is proposed to design an active diagnosis supervisor for a give plant net. Moreover, the supervisor designed guarantees that the closed-loop system is deadlock-free if no fault occurs.

In Chapter 5, the active diagnosis problem under the condition that a diagnostic agent may be asynchronously activated with the plant is studied. The existing fault diagnosis techniques for discrete event systems require to initialize and run a diagnoser synchronously with a plant. However, such a condition may not be easy to meet in some cases [56]. Therefore, a notion called asynchronous diagnosis is proposed. Asynchronous diagnosability is the property of a plant such that the occurrence of any fault can be detected after a finite number of observations in the case of asynchronous activation of the diagnostic agent and the plant. This chapter develops a supervisor for an asynchronous undiagnosable plant such that the closed-loop system is asynchronous diagnosable.

Chapter 2 Preliminaries

In this chapter, we recall some basics of automata and Petri nets.

2.1 Automaton

A deterministic finite-state automaton (DFA) is a four-tuple structure $A = (X, E, \delta, x_0)$, where:

- X is the set of states;
- E is the set of events;
- $\delta : X \times \Sigma \to X$ is the partial state transition function;
- x_0 is the initial state.

Example 2.1. *Figure 2.1 shows a DFA with $X = \{0, 1, 2, 3\}$, alphabet $E = \{a, b, c\}$, and initial state $x_0 = 0$. The transition function is given by Table 2.1.* □

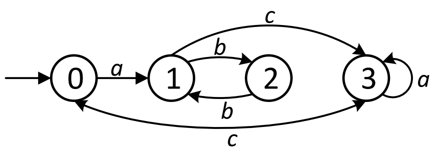


Fig. 2.1 A deterministic finite-state automaton.

Table 2.1 The transition function of the DFA in Fig. 2.1.

δ	a	b	c
0	1	–	3
1	–	2	3
2	–	1	–
3	3	–	0

We use $\Gamma(x)$ to denote the set of events that are feasible at state x. A state $x \in X$ is a *dead state* if $\Gamma(x) = \emptyset$. A plant A is *live* if it does not contain dead states.

The event set E can be partitioned into the set of observable events E_o and the set of unobservable events E_{uo}, i.e., $E = E_o \cup E_{uo}$. The unobservable event set can be further partitioned into the set of regular unobservable events E_{reg} and the set of fault events E_f, i.e., $E_{uo} = E_{reg} \cup E_f$. According to different fault types, we have $E_f = \{f_1, f_2, ..., f_n\}$.

Let E^* (the Kleene closure of E) denote the set of all finite sequences over E, including the empty sequence ε. Let $\sigma \in E^*$ be an event sequence, and we use $|\sigma|$ to denote the length of σ. The transition function δ is extended to $\delta^* : X \times E^* \to X$ by recursively defining $\delta(x, \varepsilon) = x$ and $\delta(x, \sigma e) = \delta(\delta(x, \sigma), e)$, where $\sigma \in E^*$ and $e \in E$.

The set of sequences that can be generated by plant A from state x is denoted by $L(A, x) = \{\sigma \in E^* \mid \delta(x, \sigma)!\}$ where "!" means "is defined". The generated language of A is defined as $L(A) = L(A, x_0) = \{\sigma \in E^* \mid \delta(x_0, \sigma)!\}$. Given a sequence $\sigma \in L(A)$, we use $L(A)/\sigma$ to denote the set of sequences that occurs after σ, i.e., $L(A)/\sigma = \{\sigma' \in E^* \mid \sigma\sigma' \in L(A)\}$. A sequence $\sigma \in L(A)$ is *terminal* if $L(A)/\sigma = \emptyset$. The extension closure of the language $L(A)$, denote by $ext(L(A))$, is defined as $ext(L(A)) = \{\sigma' \in E^* \mid \exists \sigma \in L(A), \sigma\sigma' \in L(A)\}$.

The *prefix closure* of a language $L \subseteq E^*$ is the set $\overline{L} = \{\sigma \in E^* \mid \exists \sigma' \in E^*, \sigma\sigma' \in L\}$. A language $L \subseteq E^*$ is *prefix-closed* if $L = \overline{L}$.

Given a sequence $\sigma \in E^*$, the observation mask is the natural projection $\mathcal{P} : E^* \to E_o^*$, which is defined as:

$$\begin{cases} \mathcal{P}(\varepsilon) = \varepsilon; \\ \mathcal{P}(e) = e, \ if \ e \in E_o; \\ \mathcal{P}(e) = \varepsilon, \ if \ e \in E_{uo}; \\ \mathcal{P}(\sigma e) = \mathcal{P}(\sigma)\mathcal{P}(e), \sigma \in E^*, e \in E. \end{cases} \tag{2-1}$$

If a sequence $\sigma = e_1 e_2 \ldots e_k \in L(A)$ is executed in a plant A, through the observation mask $\mathcal{P}$ one observes the *word* $w = \mathcal{P}(\sigma) = \mathcal{P}(e_1)\mathcal{P}(e_2) \ldots \mathcal{P}(e_k)$. We use $L_o(A)$ to denote the observed language of A, i.e., $L_o(A) = \{\mathcal{P}(\sigma) \mid \sigma \in L(A)\}$. The inverse projection $\mathcal{P}^{-1} : E_o^* \to 2^{E^*}$ is defined as: $\mathcal{P}^{-1}(w) = \{\sigma \in L(A) \mid \mathcal{P}(\sigma) = w\}$, i.e., $\mathcal{P}^{-1}(w)$ consists of all sequences in $L(A)$ whose observations are w.

For an automaton plant $A = (X, E, \delta, x_0)$, we define the unobservable reach [57, 58] of a state $x \in X$ as $UR(x) = \{x' \in X \mid \exists \sigma \in E_{uo}^*, \delta(x, \sigma) = x'\}$, i.e., $UR(x)$ is the set of states that are reachable from state x via sequences consisting of unobservable events only. Given a state $x \in X$ and an observable event $e \in E_o$, we define $U_e(x) = \{\sigma \in E_{uo}^* \mid e\sigma \in L(A, x)\}$, i.e., the set of unobservable sequences σ such that $e\sigma$ is generated from state x.

For an automaton plant $A = (X, E, \delta, x_0)$, we say that $x_1 \xrightarrow{e_1} x_2 \xrightarrow{e_2} ... \xrightarrow{e_{n-1}} x_n$ (where $x_i \in X$ and $e_i \in E$) is a *path* if $\delta(x_i, e_i) = x_{i+1}$ for all $i \in \{1, 2, ..., n-1\}$. A *cycle* is path with $x_1 = x_n$.

2.2 Petri Net

A Petri net is a four-tuple $PN = (P, T, Pre, Post)$, where P is a set of m places represented by circles and T is a set of n transitions represented by bars; $Pre : P \times T \to \mathbb{N}$ and $Post : P \times T \to \mathbb{N}$ are the *pre*- and *post*- incidence matrices which specify the arcs from places to transitions and from transitions to places, respectively. Here, $\mathbb{N}$ is the set of non-negative integers. $C = Post - Pre \in \mathbb{N}^{m\times n}$ is the incidence matrix of the net. For a transition $t \in T$, the preset of t is defined as ${}^\bullet t = \{p \in P \mid Pre(p,t) > 0\}$, while the postset of t is defined as $t^\bullet = \{p \in P \mid Post(p,t) > 0\}$.

A *marking* of a Petri net is a function M: $P \to \mathbb{N}$, which assigns to each place a non-negative integer number of *tokens*, represented by black dots. A Petri net with an initial marking M_0 is called a Petri net system and is denoted by $\langle PN, M_0\rangle$.

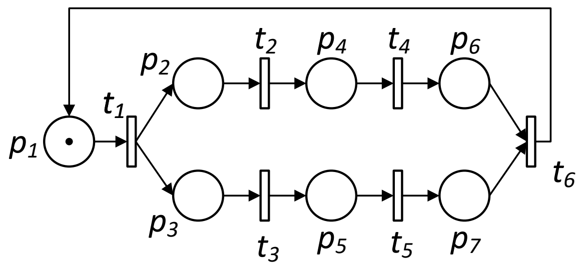


Fig. 2.2 A Petri net system.

Example 2.2. *Fig. 2.2 presents a Petri net system with* $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$ *and* $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$. *The initial marking* $M_0 = [1, 0, 0, 0, 0, 0, 0]^T$. *The* $Post$ *and* Pre *incidence matrices of the net are as follows:*

$$Pre = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \qquad Post = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Accordingly, the incidence matrix can be computed as follows:

$$C = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix}.$$

A transition t is *enabled* at a marking M if $M \geq Pre(\cdot, t)$ holds, which is denoted by $M[t\rangle$. At a marking M, an enabled transition t may fire reaching a new marking $M' = M + C(\cdot, t)$, which is denoted by $M[t\rangle M'$. We use $t \in \sigma$ to denote that transition t appears at least once in a sequence $\sigma \in T^*$. We write $M[\sigma\rangle M'$ with $\sigma = t_1 \cdots t_k$ to denote that at marking M transitions $t_1, \cdots, t_k$ can fire sequentially, which eventually yields marking M'. We say that marking M' is *reachable* from marking M if there exists a firing sequence $\sigma \in T^*$ such that $M[\sigma\rangle M'$. We use $L(PN, M_0)$ to represent the set of all sequences that are enabled at the initial marking M_0, i.e., $L(PN, M_0) = \{\sigma \in T^* \mid M_0[\sigma\rangle\}$.

The set of all markings that are reachable from M_0 is the *reachability set*, denoted by $R(PN, M_0)$. A marked net $\langle PN, M_0\rangle$ is *bounded* if there exists a number $k \in \mathbb{N}$ such that for all $M \in R(PN, M_0)$, and all $p \in P$, $M(p) \leq k$ holds.

A marking M is said to be *dead* if no transition is enabled at M. A sequence $\sigma \in L(PN, M_0)$ is *terminal* if the firing of σ reaches a dead marking M. A marked net $\langle PN, M_0\rangle$ is said to be *deadlock-free* if for all $M \in R(PN, M_0)$, M is not dead.

For a sequence $\sigma \in T^*$, we use $\sigma_{\uparrow T'}$ with $T' \subseteq T$ to denote the projection of sequence σ onto the transition set T', and we write $\mathbf{y}_\sigma$ to denote the firing vector of σ, i.e., $\mathbf{y}_\sigma(t) = k$ if transition t appears k times in σ.

A Petri net is *acyclic* if it does not contain any cycle. For an acyclic net, the following result holds [59, 60].

Proposition 2.1. *Given an acyclic Petri net, a marking M is reachable from M_0 if and only if there exists a vector $\mathbf{y} \in \mathbb{N}^n$ satisfying the* state equation $M = M_0 + C \cdot \mathbf{y}$. □

A *labeled Petri net* (LPN) is a structure $G = (PN, M_0, \Sigma, \ell)$, where PN is a Petri net, M_0 is the initial marking, Σ is an alphabet (a set of labels) and $\ell : T \to \Sigma \cup \{\varepsilon\}$ is the labeling function that assigns to each transition $t \in T$ either a symbol from the given event set Σ or the empty sequence ε. The set of transitions T is partitioned into two disjoint sets as follows: $T = T_o \cup T_{uo}$, where $T_o = \{t \in T \mid \ell(t) \in \Sigma\}$ is *the observable transition set* and $T_{uo} = \{t \in T \mid \ell(t) = \varepsilon\}$ is *the unobservable transition set.* The labeling function is also naturally extended to firing sequences $\ell : T^* \to \Sigma^*$.

In an LPN, the observation of a sequence $\sigma \in T^*$ is denoted as $w = \ell(\sigma) \in \Sigma^*$. The *language* of an LPN G is defined as $\mathcal{L}(G) = \{w \in \Sigma^* \mid \exists \sigma \in L(PN, M_0) : \ell(\sigma) = w\}$. Given an observation $w \in E^*$, we define $\ell^{-1}(w) = \{\sigma \in L(PN, M_0) \mid \ell(\sigma) = w\}$ as the set of firing sequences consistent with w.

Given a net $PN = (P, T, Pre, Post)$ and a subset $T' \subseteq T$ of its transitions, we define the T'-induced subnet of PN as a new net $PN' = (P, T', Pre', Post')$ where Pre' and

$Post'$ are the restrictions of Pre and $Post$ to T', i.e., PN' is the net obtained from PN by removing all transitions in $T \setminus T'$. In the following, matrices C_u and C_o denote the restriction of the incidence matrix C to T_u and T_o, respectively.

The study in [21, 61] develops a semi-structural approach called basis reachability graph to represent the reachability set of a bounded Petri net. Basis reachability graph approach is a state space abstraction technique in Petri nets. In this method, only a subset of reachable markings, called basis markings, requires to be enumerated, while all other reachable markings are abstracted by linear algebraic equations. For a Petri net, the number of basis markings in the BRG is much smaller than the markings in the reachability graph.

Definition 2.1. *[61] Given a Petri net $PN = (P, T, Pre, Post)$, a pair $\pi = (T_E, T_I)$ is called a* basis partition[1] *of T if (1) $T_I \subseteq T$, $T_E = T \setminus T_I$; and (2) the T_I-induced subnet is acyclic. The sets T_E and T_I are called the set of* explicit transitions *and the set of* implicit transitions*, respectively.* □

Definition 2.2. *[61] Given a Petri net $PN = (P, T, Pre, Post)$, a basis partition $\pi = (T_E, T_I)$, a marking M, and a transition $t \in T_E$, we define:*

- $\Sigma(M, t) = \{\sigma \in T_I^* \mid M[\sigma\rangle M', M' \geq Pre(\cdot, t)\}$ *as the set of* explanations *of transition t at marking M;*
- $Y(M, t) = \{\mathbf{y}_\sigma \in \mathbb{N}^{|T|} \mid \sigma \in \Sigma(M, t)\}$ *as the set of* explanation vectors *of transition t at marking M;*
- $Y_{min}(M, t)$ *denotes the set of all minimal elements of $Y(M, t)$, i.e., the* minimal explanation vectors. □

The set of *basis markings* $\mathcal{M}$ is recursively defined as follows:

- $M_0 \in \mathcal{M}$;
- If $M \in \mathcal{M}$, then for all $t \in T_E$, for all $\mathbf{y} \in Y_{min}(M, t)$,

$$(M' = M + C_I \cdot \mathbf{y} + C(\cdot, t)) \Rightarrow (M' \in \mathcal{M}).$$

Given a partition $\pi = (T_E, T_I)$, the corresponding *basis reachability graph (BRG)* is a deterministic finite state automaton (DFA) defined in [61]. In short, a BRG $\mathcal{B}$ is a quadruple $(\mathcal{M}, Tr, \Delta, M_0)$, where:

[1]In general, there may exist multiple valid basis partitions for a given plant net. The selection of explicit and implicit transitions is not necessarily associated with physical meanings. However, for certain problems some particular basis partitions are useful.

- the state set $\mathcal{M}$ is the set of basis markings;
- the event set Tr is the set of pairs $(t, \mathbf{y}) \in T_E \times \mathbb{N}^{|T_I|}$;
- the transition function Δ is:

$$\begin{aligned}\Delta = \{(M_1, (t, \mathbf{y}), M_2) \mid t \in T_E, \mathbf{y} \in Y_{min}(M_1, t),\\ M_2 = M_1 + C_I \cdot \mathbf{y} + C(\cdot, t)\}\end{aligned}$$

- the initial state is the initial marking M_0.

Definition 2.3. *[61] Given a net $PN = (P, T, Pre, Post)$, a basis partition $\pi = (T_E, T_I)$, and a basis marking M_b, the* implicit reach *of M_b is defined as $R_I(M_b) = \{M \in \mathbb{N}^m \mid (\exists \sigma \in T_I^*) M_b[\sigma\rangle M\}$.* □

Theorem 2.1. *[61] Given a Petri net $PN = (P, T, Pre, Post)$, a basis partition $\pi = (T_E, T_I)$, and a marking M', the following condition holds:*

$$(\exists \sigma \in T^*)\sigma_{\uparrow T_E} = \sigma_E \wedge M_0[\sigma\rangle M' \Leftrightarrow (\exists M_b \in \mathcal{M})(M_0, \sigma_E, M_b) \in \Delta^* \wedge M' \in R_I(M_b)$$

where $(M_0, \sigma_E, M_b) \in \Delta^*$ denotes that, in the BRG, there exists a path from M_0 to M_b labeled by $(t_1, \mathbf{y_1}), \cdots, (t_2, \mathbf{y_2})$ such that $t_1 \cdots t_k = \sigma_E$. □

2.3 Supervisory Control

Supervisory control theory of DESs was first proposed by Ramadge and Wonham [62–64] in which they model a plant as an automaton. Given an plant $A = (X, E, \delta, x_0)$, the event set E is partitioned into two disjoint subsets $E = E_c \cup E_{uc}$ where E_c is the set of *controllable events* and E_{uc} is the set of *uncontrollable events*. The control objective is defined by a regular language $K \subseteq E^*$ called the *specification*. A supervisor S that dynamically disables events of a plant such that the closed-loop language of S over A is restricted within K.

A supervisor S runs in parallel with the plant and, when the plant generates a sequence $\sigma \in L(A)$, the supervisor observes the word $w = \mathcal{P}(\sigma)$ and makes the control policy $S(w)$ that disables all controllable events not in $S(w)$. Note that the supervisor cannot disable any uncontrollable $e_{uc} \in E_{uc}$.

A language $K \subseteq E^*$ is *controllable* (w.r.t. A and E_{uc}) if

$$(\forall \sigma \in \overline{K}, e \in E_{uc})(\sigma e \in L(A)) \Rightarrow \sigma e \in \overline{K}.$$

We say that K is *observable* (w.r.t. A and E_o) if for every pair of strings $\sigma, \sigma' \in E^*$ such that $\mathcal{P}(\sigma) = \mathcal{P}(\sigma')$, the following statement holds:

$$(\forall e \in E)(\sigma e \in \overline{K}, \sigma' \in \overline{K}, \sigma' e \in L(A)) \Rightarrow \sigma' e \in \overline{K}.$$

Theorem 2.2 ([65]). *Let $K \subseteq L(A)$ be a prefix-closed nonempty language. There exists a proper supervisor S such that the closed-loop language of S over A is K if and only if K is controllable and observable.*

In this paper, we use (A, S) to denote the closed-loop system composed by the plant A under the supervision of S, and we use $L(A, S)$ to denote the language of (A, S).

Chapter 3 Active Diagnosis Problem in the Framework of Automata

In the literature, many works have been done to deal with the active diagnosis problem in automata [45–49, 52, 54, 55]. Although theses contributions can be used to enforce diagnosability, they are all based on some observer-like structure whose complexity is exponential with respect to the number of states in a plant. Therefore, these approaches may encounter the state explosion problem. Differently from the existing works, in this chapter, we present an active diagnosis method based on the structural analysis of the verifier. Note that although the verifier can be used to efficiently test diagnosability, it is not straightforward in designing a supervisor such that the closed-loop system is diagnosable. As far as we know, this is the first work on the active diagnosis whose structural complexity of the control structure is polynomial with respect to the number of states of the plant.

3.1 Diagnosability Verification in Automata

In the section, we first review the notion of diagnosability in the framework of automata [3, 66]. Here we use "$e \in \sigma$" to denote that event e appears at least once in sequence σ.

Definition 3.1 (Diagnosability). *Given a live plant $A = (X, E, \delta, x_0)$ with $E = E_o \cup E_{uo}$ and $E_{uo} = E_{reg} \cup E_f$, a fault type $f_i \in E_f$ is* diagnosable *if for all $\sigma' f_i \in L(A)$, there exists $k_{\sigma'} \in \mathbb{N}$ such that for all $\sigma' f_i \sigma'' \in L(A)$,*

$$|\sigma''| \geq k_{\sigma'} \quad \Rightarrow \quad \forall \sigma \in P^{-1}(P(\sigma' f_i \sigma'')) : f_i \in \sigma.$$

where $\mathbb{N}$ is a set of natural numbers. □

The notion of diagnosability can be interpreted as follows. Suppose that fault f_i occurs after a sequence σ', i.e., sequence $\sigma' f_i$ has occurred. Fault f_i is diagnosable if after a sufficiently long continuation σ'', all possible sequences that have the same observation as that of $\sigma' f_i \sigma''$ contain fault f_i at least once. This implies that fault f_i can be detected within a finite delay after its occurrence. Note that the number $k_{\sigma'}$ depends on the particular sequence σ'.

Plant A is *diagnosable* if all fault types of A are diagnosable. For the sake of simplicity, in this work we consider a single type of faults, i.e., $\Sigma_f = \{f\}$. However, we note that the approach proposed in this chapter can be easily extended to plants with multiple fault types.

3.1.1 Diagnoser

Given a plant automaton A, in this section we recall the diagnoser in [3] that is used both for online diagnosis and diagnosability verification. For a plant $A = (X, E, \delta, x_0)$, the corresponding diagnoser is a four-tuple structure:

$$A_d = (Y, E_d, \delta_d, y_0)$$

where $Y \subseteq 2^{X\times\{N,F\}}$ is the set of diagnostics states, $E_d = E_o$ is the set of events, $\delta_d : Y \times \Sigma_d \to Y$ is the transition function, and y_0 is the initial state. For an observation $w \in E_o^*$, a diagnostic state is in the form of $y(w) = \{(x_1, l_1), ..., (x_n, l_n)\}$ where $x_i \in X$ and $l_i \in \{N, F\}$ that carries the fault information at state x_i. Hence, according to the current diagnostic state $y(w)$, a diagnoser reports that by observing w the plant is in either of the three cases:

- *Normal* (no fault has occurred so far) if for all $(x_i, l_i) \in y(w)$, $l_i = N$, i.e., all pairs are flagged by N;
- *Faulty* (fault f must have occurred at least once) if for all $(x_i, l_i) \in y(w), l_i = F$, i.e., all pairs are flagged by F;
- *Uncertain* (fault f may have occurred) if there exist $(x_i, l_i), (x_j, l_j) \in y(w)$, $l_i = N$ and $l_j = F$, i.e., some pairs are flagged by N and some are flagged by F;

Before defining the initial diagnostic state y_0 and the transition function δ_d, in the following, we first introduce a *labeling function* $\Delta : \{N, F\} \times \Sigma^* \to \{N, F\}$.

Definition 3.2. *Given a sequence $\sigma \in E^*$ and a label $l \in \{N, F\}$, the labeling function Δ is defined as follows:*

$$\Delta(l, \sigma) = \begin{cases} N, & \textit{if } l = N \wedge f \notin \sigma \\ F, & \textit{otherwise.} \end{cases} \tag{3-1}$$

□

The usage of the labeling function can be explained as follows. Suppose that a plant is currently at a state x at which a label l is attached indicating that fault f has occurred ($l = F$) or not ($l = N$). After the occurrence of a sequence σ, the current state is updated to a new one x' that attached a new label l'. If sequence σ does not contain fault f and label $l = N$, it is obviously that no fault has occurred at state x', i.e., $l' = N$. Otherwise, fault f has occurred and the corresponding label $l' = F$.

The initial diagnostic state y_0 contains the information about the states that the plant initially may be in and the condition of the plant, which is defined as follows:

$$y_0 = \{(x, \Delta(N, \sigma)) \mid \sigma \in L(A) \cap E_{uo}^*, x \in \delta(x_0, \sigma)\}.$$

The transition function $\delta_d : Y \times E_o \to Y$ of A_d is defined as follows:

$$\delta_d(y, e) = \{(x', \Delta(l, e\sigma)) \mid (x, l) \in y, \sigma \in U_e(x), x' \in \delta(x, e\sigma)\}.$$

In [3], an important notion related to diagnosability is indeterminate cycle in diagnoser A_d. The diagnosability of a plant can be determined by checking the existence of indeterminate cycles in the diagnoser of the plant.

Definition 3.3 (Indeterminate cycle). *Given a diagnoser $A_d = (Y, E_o, \delta_d, y_0)$ of a plant $A = (X, E, \delta, x_0)$ with $E = E_o \cup E_{uo}$, a sequence of uncertain diagnostic states $y_1 y_2 \cdots y_n$ forms an* indeterminate cycle *if:*

1. *the uncertain diagnostic states compose a cycle:*

$$y_1 \xrightarrow{e_1} y_2 \xrightarrow{e_2} \ldots \xrightarrow{e_{n-1}} y_n \xrightarrow{e_n} y_1,$$

where $e_i \in E_o$, $i \in \{1, 2, \ldots n\}$;

2. *there exist two cycles of states:*

   (a) $x'_1 \to x'_2 \to \cdots \to x'_n \to x'_1$ such that $x'_{i+1} \in \Delta(x'_i, e_i)$ and $(x'_i, N) \in y_i$;

   (b) $x''_1 \to x''_2 \to \cdots \to x''_n \to x''_1$ such that $x''_{i+1} \in \Delta_F(x''_i, e_i)$ and $(x''_i, F) \in y_i$;

where $\Delta(x, e) = \{\bar{x} \in X \mid \exists \sigma \in (E \setminus \{f\})^*, \mathcal{P}(\sigma) = e, \delta^*(x, \sigma) = \bar{x}\}$ and $\Delta_F(x, e) = \{\bar{x} \in X \mid \exists \sigma \in E^*, \mathcal{P}(\sigma) = e, \delta^*(x, \sigma) = \bar{x}\}$. □

An indeterminate cycle in the diagnoser A_d implies that there exist two arbitrary long sequences σ_1 and σ_2 such that sequence σ_1 contains a fault event while sequence σ_2 does not, and they both have the same observation. The following theorem provides a method to test the diagnosability of a given system.

Theorem 3.1. *For a plant A that does not have any cycle of unobservable events, it is diagnosable if and only if there does not exist any indeterminate cycle in its diagnoser A_d.*□

3.1.2 Verifier

Given a plant $A = (X, E, \delta, x_0)$, the diagnosability of A can be verified by using the diagnoser A_d, as mentioned in the previous subsection. However, it is obviously that the structure complexity of A_d is $O(2^{2|X|})$. Therefore, to reduce the complexity, some researchers [11, 12] develop a structure called verifier whose structure complexity is polynomial with respect to the number of states in the plant. In the following, we give the definition of verifier.

Definition 3.4 (Verifier). *Given a plant $A = (X, E, \delta, x_0)$ with $E = E_o \cup E_{uo}$ and $E_{uo} = E_{reg} \cup \{f\}$, its verifier (of fault f) is a* nondeterministic finite automaton $A_v = (V, E, \delta_v, v_0)$, *where (i) a state $v \in V$ is a set of pairs $\{(x_i, l_i), (x_j, l_j)\}$, where $x_i, x_j \in X$ and $l_i, l_j \in \{N, F\}$ are* flags *of the fault: N and F denote "the fault has not occurred" and "the fault has occurred", respectively; (ii) the initial verifier state is $v_0 = \{(x_0, N), (x_0, N)\}$; (iii) the transition rule δ_v is defined as follows:*

For $e \in E_{uo} \setminus E_f$,

$$\delta_v(\{(x_i, l_i), (x_j, l_j)\}, e) = \begin{cases} \{(\delta(x_i, e), l_i), (x_j, l_j)\} \\ \{(x_i, l_i), (\delta(x_j, e), l_j)\} \\ \{(\delta(x_i, e), l_i), (\delta(x_j, e), l_j)\}. \end{cases}$$

For $e \in E_f$,

$$\delta_v(\{(x_i, l_i), (x_j, l_j)\}, e) = \begin{cases} \{(\delta(x_i, e), F), (x_j, l_j)\} \\ \{(x_i, l_i), (\delta(x_j, e), F)\} \\ \{(\delta(x_i, e), F), (\delta(x_j, e), F)\}. \end{cases}$$

For $e \in E_o$,

$$\delta_v(\{(x_i, l_i), (x_j, l_j)\}, e) = \{(\delta(x_i, e), l_i), (\delta(x_j, e), l_j)\}.$$

□

For simplicity, in the sequel a verifier state $\{(x_i, l_i), (x_j, l_j)\}$ is denoted as $(x_i l_i, x_j l_j)$. A verifier state $(x_i l_i, x_j l_j)$ is called normal if $l_i = l_j = N$, and faulty if $l_i = l_j = F$. The diagnosability of a plant A can be determined by checking the existence of *confused cycles* in the verifier.

Definition 3.5 (Confused cycle). *Let A_v be the verifier of a plant $A = (X, E, \delta, x_0)$ with*

$E = E_o \cup E_{uo}$ *and* $E_{uo} = E_{reg} \cup \{f\}$. *A cycle*

$$(x'_1 l'_1, x''_1 l''_1) \xrightarrow{e_1} (x'_2 l'_2, x''_2 l''_2) \xrightarrow{e_2} \cdots \xrightarrow{e_{n-2}} (x'_{n-1} l'_{n-1}, x''_{n-1} l''_{n-1}) \xrightarrow{e_{n-1}} (x'_1 l'_1, x''_1 l''_1)$$

is confused *if for all* $v_i = (x'_i l'_i, x''_i l''_i)$, $l'_i = N$, $l''_i = F$, $i \in \{1, 2, ..., n-1\}$. □

Theorem 3.2. *[12] For a plant automaton A that does not have any cycle of unobservable events, it is diagnosable with respect to observation mask $\mathcal{P}$ and fault f if and only if there does not exist any confused cycle in its verifier A_v.* □

The idea here is that a fault f is diagnosable if and only if there is no pair of arbitrary long sequences having the same observation, such that f occurs in the first sequence but not in the second. A sequence σ in the verifier, i.e., $(x_0 N, x_0 N) \xrightarrow{\sigma} (x_i N, x_j F)$, implies that there exist two sequences in the plant having the same observation where one contains a fault while another not. Obviously, by checking the existence of confused cycle, one can determine the diagnosability of the plant.

3.2 Active Diagnosis Formulation

By Defintion 3.1, a fault in an automaton plant A is diagnosable if for any sequence $\sigma f \in L(A)$, after a finite continuation σ', all possible sequences having the same observation as $\sigma f \sigma'$ contain the fault at least once. However, in some plants it may happen that when a fault has occurred, by executing σf, there exists an infinite continuation σ' such that there always exists some non-faulty sequence that has the same observation as $\sigma f \sigma'$. Since by Theorem 3.2 the existence of confused cycles in the verifier is a sufficient and necessary condition for undiagnosability, a classical scheme to design a system is the following.

1. Model a physical plant as an automaton A;
2. Test if A is diagnosable (by checking its verifier);
3. If A is not diagnosable, modify A and go to Step 2;
4. Compute a diagnostic agent, and put A and its diagnostic agent online;

The way of modifying A in Step 3 requires to modify the plant structure including the states, transition relations, and observation masks. Such a modification requires to redesign the plant and/or to physically deploy new sensors, which may not be possible due to technical and financial difficulties. On the other hand, people are interested in the *active*

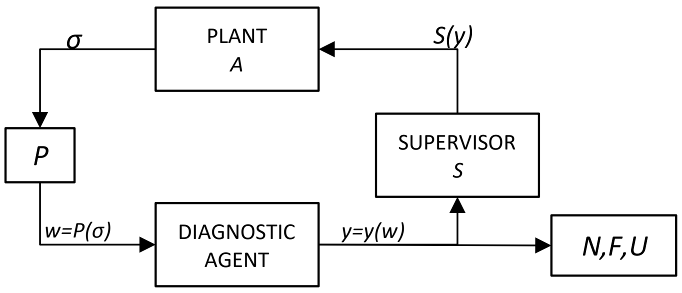


Fig. 3.1 The active diagnosis scheme.

diagnosis using supervisory control theory. In the active diagnosis scheme, a supervisor forces the plant to leave the uncertain diagnostic states by disabling some events such that the ambiguity of fault is clarified in a finite number of steps.

Classical supervisory control methods are usually in the context of languages. In this thesis, however, we consider a slight different type of supervisor whose control policy is made according to the current diagnostic state. State-based and language-based control frameworks are proved to be equivalent [67], but one will shortly see that state-based control is convenient and sufficient for active diagnosis. The scheme of active diagnosis in this chapter is illustrated in Figure 3.1. When a plant generates a sequence $\sigma \in L(A)$, the diagnostic agent observes the word $w = \mathcal{P}(\sigma)$ and obtains a diagnostic state $y(w)$. Based on $y(w)$ the supervisor allows a subset of controllable events $S(y) \subseteq E_c$ to execute, while other events in $E_c \setminus S(y)$ are disabled.

Since diagnosability can be verified in polynomial complexity [11, 12], to focus on the development of our control policy S for active diagnosis, we assume that a plant is known to be undiagnosable *a priori*. Nevertheless, it is worth noting that the proposed approach can be generalized to design a supervisor for a k-diagnosable plant such that the closed-loop system is k'-diagnosable with $k' < k$. The problem investigated in this chapter is formulated as follows.

Problem 3.1 (Active Diagnosis)**.** *Given a plant DFA A, determine a control policy S for A such that the closed-loop system (A, S) is diagnosable.* □

Moreover, we assume that a plant considered in this chapter satisfies the following assumptions:

A1 It is live;

A2 There does not exist any cycle of unobservable events in it;

A3 $E_c \subseteq E_o \subseteq E$.

We note that Assumption 1 is purely technical. In fact, if a plant contains dead states (at which no event can execute), it can be converted to an equivalent automaton without dead states by adding a self-loop labeled with the *quiescent event* at each dead states (e.g., see [45]). We also note that if the original plant is live, the supervisor designed in this chapter guarantees that the closed-loop system is also live. Assumption 2 is a commonly used assumption in the context of diagnosis, which requires that the plant does not generate an infinite long sequence that contains only unobservable events. Assumption 3 requires that only part of observable events should be controllable, while all unobservable events are uncontrollable. Such an assumption is widely used in the context of supervisory control in automata with unobservable events. The proposed method can also be generalized to cases where Assumption 3 is not satisfied by using the *All Enforcement* structure in [49].

3.2.1 Diagnoser-based Approach for Active Diagnosis in Automata

In the classical setting of diagnosis, a plant is assumed to be live. However, by enforcing a control policy S, the closed-loop system may get blocked at a state while all executable events are disabled by the supervisor. In other words, the closed-loop system (A, S) may be non-live. In [45], the author introduce a particular "$stop$" event at each dead state such that the non-live closed-loop system (A, S) is refined to a system $(A, S)^{live}$ that is always live. Hence both the diagnosability checking and the diagnosis can be done in the refined system $(A, S)^{live}$. An iterative algorithm is proposed in [45] to design a supervisor, and here we briefly recall it by the following example.

Example 3.1. *Figures 3.2(a) depicts a plant A in which $E_c = E_o = \{a, b, c, d, e\}$ and $E_f = \{f\}$. The diagnoser of A, i.e., A_d, is shown in Figure 3.2(b). According to Theorem 3.1, the plant is not diagnosable since diagnostic states y_2 and y_3 in the diagnoser form an indeterminate cycle.*

To make the system diagnosable by supervisory control, the indeterminate cycle must be cut[1] *to prevent the system from staying in such a loop for an infinite long time. Hence, we apply a control policy S' that disables event d at diagnostic state y_3. Since by observing abc the plant may be at state 7 (non-faulty), 4 (faulty), or 10 (faulty), this disablement of d at y_3 makes the plant be blocked in all cases. As a result, at diagnostic state y_3 the plant will generate the event $stop$ due to the disablement of event d.*

The diagnoser of the refined closed-loop system (A, S) is shown in Figure 3.2(c), which

[1]In [45] the control policy S is described by a language K_l that completes any elementary indeterminate cycle at most l times. Since in practical cases it is more preferable that a fault be detected as early as possible, here we omit this distinction and consider the case $l = 0$.

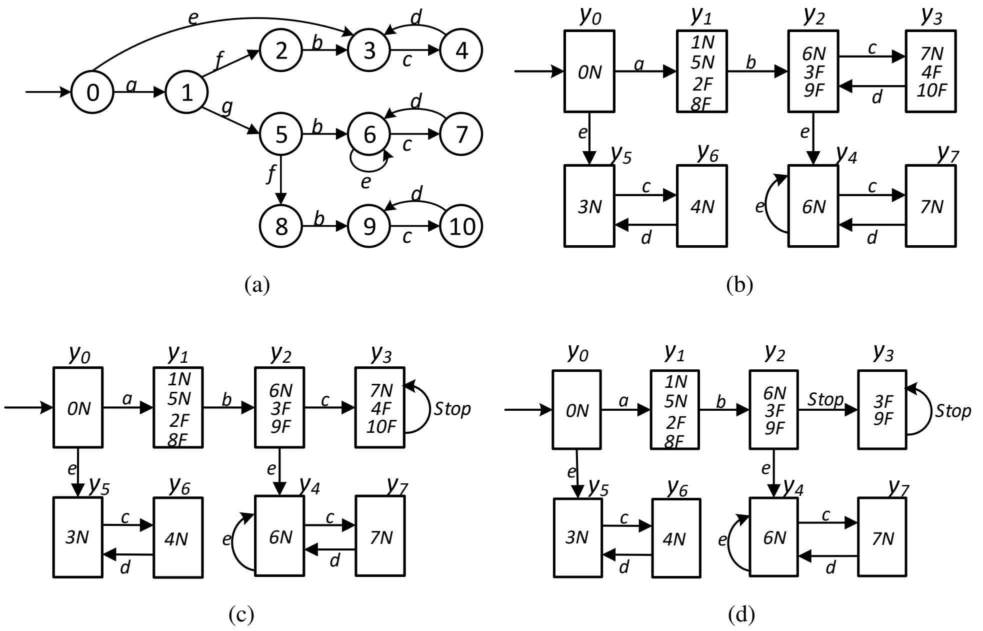


Fig. 3.2 (a) Plant A for Example 3.1, (b) diagnoser of A, (c) diagnoser of the closed-loop system $(A, S')^{live}$, and (d) diagnoser of the closed-loop system $(A, S'')^{live}$.

is still not diagnosable due to the indeterminate cycle at $y_3 = \{(7, N), (4, F), (10, F)\}$ *labelled with event* $stop$*. To eliminate the indeterminate cycle at* y_3 *in the refined diagnoser in Figure 3.2(c), we enforce a more restrictive control policy* S'' *that disables event* c *at diagnostic state* y_2 *which indicates that the plant currently may be at state 3 (faulty), 6 (non-faulty), or 9 (faulty). Since the supervisor cannot differentiate states 3, 6, and 9 at this moment, to be on the safe side, it has to disable event* c *regardless the actual current state. With the disablement of event* c *at* y_2*, event* e *is executable if the plant is at state 6, while no event is executable if the plant is at state 3 or 9. Hence if the fault has occurred, by observing the event* $stop$ *we can conclude that the plant is either at state* $3F$ *or* $9F$ *by which we know that the fault must have occurred. The diagnoser of the refined closed-loop system* (A, S'') *is shown in Figure 3.2(d) and does not contain any indeterminate cycle. Therefore, the automaton in Figure 3.2(d) can be used as a supervisor for active diagnosis.* □

Remark 3.1. *Since the control decision made by the supervisor is based on diagnostic states, the trimming procedure can be directly done on the diagnoser structure as shown in Example 3.1. Another way of trimming the system is to modify the plant structure at each iteration. For example, if the supervisor disables event* d *at diagnostic state* $y_3 = \{(7, N), (4, F), (10, F)\}$ *in Figure 3.2(b), all three events labelled with* d *at states* $4, 7, 10$ *are removed and* $stop$ *events are added to these states. However, this way of trimming requires to completely recompute the diagnoser at each iteration, and the resultant supervisor*

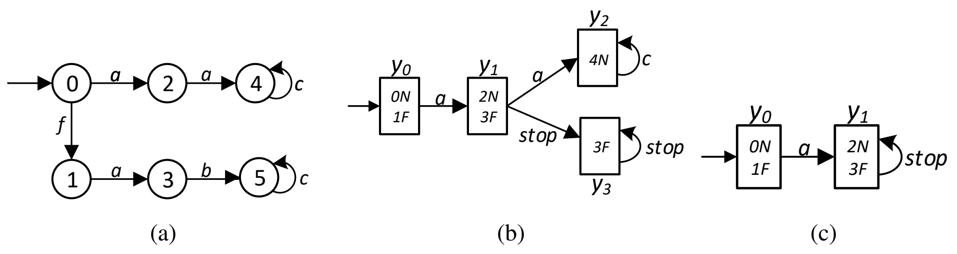


Fig. 3.3 (a) Automaton A for Example 3.2, (b) the closed-loop system (A, S_1), and (c) the closed-loop system (A, S_2).

may be more restrictive: the sequence $e(cd)^k$ ($k \in \mathbb{N}$) associated to $y_0 \to y_5 \to y_6 \to \cdots$ is not undiagnosable but is no longer executable. □

3.2.2 Properties of Supervisor-induced Deadlocks

Diagnoser-based methods provide a way to enforce diagnosability by supervisory control. However, they are computationally expensive since the diagnoser structure is exponential with respect to the number of states of a plant. In this subsection we show that the computation of the diagnoser is unavoidable due to the introduction of "*stop*" event. Let us first recall the *restrictiveness* of control policies.

Definition 3.6. *Given a plant A, a control policy S'' is* more restrictive *than another control policy S', denoted by $S'' \prec S'$, if for all diagnostic state y of A, $S''(y) \subseteq S'(y)$ holds, and there exists at least one diagnostic state y in A such that the inclusion is strict.* □

If S'' is more restrictive than S', then S' is *more permissive* than S''. In many supervisory control problems, since $L(A, S'') \subseteq L(A, S')$ holds where S'' is more restrictive than S', the fact that one control policy is valid implies that any control policy that is more restrictive than it is also valid[2]. However, with the *stop* event in the plant, the language of the closed-loop system $L(A, S)^{live}$ is not a sublanguage of $L(A)$, and in general $L(A, S'')^{live} \nsubseteq L(A, S')^{live}$ holds, where S'' is more restrictive than S'. This indicates that *diagnosability* is not preserved when the permissiveness of a supervisor decreases. In other words, that the closed-loop system $(A, S)^{live}$ is diagnosable does not imply that $(A, S'')^{live}$ is also diagnosable, as illustrated by the following example.

Example 3.2. *Consider the plant in Figure 3.3(a) in which $E_o = \{a, b, c\}$, $E_c = \{a, b\}$, and $E_f = \{f\}$. The initial diagnostic state $y_0 = \{(0, N), (1, F)\}$ is* Uncertain. *Suppose that a supervisory control policy S_1 satisfies $S_1(y_1) = \{a\}$, i.e., it only allows event a to*

[2]Sometimes the control goal includes to preserve a minimum legal behavior, e.g., in [68]. In such cases, if S' is valid, then for any more restrictive S'' that includes the required minimal legal behavior, S'' is also valid.

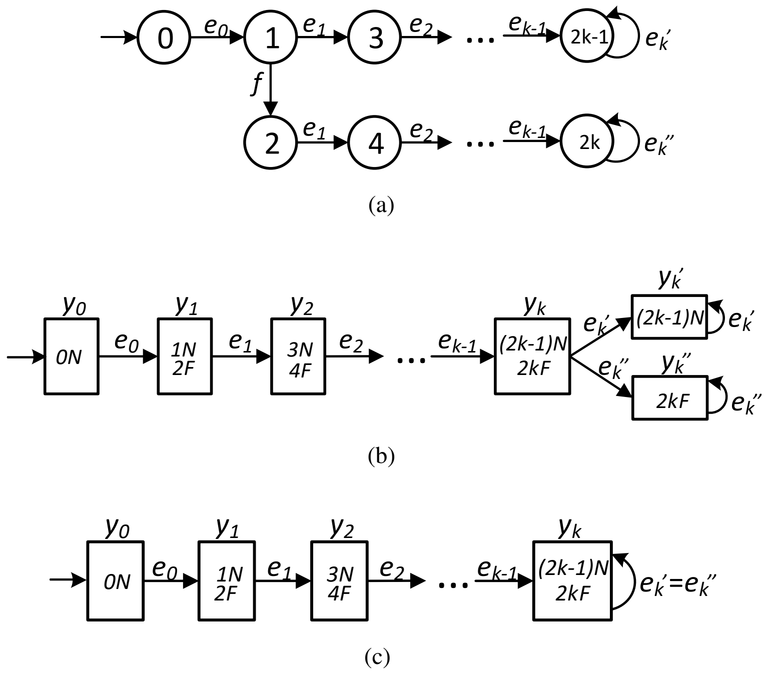


Fig. 3.4 (a) Automaton A for Example 3.3, (b) the diagnoser of A ($e'_k \neq e''_k$), (c) the diagnoser of A ($e'_k = e''_k$).

execute. The closed-loop behavior of (A, S_1) is shown in Figure 3.3(b) and it is clearly diagnosable. However, for a control policy S_2 such that $S_2(y_1) = \emptyset$, it disables both a and b. The closed-loop behavior of (A, S_2) is undiagnosable, as shown in Figure 3.3(c). □

Suppose that the current diagnostic state of a system is y and the execution of event e yields a new diagnostic state y'. The supervisor must decide if event e should be permitted. According to Theorem 3.1, event e should be permitted if y' does not belong to an indeterminated cycle. However, the control policy at a diagnostic state y cannot be made only by the knowledge of y. In other words, the decision $S(y)$ can only be made after computing the entire closed-loop behavior, i.e., the diagnoser.

Example 3.3. *Consider the automaton A shown in Figure 3.4(a) where $E_o = E_c = \{e_0, e_1, e_2, ..., e_{k-1}, e'_k, e''_k\}$ and $E_f = \{f\}$. The diagnosers for $e'_k \neq e''_k$ and $e'_k = e''_k$ are shown in Figure 3.4(b) and (c), respectively. Now let us consider the initial control decision "whether event e_0 should be permitted" at diagnostic state y_0. Notice that $y_0 \xrightarrow{e_0} y_1$ that is an uncertain diagnostic state, and the most restrictive control policy at y_1 that disables event e_1 at y_1 makes the closed-loop system undiagnosable (since the plant is blocked at $y_1 = \{(1, N), (2, F)\}$). This indicates that e_0 should be permitted only if the control decision at y_1 permits event e_1. By the same reasoning, at y_1 event e_1 should be permitted only if the control decision at y_2 permits event e_2. This reasoning can be repeatedly applied such that* *the decision "whether event e_0 should be permitted" can only be made by analyzing all subsequent diagnostic states from y_0, which is in fact to enumerate all diagnostic states in the diagnoser.* □

Since the computation of diagnoser is unavoidable due to the control-induced deadlock, in the next section we introduce the notion of $stop$-free control policy whose supervision does not introduce $stop$ event. A particular property of this class of control policies S' and S'' is that the diagnosability of (A, S') implies that of (A, S'') if $S'' \prec S'$. This property will be further used to establish our active diagnosis algorithm.

3.3 $Stop$-free Control Policy and Properties

3.3.1 $Stop$-free Event Set

Since we are interested in finding a control policy S such that the closed-loop system is diagnosable without creating the silent-blocking, the control policy S must satisfy the following condition: For any plant state x that belongs to any reachable diagnostic state y, there exists at least one event $e \in E$ that is executable at state x. The $stop$-free control policy is defined as follows.

Definition 3.7. *Given a plant A, a control policy S is stop-free if for any reachable diagnostic state y and any state $x \in y$, $(S(y) \cup E_{uc}) \cap \Gamma(x) \neq \emptyset$ holds.* □

Unfortunately, there does not exist an efficient method to find all reachable diagnostic states of A except by enumerating the entire *diagnoser*. Hence in the following we introduce the notion of $stop$-free event set. A $stop$-free event set $\hat{E}_c$ is a subset of the controllable event set E_c, and for each state x there exists at least one event $e \in \Gamma(x)$ that does not belong to $\hat{E}_c$.

Definition 3.8. *Given a plant $A = (X, E, \delta, x_0)$ with $E = E_c \cup E_{uc}$, $\hat{E}_c$ is a stop-free event set if $(\hat{E}_c \subseteq E_c) \wedge (\forall x \in X, \Gamma(x) \nsubseteq \hat{E}_c)$.* □

The physical meaning of the $stop$-free event set $\hat{E}_c$ is presented as follows. Although a control agent can disable controllable events in E_c, to guarantee the $stop$-freeness (i.e., the supervisor does not cause a deadlock) the control agent only controls a subset of controllable events, while the events in $E_c \setminus \hat{E}_c$ are never disabled by the supervisor. We note that *uncontrollable events* in $E \setminus E_c$ are different from the events *uncontrolled* with respect to a $stop$-free event set: an uncontrollable event in $E \setminus E_c$ cannot be disabled by a supervisor in any case, while an event in $E_c \setminus \hat{E}_c$ is controllable, but a supervisor that is designed will never attempt to disable it.

Proposition 3.1. *Given a stop-free event set $\hat{E}_c$ in a live plant A, the closed-loop system (A, S) is live if S never disables any event not in $\hat{E}_c$.*

Proof. Since S never disables any event not in $\hat{E}_c$, for any diagnostic sate y, $S(y) \supseteq (E_c \backslash \hat{E}_c)$ holds. Then we can obtain that $(S(y) \cup E_{uc}) \supseteq (E \setminus \hat{E}_c)$. By Definition 3.8, for each state $x \in y$, $\Gamma(x) \cap (E \setminus \hat{E}_c) \neq \emptyset$, which indicates that $(S(y) \cup E_{uc}) \cap \Gamma(x) \neq \emptyset$, i.e., there exists at least one executable event at x. Therefore, the closed-loop system (A, S) is live. □

It is not difficult to understand that $stop$-free event set $\hat{E}_c$ always exists ($\hat{E}_c = \emptyset$ if G is *live*) and in general is not unique. However, not all $stop$-free event sets are feasible to design a control policy to guarantee diagnosability. The necessary condition of the existence of a feasible $stop$-free event set $\hat{E}_c$ will be discussed in the next subsection, where an algorithm is also proposed. Given a $stop$-free event set $\hat{E}_c$, we say that a control policy S is $stop$-free with respect to $\hat{E}_c$ if it never disables any event that is not in $\hat{E}_c$. In the following, we propose two properties for $stop$-free control policy.

Proposition 3.2. *Given a plant $A = (X, E, \delta, x_0)$ with $E = E_c \cup E_{uc}$ and a stop-free event set $\hat{E}_c \subseteq E_c$, let S' and S'' be two stop-free control policies with respect to $\hat{E}_c$ with $S'' \prec S'$. Then $L(A, S'') \subseteq L(A, S')$.*

Proof. Since the two control policies S' and S'' are $stop$-free with respect to $\hat{E}_c$, the closed-loop systems (A, S') and (A, S'') do not contain the $stop$ event. Since the precondition making $L(A, S')$ and $L(A, S'')$ incomparable is excluded, for any sequence $\sigma \in L(A, S'')$, we can infer that $\sigma \in L(A, S')$. □

Note that Proposition 3.2 does not hold for non-$stop$-free control policies, as discussed in the previous section: due to the existence of quiescent event $stop$, $L(A, S'')^{live} \nsubseteq L(A, S')^{live}$.

Proposition 3.3. *Given a plant $A = (X, E, \delta, x_0)$ where $E = E_c \cup E_{uc}$ and a stop-free event set $\hat{E}_c \subseteq E_c$, let S' and S'' be two stop-free control policies with respect to $\hat{E}_c$ with $S'' \prec S'$. Then it holds:*

$$(A, S') \textit{ is diagnosable } \Rightarrow (A, S'') \textit{ is diagnosable}.$$

Proof. Since (A, S') is diagnosable, for any faulty sequence $\sigma_1 f$ there exists an integer k such that for all $\sigma_1 f \sigma_2 \in L(A, S')$ with $|\sigma_2| \geq k$, any non-faulty sequence $\sigma \in L(A, S')$ satisfies $\mathcal{P}(\sigma) \neq \mathcal{P}(\sigma_1 f \sigma_2)$. Since both (A, S') and (A, S'') are $stop$-free, by Proposition 3.2, $L(A, S'') \subseteq L(A, S')$ holds. Hence for any faulty sequence $\sigma_1' f \in L(A, S'')$, there exists an integer k' such that for all $\sigma_1' f \sigma_2' \in L(A, S'')$ with $|\sigma_2'| \geq k'$, any non-faulty sequence $\sigma' \in L(A, S'')$ satisfies $\mathcal{P}(\sigma') \neq \mathcal{P}(\sigma_1' f \sigma_2')$, which concludes the proof. □

Proposition 3.3 reveals an important fact: if at diagnostic state y' we apply a most restrictive control policy such that all events in $\hat{E}_c$ are disabled from now on, and if the plant can still reach some indeterminate cycles and stay for infinite long time, then the supervisor should disable event e to prevent the system to reach diagnostic state y'. As a result, the decision of whether an event e should be permitted at y can be determined by simply looking at the new diagnostic state y'. To characterize this we introduce the notion of *$\hat{E}_c$-uncontrollable confused cycles*.

Definition 3.9. *Given a stop-free event set $\hat{E}_c \subseteq E_c$, a sequence $\sigma \in E^*$ is $\hat{E}_c$-*uncontrollable *if $\sigma \in (E \setminus \hat{E}_c)^*$; otherwise σ is $\hat{E}_c$-*controllable. □

Definition 3.10. *Given a plant $A = (X, E, \delta, x_0)$ with $E = E_c \cup E_{uc}$ and a stop-free event set $\hat{E}_c \subseteq E_c$, a confused cycle $\mathcal{C}$ in the verifier A_v is a $\hat{E}_c$-*uncontrollable confused cycle *(with respect to $\hat{E}_c$) if $E_{\mathcal{C}} \cap \hat{E}_c = \emptyset$ holds, where $E_{\mathcal{C}}$ is the set of events in cycle $\mathcal{C}$. The cycle is called a $\hat{E}_c$-*controllable confused cycle *(with respect to $\hat{E}_c$) if $E_{\mathcal{C}} \cap \hat{E}_c \neq \emptyset$ holds.* □

Given a $stop$-free control policy S with respect to $\hat{E}_c$, let σ be a sequence that is allowed by control policy S. Suppose that in the verifier A_v, after the execution of sequence σ, there exists a continuation of sequence $\sigma'\sigma''$ such that:

$$v_0 \xrightarrow{\sigma} (x_i l_i, x_j l_j) \xrightarrow{\sigma'} (x_i' l_i', x_j' l_j') \xrightarrow{\sigma''} (x_i' l_i', x_j' l_j')$$

where σ' is a $\hat{E}_c$-uncontrollable sequence and $(x_i' l_i', x_j' l_j') \xrightarrow{\sigma''} (x_i' l_i', x_j' l_j')$ is a $\hat{E}_c$-uncontrollable confused cycle. This means that from $(x_i l_i, x_j l_j)$ sequence $\sigma'(\sigma'')^k$ can be executed with an arbitrary large integer k and cannot be disabled by S. Hence, the closed-loop system (A, S) is not diagnosable. To prevent this from happening, in the following we introduce the notion of *undiagnosable verifier state*.

Definition 3.11. *Given a plant $A = (X, E, \delta, x_0)$ with $E = E_c \cup E_{uc}$ and a stop-free event set $\hat{E}_c \subseteq E_c$, let $A_v = (V, E, \delta_v, v_0)$ be the verifier of A. A verifier state $v = (x_i l_i, x_j l_j) \in V$ is an* undiagnosable verifier state *(with respect to $\hat{E}_c$) if there exists a path in A_v from v to v'*

such that (i) the path has no events in $\hat{E}_c$, and (ii) v' belongs to a $\hat{E}_c$-uncontrollable confused cycle. □

By the discussion above, such a verifier state (x_il_i, x_jl_j) must be prevented, i.e., a control policy S must guarantee $\sigma \notin L(A, S)$, which is formalized in the following theorem.

Theorem 3.3. *Given a plant $A = (X, E, \delta, x_0)$ with $E = E_c \cup E_{uc}$ and a stop-free control policy S with respect to a stop-free event set $\hat{E}_c \subseteq E_c$, the closed-loop system (A, S) is not diagnosable if there exists a reachable diagnostic state y that contains an undiagnosable verifier state (x_il_i, x_jl_j).*

Proof. Suppose that there exists a reachable diagnostic state that contains an undiagnosable verifier state $v = (x_il_i, x_jl_j)$. By Definition 3.11, there exists a $\hat{E}_c$-uncontrollable sequence σ' from v to $v' = (x_i'N, x_j'F)$ that belongs to a $\hat{E}_c$-uncontrollable confused cycle. Hence, from the trajectory $v_0 \to v \to v' \to v'$ we can extract a non-faulty sequence $\sigma_1'\sigma_2'\sigma_3'$ and a faulty sequence $\sigma_1''\sigma_2''\sigma_3''$, where $f \in \sigma_1''\sigma_2''$ such that $\mathcal{P}(\sigma_1'\sigma_2'(\sigma_3')^k) = \mathcal{P}(\sigma_1''\sigma_2''(\sigma_3'')^k)$ ($k \in \mathbb{N}$). Since $\sigma_2'\sigma_3'$ and $\sigma_2''\sigma_3''$ are $\hat{E}_c$-uncontrollable, they cannot be disabled by a supervisor. Thus, $\sigma_1'\sigma_2'(\sigma_3')^k, \sigma_1''\sigma_2''(\sigma_3'')^k \in L(A, S)$ holds. This indicates that there exist two infinite long sequences having the same observation and one contains fault while the other does not. Therefore, (A, S) is not diagnosable. □

Remark 3.2. *We note that the flags l_i and l_j of an undiagnosable verifier state $v = (x_il_i, x_jl_j)$ are not necessarily different. A verifier state (x_iN, x_jN) with both flags N may also lead to undiagnosability since from such a state a sequence of events that are not in $\hat{E}_c$ including the fault f (recall that fault f is not in $\hat{E}_c$) may be uncontrollably executed to yield a verifier state $(x_i'N, x_j'F)$ that belongs to a $\hat{E}_c$-uncontrollable confused cycle. Such a type of verifier states, although double-flagged by N, must also be prevented.* □

3.3.2 Computing a Feasible $Stop$-free Event Set

Theorem 3.3 shows that a control policy S must prevent the system from reaching any diagnostic state that contains undiagnosable verifier states. However, for some $\hat{E}_c$ it may happen that the initial diagnostic state y_0 contains undiagnosable verifier states. In other words, for some $stop$-free event set $\hat{E}_c$ there does not exist any $stop$-free control policy with respect to $\hat{E}_c$ such that the closed-loop system is diagnosable. The following proposition characterizes this fact.

Proposition 3.4. *Given a plant $A = (X, E, \delta, x_0)$ with $E = E_c \cup E_{uc}$ and a stop-free event set $\hat{E}_c \subseteq E_c$, if the initial verifier state v_0 is an undiagnosable verifier state, then for any stop-free control policy S with respect to $\hat{E}_c$, the system (A, S) is not diagnosable.*

Proof. If the initial verifier state is an undiagnosable verifier state, by Definition 3.11, there exists a $\hat{E}_c$-uncontrollable sequence σ from v_0 to $v = (x_i'N, x_j'F)$ that belongs to a $\hat{E}_c$-uncontrollable confused cycle. Hence in (A, S) there exist a non-faulty sequence $\sigma_1(\sigma_2)^k$ ($k \in \mathbb{N}$) and a faulty sequence $\sigma_1'(\sigma_2')^k$ where $\mathcal{P}(\sigma_1(\sigma_2)^k) = \mathcal{P}(\sigma_1'(\sigma_2')^k)$. Sequences $\sigma_1(\sigma_2)^k, \sigma_1'(\sigma_2')^k$ are $\hat{E}_c$-uncontrollable and cannot be disabled by a supervisor. Therefore, the result holds. □

Proposition 3.4 indicates the following fact. Although a $stop$-free event set always exists and in general is not unique, for some $stop$-free event sets it may not be possible to design a supervisor to guarantee diagnosability. In other words, a smaller $\hat{E}_c$ may guarantee $stop$-freeness, but a too small $stop$-free event set is insufficient to enhance the diagnosability. We note that Proposition 3.4 shows also a sufficient condition that will be proved in the next section, i.e., given a $stop$-free event set $\hat{E}_c$, if the initial diagnostic state does not contain an undiagnosable verifier state, then we can always find a $stop$-free control policy S such that (A, S) is diagnosable. Moreover, from Proposition 3.4 we have the following necessary conditions for the existence of a diagnosability enhancing control policy.

Corollary 3.1. *If the initial verifier state v_0 is an undiagnosable verifier state with respect to $\hat{E}_c = E_c$, then there does not exist a control policy such that (A, S) is diagnosable.*

Proof. Notice that the initial diagnostic state y necessarily contains the initial verifier state v_0, which cannot be prevented by any control policy. If v_0 is an undiagnosable verifier state with respect to E_c, by Proposition 3.4, (A, S) is undiagnosable for all control policy S. □

We say that a $stop$-free event set $\hat{E}_c$ is *feasible* if there exists a control policy with respect to $\hat{E}_c$ that ensures diagnosability. By the above discussion, a valid $stop$-free event set $\hat{E}_c$ necessarily satisfies the following two conditions:

- *stop-freeness*: $(\hat{E}_c \subseteq E_c) \wedge (\forall x \in X, \Gamma(x) \nsubseteq \hat{E}_c)$;
- *feasibility*: the initial verifier state with respect to $\hat{E}_c$ is not an undiagnosable verifier state.

In the following, we introduce the notion of *sub-verifier* that can be computed by Algorithm 1 and can be used to test the feasibility of a given set $\hat{E}_c$.

Definition 3.12. *Given a verifier $A_v = (V, E, \delta_v, v_0)$ and a sub-alphabet $E' \subseteq E$, the E'-*induced sub-verifier $A_{v,E'} = (V', E', \delta'_v, v_0)$ *is the sub-verifier obtained from A_v by removing all edges labeled with event $e \in E \setminus E'$, followed by removing all unreachable states.* □

Algorithm 1: Computation of E'-induced sub-verifier.

```
  Input: $A_v = (V, E, \delta_v, v_0)$ and a set $E' \in E$
  Output: $A_{v,E'} = (V', E', \delta'_v, v_0)$
1 Let $V' = V, \delta'_v = \delta_v$;
2 foreach $e \in E \setminus E'$ and $v \in V'$ such that $\delta'_v(v, e) \in \delta'_v$ do
3     Delete $\delta'_v(v, e)$ from $\delta'_v$;
4 end
5 Output the accessible part of $A_{v,E'}$;
```

A property of the *stop*-freeness and feasibility is depicted by the following proposition, which provides a way to establish a heuristic algorithm to find a feasible *stop*-free event set.

Proposition 3.5. *Given two sets $\hat{E}_1$, $\hat{E}_2 \subseteq E_c$ and $\hat{E}_1 \subseteq \hat{E}_2$:*

- *the stop-freeness of $\hat{E}_2$ implies the stop-freeness of $\hat{E}_1$;*
- *the feasibility of $\hat{E}_1$ implies the feasibility of $\hat{E}_2$.*

Proof. Suppose that $\hat{E}_2$ meets the condition of *stop*-freeness. It indicates that for all $x \in X$, there exists at least one event $e \in E$ such that $\delta(x, e)$ is defined and $e \notin \hat{E}_2$. By $\hat{E}_1 \subseteq \hat{E}_2$, we have $e \notin \hat{E}_1$. Therefore, $\hat{E}_1$ is *stop*-free.

Then, we prove the second conclusion. By $\hat{E}_1 \subseteq \hat{E}_2$, we have $(E \setminus \hat{E}_1) \supseteq (E \setminus \hat{E}_2)$. Due to Definition 3.12, $A_{v,E\setminus\hat{E}_i}$ only contains the events in $E \setminus \hat{E}_i$. If in $A_{v,E\setminus\hat{E}_1}$ there does not exist a confused cycle that can be reached from the initial verifier state, it indicates that in $A_{v,E\setminus\hat{E}_2}$ there must not exist a confused cycle that can be reached from the initial verifier state. Therefore, the event set $\hat{E}_2$ is feasible. □

Given a plant A, the set $\hat{E}_c$ (both *stop*-free and feasible) is in general not unique. In practice a larger *stop*-free event set is preferable since it provides more flexibility in designing the control policy. In the following, a heuristic algorithm is proposed to find a feasible *stop*-free event set with the maximal cardinality. First, a pre-screening procedure is executed to reduce the number of event to be tested. Then a heuristic search is applied to find a minimal set of events to remove from E_c. At each iteration, a minimal set of events is selected such that a candidate of $\hat{E}_c$ with a maximal cardinality is tested for *stop*-freeness

Algorithm 2: Computation of a feasible $stop$-free event set $\hat{E}_c$

```
    Input: Plant $A = (X, E, \delta, x_0)$ with $E = E_{uc} \cup E_c$
    Output: A feasible $stop$-free event set $\hat{E}_c$
 1  Let $q_0 = \emptyset$;
 2  foreach $x \in X$ do
 3      if $\Gamma(x) = \{e\}$, where $e \in E_c$ then
 4          $q_0 = q_0 \cup \{e\}$;
 5      end
 6  end
 7  Let $Q = 2^{E_c \setminus q_0}$;
 8  while $Q \neq \emptyset$ do
 9      Select an event set $q \in Q$ with minimal cardinality;
10      Let $\hat{E}_c = E_c \setminus (q \cup q_0)$;
11      Compute the $(E \setminus \hat{E}_c)$-induced sub-verifier $A_{v,E\setminus\hat{E}_c}$;
12      if $\hat{E}_c$ is stop-free then
13          if $\hat{E}_c$ is feasible then
14              Output $\hat{E}_c$, END;
15          else
16              Let $Q = Q \setminus \{q' \in Q \mid q' \supseteq q\}$;
17          end
18      else
19          Let $Q = Q \setminus \{q' \in Q \mid q' \subseteq q\}$;
20      end
21  end
22  Output: No solution;
```

and feasibility. The two properties in Proposition 3.5 are used (in a contrapositive way) to reduce the searching space.

We explain how Algorithm 2 works step-by-step. Steps 1 to 6 are pre-screening procedures. For a state $x \in X$, if there is only one event e that is defined at state x, i.e., $|\Gamma(x)| = 1$, and e is controllable (i.e., $\Gamma(x) = \{e\}$, where $e \in E_c$), then event e is put into set q_0 by Step 4, which means that it is exempted from the future test (since such an event must not belong to $\hat{E}_c$ to ensure $stop$-freeness). In Step 7, the power set $Q = 2^{E_c \setminus q_0}$ is generated as the set of candidates. Steps 8 to 21 compose the heuristic module. At each iteration a minimal set q among all untested cases is selected, and the corresponding $\hat{E}_c = E_c \setminus (q \cup q_0)$ with maximal cardinality is tested for both $stop$-freeness and feasibility. The $stop$-freeness is tested by checking if for each state $x \in X$, at least one event $e \notin \hat{E}_c$ is defined at state x, and the feasibility is tested by checking if there does not exist any confused cycle in $A_{v,E\setminus\hat{E}_c}$. If the two conditions are satisfied, then $\hat{E}_c$ is a feasible $stop$-free event set with the maximal

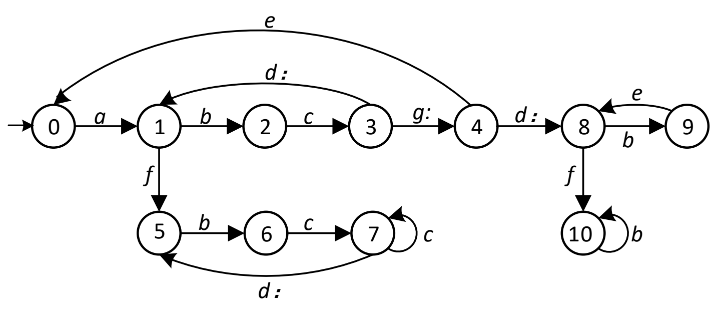


Fig. 3.5 Automaton A for Example 3.4.

cardinality which will be used to design a control policy in the next section. On the other hand, if at least one test fails, q is removed from the candidate set Q, and by Steps 16 and 19 some other candidates may also be removed according to the contraposition of Proposition 3.5.

Theorem 3.4. *The event set $\hat{E}_c$ obtained by Algorithm 2 is stop-free and feasible.*

Proof. Steps 12 and 13 in Algorithm 2 ensure that the output set $\hat{E}_c$ is $stop$-free and feasible, respectively. Therefore, the event set $\hat{E}_c$ obtained by Algorithm 2 is a feasible $stop$-free event set. □

Example 3.4. *Let us consider automaton A depicted in Figure 3.5. The controllable event set is $E_c = \{d, g\}$ and the uncontrollable event set is $E_{uc} = \{a, b, c, e, f\}$. The unobservable event set is $E_{uo} = \{e, f\}$ and the fault event is f. The corresponding verifier A_v is shown in Figure 3.6. Plant A is not diagnosable due to the two confused cycles in A_v (in dashed ellipses).*

Now let us apply Algorithm 2 to compute a feasible $stop$-free event set. In this example, set $q_0 = \emptyset$ since in the plant there does not exist a state at which only a controllable event is defined. Then the power set $Q = 2^{E_c} = \{\emptyset, \{d\}, \{g\}, \{d, g\}\}$ and the verifier A_v are computed. In the first loop, $q_1 = \emptyset$ is selected from Q such that $\hat{E}_c = E_c$. In Step 12, $\hat{E}_c$ is tested and it is not stop-free since $\Gamma(3) = \{d, g\} \subseteq \hat{E}_c$ holds. Hence q_1 is removed from Q. In the second loop, $q_2 = \{d\}$ is selected such that $\hat{E}_c = E_c \setminus q_2 = \{g\}$. Such a set $\hat{E}_c$ is stop-free. To test the feasibility of $\hat{E}_c$ in Step 13, the $(E \setminus \{g\})$-induced verifier $A_{v,E\setminus\{g\}}$ is computed by removing all edges in A_v labelled with event g followed by removing all unreachable states v_4, and $v_{11} - v_{17}$. Since in $A_{v,E\setminus\{g\}}$ there exists a confused cycle, $\hat{E}_c = \{g\}$ is not feasible, and hence q_2 is removed from Q. Since, by Proposition 3.5, any $q' \supseteq q$ is neither feasible, in Step 16 $q_4 = \{g, d\}$ is also removed from Q. Finally, in the third loop, for $q_3 = \{g\}$, we have $\hat{E}_c = \{d\}$ that is stop-free. To test feasibility, $A_{v,E\setminus\{d\}}$ is

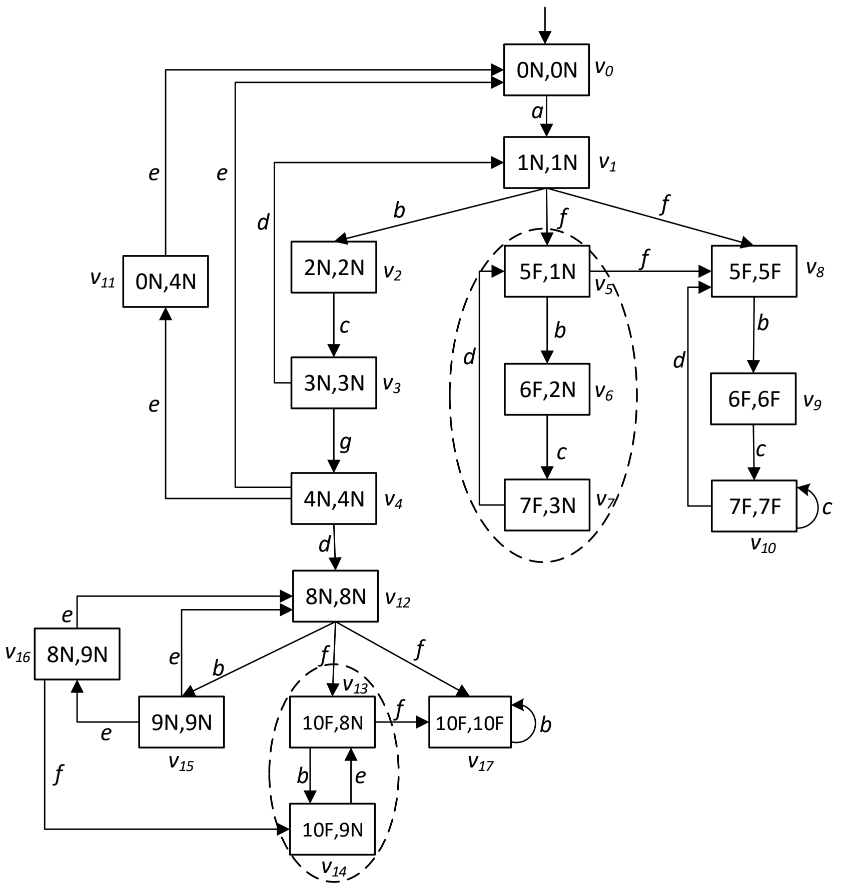


Fig. 3.6 The verifier of automaton A in Figure 3.5.

computed by removing all edges in A_v labelled with event d ($v_3 \to v_1$, $v_7 \to v_5$, $v_{10} \to v_8$, and $v_4 \to v_{12}$) followed by removing all unreachable states $v_{12} - v_{17}$. Since in $G_{v,E\setminus\{d\}}$ there do not exist confused cycles, $\hat{E}_c = \{d\}$ is output which is a feasible stop-free event set. $\square$

3.3.3 Complexity Analysis

In this subsection we have the following remarks on the complexity of finding a $stop$-free and feasible set $\hat{E}_c$ with maximal cardinality.

First, let us consider the complexity of Algorithm 2. The computation of set q_0 needs to check all states in the plant, whose complexity is $O(|X|)$. The $stop$-freeness is tested by checking if for all states $x \in X$ at least one event $e \notin \hat{E}_c$ is defined at state x, whose complexity is $O(|X|)$. On the other hand, the feasibility is tested by checking if there exists a confused cycle in $A_{v,E\setminus\hat{E}_c}$. The computation of verifier has complexity $O(|V| \cdot |E|)$ (where V is the set of states in A_v), and to check confused cycles in the worst case has complexity $O(|V| \cdot |E|)$. Therefore, the complexity to test feasibility is $O(|V| \cdot |E|) = O(|X|^2 \cdot |E|)$. Since the outer loop executes at most $2^{|E_c|}$ times when q equals to E_c,[3] the global complexity of Algorithm 2 is $O(2^{|E_c|} \cdot |X|^2 \cdot |E|)$.

The complexity of computing $\hat{E}_c$ by Algorithm 2 is exponential to $|E_c|$, which naturally leads to the question that if such a method can be further improved such that the exponentiality can be avoided. Unfortunately this is unlikely possible: we show that the *set cover problem* (SCP), whose optimization version is *NP-hard* [69], can be reduced to the problem of finding a $stop$-free event set $\hat{E}_c$ with the maximal cardinality. As a result, finding $stop$-free and feasible $\hat{E}_c$ with the maximal cardinality is also NP-hard.

Problem 3.2 (Set Cover Problem). *For a set $S = \{a_1, \ldots, a_m\}$ and a collection of subsets of S, i.e., $\mathcal{S} = \{S_1, \ldots, S_n\}$ such that $S_i \subseteq S$, find a minimal set $\mathcal{S}_{min} \subseteq \mathcal{S}$ such that $\bigcup_{S_i \in \mathcal{S}_{min}} = S$.*

Proposition 3.6. *Problem 3.2 can be reduced to the problem of finding a stop-free event set $\hat{E}_c$ with the maximal cardinality in an automaton.*

Proof. Given an instance of SCP, we construct an automaton $A_{SC} = (X, E, \delta, x_0)$ by the following procedure:

1. let $X = \{x_0, x_1, \cdots, x_m, x'\}$, $E = \mathcal{S} \cup \{e\}$;
2. for all x_i, $1 \leq i \leq m$, let $\delta(x_0, e) = x_i$;

[3]If only $q = E_c$ meets the condition of stop-freeness, then $\hat{E}_c = \emptyset$, which implies that v_0 is an undiagnosable verifier state. By Proposition 3.4, there is no $stop$-free control policy for active diagnosis.

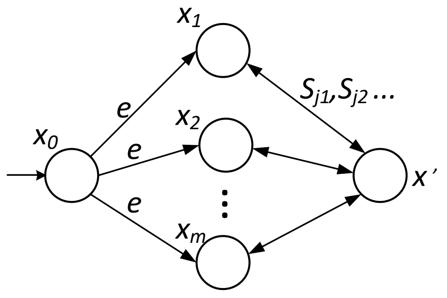


Fig. 3.7 Automaton A_{SC}.

3. for each $S_j \in \mathcal{S}$, for each $a_i \in S_j$, let $\delta(x_i, S_j) = x'$ and $\delta(x', S_j) = x_i$;

In short words, state x_0 can reach to each state x_i ($1 \leq i \leq m$) by event e, and there is a loop between each state x_i and x' labelled with S_j if and only if $a_i \in S_j$. Such a construction is shown in Figure 3.7 and is clearly polynomial. Let $E_{uc} = \{e\}$ and $E_c = E \setminus E_{uc} = \mathcal{S}$. It is not difficult to understand that a set $\mathcal{S}' \subseteq \mathcal{S}$ is a solution to Problem 3.2 if and only if $\hat{E}_c = E_c \setminus \mathcal{S}'$ is a $stop$-free set in A_{SC} with the maximal cardinality. Therefore, finding a $stop$-free $\hat{E}_c$ with the maximal cardinality is NP-hard. □

Although finding $\hat{E}_c$ with the maximal cardinality is exponential with respect to $|E_c|$ (i.e., in the worst case $2^{|E_c|}$ cases have to be tested), we believe that in general the computational load of Algorithm 2 is reasonable due to the following reasons: (i) some controllable events are removed by the pre-screening step, and (ii) in the heuristic procedure, the conditions in Proposition 3.5 are used to iteratively reduce the search space. Moreover, note that the complexity of a diagnoser is exponential with respect to the number of states in the plant and in general the number of controllable events is much less than that of states, i.e., $2^{|E_c|} \ll 2^{|X|}$. Therefore, Algorithm 2 is more efficient than active diagnosis method using diagnoser [45].

3.4 Online Active Diagnosis Method

In this section we propose an algorithm to synthesize an online control policy for active diagnosis. For a plant that is undiagnosable, the plant may enter some confused cycles and stay for an infinite long time. To prevent this from happening, different types of confused cycles can be treated differently.

1. For $\hat{E}_c$-uncontrollable confused cycles (which does not contain any event in $\hat{E}_c$ due to Definition 3.10), if the system enters such cycles, there does not exist any $stop$-free

control policy to force the plant to leave. Hence by Theorem 3.3 all undiagnosable verifier states (Definition 3.11) must not be reachable;

2. For $\hat{E}_c$-controllable confused cycles (which contain at least one event in $\hat{E}_c$), a supervisor can disable some events in $\hat{E}_c$ such that the plant will not stay in such cycles for an infinite long time.

The former condition can be enforced by a control policy such that the system should not reach a diagnostic state that contains any undiagnosable verifier state, while the latter can be enforced by the following method: if the current diagnostic state contains a verifier state v_1 that belongs to a $\hat{E}_c$-controllable confused cycle $\mathcal{C} = v_1 v_2 \cdots v_k v_1$ where event e from v_1 to v_2 belongs to $\hat{E}_c$, then by disabling event e the system is forced to leave cycle $\mathcal{C}$.

Example 3.5. *Let us consider again the plant A in Figure 3.5. Its verifier is shown in Figure 3.6, and the feasible stop-free event set is $\hat{E}_c = \{d\}$. Cycle $\mathcal{C}_1 = v_{13} \xrightarrow{b} v_{14} \xrightarrow{e} v_{13}$ is a $\hat{E}_c$-uncontrollable confused cycle since it only consists of events in $E \setminus \hat{E}_c$. By Theorem 3.3, all diagnostic states containing v_{13} and/or v_{14} must not be reachable, which means that the supervisor must disable event d at a diagnostic state y such that $v_4 \subseteq y$.*

On the other hand, cycle $\mathcal{C}_2 = v_5 \xrightarrow{b} v_6 \xrightarrow{c} v_7 \xrightarrow{d} v_5$ is a $\hat{E}_c$-controllable confused cycle since it contains event $d \in \hat{E}_c$. When the system reaches a diagnostic state y such that $v_7 \subseteq y$, the supervisor should disable event d to force the plant to leave cycle $\mathcal{C}_2$ such that the fault can be eventually detected (if it has occurred). □

Now we present Algorithm 3 to compute the online control policy. Note that control policy S is a feedback function based on the current diagnostic state y, and the control decision at diagnostic state y is $S(y) \subseteq E_c$. Algorithm 3 consists of the offline and online parts. In the offline part, a feasible $stop$-free event set $\hat{E}_c$ is first computed. Then the verifier A_v and the set of undiagnosable verifier states B are obtained by Steps 5 and 6, respectively. In Steps 7 to 16 the set of *disabled edges* $\mathcal{D}$ is computed. Set $\mathcal{D}$ contains the edges of the verifier such that they must be disabled by the supervisor. The physical meaning of an edge $(v_i, e, v_j) \in \mathcal{D}$ is that the supervisor disables event e if the closed-loop system reaches a diagnostic state y containing v_i. In Steps 7 to 10, each edge (v_i, e, v_j) is added to $\mathcal{D}$ if $e \in E_{\mathcal{C}} \cap \hat{E}_c$ and v_j is an undiagnosable verifier state. In Step 11, all states in B (i.e., the undiagnosable verifier states) are removed from A_v.

In Steps 12 to 16, a $\hat{E}_c$-controllable confused cycle $\mathcal{C}$ is found, and then one of its edges (v_i, e, v_j) with event $e \in E_{\mathcal{C}} \cap \hat{E}_c$ is added to $\mathcal{D}$. Note that in Step 12, determining one

Algorithm 3: Online control policy S

```
   Input: A plant $A = (X, E, \delta, x_0)$
   Output: Online control policy
   Offline Stage:
1  Call Algorithm 2 to compute a feasible $stop$-free event set $\hat{E}_c$;
2  if Algorithm 2 returns no solution then
3      Exit;
4  end
5  Compute the verifier $A_v = (V, E, \delta_v, v_0)$;
6  Compute the set of undiagnosable verifier state $B$ according to Definition 3.11;
7  Let $\mathcal{D} = \emptyset$;
8  foreach $v_i \in (V \setminus B), e \in \hat{E}_c$ such that $\delta_v(v_i, e) = v_j \in B$ do
9      $\mathcal{D} = \mathcal{D} \cup \{(v_i, e, v_j)\}$;
10 end
11 Remove all $v \in B$ from $A_v$;
12 while there exists a confused cycle $\mathcal{C}$ in $A_v$ such that $E_{\mathcal{C}} \cap \hat{E}_c \neq \emptyset$ do
13     Select an event $e \in E_{\mathcal{C}} \cap \hat{E}_c$ such that $\delta_v(v_i, e) = v_j$ where $v_i$, $v_j$ are two states
       in cycle $\mathcal{C}$;
14     Let $\mathcal{D} = \mathcal{D} \cup \{(v_i, e, v_j)\}$;
15     Remove edge $(v_i, e, v_j)$ from $A_v$;
16 end
   Online Stage:
17 Compute the current diagnostic state $y = y(w)$;
18 Let $S(y) = E_c$;
19 foreach $(v_i, e, v_j) \in \mathcal{D}$ do
20     if $v_i \subseteq y$ then
21         let $S(y) = S(y) \setminus \{e\}$;
22     end
23 end
24 Wait until an observable event $e$ occurs, update $w = we$, goto Step 17;
```

confused cycle $\mathcal{C}$ such that $E_{\mathcal{C}} \cap \hat{E}_c \neq \emptyset$ can be done by the following two steps: (i) remove all normal and faulty verifier states (and their corresponding edges) from A_v; (ii) determine the existence of cycles in A_v. If a cycle $\mathcal{C}$ is found in A_v, then $\mathcal{C}$ is necessarily a confused cycle such that $E_{\mathcal{C}} \cap \hat{E}_c \neq \emptyset$ (otherwise all states in $\mathcal{C}$ are undiagnosable verifier states and have already been removed due to Step 11). Both steps can be done in polynomial time [69]. Since in each loop of Steps 12 to 16 at least one edge is added to set $\mathcal{D}$, the number of executions of this loop is bounded by the total number of edges in A_v, which is polynomial with respect to the number of states of the verifier and that of events of the plant.

The online control policy is based on the set of disabled edge $\mathcal{D}$. For each observation w, the supervisor first updates its current diagnostic state $y = y(w)$. If the diagnostic state

$y(w)$ contains a verifier state v_i such that there exists an edge $(v_i, e, v_j) \in \mathcal{D}$, event e is disabled by the supervisor. Then the supervisor enforces its control policy and waits until the next observable event occurs.

Remark 3.3. *Algorithm 3 does not guarantee that the number of disabling actions, i.e., the size of the set $\mathcal{D}$, is minimal. However, a smaller set $\mathcal{D}$ does not necessarily imply that the closed-loop system has a larger reachable state space. In fact, it may happen that a small $\mathcal{D}$ is obtained with some crucial transitions whose disablement will greatly reduce the reachable state space or affect the normal functionality of the plant. As a result, additional information can be embedded into Algorithm 3 to avoid disabling these crucial transitions. For example, each event is assigned a number according to the cost of disabling it, and a set of edges with minimal sum of weights can be selected by optimization. To explore this will be part of our future work.* □

Theorem 3.5. *Given a plant $A = (X, E, \delta, x_0)$ that satisfies Assumptions 1–3, the closed-loop system (A, S) is diagnosable where S is the control policy designed by Algorithm 3.*

Proof. By contradiction, suppose that the closed-loop system (A, S) is not diagnosable. It implies that there exists a cycle of uncertain diagnostic states:

$$y_1 \xrightarrow{e_1} y_2 \cdots \xrightarrow{e_{k-1}} y_k \xrightarrow{e_k} y_1$$

such that $e_i \in S(y_i)$. Following the property of *verifiers* in [12], from the cycle we can extract a confused cycle of verifier states:

$$v_1 \xrightarrow{e_1} v_2 \cdots \xrightarrow{e_{k-1}} v_k \xrightarrow{e_k} v_1$$

where $v_i = (x_i'N, x_i''F)$ such that $v_i \subseteq y_i$. If such a cycle is $\hat{E}_c$-uncontrollable confused (i.e., all events $e_i \in E \setminus \hat{E}_c$), then all v_i's are undiagnosable verifier states, which cannot happen due to Steps 7 to 10 in Algorithm 3, guaranteeing that all undiagnosable verifier states are not reachable. On the other hand, there does not exist an event $e_i \in \hat{E}_c$ since otherwise at least one event $e_i \in \hat{\Sigma}_c$ is a disabled edge by Steps 12 to 16 in Algorithm 3, which contradicts $e_i \in S(y_i)$. Therefore we conclude that the uncertain diagnostic states of (A, S) cannot form a cycle, and hence (A, S) is diagnosable. □

Example 3.6. *Let us consider again the automaton A in Figure 3.5 and the corresponding verifier A_v in Figure 3.6. By applying Algorithm 2 a feasible stop-free event set $\hat{E}_c = \{d\}$*

is obtained. There are two confused cycles in A_v. The cycle

$$v_{13} \xrightarrow{b} v_{14} \xrightarrow{e} v_{13}$$

is $\hat{E}_c$-uncontrollable confused (i.e., $E_C \cap \hat{E}_c = \emptyset$). By Step 6 in Algorithm 3, the set of undiagnosable verifier states is $B = \{v_{12}, v_{13}, v_{14}, v_{15}, v_{16}\}$. Since v_{12} can be reached by executing event d from v_4, by Steps 8 to 10 in Algorithm 3 the edge (v_4, d, v_{12}) is added to set $\mathcal{D}$. On the other hand, the cycle

$$v_5 \xrightarrow{b} v_6 \xrightarrow{c} v_7 \xrightarrow{d} v_5$$

is $\hat{E}_c$-controllable confused since it contains $d \in \hat{E}_c$. By Steps 12 to 16 in Algorithm 3 the edge (v_7, d, v_5) is added to set $\mathcal{D}$. Finally we have $\mathcal{D} = \{(v_7, d, v_5), (v_4, d, v_{12})\}$. The online control policy can be obtained according to set $\mathcal{D}$.

Now let us consider the online control policy. Figure 3.8 depicts the closed-loop system (A, S) that runs online. The initial diagnostic state is $y_0 = y(\varepsilon) = \{(0, N)\}$. Since $v_4, v_7 \nsubseteq y_0$, no event is disabled, i.e., $S(y_0) = \{d, g\}$. By observing events a and b, the new diagnostic states are $y_1 = y(a) = \{(1, N), (5, F)\}$ and $y_2 = y(ab) = \{(2, N), (6, F)\}$, respectively, and hence $S(y_1) = S(y_2) = \{d, g\}$. By observing event c, the current diagnostic state is $y_3 = \{(3, N), (7, F)\}$. Since $v_7 \subseteq y_3$ and $(v_7, d, v_5) \in \mathcal{D}$, event d is disabled at y_3, i.e., $S(y_3) = E_c \setminus \{d\} = \{g\}$.

From diagnostic state y_3 if another event c is observed, then we conclude that fault f must have occurred. On the other hand, if we observe event g, the new diagnostic state is $y_4 = y(abcg) = \{(4, N), (0, N)\}$ by which the fault does not occur. Since $v_4 \subseteq y_4$ and $(v_4, d, v_{12}) \in \mathcal{D}$, event d is disabled at y_3, i.e., $S(y_3) = E_c \setminus \{d\} = \{g\}$. Finally we observe another event a such that the new diagnostic state is again y_1, and the corresponding control decision is $S(y_1) = \{d, g\}$. □

The following example shows the scalability of our method. The automaton in this example is modified from [64].

Example 3.7. *Consider the automaton A in Fig 3.9, where $E_o = E_c = \{a, b, c, e\}$, $E_{uo} = \{d, f\}$, and the fault event is f. Let us consider the case $n = 5$. The diagnoser of A has 85 states, while its verifier has 36 states only. Since the whole verifier is too large to be graphically depicted, only confused cycles related to our control design is presented while other unrelated parts are abstracted as dashed arrows. By Algorithm 2, we have $\hat{E}_c = \{b, c\}$. All confused cycles of the verifier are $\hat{E}_c$-controllable. The* disabled edge set

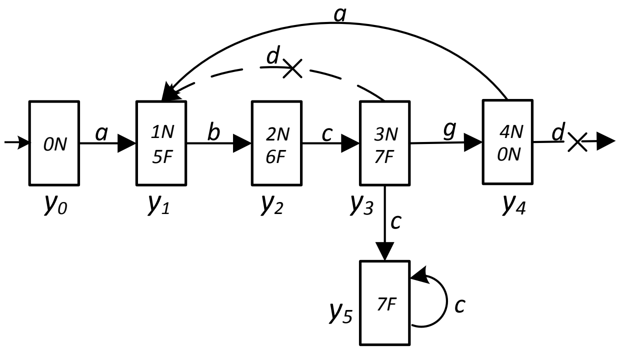


Fig. 3.8 The closed-loop system (A, S) for Example 3.6.

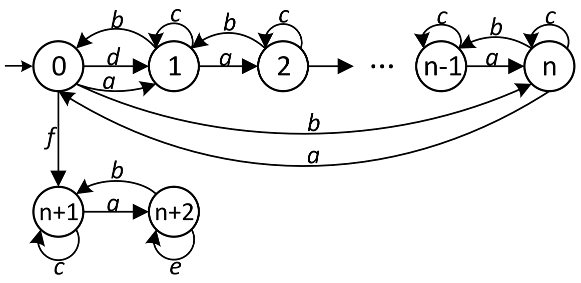


Fig. 3.9 Automaton A for Example 3.7.

$\mathcal{D} = \{(v_2, b, v_1), (v_3, c, v_3), (v_4, b, v_3), (v_5, c, v_5), (v_6, b, v_5), (v_7, c, v_7), (v_8, b, v_7), (v_9, c, v_9), (v_{10}, b, v_9), (v_{11}, c, v_{11}), (v_{12}, b, v_{11})\}$.

Moreover, if $n = 6, 7, 8$, the numbers of states of diagnoser are 169, 337, 673, respectively, while the numbers of states of verifier are 45, 55, 66, respectively[4]. *Obviously, our method has the computational advantage.* □

At the end of this section we discuss the computational complexity of Algorithm 3. In the offline stage, calling Algorithm 2 has the complexity $O(2^{|E_c|} \cdot |X|^2 \cdot |E|)$, as discussed

[4]The results are computed by using the DESUMA software.

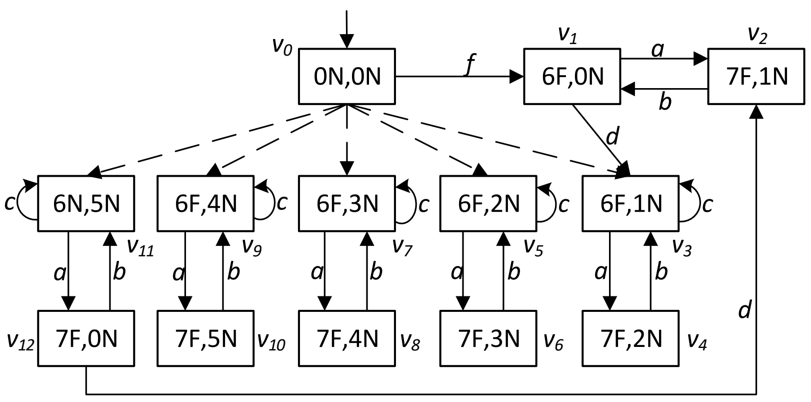


Fig. 3.10 The verifier (partly) of automaton A in Figure 3.9.

in Section 3.3. The computation of the set of undiagnosable verifier states B includes: (1) to check all elementary $\hat{E}_c$-uncontrollable confused cycles in A_v, whose complexity is $O(|V| \cdot |E|) = O(|X|^2 \cdot |E|)$; (2) to check if from each state in V some undiagnosable cycles can be reached by executing events not in $\hat{E}_c$, whose complexity is $O(|V|) = O(|X|^2)$. Therefore, the computation of set B has complexity $O(|X|^2 \cdot |E|)$. Moreover, the complexity to compute the set of disabled edges $\mathcal{D}$ is $O(|V|^2 \cdot |E|^2) = O(|X|^4 \cdot |E|^2)$. Thus, the global offline computational load is $O(2^{|E_c|} \cdot |X|^2 \cdot |E| + |X|^4 \cdot |E|^2)$. In the online stage, for any observed event the current diagnostic state is recursively updated, and the control policy is made by analyzing at set $\mathcal{D}$.

3.5 Conclusion

In this section, we summarize the contributions of this chapter as follows:

- First, we show that a closed-loop system being diagnosable does not necessarily imply the diagnosability with a more restrictive control policy. As is shown in Section 3.2, this fact indicates that the control policy at a diagnostic state cannot be correctly made locally.

- To avoid the possible silent-blocking, we introduce the notion of $stop$-free event set such that at any state of a plant at least one event that does not belong to such a set is active. A $stop$-free control policy involves only the disabling of the events in a $stop$-free event set, and hence it does not cause deadlocks.

- Several properties of $stop$-freeness are studied. For some $stop$-free event sets, there may not exist a $stop$-free control policy that enhances diagnosability. We propose a heuristic algorithm to compute a *feasible* $stop$-free event set with the maximal cardinality. Moreover, we also prove that finding such a set is *NP-hard* with respect to the number of controllable events.

- Finally, we propose an online control policy such that the closed-loop system is diagnosable. The confused cycles in the *verifier* are classified into $\hat{\Sigma}_c$*-uncontrollable cycles* and $\hat{\Sigma}_c$*-controllable cycles*. It is proved that the diagnosability of the closed-loop system is ensured by (i) preventing a plant from reaching a diagnostic state containing some verifier states that can reach $\hat{\Sigma}_c$-uncontrollable cycles by executing events not in $\hat{\Sigma}_c$, and (ii) preventing a plant from staying in some $\hat{\Sigma}_c$-controllable cycles forever by disabling some events in $\hat{\Sigma}_c$.

Chapter 4 Active Diagnosis Problem in Labeled Petri Nets

Petri net is a graphical and mathematical modeling tool with a higher modeling power than finite state automata. In Petri nets, the structure analysis and abstraction techniques can be used to reduce the computation complexity of analysis and control [17, 18, 21]. However, there are few works in the literature to deal with the active diagnosis problem in Petri nets. Based on a notion called a regulation circuit controller, an approach is presented in [70] to enforce diagnosability in interpreted Petri nets. Moreover, the nets considered in [70] are live, binary and event-detectable. Differently from the framework in [70], in this chapter, we study the active diagnosis problem in labeled Petri nets (LPNs). The active diagnosis problem in LPNs can be solved by computing the reachability graph of a plant and using the graph to design a supervisor by means of the automaton-based algorithms, e.g., [45]. However, such a method is rather inefficient since it requires a full enumeration of the reachability space of a net. An alternative approach consists in adopting state abstraction techniques such as the basis reachability graph (BRG) that has been successfully applied to fault diagnosis [21, 37, 71], prognosis [72], and marking estimation [73, 74]. Since the supervisor designed for active diagnosis may induce deadlocks in a plant and the corresponding BRG does not explicitly characterize this phenomenon, the active diagnosis problem in LPNs cannot be solved by simply applying the automaton based method in [45] to the BRG of a plant net.

In this chapter, we propose a new BRG-based structure that contains the information required to analyze the presence of deadlocks in a plant. Differently from the method in [45] where all dead states are enumerated, we do not explicitly enumerate all dead markings of the plant. Instead, for each basis marking, at most one virtual basis marking is introduced to represent all dead markings. Moreover, the supervisor designed in this chapter guarantees that the closed-loop system is deadlock-free when no fault occurs.

4.1 Active Diagnosis Problem Formulation in Labeled Petri Nets

Given an LPN $G = (PN, M_0, \Sigma, \ell)$, its unobservable transition set T_{uo} is partitioned into the *set of regular unobservable transitions* T_{reg} and the *set of fault transitions* T_f. The latter can be further partitioned into different fault classes $T_f^i (i = 1, 2, ..., r)$ that model different types of faults affecting the plant, i.e., $T_f = \bigcup_{i=1}^{r} T_f^i$. For the sake of simplicity, in this chapter we consider an LPN with a single fault class, i.e., $T_f = T_f^1$. However, our approach can be extended to the active diagnosis of nets with multiple fault classes with a slight modification of the methodology proposed in [3] for this purpose. In the sequel we use $\psi(T_f^i)$ to denote the set of all sequences in $L(PN, M_0)$ that ends with a fault transition in T_f^i, i.e., $\psi(T_f^i) = \{\sigma t_f \in L(PN, M_0) : t_f \in T_f^i\}$.

Definition 4.1. *[32] Given an LPN $G = (PN, M_0, \Sigma, \ell)$ that is deadlock-free, G is* diagnosable w.r.t. a fault class T_f^i *if for all $\sigma' \in \psi(T_f^i)$, there exists a non-negative integer $k \in \mathbb{N}$ such that for all $\sigma'\sigma'' \in L(PN, M_0)$,*

$$|\sigma''| \geq k \Rightarrow (\forall \sigma \in \ell^{-1}(\ell(\sigma'\sigma'')))(\exists t_f \in T_f^i) t_f \in \sigma. \ \square$$

In other words, given a sequence σ' that ends with a fault transition in T_f^i, let σ'' be a continuation of σ'. A labeled Petri net G is diagnosable *w.r.t. the fault class T_f^i* if when σ'' is sufficiently long (i.e., $|\sigma''| \geq k$), all possible sequences that have the same observation as that of $\sigma'\sigma''$ contain a fault transition in T_f^i. This implies that the firing of any fault transition in T_f^i can be detected within a finite number of future observations.

According to its definition, diagnosability is a behavioral property, i.e., it depends on the language generated by a plant. Assume that we are given an undiagnosable plant represented by a labeled Petri net. We consider here a problem of active diagnosis: it consists in modifying the plant's behavior by supervisory control, thus ensuring that the closed-loop system is diagnosable.

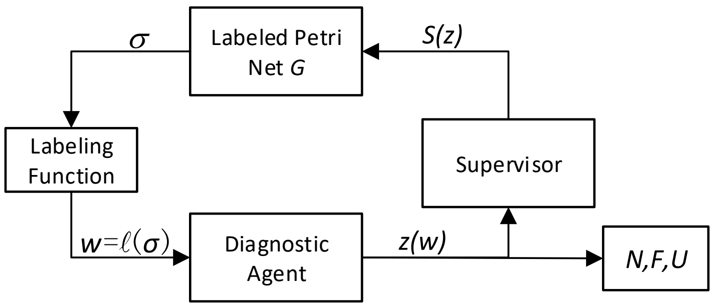


Fig. 4.1 The active diagnosis scheme.

The set-up of supervisory control in LPNs is proposed in [75]. Given an LPN $G = (PN, M_0, \Sigma, \ell)$, the set of labels Σ is partitioned into the set of controllable events (labels) Σ_c and the set of uncontrollable events Σ_{uc}, i.e., $\Sigma = \Sigma_c \cup \Sigma_{uc}$. In this chapter, we aim to design a control policy based on the current *diagnostic state*. The scheme of active diagnosis is depicted in Figure 4.1. Specifically, when a plant net generates a sequence σ, through the labeling function, a diagnostic agent observes $w = \ell(\sigma)$ and computes the corresponding diagnostic state $z(w)$ that contains information of both the current set of markings and possible fault occurrences. As discussed in Section 3.1, a diagnostic state is a set of pairs $z(w) = \{(M_1, l_1), (M_2, l_1), \ldots, (M_n, l_n)\}$, where each pair (M_i, l_i) denotes that the plant may be currently at marking M_i having previously fired a fault transition ($l_i = F$) or not ($l_i = N$). When receiving a current diagnostic state $z(w)$, a *supervisor* makes a control decision that specifies a subset of controllable events $S(z) \subseteq \Sigma_c$ to execute, while all other controllable events in $E_c \backslash S(z)$ are disabled. Note that: (i) a supervisor cannot disable any transitions with uncontrollable labels, and (ii) if a supervisor disables an event $e \in \Sigma_c$, all transitions labeled e are disabled. Here, we use (G, S) to denote the closed-loop system. The problem investigated in this paper is formulated as follows.

Problem 4.1 (Active diagnosis). *Given an undiagnosable plant modeled by an LPN G, we want to determine a control policy S for G such that the closed-loop system (G, S) is diagnosable.* □

Since in this paper we will apply the BRG approach to represent the reachability space of a net, the LPN G considered satisfies the following assumptions:

A1) G is bounded;

A2) The T_{uo}-induced subnet is acyclic.

Assumption A1 guarantees that the BRG of a plant net is always finite. Assumption A2 allows to use the state equation to characterize the implicit reach of a basis marking.

Remark 4.1. *Note that in the literature, an ILPP technique [29, 76, 77] is used to characterize the reachability set of a bounded net system, which does not rely on Assumption A2. However, in this paper the proposed supervisor is computed based on the BRG approach. Assumption A2 ensures that a BRG contains a correct abstract representation of a net reachability set.* □

4.1.1 Diagnosability of LPNs with Deadlocks

In the literature, it is commonly assumed that a plant net to be diagnosed is deadlock-free [31–33, 37]. However, the action of a supervisor for active diagnosis may create deadlocks in the closed-loop system. This happens when the closed-loop system reaches a marking while receiving a control decision that disables all plant-enabled transitions: this situation is called a *control-induced deadlock*. Therefore, to perform active diagnosis, the notion of diagnosability in LPNs needs to be generalized.

Definition 4.2. *An LPN* $G = (PN, M_0, \Sigma, \ell)$ *is* diagnosable *with respect to a set of fault transitions* T_f *if for all* $\sigma' \in \psi(T_f)$*, there exists a non-negative integer* $k \in \mathbb{N}$ *such that for all* $\sigma'\sigma'' \in L(PN, M_0)$*, the following two conditions hold:*

- *if* $|\sigma''| \leq k$ *and sequence* $\sigma'\sigma''$ *is terminal, then:*

$$\sigma \in \ell^{-1}(\ell(\sigma'\sigma'')) \wedge (\nexists t \in T)\sigma t \in L(PN, M_0) \Rightarrow (\exists t_f \in T_f)t_f \in \sigma;$$

- *if* $|\sigma''| \geq k$*, then:*

$$(\forall \sigma \in \ell^{-1}(\ell(\sigma'\sigma'')))(\exists t_f \in T_f)t_f \in \sigma. \quad \square$$

The second condition in Definition 4.2 is the classical notion of diagnosability for deadlock-free LPNs. On the other hand, the first condition means that if a plant reaches a dead marking after a fault transition has fired, then all consistent sequences that have the same observation and yield dead markings contain a fault. By generalizing the definition of diagnosability, we do not require the assumption that the original net is deadlock-free, i.e., our approach can be applied to LPNs containing deadlocks.

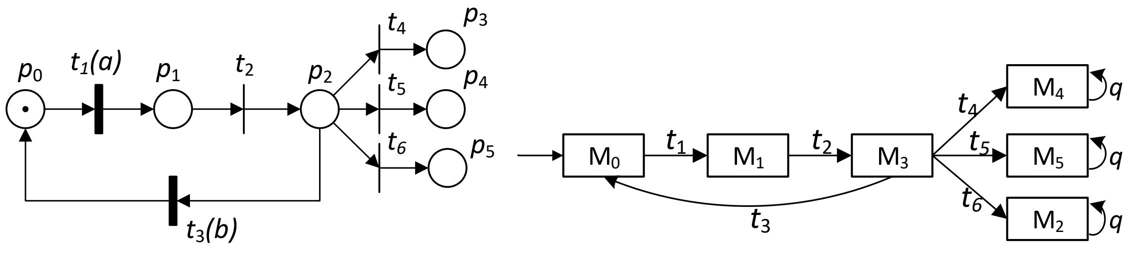
4.1.2 Quiescent Behavior and Quiescent Event

When a plant reaches a marking M that is either a dead marking or a control-induced deadlock, the system halts and no observation is produced in the future. Since in many practical cases the time needed to fire a transition has an upper bound that can be assumed to be known, if the plant does not produce any observation for a sufficiently long time, one can infer that the plant must be deadlocked at some marking[1]. To this end, deadlocks can be indirectly "observed". Such a time-out behavior is called the *quiescent behavior*, which can be encoded into the model by the following mechanism:

- Once the plant is deadlocked, it will repetitively generate a particular event q called the *quiescent event*. Event q is observable and uncontrollable;
- Event q is generated only when the plant is deadlocked, i.e., the plant does not generate event q if any transition is enabled.

Note that event q is not an event associated to a sensor signal: it is a logical event that represents the condition that a plant does not produce any observation for a sufficient long

[1]Note that a *divergent* plant [78] may similarly renounce to engage in any further communication with the environment even if not deadlocked. However, the models that we consider satisfy Assumptions A1 and A2, and thus they are necessarily divergence-free.



(a) (b)

Fig. 4.2 (a) The LPN used in Example 4.1, and (b) its reachability graph after adding the quiescent event q.

Table 4.1 The markings of the net in Figure 4.2(b).

M_0	$[1\ 0\ 0\ 0\ 0\ 0]^T$
M_1	$[0\ 1\ 0\ 0\ 0\ 0]^T$
M_2	$[0\ 0\ 0\ 0\ 0\ 1]^T$
M_3	$[0\ 0\ 1\ 0\ 0\ 0]^T$
M_4	$[0\ 0\ 0\ 1\ 0\ 0]^T$
M_5	$[0\ 0\ 0\ 0\ 1\ 0]^T$

time. We use the following example to illustrate this.

Example 4.1. *Consider the LPN in Figure 4.2(a) in which the set of observable transitions is $T_o = \{t_1, t_3\}$. By firing sequence $\sigma_1 = t_1t_2t_4$, the plant reaches a dead marking M_4 at which the plant generates the quiescent event q. Besides, the plant may also reach dead markings M_2, M_5 and then generate event q. This mechanism can be illustrated by adding a self-loop labeled by q at dead markings in the reachability graph shown in Figure 4.2(b).* □

4.2 Basis Reachability Graph with Quiescence and Q-diagnosers

4.2.1 Diagnosability of Deadlock-free LPNs and Basis Diagnosers

Given a bounded LPN, one can construct its reachability graph (RG) and use the automata approaches presented in [45] for active diagnosis. Nevertheless, the construction of the RG needs to explicitly enumerate all reachable markings of a net. In this paper, we use the notion of basis reachability graph that is a compact representation of the reachability space of a net.

In [32], the authors use a basis reachability graph (BRG) with respect to a particular partition to study the diagnosability verification problem in LPNs. Such a BRG is called a *diagnostic BRG*.

Definition 4.3. *A diagnostic BRG is a basis reachability graph with respect to a partition where $T_E = T_{uo} \cup T_f$ and $T_I = T \setminus T_E$.* □

In a diagnostic BRG, each state is a basis marking and each arc is labeled with a pair $(t, \mathbf{y})$, where $t \in T_o \cup T_f$ and $\mathbf{y}$ is a minimal explanation vector to enable t. Based on the diagnostic BRG, an automaton called a *basis reachability diagnoser* (BRD) [32] is computed and used to verify the diagnosability of the net. To compute a BRD, a series of ILPPs need to be solved to flag the occurrence of faults. In this thesis, we will also use the diagnostic BRG structure and will develop a simplified BRD structure to verify the diagnosability of an LPN. In our case, no ILPP has to be solved and the simplified diagnoser structure will be later used to design a supervisor.

Definition 4.4. *Given an LPN $G = (PN, M_0, \Sigma, \ell)$ and its diagnostic BRG $\mathcal{B} = (\mathcal{M}, Tr, \Delta, M_0)$, the* underlying automaton *of $\mathcal{B}$ is a nondeterministic finite state automaton $G_l = (\mathcal{M}, \Sigma \cup \{\varepsilon_f\}, \Delta_l, M_0)$, where:*

- *$\mathcal{M}$ is the set of states;*
- *$\Sigma \cup \{\varepsilon_f\}$ is the event set, where ε_f denotes a fault event that is unobservable.*
- *$\Delta_l \subseteq \mathcal{M} \times (\Sigma \cup \{\varepsilon_f\}) \times \mathcal{M}$ is the transition relation defined as follows: for any $M_1, M_2 \in \mathcal{M}$ and $(t, \mathbf{y}) \in Tr$,*
  $(M_1, (t, \mathbf{y}), M_2) \in \Delta \wedge \ell(t) \in \Sigma \Rightarrow (M_1, \ell(t), M_2) \in \Delta_l$,
  $(M_1, (t, \mathbf{y}), M_2) \in \Delta \wedge t \in T_f \Rightarrow (M_1, \varepsilon_f, M_2) \in \Delta_l$;
- *M_0 is the initial state.* $\square$

In other words, the *underlying automaton* of a diagnostic BRG $\mathcal{B}$, denoted by G_l, can be obtained by changing the label of each arc in $\mathcal{B}$ from $(t, \mathbf{y})$ to $\ell(t)$ (if $t \in T_o$) or ε_f (if $t \in T_f$). Now we show that the diagnosability of a deadlock-free LPN implies the diagnosability of the corresponding automaton G_l, and vice versa.

Theorem 4.1. *Given a deadlock-free LPN $G = (PN, M_0, \Sigma, \ell)$, let $G_l = (\mathcal{M}, \Sigma \cup \{\varepsilon_f\}, \delta_l, M_0)$ be the underlying automaton of its diagnostic BRG $\mathcal{B}$. The net G is diagnosable if and only if G_l is diagnosable with respect to fault ε_f.*

Proof. (If) Assume that G_l is diagnosable with respect to fault ε_f. For any sequence $\sigma_1 \varepsilon_f \in L(G_l)$, there exists $k \in \mathbb{N}$ such that by observing subsequent k events, the occurrence of the fault can be detected. According to Theorem 2.1, in net G for any sequence $\sigma_1' \in \psi(T_f)$, and all sequences $\sigma_1' \sigma_2' \in L(PN, M_0)$ such that $|\ell(\sigma_2')| \geq k$, all sequences in $\ell^{-1}(\ell(\sigma_1' \sigma_2'))$ contain a fault in T_f. Since the unobservable subnet is acyclic, the length of σ_2' is bounded, which indicates that G is diagnosable.

(Only If) If net G is not diagnosable, there exists two arbitrary long sequence $\sigma_1, \sigma_2 \in L(PN, M_0)$, such that $\ell(\sigma_1) = \ell(\sigma_2)$ and σ_1 contains a fault while σ_2 does not. By Theorem 2.1 and the definition of G_l, it is obvious that there exist two arbitrary long sequence $\sigma_1', \sigma_2' \in L(G_l)$ such that $\mathcal{P}(\sigma_1') = \mathcal{P}(\sigma_2')$ and one contains fault ε_f while the other does not. Therefore, G_l is not diagnosable. □

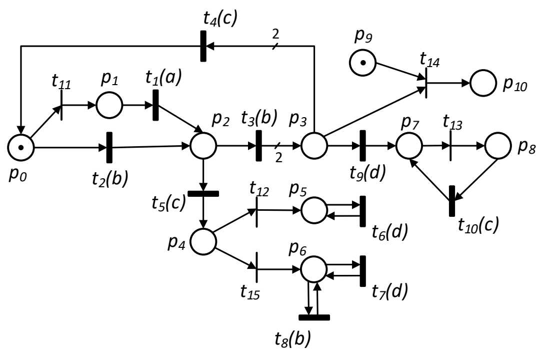
In [3], a structure called *diagnoser* are used to verify the diagnosability of an automaton. The diagnoser of an automaton can be constructed by a standard procedure [1, 3]. An important notion related to diagnosability is the *indeterminate cycle* [3]. An indeterminate cycle in a diagnoser is a cycle composed exclusively of uncertain diagnostic states for which there exist: (i) a corresponding cycle in the plant automaton involving only states tagged "N" in the cycle of diagnoser; and (ii) a corresponding cycle in the plant automaton involving only states tagged "F" in the cycle of diagnoser. Sampath *et. al* [3] show that a plant automaton is diagnosable if and only if its diagnoser does not contain any indeterminate cycle. Therefore, by Theorem 4.1, we immediately have the following corollary.

Corollary 4.1. *Given an LPN G that is deadlock-free, let G_l be the underlying automaton of its diagnostic BRG $\mathcal{B}$. The net G is diagnosable if and only if the diagnoser of G_l does not contain any indeterminate cycle.* □

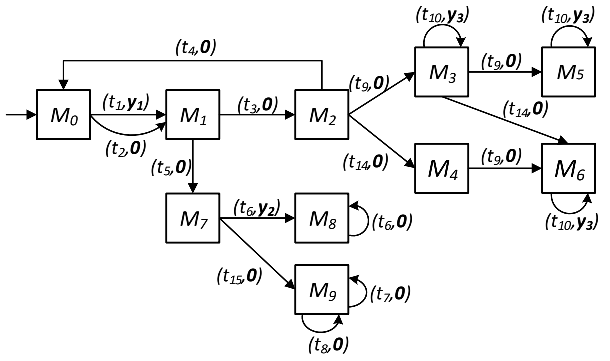
Example 4.2. *Consider the LPN in Figure 4.3(a), where $T_o = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$, $T_{uo} = \{t_{11}, t_{12}, t_{13}, t_{14}, t_{15}\}$, and $T_f = \{t_{14}, t_{15}\}$. The labeling function is defined as follows: $\ell(t_1) = a$, $\ell(t_2) = \ell(t_3) = \ell(t_8) = b$, $\ell(t_4) = \ell(t_5) = \ell(t_{10}) = c$, and $\ell(t_6) = \ell(t_7) = \ell(t_9) = d$. The diagnostic BRG of this net is shown in Figure 4.3(b). The basis markings are listed in Table 4.3. The corresponding underlying automaton G_l of the diagnostic BRG is shown in Figure 4.3(c) and its diagnoser is shown in Figure 4.3(d). There exist two indeterminate cycles in the diagnoser, i.e., $z_3 \xrightarrow{c} z_3$ and $z_6 \xrightarrow{d} z_6$. According to Corollary 4.1, the plant net is not diagnosable. In fact, the plant can generate a normal sequence $\sigma_1 = t_{11}t_1t_3t_9(t_{13}t_{10})^*$ and a faulty sequence $\sigma_2 = t_{11}t_1t_3t_{14}t_9(t_{13}t_{10})^*$ having the same observation, which means that it is not diagnosable.* □

4.2.2 Quiescent Basis Reachability Graph and Q-diagnoser

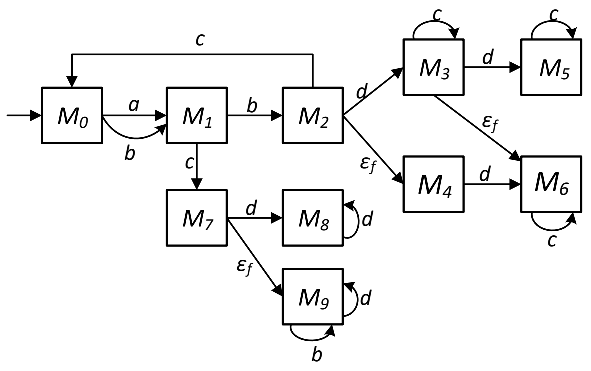
As mentioned in the previous section, a diagnostic BRG can be used to characterize the behavior of a deadlock-free LPN and to verify its diagnosability without explicitly computing the reachability graph. Since a diagnostic BRG is an abstract model of the plant



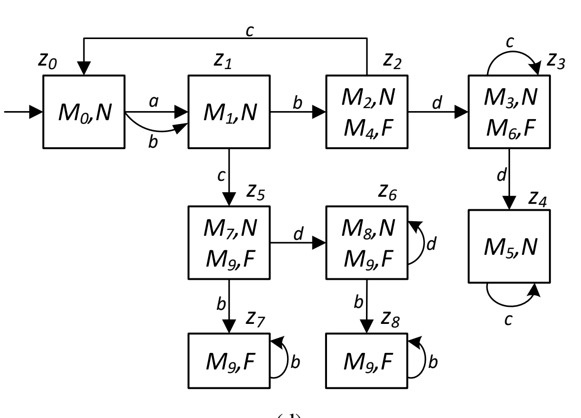
(a)



(b)



(c)



(d)

Fig. 4.3 (a) The LPN for Example 4.2, (b) its diagnostic BRG, (c) the underlying automaton of the BRG, and (d) the diagnoser of G_l.

Table 4.2 The basis markings of the BRG in Figure 4.3(b).

M_0	$[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^T$
M_1	$[0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^T$
M_2	$[0\ 0\ 0\ 2\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^T$
M_3	$[0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1\ 0]^T$
M_4	$[0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^T$
M_5	$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 0\ 1\ 0]^T$
M_6	$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1]^T$
M_7	$[0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 0]^T$
M_8	$[0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 0]^T$
M_9	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0]^T$

Table 4.3 The minimal explanation vectors of the BRG in Figure 4.3(b).

$\mathbf{y_1}$	$[1\ 0\ 0]^T$
$\mathbf{y_2}$	$[0\ 1\ 0]^T$
$\mathbf{y_3}$	$[0\ 0\ 1]^T$

LPN and does not preserve the information needed to characterize deadlocks, the automaton-based approach [45] for active diagnosis cannot be directly applied to a diagnostic BRG. In fact, a basis marking that has an outgoing arc in a diagnostic BRG does not necessarily imply that all markings reachable from it by firing regular unobservable transitions are not dead. Therefore, for the purpose of active diagnosis in LPNs with deadlocks, the structure of conventional diagnostic BRGs needs to be augmented to encode the information of deadlocks. The following proposition shows that the set of dead markings can be described by a set of linear equalities that characterize the enabling conditions of transitions.

Proposition 4.1. *Given a Petri net $PN = (P, T, Pre, Post)$, a marking $M \in R(PN, M_0)$ is dead if and only if the following constraint set, denoted by $\rho(M)$, is feasible:*

$$\rho(M) : \bigwedge_{t\in T}(\bigvee_{p\in{}^\bullet t} M(p) \leq Pre(p,t) - 1) \tag{4-1}$$

Proof. (Only If) Since M is dead, for each transition $t \in T$, there exists at least one place $p \in {}^\bullet t$ such that $M(p) \leq Pre(p,t) - 1$ holds. Therefore $\rho(M)$ is feasible.

(If) If for each transition $t \in T$ there exists at least one place $p \in {}^\bullet t$ such that $M(p) \leq Pre(p,t) - 1$ holds, transition t is not enabled at marking M, which indicates that M is a dead marking. □

The logical *OR* condition in the constraint set $\rho(M)$ in Eq. (4-1) can be converted to its equivalent conjunctive normal form by the method in [79]. Proposition 4.1 provides us a way to verify if the regular unobservable reach of a (basis) marking contains dead markings, which is stated by the following proposition. Here we denote $R_{reg}(M) = \{M' \mid M[\sigma_{reg}\rangle M', \sigma_{reg} \in T^*_{reg}\}$.

Proposition 4.2. *Given an LPN $G = (PN, M_0, \Sigma, \ell)$ and a marking M, there exists at least one dead marking in $R_{reg}(M)$ if and only if the following linear integer constraint $D(M)$ is feasible:*

$$D(M) = \begin{cases} M_d = M + C_{uo} \cdot \mathbf{y} \geq \mathbf{0}, \\ \mathbf{y} \in \mathbb{N}^{|T_{uo}|}, \\ \sum\limits_{t_f \in T_f} \mathbf{y}(t_f) = 0 \\ \rho(M_d). \end{cases} \tag{4-2}$$

Proof. (Only If) Suppose that there exists a dead marking $M_d \in R_{reg}(M)$, i.e., there exists a firing vector $\mathbf{y} \in \mathbb{N}^{|T_{uo}|}$ such that $M + C_{uo} \cdot \mathbf{y} = M_d \geq \mathbf{0}$ and $\mathbf{y}(t_f) = 0$ for all $t_f \in T_f$. By Proposition 4.1, marking M_d satisfies $\rho(M_d)$. Therefore, ILPP (4-2) is feasible.

(If) Suppose that ILPP (4-2) is feasible. By Assumption 2, the T_{uo}-induced subnet is acyclic. If there exists a firing vector $\mathbf{y} \in \mathbb{N}^{|T_{uo}|}$ such that $M + C_{uo} \cdot \mathbf{y} = M_d \geq \mathbf{0}$ and $\mathbf{y}(t_f) = 0$ for all $t_f \in T_f$, then there necessarily exists a firing sequence $\sigma \in T_{reg}^*$ such that $M[\sigma\rangle M_d$ whose firing vector is $\mathbf{y}$. Since M_d satisfies $\rho(M_d)$, by Proposition 4.1, M_d is a dead marking. □

Note that the dead markings in $R_{reg}(M_i)$ may not be unique. However, we will shortly see that for the purpose of active diagnosis, it is sufficient to use a single virtual marking $M_{i,d}$ to denote the existence of some dead markings in $R_{reg}(M_i)$ without explicitly enumerating them. In the following, we introduce a structure called a *quiescent-BRG* (QBRG) that is an augmented basis reachability graph in which the information of the quiescent behavior of a plant net is encoded.

Definition 4.5. *Given an LPN $G = (PN, M_0, \Sigma, \ell)$, let $G_l = (\mathcal{M}, \Sigma \cup \{\varepsilon_f\}, \Delta_l, M_0)$ be the underlying automaton of its diagnostic BRG. The* quiescent-BRG *(QBRG) of G is a nondeterministic finite state automaton $G_q = (\mathcal{M}_q, \Sigma_q, \Delta_q, M_0)$, where:*

- *the state set $\mathcal{M}_q = \mathcal{M} \cup \{M_{i,d} \mid M_i \in \mathcal{M}, D(M_i) \text{ is feasible}\}$;*
- *$\Sigma_q = \Sigma \cup \{\varepsilon_f, q\}$ is the event set;*
- *the transition relation Δ_q is defined as follows:*

$$\Delta_q = \Delta_l \cup \{(M_i, q, M_{i,d}) \mid M_i \in \mathcal{M}, D(M_i) \text{ is feasible}\}$$
$$\cup \{(M_{i,d}, q, M_{i,d}) \mid M_{i,d} \in \mathcal{M}_q \setminus \mathcal{M}\}$$

- *M_0 is the initial state.* □

Given an LPN, Algorithm 4 can be used to compute its QBRG. The difference between the QBRG and the diagnostic BRG of an LPN can be explained as follows. For each basis marking M_i in a diagnostic BRG, if constraint $D(M_i)$ is feasible, then virtual basis marking $M_{i,d}$ is added with arcs $M_i \xrightarrow{q} M_{i,d}$ and $M_{i,d} \xrightarrow{q} M_{i,d}$. Again, we note that $M_{i,d}$ is not a real marking of a Petri net: it is just a modeling primitive to denote the existence of some dead markings in $R_{reg}(M_i)$. However, for simplicity, we also call $M_{i,d}$ a "basis marking" by omitting the term "virtual", since there will be no confusion. If an LPN is deadlock-free, then its QBRG is identical to the underlying automaton of its diagnostic BRG.

Algorithm 4: Computation of QBRG G_q.

```
   Input: An LPN $G = (PN, M_0, \Sigma, \ell)$
   Output: $G_q = (\mathcal{M}_q, \Sigma \cup \{\varepsilon_f, q\}, \Delta_q, M_0)$
1  compute the diagnostic BRG $\mathcal{B}$ of the LPN $G$;
2  compute the underlying automaton $G_l = (\mathcal{M}, \Sigma \cup \{\varepsilon_f\}, \Delta_l, M_0)$ of $\mathcal{B}$;
3  let $\mathcal{M}_q = \mathcal{M}$, $\Delta_q = \Delta_l$;
4  for each basis marking $M_i \in \mathcal{M}$ do
5      if $D(M_i)$ is feasible then
6          let $\mathcal{M}_q = \mathcal{M}_q \cup \{M_{i,d}\}$;
7          let $\Delta_q = \Delta_q \cup \{(M_i, q, M_{i,d})\} \cup \{(M_{i,d}, q, M_{i,d})\}$;
8      end
9  end
10 output $G_q = (\mathcal{M}_q, \Sigma \cup \{\varepsilon_f, q\}, \Delta_q, M_0)$;
```

In the following, we introduce a new projection function $\mathcal{P}' : T^* \to (\Sigma \cup \{\varepsilon_f\})^*$, defined as follows:

$$\begin{cases} \mathcal{P}'(t) = e \ \text{ if } t \in T_o,\ \ell(t) = e; \\ \mathcal{P}'(t) = \varepsilon \ \text{ if } t \in T_{reg}; \\ \mathcal{P}'(t) = \varepsilon_f \ \text{ if } t \in T_f; \\ \mathcal{P}'(\sigma t) = \mathcal{P}'(\sigma)\mathcal{P}'(t) \ \text{ if } \sigma \in T^*,\ t \in T. \end{cases} \tag{4-3}$$

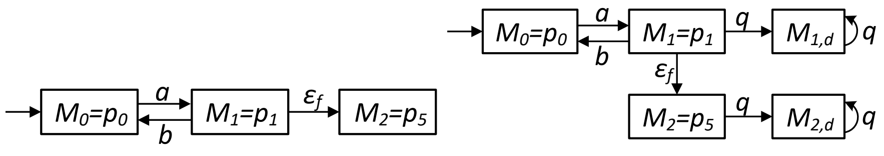
Theorem 4.2. *Consider an LPN $G = (PN, M_0, \Sigma, \ell)$ and its QBRG $G_q = (\mathcal{M}_q, \Sigma_q, \Delta_q, M_0)$. There exists a sequence $\sigma \in L(PN, M_0)$ satisfying $\mathcal{P}'(\sigma) = w$ and $M_0[\sigma\rangle M$ where M is a dead marking if and only if in the QBRG G_q there exists a path: $M_0 \xrightarrow{w} M_i \xrightarrow{q} M_{i,d}$ such that $M \in R_{reg}(M_i)$.*

Proof. (Only If) Assume that there exists a sequence $\sigma \in L(PN, M_0)$ reaching to a dead marking M. Let G_l be the underlying automaton of the diagnostic BRG. By Theorem 2.1, in G_l there exists a path labeled by $w = \mathcal{P}'(\sigma)$ leading to a basis marking M_i and $M \in R_{reg}(M_i)$. By the definition of G_q, there exists a path $M_0 \xrightarrow{w} M_i \xrightarrow{q} M_{i,d}$ in G_q such that

$M \in R_{reg}(M_i)$.

(If) Assume that there exits a path $M_0 \xrightarrow{w} M_i \xrightarrow{q} M_{i,d}$ in G_q. We can infer that in G_l there exists a path $M_0 \xrightarrow{w} M_i$ and $D(M_i)$ is feasible, which implies that there exists a dead marking $M \in R_{reg}(M_i)$. By Theorem 2.1, there exists a sequence $\sigma \in L(PN, M_0)$ such that $M_0[\sigma\rangle M$ and $\mathcal{P}'(\sigma) = w$. □

Example 4.3. *Consider the LPN in Figure 4.2(a), where $T_o = \{t_1, t_3\}$ and $T_f = \{t_6\}$. The underlying automaton G_l of its diagnostic BRG is shown in Figure 4.4(a). By applying Algorithm 4, its QBRG is depicted in Figure 4.4(b).*



(a) (b)

Fig. 4.4 (a) The structure G_l for Example 4.3, and (b) the corresponding QBRG G_q.

Since there exists a path $M_0 \xrightarrow{aq} M_{1,d}$ in G_q, By Theorem 4.2, the plant can reach to a dead marking $M \in R_{reg}(M_1)$. In fact, there are two dead markings, i.e., $[0,0,0,1,0,0]^T$ and $[0,0,0,0,1,0]^T$, in the regular unobservable reach of basis marking M_1. □

Given a QBRG G_q, let $A_d = (Z, \Sigma \cup \{q\}, \delta_d, z_0)$ be the diagnoser of G_q. A_d is called a Q-diagnoser that can be computed by the standard diagnoser construction [1, 3]. The following theorem provides us a way to verify the diagnosability of an LPN that contains deadlocks by using its Q-diagnoser A_d.

Theorem 4.3. *Given an LPN $G = (PN, M_0, \Sigma, \ell)$ and its Q-diagnoser $A_d = (Z, \Sigma \cup \{q\}, \delta_d, z_0)$, G is diagnosable if and only if A_d does not contain any indeterminate cycle.*

Proof. By Corollary 4.1, there does not exist an undiagnosable non-terminal sequence if and only if in the Q-diagnoser A_d there does not exist an indeterminate cycle labeled by $w \in E^*$. Now we prove that there does not exist an undiagnosable terminal sequence if and only if in the Q-diagnoser A_d there does not exist an indeterminate cycle labeled by event q.

(Only If) Let σ be a faulty sequence in $L(PN, M_0)$ that leads to a dead marking M. If σ does not meet the first condition in Definition 4.2, then there exists another non-faulty sequence σ' that yields a dead marking M' such that $\ell(\sigma) = \ell(\sigma') = w$. This indicates that by observing wq the corresponding diagnostic state $z = \delta_d^*(z_0, wq)$ necessarily contains two pairs (M'_d, N) and (M_d, F). Since by the construction of the Q-diagnoser, $\delta_d(z, q) = z$ holds, there exists an indeterminate cycle at diagnostic state z labeled by q, i.e., $z \xrightarrow{q} z$.

(If) Suppose that the Q-diagnoser contains an indeterminate cycle $z \xrightarrow{q} z$ such that z contains two pairs (M_d, F) and (M_d', N). It means that there necessarily exist two dead markings M and M' that can be reached form the initial marking M_0 by firing a faulty sequence σ and a non-faulty sequence σ', respectively. Since sequences σ and σ' have the same observation, the first condition in Definition 4.2 is not satisfied. Therefore, the statement holds. □

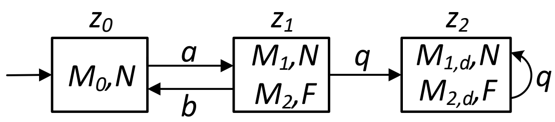


Fig. 4.5 The Q-diagnoser A_d of the plant in Figure 4.2(a).

Example 4.4. *Consider the net in Figure 4.2(a), where $T_o = \{t_1, t_3\}$ and $T_f = \{t_6\}$. Its QBRG G_q is visualized in Figure 4.4(b). Thanks to G_q, its Q-diagnoser A_d is shown in Figure 4.5. Since in A_d there exists an indeterminate cycle $z_2 \xrightarrow{q} z_2$, by Theorem 4.3, the plant is not diagnosable. In fact, the plant can fire normal sequence $\sigma_1 = t_1t_2t_4$ and faulty sequence $\sigma_2 = t_1t_2t_6$ that are both dead and have the same observation, which violates the first condition in Definition 4.2.* □

4.3 Diagnosability Enforcing Supervisor

In this section, we develop a method to design a supervisor for the active diagnosis of a plant LPN. By Theorem 4.3, a plant LPN is undiagnosable if and only if its Q-diagnoser contains indeterminate cycles. Therefore, to enforce diagnosability, a supervisor must forbid all behaviors of the plant that may evolve along those indeterminate cycles in the Q-diagnoser. Given a Q-diagnoser, indeterminate cycles are classified into two types by the controllability of events in them:

- for an indeterminate cycle that contains controllable events, at least one of these controllable events has to be disabled to prevent the plant circulating in this cycle for infinite times;
- for an indeterminate cycle that does not contain any controllable event, all diagnostic states in it (and all diagnostic states that may uncontrollably reach it) must be forbidden.

As mentioned in Section 4.1, the supervisor makes control decisions from the knowledge of the consistent diagnostic state obtained by the Q-diagnoser. More precisely, for an observation $w \in (\Sigma \cup \{q\})^*$ whose consistent diagnostic state is z_i, the control decision at state z_i is $S(z_i) = \Sigma_c \setminus \Sigma_{d,i}$ where $\Sigma_{d,i}$ is the set of disabled events at diagnostic state z_i. Given a set $\Sigma_{d,i}$, we use $T_{d,i}$ to denote the corresponding set of disabled transitions, i.e., $T_{d,i} = \{t \in T \mid \ell(t) \in \Sigma_{d,i}\}$.

In automata, the two control specifications mentioned above can be easily enforced by inspecting the plant automaton [45]. However, a Q-diagnoser is an abstract model of the plant LPN, in which the firing of implicit transitions is omitted. Hence, a control decision at a diagnostic state z may result in deadlocks that are not explicitly represented in the Q-diagnoser. Therefore, the method to trim a diagnoser in automata in [45] cannot be applied to trim a Q-diagnoser. To detect if there exists a control-induced deadlock at a diagnostic state z, we rewrite Eqs. (4-1) and (4-2) as the following Eqs. (4-4) and (4-5), respectively.

$$\rho'(M) : \bigwedge_{t \in T \setminus T_{d,i}} \left(\bigvee_{p \in {}^{\bullet}t} M(p) \leq Pre(p,t) - 1 \right) \tag{4-4}$$

$$D'(M) = \begin{cases} M_d = M + C_{uo} \cdot \mathbf{y} \geq 0, \\ \mathbf{y} \in \mathbb{N}^{|T_{uo}|}, \\ \sum\limits_{t_f \in T_f} \mathbf{y}(t_f) = 0, \\ \rho'(M_d). \end{cases} \tag{4-5}$$

Comparing with Eq. 4-1, in Eq. 4-4 the token-disabling constraints for control-disabled transitions (i.e., transition in $T_{d,i}$) are removed. On the other hand, Eq. (4-5) is analogous to Eq. (4-2) while the constraint ρ from Eq. (4-1) is replaced by ρ' from Eq. (4-4). Hence, if Eq. (4-5) is feasible for a marking M, by firing regular unobservable transitions, some marking M_d is reached from M such that all transitions are either control-disabled or lack of tokens to fire, and vice versa.

Proposition 4.3. *Given an LPN $G = (PN, M_0, \Sigma, \ell)$ with $\Sigma = \Sigma_c \cup \Sigma_{uc}$ and its Q-diagnoser $A_d = (Z, \Sigma \cup \{q\}, \delta_d, z_0)$, suppose that the control decision at diagnostic state z_i is $S(z_i) = \Sigma_c \setminus \Sigma_{d,i}$. For a basis marking M such that $(M, l) \in z_i$, there exists at least one marking $M_d \in R_{reg}(M)$ at which no transition can fire if and only if constraint $D'(M)$ in Eq.* (4-5) *is feasible.*

Proof. (Only If) Suppose that after disablement of transitions in $T_{d,i}$, there exists a dead marking $M_d \in R_{reg}(M)$, i.e., there exists a firing vector $\mathbf{y} \in \mathbb{N}^{|T_{uo}|}$ such that $M + C_{uo} \cdot \mathbf{y} =$

$M_d \geq \mathbf{0}$ and $\mathbf{y}(t_f) = 0$ for all $t_f \in T_f$. Moreover, marking M_d satisfies $\rho'(M_d)$. Therefore, ILPP 4-5 is feasible.

(If) Suppose that ILPP 4-5 is feasible. By Assumption 2, the T_{uo}-induced subnet is acyclic. If there exists a firing vector $\mathbf{y} \in \mathbb{N}^{|T_{uo}|}$ such that $M + C_{uo} \cdot \mathbf{y} = M_d \geq \mathbf{0}$ and $\mathbf{y}(t_f) = 0$ for all $t_f \in T_f$, then there exists a firing sequence $\sigma \in T_{ref}^*$ such that $M[\sigma\rangle M_d$ whose firing vector is $\mathbf{y}$. Since M_d satisfies $\rho'(M_d)$, at marking M_d all transitions are either control-induced or lack of tokens to fire. Therefore, M_d is a dead marking. □

Example 4.5. *Consider the plant in Figure 4.3(a), where $\Sigma_c = \{c, d\}$. Its Q-diagnoser is shown in Figure 4.3(c). Suppose that a supervisor disables event c at diagnostic state z_3, i.e., the disabled transition set $T_{d,3} = \{t_4, t_5, t_{10}\}$. Since there are two pairs (M_3, N) and (M_6, F) in state z_3, according to Proposition 4.3 markings M_3 and M_6 need to be considered. For marking M_3, constraint $D'(M_3)$ is not feasible, which means that if the plant is at a marking in the regular unobservable reach of M_3, by disabling event c, there is no control-induced deadlock. On the other hand, constraint $D'(M_6)$ is feasible, which means that if the plant is at a marking in the regular unobservable reach of M_6, by disabling event c, the plant can reach some dead marking in $R_{reg}(M_6)$.* □

In reality, a plant is expected to be deadlock-free, since an unexpected deadlock may greatly reduce the rate of productivity (e.g., long down-time and low use of some critical and expensive resources) or even cause severe consequence [80–82]. On the other hand, if a fault has occurred, then a deadlock, i.e., a "planned shutdown", is usually harmless and acceptable, since the operator of a plant may examine the plant when it is offline and initiate a recovering process to repair the fault. Therefore, in this section we aim to design a supervisor for active diagnosis which meets the two criteria:

1. the closed-loop system is diagnosable, i.e., the firing of fault transitions can be detected in finite future steps;
2. the closed-loop system is not deadlocked when no fault transition has fired.

To prevent the plant from reaching unfaulty deadlocks, in the following we introduce a notion called a *q-normal cycle*.

Definition 4.6. *Let $A_d = (Z, \Sigma \cup \{q\}, \delta_d, z_0)$ be the Q-diagnoser of an LPN G. A cycle $\mathcal{C}$: $z \xrightarrow{q} z$ in A_d is called a q-normal cycle if for all $(M_i, l_i) \in z, l_i = N$ holds.* □

In other words, a cycle $\mathcal{C} : z \xrightarrow{q} z$ is *q-normal* if at diagnostic state z no fault transition has fired. As we have discussed at the beginning of this section, to guarantee diagnosability a supervisor should prevent the plant from circulating in any indeterminate cycle. Besides, to guarantee that the plant is not deadlocked when no fault transition has fired, a supervisor should also prevent the closed-loop system reaching any q-normal cycle.

In the following, we propose Algorithm 5 to design a supervisor for a given labeled Petri net plant such that the closed-loop system is diagnosable and cannot reach deadlock if no fault occurs. Algorithm 5 recursively trims the Q-diagnoser by eliminating all indeterminate cycles and q-normal cycles in it. In Algorithm 5, we use $E_{\mathcal{C}}$ to denote the set of events in a cycle $\mathcal{C}$ in A_d and use set Z_{dis} to denote the set of diagnostic states at which the supervisor makes disablement actions.

We explain how Algorithm 5 works step-by-step. Steps 1 and 2 compute the QBRG and the Q-diagnoser A_d, respectively. If there exists an indeterminate cycle or a q-normal cycle $\mathcal{C}$ that contains only uncontrollable events and can be reached from the initial diagnostic state z_0 by executing a sequence of uncontrollable events, then there does not exist a supervisor that meets the two criteria. In such a case, the algorithm terminates.

The main body of Algorithm 5 consists of two parts. In the first part (Steps 4 to 16) an indeterminate cycle or a q-normal cycle $\mathcal{C}$ is found and treated. If $\mathcal{C}$ contains at least one arc labeled by a controllable event $e \in \Sigma_c$, by Steps 7 to 10 one of such arcs, denoted by (z_i, e), is put into set $\mathcal{D}$, meaning that event e is disabled at diagnostic state z_i. On the other hand, if $\mathcal{C}$ contains no controllable events, then in Steps 12 to 14 all controllable arcs leading to some states that may uncontrollably reach $\mathcal{C}$ are put in $\mathcal{D}$. Steps 15 and 16 remove the unreachable states and the corresponding arcs in $\mathcal{D}$ which are no longer necessary.

Once the control policy is updated, the second part of the algorithm (Steps 17 to 26) updates the information of control-induced deadlock accordingly. For each diagnostic state $z_i \in Z_{dis}$, Steps 18 and 19 compute the disabled event set $\Sigma_{d,i}$ and the disabled transitions set $T_{d,i}$ at state z_i, respectively. By Step 20, for each pair $(M_j, l_j) \in z_i$, ILPP 4-5 is solved to test if the disablement of transitions in $T_{d,i}$ will block the plant at some marking in $R_{reg}(M_j)$. In Steps 22 to 26 a new diagnostic state is added to Z that contains all $(M_{j,d}, l_j)$ from all (M_j, l_j) in z_i such that ILPP 4-5 is feasible. The above procedures (Steps 4 to 27) are iteratively done until all indeterminate cycles and q-normal cycles have been removed. In Algorithm 5, we separate the function that updates the Q-diagnoser (Steps 17 to 26, which adds q-cycles) and the function that trims the Q-diagnoser (Steps 7 to 14). The reason for this is to modularize the algorithm and improve its readability.

Finally, we discuss the complexity of the proposed approach. Consider an LPN $G = (PN, M_0, \Sigma, \ell)$ whose diagnostic BRG is $\mathcal{B} = (\mathcal{M}, Tr, \Delta, M_0)$ with the QBRG G_q. Since

Algorithm 5: Computation of an active diagnosis supervisor

```
Input: A labeled Petri net $G = (PN, M_0, \Sigma, \ell)$, where $\Sigma = \Sigma_c \cup \Sigma_{uc}$
Output: Diagnosability enforcing supervisor $S$.
1  compute the QBRG $G_q$ using Algorithm 4;
2  compute the Q-diagnoser $A_d = (Z, \Sigma \cup \{q\}, \delta_d, z_0)$ that is the diagnoser of $G_q$;
3  let $\mathcal{D} = \emptyset$, $Z_{dis} = \emptyset$;
4  while there exists an indeterminate cycle or a q-normal cycle $\mathcal{C}$ in $A_d$ do
5      if $\Sigma_{\mathcal{C}} \cap \Sigma_c = \emptyset \wedge (\exists \sigma \in \Sigma_{uc}^*)\delta_d(z_0, \sigma) = z' \in \mathcal{C}$ then
6          output: No solution, END;
7      end
8      if $\Sigma_{\mathcal{C}} \cap \Sigma_c \neq \emptyset$ then
9          select $e \in \Sigma_{\mathcal{C}} \cap \Sigma_c$ such that $\delta_d(z_i, e) = z'$, $z_i, z' \in \mathcal{C}$;
10         let $\mathcal{D} = \mathcal{D} \cup \{(z_i, e)\}$, $Z_{dis} = \{z_i\}$;
11         remove $\delta_d(z_i, e)$ from $\delta_d$;
12     else
13         for each $z_i \in Z$, $e \in \Sigma_c$, such that $\delta_d(z_i, e) = z'$ and $\exists \sigma \in \Sigma_{uc}^*$ such that
           $\delta_d(z', \sigma) = z'' \in \mathcal{C}$ do
14             let $\mathcal{D} = \mathcal{D} \cup \{(z_i, e)\}$, $Z_{dis} = Z_{dis} \cup \{z_i\}$;
15             remove $\delta_d(z_i, e)$ from $\delta_d$;
16         end
17     end
18     remove all unreachable states from $Z$;
19     remove $(z, e)$ from $\mathcal{D}$ if $z \notin Z$ ;
20     for each $z_i \in Z_{dis}$ do
21         let $\Sigma_{d,i} = \{e \mid \exists (z_i, e) \in \mathcal{D}\}$;
22         compute the disabled transitions set $T_{d,i}$;
23         for each $(M_j, l_j) \in z_i$ do
24             if $D'(M_j)$ is feasible then
25                 if $\exists z \in Z$, such that $\delta_d(z_i, q) = z$ then
26                     let $z = z \cup \{(M_{j,d}, l_j)\}$
27                 else
28                     let $z = \{(M_{j,d}, l_j)\}$, $Z = Z \cup \{z\}$;
29                     let $\delta_d(z_i, q) = z$, $\delta_d(z, q) = z$;
30                 end
31             end
32         end
33     end
34     let $Z_{dis} = \emptyset$
35 end
36 output $S = (Z, \Sigma \cup \{q\}, \delta_d, z_0)$.
```

in QBRG G_q each basis marking is associated with at most one virtual basis marking, there are at most $2|\mathcal{M}|$ states in G_q, i.e., the structural complexity of G_q is $O(2|\mathcal{M}|)$. On the other hand, both Q-diagnoser A_d and active diagnosis supervisor S contain two types of nodes: (1) basis marking nodes, i.e., $z = \{(M_1, l_1), (M_2, l_2), \ldots, (M_n, l_n)\}$, where $M_i \in \mathcal{M}$; and (2) virtual basis marking nodes, i.e., $z' = \{(M_{1,d}, l_1), (M_{2,d}, l_2), \ldots, (M_{n,d}, l_n)\}$. Since the number of each type of nodes is at most $2^{2|\mathcal{M}|}$, both A_d and S contain at most $2 \cdot 2^{2|\mathcal{M}|}$ states, i.e., their structural complexity is $O(2^{2|\mathcal{M}|})$. Such exponential complexity of $|\mathcal{M}|$ seems unavoidable due to the construction of the diagnoser of QBRG. However, it has been acknowledged that in practice the number of basis markings is usually much smaller than the markings in the reachability graph [32, 61]. Thus, our method is practically more efficient than automaton-based methods (such as [45]).

Remark 4.2. *Note that in general there exist multiple ways to break an indeterminate cycle with controllable events in a Q-diagnoser. It may happen that the control action prunes some crucial arcs such that the state space is greatly reduced or the normal functionality is greatly affected. As a result, some additional information can be embedded into Algorithm 5 to avoid removing these crucial arcs. To explore this will be part of our future work.* □

Example 4.6. *Consider again the LPN in Figure 4.3(a) where $T_o = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10}\}$ and $T_f = \{t_{14}, t_{15}\}$. The controllable event set $\Sigma_c = \{c, d\}$. Its Q-diagnoser is shown in Figure 4.3(d). In the Q-diagnoser there exist two indeterminate cycles: $z_3 \xrightarrow{c} z_3$ and $z_6 \xrightarrow{d} z_6$.*

In the first iteration, indeterminate cycle $z_3 \xrightarrow{c} z_3$ is picked. Since event c is controllable, by Step 9 event c is disabled at state z_3, i.e., $\mathcal{D} = \{(z_3, c)\}$ and $Z_{dis} = \{z_3\}$. For diagnostic state z_3, the disabled event set $\Sigma_{d,3} = \{c\}$ and the disabled transition set is $T_{d,3} = \{t_4, t_5, t_{10}\}$. There are two pairs (M_3, N) and (M_6, F) in state z_3. For marking M_6, constraint $D'(M_6)$ is feasible, and according to Step 25, a new state $z_9 = (M_{6,d}, F)$ is added to the state set. By Step 26, an arc labeled event q from state z_3 to z_9 and a self-loop labeled event q at state z_9 are also added.

In the second iteration, indeterminate cycle $z_6 \xrightarrow{d} z_6$ is picked. Since event d is controllable, we have $\mathcal{D} = \{(z_3, c), (z_6, d)\}$ and $Z_{dis} = \{z_6\}$. By Steps 17 to 26, diagnostic state $z_6 = \{(M_4, N), (M_5, F)\}$ is examined to update the quiescent behavior after disabling event d at it. The trimmed Q-diagnoser A'_d is shown in Figure 4.6(a).

Since the structure A'_d contains a new q-normal cycle, i.e., $z_{10} \xrightarrow{q} z_{10}$, it is then trimmed in the third iteration. Since event q is uncontrollable, according to steps 12 to 14, arc (z_5, d) is added to set $\mathcal{D}$, implying that the supervisor should disable event d at state z_5, i.e.,

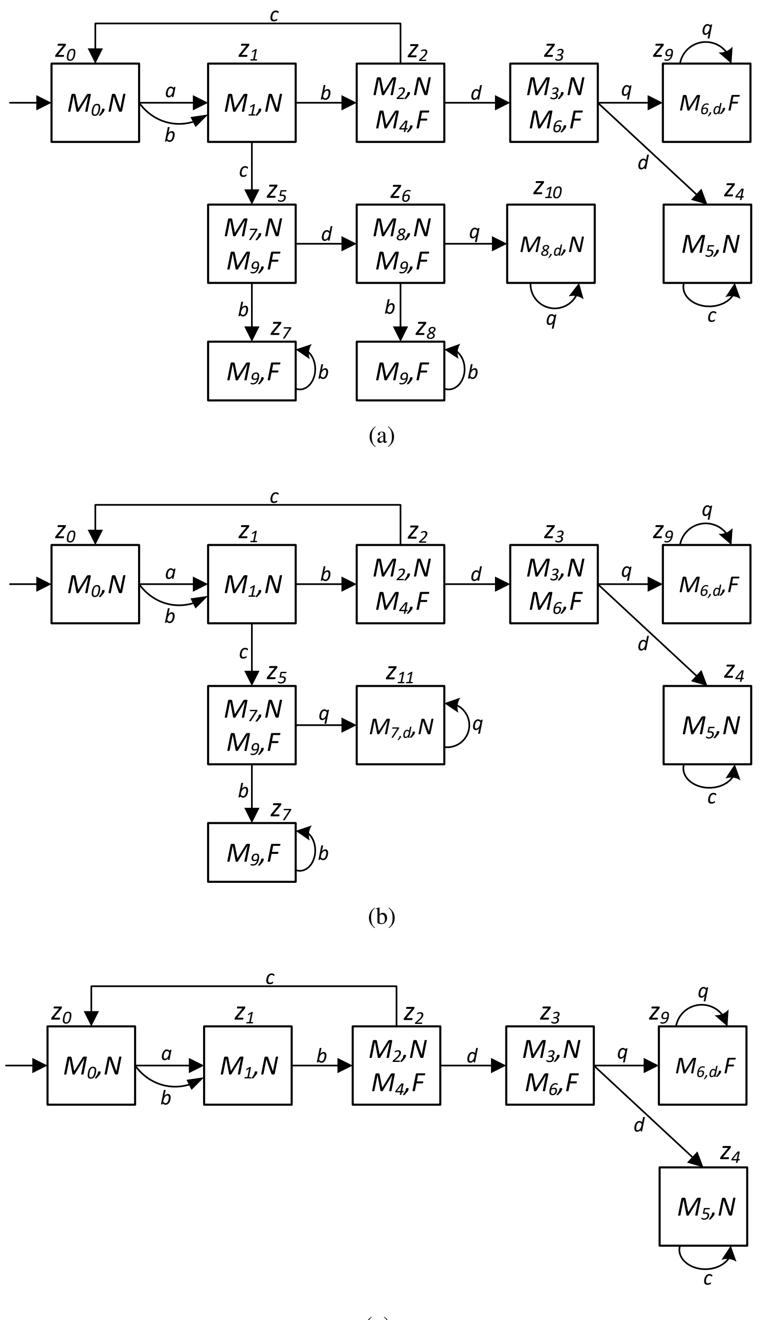


Fig. 4.6 (a) The trimmed structure A'_d in Example 4.6, (b) the trimmed structure A''_d, and (c) the diagnosability enforcing supervisor S.

$\mathcal{D} = \{(z_3, c), (z_6, d), (z_5, d)\}$. Step 15 removes the unreachable states z_6, z_8, z_{10} and Step 16 removes arc (z_6, d) from set $\mathcal{D}$ which is no longer necessary. Steps 17 to 26 update the quiescent behavior at state z_5. For a marking M_7 in z_5, the constraint $D'(M_7)$ is feasible. In Step 25, a new state $z_{11} = \{(M_{7,d}, N)\}$ is added to state set and a self-loop labeled event q is added at state z_{11}, which is a new q-normal cycle, i.e., $z_{11} \xrightarrow{q} z_{11}$. The current structure A''_d is shown in Figure 4.6(b).

Finally, in the fourth iteration, to eliminate the q-normal cycle $z_{11} \xrightarrow{q} z_{11}$, the event c at state z_1 is disabled and arc (z_1, c) is added to the set $\mathcal{D}$. After updating the quiescent behavior at state z_1, the final result is shown in Figure 4.6(c). Since there does not exist any indeterminate cycle and q-normal cycle in the current structure, Algorithm 2 terminates and outputs the finial structure in Figure 4.6(c). □

Once $S = (Z, \Sigma \cup \{q\}, \delta_s, z_0)$ is obtained, the corresponding control policy is given as follows. Given an observation w whose consistent diagnostic state in diagnosability enforcing supervisor is z, the control decision $S(z)$ is to permit all controllable events that are defined at z, i.e.,

$$S(z) = \{e \in \Sigma_c \mid \delta_s(z, e) \text{ is defined}\}. \tag{4-6}$$

For example, for the net in Example 4.2, for an observation $w = abd$ whose consistent diagnostic state is z_3, according to the diagnosability enforcing supervisor in Figure 4.6(c), the control decision is $S(z_3) = \{d\}$, i.e., event c is disabled.

Remark 4.3. *It is worth noting that the observable quiescence q provides us extra information about the current marking and the fault status of the system. Consider, for instance, the Q-diagnoser in Figure 4.6(a) in which the supervisor disables event d at diagnostic state z_6. When observing event sequence acd, one can infer that the plant may be at some marking either in the regular unobservable reach of M_8 while no fault transition has fired, or in the regular unobservable reach of M_9 while a fault transition has fired. However, by further observing quiescent event q and inspecting the new consistent diagnostic state z_{10}, one can infer that the system must be blocked at some marking in the regular unobservable reach of M_8 while no fault transition has fired, i.e., the possibility of firing a fault transition is excluded. In such a case, the supervisor may re-enable event d, which may lead to a more permissive control result. However, to re-enable events, all subsequent diagnostic states from z_{10} and the control decisions at those states need to be further explored, since the subsequence diagnostic states may not be already in A_d and may contain new indeterminate cycles. To keep this paper focused, we do not address this issue and simply disable the* *precursor event that leads to unfaulty deadlocks.* □

Theorem 4.4. *Given an LPN $G = (PN, M_0, \Sigma, \ell)$ with $\Sigma = \Sigma_c \cup \Sigma_{uc}$ and a set of fault transitions T_f, the closed-loop system (G, S) is diagnosable and free of unfaulty deadlocks, where S is the control policy designed by Algorithm 5 and Eq. 4-6.*

Proof. Algorithm 5 ensures that the diagnosability enforcing supervisor does not contain any indeterminate cycle and q-normal cycle. Since the diagnosability enforcing supervisor represents the behavior of the closed-loop system, the closed-loop system is diagnosable and does not have unfaulty deadlocks. □

4.4 Conclusion

In this section, we summarize the contributions of this chapter as follows:

- First, we generalize the notion of diagnosability to LPNs that are not necessarily deadlock-free. This is necessary because control actions enforcing active diagnosis may induce deadlocks even if an original net is deadlock-free. Moreover, we assume that a deadlock occurring in a plant can be indirectly "observed" by a modeling primitive called *quiescence* in [45] (and *time-out* in [83]). In plain words, if no firing of transitions is observed for a sufficient long time, then one can infer that a plant is blocked (due to either a deadlock or a control-induced deadlock). This inference can be modeled by a particular *quiescent event* in a logical framework.

- The conventional BRG developed in [21] does not contain sufficient information to characterize deadlocks and quiescent behavior of a plant. Thus, we develop a new BRG-like structure called the *quiescent basis reachability graph* (QBRG), in which the quiescent behavior is encoded. Analogously to a BRG, a QBRG models the quiescent behavior of an LPN without explicitly listing all reachable markings. To compute a QBRG, an integer linear programming technique is proposed to characterize the quiescent behavior of a plant net. Then a structure called a *Q-diagnoser* is developed based on a QBRG to verify the diagnosability of a plant.

- Finally, based on the notion of Q-diagnoser, we propose an algorithm to design a *diagnosability enforcing supervisor* for a given plant. The supervisor is obtained by recursively removing all *indeterminate cycles* in the Q-diagnoser, which circumvents the need of a complete marking enumeration. Moreover, the supervisor designed by our approach guarantees that the closed-loop system is deadlock-free if no fault occurs.

Chapter 5 Asynchronous Diagnosability Enforcement in Discrete Event Systems Based on Supervisory Control Theory

5.1 Motivation

As mentioned in previous sections, the aim of diagnosis [2, 3, 5, 10, 20, 23, 25, 27, 84–86] in discrete event systems is to infer the occurrence of faults in a plant by observing the events generated by the plant. A diagnostic agent called *diagnoser* that runs parallel with a plant can be used for online diagnosis. However, it is required to initialize and run the diagnoser synchronously with the plant. Such a condition may not be easy to meet in practice [56]. Therefore, the work in [56, 87] extends the original notion of diagnosis and proposes a notion called *asynchronous diagnosis* that is to detect the occurrence of faults in a plant under the condition that a diagnostic agent is activated asynchronously with the plant. In [56], a structure called *asynchronous diagnoser* is proposed for asynchronous diagnosis. The developed asynchronous diagnoser is able to detect the occurrence of faults in a plant without the past information of the plant before activating the diagnoser. This allows that the asynchronous diagnoser can be activated at any time instance, even after the occurrence of faults. Therefore, compared with the traditional diagnoser, the asynchronous diagnoser structure has a wider range of applications.

Moreover, in the case of asynchronous diagnosis, the notion corresponding to the traditional diagnosability is the *asynchronous diagnosability* [56]. In other words, asynchronous diagnosability is the property of a plant such that the occurrence of any fault can be detected after a finite number of observations under the condition that a diagnostic agent may be asynchronously activated with the plant. A necessary and sufficient condition is proposed in [56] to verify the asynchronous diagnosability of a plant. In practice, a plant should also be asynchronous diagnosable. However, given an asynchronous undiagnosable plant, there is no work to enforce the asynchronous diagnosability as far as we know. Therefore, in this chapter we address the asynchronous diagnosability enforcement problem based on the supervisor control theory.

5.2 Problem Formulation

5.2.1 Asynchronous Diagnosis and Diagnosability

Given an automaton plant $A = (X, E, \delta, x_0)$, as mentioned in Section 3, the event set E can be partitioned into the set of observable events E_o and the set of unobservable events E_{uo}, i.e., $E = E_o \cup E_{uo}$. The unobservable event set E_{uo} is further partitioned into the set of regular unobservable events E_{reg} and the the set of fault event $E_f = \{f\}$, i.e., $E_{uo} = E_{reg} \cup E_f$.

Assume that plant A has been running for a while and has generated a sequence σ. Not knowing the information about the past behavior of the plant, i.e., the generated sequence σ and its observation $\mathcal{P}(\sigma)$, one starts the diagnosis process at this time and determines if fault f has occurred by observing the events generated by the plant. This is the so-called *asynchronous diagnosis* problem [56]. The work in [56] develops a diagnoser structure for asynchronous diagnosis such that it is not required to synchronously initialize and run the diagnoser with the plant. In other words, the diagnoser can be activated at any time, even after the occurrence of a fault. In [56], an important notion called *asynchronous diagnosability* is introduced for live plants.

Definition 5.1. *[56] Given a live plant $A = (X, E, \delta, x_0)$, where $E = E_o \cup E_{uo}$ and fault event $f \in E_{uo}$, A is* asynchronous diagnosable *if for all $\sigma \in L(A)$, $f \in \sigma$, there exists $k \in \mathbb{N}$ such that for all $\sigma' \in L(A)/\sigma$ with $|\sigma'| \geq k$, the following condition holds:*

$$\hat{\sigma}\hat{\sigma}' \in L(A) \wedge \hat{\sigma}' \in \mathcal{P}^{-1}_{ext(L(A))}(\mathcal{P}(\sigma')) \Rightarrow f \in \hat{\sigma}\hat{\sigma}'. \ \square$$

In other words, given a faulty sequence σ, let σ' be a continuation of σ. The word $\mathcal{P}(\sigma')$ represents the observations after beginning the diagnosis process. A plant A is asynchronous diagnosable w.r.t. the fault f if when σ' is sufficiently long ($|\sigma'| \geq k$), all sequences $\hat{\sigma}\hat{\sigma}' \in L(A)$ contain at least one fault event where sequence $\hat{\sigma}'$ belongs to the extension closure of $L(A)$ ($\hat{\sigma}' \in ext(L(A))$) and has the same observations as σ' ($\mathcal{P}(\hat{\sigma}') = \mathcal{P}(\sigma')$). If we cannot determine whether the fault occurred or not by observing a finite number of observations after beginning the diagnosis process, then the plant is asynchronous undiagnosable.

According to Definition 5.1, asynchronous diagnosability of a plant is a behavior property that is determined by the language generated by the plant. In practice, an asynchronous undiagnosable plant should necessarily be refined to be diagnosable before being put online. Therefore, in this chapter we study the asynchronous diagnosability enforcement problem, i.e., to design a supervisor for an asynchronous undiagnosable plant to restrict the behavior of the plant such that the closed-loop system is asynchronous diagnosable.

In the classical supervisor control theory, a control policy is made based on the sequences observed by the supervisor. However, in this chapter, we develop a state-based supervisor. Specifically, given a plant A, let $\sigma \in ext(L(G))$ be a sequence that occurs after starting the diagnosis process. A diagnostic agent observes the word $w = \mathcal{P}(\sigma)$ and deduces a *diagnostic state* $y(w)$ that contains the information of both the consistent states and the status of the fault. A diagnostic state is a set of pairs $y = \{(x_1, l_1), (x_2, l_2), ..., (x_n, l_n)\}$, where each state $x_i \in X$ represents that the plant currently may be at x_i and $l_i \in \{N, F\}$ represents that at state x_i, a fault has occurred ($l_i = F$) or not ($l_i = N$). Based on a diagnostic state $y(w)$, a supervisor makes a control decision $S(y) \subseteq E_c$ such that the controllable events not in $S(y)$ are disabled by the supervisor. Note that a supervisor cannot disable any uncontrollable event. The problem studied in this chapter is formulated as follows.

Problem 5.1. *Given an automaton plant A that is asynchronous undiagnosable, determine a supervisor S for A such that the closed-loop system (A, S) is asynchronous diagnosable.* □

Moreover, the plant A considered in this paper satisfies the following assumptions:

A1 The plant A does not contain any cycle composed by unobservable events only;

A2 $E_c \subseteq E_o \subseteq E$.

Assumption A1 guarantees that the plant does not generate an infinite long sequence that contains only unobservable events, which is a common assumption in the field of fault diagnosis. Assumption A2 requires that all controllable events are also observable. This assumption is widely used in the context of supervisory control for partially observed discrete event systems. The developed approach can also be generalized to cases where Assumption A2 is relaxed based on the basis of the *BTS structure* developed in [49, 88, 89]. Note that in many practical situations the ability to disable an event is usually coupled with the ability to detect the occurrence of it: Assumption A2 is satisfied in such systems.

5.2.2 Asynchronous Diagnosability of Nonlive Plants

The concept of asynchronous diagnosability proposed in [56] requires that the considered plant is live. However, in practice, a plant may not necessarily live, i.e., a workflow system may halt after finishing all its task. Moreover, event if an original plant is live, the control actions of a supervisor may introduce deadlocks. Therefore, to enforce the asynchronous diagnosability using supervisory control theory, we first need to generalize the notion of asynchronous diagnosability to nonlive plants.

Definition 5.2. *Given a plant $A = (X, E, \delta, x_0)$, where $E = E_o \cup E_{uo}$ and fault event $f \in E_{uo}$, A is* asynchronous diagnosable *if for all $\sigma \in L(A)$, $f \in \sigma$, there exists $k \in \mathbb{N}$ such that for all $\sigma' \in L(A)/\sigma$ the following conditions hold:*

- *if $|\sigma'| \leq k$ and $L(A)/\sigma\sigma' = \emptyset$, then:*

$$\hat{\sigma}\hat{\sigma}' \in L(A) \wedge \hat{\sigma}' \in \mathcal{P}^{-1}_{ext(L(A))}(\mathcal{P}(\sigma')) \wedge L(A)/\hat{\sigma}\hat{\sigma}' = \emptyset \Rightarrow f \in \hat{\sigma}\hat{\sigma}';$$

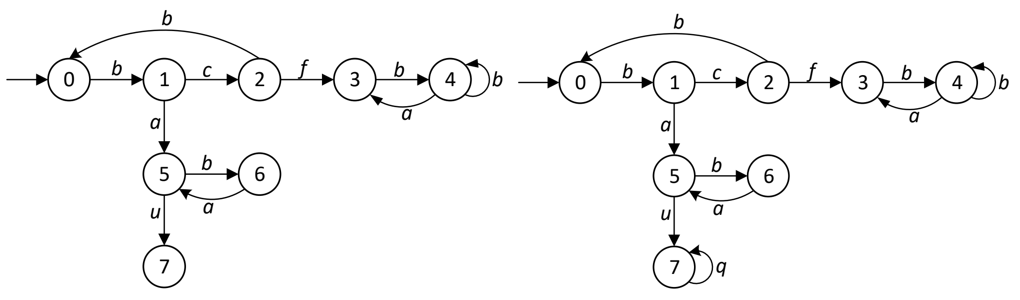
- *if $|\sigma'| \geq k$, then:*

$$\hat{\sigma}\hat{\sigma}' \in L(A) \wedge \hat{\sigma}' \in \mathcal{P}^{-1}_{ext(L(A))}(\mathcal{P}(\sigma')) \Rightarrow f \in \hat{\sigma}\hat{\sigma}'. \ \square$$

The second condition in Definition 5.2 is the conventional asynchronous diagnosability notion for live plants as in Definition 5.1. The first condition considers the case that sequence $\sigma\sigma'$ is terminal ($L(A)/\sigma\sigma' = \emptyset$). In such a case, it requires that all terminal sequences $\hat{\sigma}\hat{\sigma}' \in L(A)$ contain at least one fault where sequence $\hat{\sigma}'$ belongs to the extension closure of $L(A)$ ($\hat{\sigma}' \in ext(L(A))$) and has the same observations as σ' ($\mathcal{P}(\hat{\sigma}') = \mathcal{P}(\sigma')$). By generalizing the concept of asynchronous diagnosability, in this paper we remove the liveness assumption considered in [56].

5.2.3 Observable Quiescence

Given a nonlive plant, if it reaches a dead state, then the plant will stay at the dead state and does not produce any observation in the future. From the perspective of an external agent, the plant does not produce any observation forever. Since in many practice cases, the time to execute an event has an upper bound that is usually pre-known. In other words, the sojourn time of each non-dead state in a plant is bounded. Therefore, if a time exceeds the upper bound while no event is observed, one can infer that the plant must be blocked at some state. Hence, the deadlock and control-induced deadlock can be “observed” in an indirect way. Such an time-out mechanism is called *quiescence* that has been investigated by many researchers [90, 91] and has been used to solve the problem of diagnosis [45, 48] and supervisory control [83, 92] in DESs. We introduce a particular event q called the *quiescent event* to model that a plant does not produce any observation for a sufficient long time. The quiescence mechanism can be formulated as follows:



(a) (b)

Fig. 5.1 (a) Automaton A for Example 5.1, and (b) its quiescence automaton A_q.

- Once the plant is deadlocked (either by itself or by a control action), it will repetitively generate an observable and uncontrollable event q;
- Event q is only generated when the plant is at a dead state, i.e., the plant does not generate event q if it can execute any other event.

In other words, the occurrence of event q means that the plant is deadlocked at some state. In the following, we define a new structure called *quiescence automaton* in which the quiescence information of a plant is encoded.

Definition 5.3. *Given a plant $A = (X, E, \delta, x_0)$, a* quiescence automaton *of A is $A_q = (X, E_q, \delta_q, x_0)$, where $E_q = E \cup \{q\}$ is the set of events, and the transition function δ_q is defined as follows:*

$$\delta_q(x, e) = \delta(x, e) \ \ \forall x \in X \setminus X_d, \forall e \in E$$

$$\delta_q(x, q) = \begin{cases} x, & \mathit{if}\ x \in X_d \\ \mathit{undefined} & \mathit{if}\ x \notin X_d. \end{cases} \tag{5-1}$$

where $X_d \subseteq X$ is the set of dead states. □

Given a plant A, the quiescence automaton of A can be easily computed according to Definition 5.3. Here, we use the following example to illustrate this.

Example 5.1. *Consider the plant A in Figure 5.1(a) where $E_o = \{a, b, c\}$ and $E_c = \{a\}$. By Definition 5.3, the quiescence automaton A_q of A is computed and shown in Figure 5.1(b). When the system is at the dead state 7, it will repetitively generate the quiescent event q without changing its state anymore, as depicted in A_q.* □

5.3 Asynchronous Diagnosability Verification of Nonlive Plants

Given a plant A, in this section we develop a structure called *asynchronous-quiescent diagnoser* that is used both for online asynchronous diagnosis and asynchronous diagnosability verification. The developed asynchronous-quiescent diagnoser is a four-tuple structure:

$$A_d = (Y, E_d, \delta_d, y_0)$$

where $Y \subseteq 2^{X\times\{N,F\}}$ is the set of diagnostics states, $E_d = E_o \cup \{q\}$ is the set of events, $\delta_d : Y \times E_d \to Y$ is the transition function, and y_0 is the initial state. As mentioned in previous section, a diagnostic state is in the form of $y = \{(x_1, l_1), ..., (x_n, l_n)\}$ where $x_i \in X$ and $l_i \in \{N, F\}$ that carries the fault information at state x_i. Based on a diagnostic state y, one can infer the current plant states and the occurrence of fault f in the plant. Before defining the initial diagnostic state y_0 and the transition function δ_d, in the following, we first introduce a *labeling function* $\Delta : \{N, F\} \times \Sigma^* \to \{N, F\}$.

Definition 5.4. *Given a sequence $\sigma \in E^*$ and a label $l \in \{N, F\}$, the labeling function Δ is defined as follows:*

$$\Delta(l, \sigma) = \begin{cases} N, & \textit{if } l = N \wedge f \notin \sigma \\ F, & \textit{otherwise.} \end{cases} \tag{5-2}$$

□

Moreover, since this chapter considers the problem of asynchronous diagnosis, the proposed asynchronous-quiescent diagnoser can begin the diagnosis process at any time instance. When the diagnoser is activated, the diagnoser does not have the information about the plant's past behavior and hence does not know the exact current state and the condition of the plant. In such a case, the initial diagnostic state y_0 should contain all possible plant states and the information about the occurred fault. To this end, the initial diagnostic state y_0 is defined as follows:

$$y_0 = \{(x, \Delta(N, \sigma)) \mid \sigma \in L(A), x \in \delta_q(x_0, \sigma)\}.$$

Initially, the asynchronous-quiescent diagnoser is at the initial diagnostic state y_0. After the diagnoser is activated, it will improve the estimation of the plant states and the information about the occurred fault by using the observed events generated by the plant. The transition function $\delta_d : Y \times E_d \to Y$ of A_d is defined as follows:

$$\delta_d(y, e) = \{(x', \Delta(l, e\sigma)) \mid (x, l) \in y, \sigma \in U_e(x), x' \in \delta(x, e\sigma)\}.$$

Based on the analysis above, we then propose Algorithm 6 to compute the asynchronous-quiescent diagnoser of a plant. Compared with the diagnoser structure in [56], the proposed diagnoser A_d contains the quiescence information of the plant that can be used to improve the estimation of the plant states and the condition of the plant. We note that if a plant is live, then the asynchronous-quiescent diagnoser A_d is the same as the diagnoser structure in [56].

Algorithm 6: Computation of the asynchronous-quiescent diagnoser A_d.

```
   Input: A plant $A = (X, E, \delta, x_0)$ with $E = E_o \cup E_{uo}$, and $E_{uo} = E_{reg} \cup \{f\}$
   Output: $A_d = (Y, E_d, \delta_d, y_0)$
 1 compute the quiescence automaton $A_q = (X, E \cup \{q\}, \delta_q, x_0)$ of $A$ by Definition
   5.3;
 2 let $y_0 = \{(x, \Delta(N, \sigma)) \mid \sigma \in L(A), x \in \delta_q(x_0, \sigma)\}$;
 3 let $Y = \{y_0\}$, $Y_{temp} = \{y_0\}$;
 4 while $Y_{temp} \neq \emptyset$ do
 5     select a state $y \in Y_{temp}$;
 6     foreach $e \in E_o \cup \{q\}$ do
 7         compute $y' = \{(x', \Delta(l, e\sigma)) \mid (x, l) \in y, \sigma \in U_e(x), x' \in \delta_q(x, e\sigma)\}$;
 8         if $y' \notin Y$ then
 9             let $Y = Y \cup \{y'\}$, $Y_{temp} = Y_{temp} \cup \{y'\}$;
10         end
11         let $\delta_d(y, e) = y'$;
12     end
13     let $Y_{temp} = Y_{temp} \setminus \{y\}$;
14 end
15 output $A_d = (Y, E_o \cup \{q\}, \delta_d, y_0)$;
```

We briefly explain how Algorithm 6 works. In Step 1, the quiescence automaton A_q is computed by Definition 5.3. Step 2 computes the initial diagnostic state y_0. Step 3 puts state y_0 to sets Y and Y_{temp}. Steps 4 to 11 iteratively check the states in Y_{temp}. In the iteration cycle, if Y_{temp} is not empty, then a state $y \in Y_{temp}$ is selected in Step 5. By observing an observable event $e \in E_o \cup \{q\}$, a new diagnostic state y' can be reached from y. If y' is a new state (i.e., $y' \notin Y$), then it is put to sets Y and Y_{temp}. Step 10 defines the transition function $\delta_d(y, e) = y'$. Step 11 removes state y from set Y_{temp} to denote that y has been checked. Steps 4 to 11 run iteratively until there is no unchecked state in Y_{temp}. Finally, Step 12 outputs the final structure A_d.

Example 5.2. *Consider the plant G in Figure 5.1(a), where $E_o = \{a, b, c\}$ and $E_f = \{f\}$. We can compute the initial diagnostic state $y_0 = \{(0, N), (1, N), (2, N), (3, F), (4, F), (5, N), (6, N), (7, N)\}$. By applying Algorithm 6, the asynchronous-quiescent diagnoser A_d*

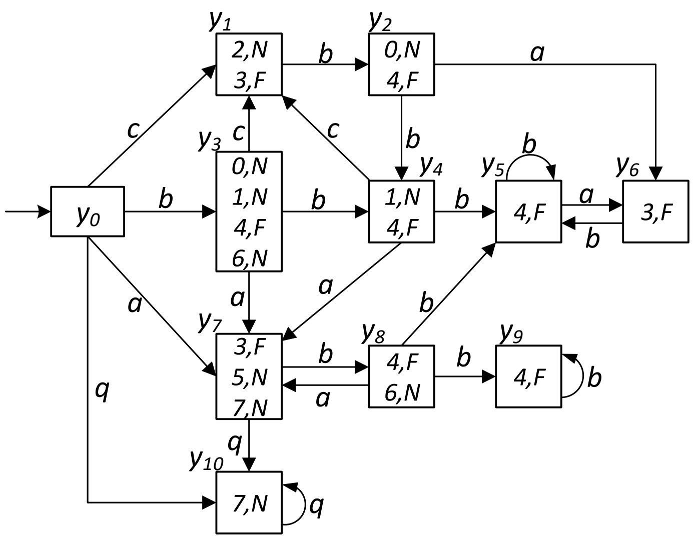


Fig. 5.2 The asynchronous-quiescent diagnoser A_d.

can be constructed, as shown in Figure 5.2.

Suppose that the plant A has generated a sequence $bcfb$, i.e., it is at state 4. Then the diagnoser is activated for asynchronous diagnosis. Initially, the diagnoser is at the initial diagnostic state y_0. If an observable event a is generated, according to Figure 5.2 the diagnoser updates its state to $y_7 = \{(3, F), (5, N), (7, N)\}$. If the plant then generates an observable event b, the diagnoser updates the diagnostic state to $y_8 = \{(4, F), (6, N)\}$. If the plant generates again an observable event b, the corresponding diagnostic state is updated to $y_5 = \{(4, F)\}$, which implies that fault f has occurred. □

Remark 5.1. *We note that the quiescence information can be used to improve the estimation of plant states and the information about the occurred fault. For example, consider the plant A in Figure 5.1(a) whose asynchronous-quiescent diagnoser is shown in Figure 5.2. If after the activation of the diagnoser, the plant generates an observable word $w = ba$, the diagnoser updates the diagnostic state to $y_7 = \{(3, F), (5, N), (7, N)\}$. However, if then the quiescence is observed, i.e., the plant does not produce any observation for a long time, one knows that the plant must be at state 7 where no fault has occurred, since from states 3 and 5 the observable event b can occur. In other words, the corresponding diagnostic states of observation $w = baq$ is $y_9 = \{(7, N)\}$.* □

As mentioned at the beginning of this section, the proposed asynchronous-quiescent diagnoser can also be used to verify the asynchronous diagnosability of a plant. To this aim, we first recall the notion of indeterminate cycle in Definition 3.3. In simple words, an indeterminate cycle in A_d is a cycle formed by uncertain diagnostic states for which there exist: (i) a corresponding cycle in A_q involving only states labeled "N" in the cycle of diagnoser; and (ii) a corresponding cycle in A_q involving only states labeled "F" in the cycle of diagnoser. The following theorem provides a necessary and sufficient condition for asynchronous diagnosability of a plant based on the asynchronous-quiescent diagnoser.

Theorem 5.1. *Given a plant $A = (X, E, \delta, x_0)$ with $E = E_o \cup E_{uo}$, and $E_{uo} = E_{reg} \cup \{f\}$, let A_d be the asynchronous-quiescent diagnoser of A. Plant A is asynchronous diagnosable if and only if there does not exist any indeterminate cycle in A_d.*

Proof. (Only If) By contradiction, suppose that there exists an indeterminate cycle in A_d, i.e., $y_1 \xrightarrow{e_1} y_2 ... \xrightarrow{e_{n-1}} y_n \xrightarrow{e_n} y_n$ According to Definition 3.3, in quiescence automaton A_q there exist a cycle $x_1 \xrightarrow{\sigma_1} x_2 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_{n-1}} x_n \xrightarrow{\sigma_n} x_1$ and a cycle $x'_1 \xrightarrow{\sigma'_1} x'_2 \xrightarrow{\sigma'_2} \cdots \xrightarrow{\sigma'_{n-1}} x'_n \xrightarrow{\sigma'_n} x'_1$ such that (x_i, N), $(x'_i, F) \in y_i$. Moreover, sequences $\hat{\sigma} = \sigma_1 \sigma_2 \cdots \sigma_n$ and $\hat{\sigma}' = \sigma'_1 \sigma'_2 \cdots \sigma'_n$ have the same observation, i.e., $\mathcal{P}(\hat{\sigma}) = \mathcal{P}(\hat{\sigma}')$. Since diagnostic state y_i is reachable from initial diagnostic state y_0, we can conclude that there exist a normal sequence σ and a faulty σ' such that $\delta(x_0, \sigma) = x_i$ and $\delta(x_0, \sigma') = x'_i$. There exist two cases:

Case 1: The events in the indeterminate cycle belong to E_o, i.e., $e_i \in E_o$. In such a case, in plant A there exists two arbitrary long sequences $\sigma(\hat{\sigma})^n$ and $\sigma'(\hat{\sigma}')^n$ where $f \in \sigma'$, $f \notin \sigma(\hat{\sigma})^n$ and $\mathcal{P}(\hat{\sigma}) = \mathcal{P}(\hat{\sigma}')$. This violates the second condition in Definition 5.2.

Case 2: The indeterminate cycle is a q-indeterminate cycle $y_i \xrightarrow{q} y_i$. In such a case, states x_i and x'_i are dead. We can conclude that there exist two terminal sequences σ, $\sigma' \in L(A)$ where $f \notin \sigma$ and $f \in \sigma'$. Therefore, the first condition in Definition 5.2 is not satisfied.

In both of the above two cases, one of the conditions in Definition 5.2 is not satisfied. Therefore, plant A is asynchronous undiagnosable.

(If) We prove the statement by showing that A is asynchronous undiagnosable implying the existence of indeterminate cycles in A_d. Let $\sigma \in L(A)$ be a faulty sequence and σ' be a continuation of σ such that sequence $\sigma\sigma'$ violates the condition in Definition 5.2 causing the plant asynchronous undiagnosable. There are two cases:

Case 1: Sequence $\sigma\sigma'$ is a terminal sequence leading to a dead state x', i.e., the first condition in Definition 5.2 is not satisfied. Therefore, there exists another non-faulty sequence $\hat{\sigma}\hat{\sigma}' \in L(A)$ leading to a dead state x such that $\mathcal{P}(\hat{\sigma}') = \mathcal{P}(\sigma')$. This implies that in A_d there exists a diagnostic state y containing two pairs (x, N) and (x', F). By the construction of A_d, there exists an indeterminate cycle in A_d, i.e., $y \xrightarrow{q} y$.

Case 2: Sequence $\sigma\sigma'$ is an arbitrary long sequence and violates the second condition in Definition 5.2. This implies that there exists another arbitrary long non-faulty sequence

$\hat{\sigma}\hat{\sigma}' \in L(A)$ such that $\mathcal{P}(\hat{\sigma}') = \mathcal{P}(\sigma')$. Since the states in A is bounded, sequences σ' and $\hat{\sigma}'$ corresponding to two cycles in plant A that have the same observation. By the construction of A_d, we conclude that there necessarily exists an indeterminate cycle in A_d. □

According to Theorem 5.1, the asynchronous diagnosability of a plant A can be verified by checking the existence of indeterminate cycles in the corresponding asynchronous-quiescent diagnoser A_d. To this aim, we first check if there exists a cycle composed by only uncertain diagnostic states; then we verify if the cycle is indeterminate or not by using Definition 3.3.

Example 5.3. *Consider the plant A in Figure 5.1(a) whose asynchronous-quiescent diagnoser A_d is shown in Figure 5.2. There are two cycles in A_d composed by only uncertain diagnostic states, i.e., $y_1 \xrightarrow{b} y_2 \xrightarrow{b} y_4 \xrightarrow{c} y_1$ and $y_7 \xrightarrow{b} y_8 \xrightarrow{a} y_7$. For the first cycle $y_1 \xrightarrow{b} y_2 \xrightarrow{b} y_4 \xrightarrow{c} y_1$, there does not exist a corresponding cycle in A_q involving only plant states labeled "F". Therefore, it is not an indeterminate cycle. For the second cycle $y_7 \xrightarrow{b} y_8 \xrightarrow{a} y_7$, we can find two cycles in A_q, i.e., $3 \xrightarrow{b} 4 \xrightarrow{a} 3$ and $5 \xrightarrow{b} 6 \xrightarrow{a} 5$. By Definition 3.3, cycle $y_7 \xrightarrow{b} y_8 \xrightarrow{a} y_7$ is an indeterminate cycle. Therefore, according to Theorem 5.1, the original plant is asynchronous undiagnosable.* □

5.4 Asynchronous Diagnosability Enforcement Supervisor

In this section, we propose an approach to compute a supervisor for an asynchronous undiagnosable plant such that the closed-loop system is asynchronous diagnosable. According to Theorem 5.1, a plant is asynchronous undiagnosable if and only if its asynchronous-quiescent diagnoser contains indeterminate cycles. Therefore, to enforce the asynchronous diagnosability of the plant, a supervisor should forbid all behaviors of the plant that correspond to the indeterminate cycles. Moreover, there are two types of indeterminate cycles in an asynchronous-quiescent diagnoser: (i) an indeterminate cycle containing controllable events, and (ii) an indeterminate cycle containing uncontrollable events only. For the first type of indeterminate cycles, to prevent the plant circulating in such a cycle for an infinite long time, some controllable event in the cycle should be disabled. On the other hand, for an indeterminate cycle belongs to the second type, all diagnostic states in the cycle (and all diagnostic diagnostic states that can uncontrollable reach to the cycle) should be forbidden.

As mentioned in Section 5.2, the developed supervisor makes control decisions based on the current diagnostic states computed by the asynchronous-quiescent diagnoser. In other words, for an observation w whose consistent diagnostic state is y_i, the corresponding control decision is $S(y_i) = E_c \setminus E_{d,i}$ where $E_{d,i}$ is the set of controllable events disabled at state y_i.

In the following, we propose Algorithm 7 to compute an asynchronous diagnosability enforcement supervisor. Given a plant A and its asynchronous-quiescent diagnoser A_d, Algorithm 7 recursively removes the indeterminate cycles in A_d until a fixed point is reached. In Algorithm 7, we use $E_{\mathcal{C}}$ and Y_{dis} to represent the set of events in indeterminate cycle $\mathcal{C}$ and the set of diagnostic states at which the supervisor makes disablement actions, respectively.

We briefly explain how Algorithm 7 works. Step 1 calls Algorithm 6 to compute the asynchronous-quiescent diagnoser of plant A. In the first part of the algorithm, i.e., Steps 4 to 13, an indeterminate cycle $\mathcal{C}$ is selected and treated. If indeterminate cycle $\mathcal{C}$ contains uncontrollable events only and can be reached from y_0 by executing an uncontrollable sequence, then there does not exist an asynchronous diagnosability enforcement supervisor. Algorithm 7 terminates in such a case. If cycle $\mathcal{C}$ contains at least one arc labeled by a controllable event $e \in E_c$, by Steps 7 to 9, one of such arc (y, e) is put in set $\mathcal{D}$, which implies that event e should be disabled at state y. On the other hand, if the cycle contains no arcs labeled by controllable events, then by Steps 11 to 13 all controllable arcs leading to some states that may uncontrollably reach cycle $\mathcal{C}$ are put in $\mathcal{D}$ to be disabled. Steps 14 to 15 remove the unreachable states and the unnecessary arcs in $\mathcal{D}$.

The second part of the algorithm, i.e., Steps 16 to 24 update the control-induced quiescence information at a state $y_i \in Y_{dis}$. Step 17 computes the disabled event set $E_{d,i}$ at y_i. Since the control decision has changed at y_i, we first remove the quiescent state corresponding to state y_i in Steps 18 to 19 and then update the quiescent behavior at state y_i. In Step 20, a new state y' is computed containing all $(x, l) \in y_i$ such that the disablement of events in $E_{d,i}$ at y_i will block the plant at state x ($\Gamma(x) \subseteq E_{d,i}$). If state $y' \neq \emptyset$, the quiescence information is encoded into the structure by Steps 22 to 24. The above procedure runs iteratively until a fixed structure is reached.

Remark 5.2. *The number of states in the proposed supervisor S, in the worst case, is $2^{2|X|}$. Therefore, the structural complexity of S is $O(2^{2|X|})$. Note that such an exponential complexity of $|X|$ seems unavoidable due to the diagnoser construction for a partially observed automaton.* $\square$

Example 5.4. *Consider again the plant A in Figure 5.1(a) where the set of observable events $E_o = \{a, b, c\}$ and the set of controllable events $E_c = \{a\}$. The asynchronous-quiescent diagnoser A_d of A is shown in Figure 5.2. Plant A is asynchronous undiagnosable, since*

Algorithm 7: Computation of an asynchronous diagnosability enforcement supervisor

```
Input: A plant $A = (X, E, \delta, x_0)$ with $E = E_o \cup E_{uo}$ and $E = E_c \cup E_{uc}$
Output: Asynchronous diagnosability enforcement supervisor $S$.
1  call Algorithm 6 to compute the asynchronous-quiescent diagnoser
   $A_d = (Y, E_d, \delta_d, y_0)$ of $A$;
2  let $Y_{dis} = \emptyset$, $\mathcal{D} = \emptyset$;
3  while there exists an indeterminate cycle $\mathcal{C}$ in $A_d$ do
4      if $E_{\mathcal{C}} \cap E_c = \emptyset \wedge (\exists\sigma \in E_{uc}^*)\delta_d(y_0, \sigma) = y' \in \mathcal{C}$ then
5          output: No solution, END;
6      end
7      if $E_{\mathcal{C}} \cap E_c \neq \emptyset$ then
8          select $e \in E_{\mathcal{C}} \cap E_c$ such that $\delta_d(y, e) = y', y, y' \in \mathcal{C}$;
9          let $\mathcal{D} = \mathcal{D} \cup \{(y, e)\}$, $Y_{dis} = \{y\}$;
10         remove $\delta_d(y, e)$ from $\delta_d$;
11     else
12         foreach $y \in Y$, $e \in E_c$, such that $\delta_d(y, e) = y'$ and $\exists\sigma \in E_{uc}^*$ such that
           $\delta_d(y', \sigma) = y'' \in \mathcal{C}$ do
13             let $\mathcal{D} = \mathcal{D} \cup \{(y, e)\}$, $Y_{dis} = Y_{dis} \cup \{y\}$;
14             remove $\delta_d(y, e)$ from $\delta_d$;
15         end
16     end
17     remove all unreachable states from $Y$;
18     remove $(y, e)$ from $\mathcal{D}$ if $y \notin Y$ ;
19     foreach $y_i \in Y_{dis}$ do
20         let $E_{d,i} = \{e \mid \exists(y_i, e) \in \mathcal{D}\}$;
21         if $\exists y \in Y$, such that $\delta_d(y_i, q) = y$ then
22             remove $\delta_d(y_i, q)$ from $\delta_d$;
23         end
24         let $y' = \{(x, l) \in y_i \mid \Gamma(x) \subseteq E_{d,i}\}$;
25         if $y' \neq \emptyset$ then
26             if $y' \notin Y$ then
27                 let $Y = Y \cup \{y'\}$;
28             end
29             let $\delta_d(y_i, q) = y'$, $\delta_d(y', q) = y'$;
30         end
31     end
32     let $Y_{dis} = \emptyset$
33 end
34 output $S = (Y, E_o \cup \{q\}, \delta_d, y_0)$.
```

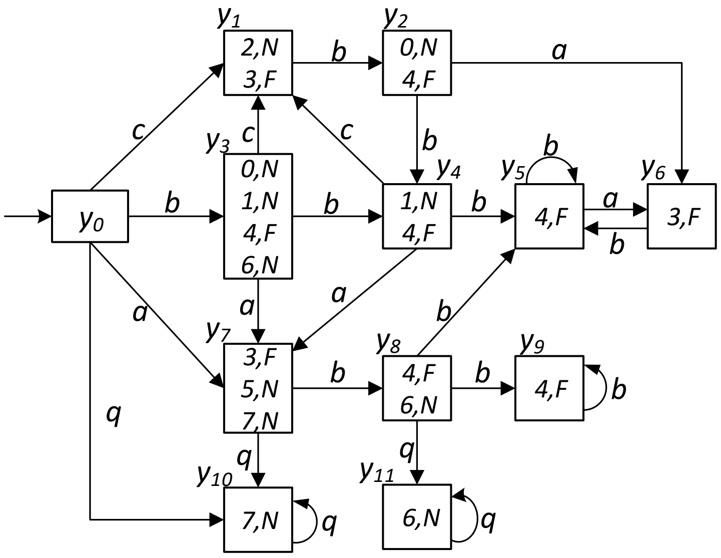


Fig. 5.3 Supervisor S for Example 5.4.

there exists an indeterminate cycle in A_d, i.e., $y_7 \xrightarrow{b} y_8 \xrightarrow{a} y_7$.

Now we compute an asynchronous diagnosability enforcing supervisor for A by Algorithm 7. Since the indeterminate cycle contains a controllable event, i.e., event a, by Step 8 event a is disabled at state y_8, i.e., $\mathcal{D} = \{(y_8, a)\}$ and $Y_{dis} = \{y_8\}$. Step 9 removes transition function $\delta_d(y_8, a)$ from δ_d. Step 17 computes the disabled event set at y_8, i.e., $E_{d,8} = \{a\}$. There exist two pairs $(4, F)$ and $(6, N)$ in state y_8. For state 6, the condition $\Gamma(6) \subseteq E_{d,8}$ holds. By Step 20, a new diagnostic state $y_{11} = \{(6, N)\}$ is computed. In Step 24, an arc labeled event q from y_8 to y_{11} and a self-loop labeled event q at y_{11} are added. The result is shown in Figure 5.3. Since there is no any indeterminate cycle in the current structure, Algorithm 7 terminates and outputs the final supervisor shown in Figure 5.3. □

Once a supervisor $S = (Y, E_d, \delta_s, y_0)$ is computed by Algorithm 2, the corresponding control policy can be easily obtained based on S. Let w be an observation after starting the diagnosis process whose corresponding diagnostic state in S is y. The control decision $S(y)$ is to permit all controllable events defined at state y, i.e.,

$$S(y) = \{e \in E_c \mid \delta_s(y, e)!\}.$$

For example, consider the plant in Figure 5.1 whose asynchronous diagnosability enforcing supervisor is shown in Figure 5.3. For an observation $w = ab$ generated after beginning the diagnosis process, the corresponding diagnostic state is y_8. By checking supervisor S, the control decision $S(y_8) = \emptyset$, i.e., event a is disabled.

Theorem 5.2. *Given a plant $A = (X, E, \delta, x_0)$ with $E = E_o \cup E_{uo}$ and $E = E_c \cup E_{uc}$, let*

S be the asynchronous diagnosability enforcing supervisor computed by Algorithm 7. The closed-loop system (A, S) is asynchronous diagnosable.

Proof. By Theorem 5.1, a system is asynchronous diagnosable if and only if there does not exist any indeterminate cycle in its asynchronous-quiescent diagnoser. Since Algorithm 7 ensures that there is no any indeterminate cycle in the supervisor, the closed-loop system (A, S) is asynchronous diagnosable. $\square$

Remark 5.3. *Notice that the supervisor S designed by Algorithm 7 is not necessarily maximally permissive. The reason is that set $\mathcal{D}$ may contain some arcs such that the prune of these arcs will greatly reduce the reachable state space or affect the normal functionality. To deal with this, some additional information can be embedded into Algorithm 7 to avoid removing these crucial arcs. To address this issue will be part of our future work.* $\square$

5.5 Conclusion

In this section, we summarize the main contributions of this chapter as follows:

- Since the control actions of a supervisor may introduce deadlocks even if the original plant is live, we first extend the classical notion of asynchronous diagnosability to plants that may contain deadlocks. Moreover, we assume that deadlocks of a plan can be indirectly "observed" when it occurs, which is the so-called *quiescence* in [48, 90, 91] (or *time-out* in [83, 92]). A logical event called *quiescent event* is introduced to model such a behavior.
- We develop a structure called *asynchronous-quiescent diagnoser* that is used both for online asynchronous diagnosis and asynchronous diagnosability verification of a given plant.
- Finally, for an asynchronous undiagnosable plant, we compute a supervisor based on the *asynchronous-quiescent diagnoser* such that the closed-loop system is asynchronous diagnosable.

Chapter 6 Conclusions and Future Works

6.1 Conclusions

This dissertation studies the active diagnosis problem in discrete event systems by using the model of automaton and Petri net, respectively. The main contributions of the thesis are summarized as follows.

(1) Active diagnosis problem in automata

In this part we present an active diagnosis method to enforce diagnosability of a plant modeled by a finite state automaton. Some properties of active diagnosis are studied. To avoid the deadlock, the notion of $stop$-free event set that is a subset of controllable event set is proposed, and a $stop$-free control policy that does not introduce the $stop$ event is formulated. We develop a heuristic method based on the verifier of the plant to compute a feasible $stop$-free event set that guarantees the existence of a valid control policy. With the $stop$-free event set, the set of disabled edges is computed, and an online control policy that is based on the current diagnostic state is computed which guarantees that the closed-loop system is diagnosable. The structural complexity of the proposed control structure is polynomial with respect to the number of states of the plant.

(2) Active diagnosis problem in labeled Petri nets

In this part, we formulate and deal with the active diagnosis problem in labeled Petri nets by developing a supervisor for a plant such that the closed-loop system is diagnosable. Since control actions may introduce deadlocks even if an original plant is deadlock-free, we first generalize the classical notion of diagnosability in labeled Petri nets to the nets that may contain potential deadlocks. To avoid enumerating all reachable markings of a plant, we develop a structure called *quiescent basis reachability graph* to characterize the behavior of a net containing deadlocks. Accordingly, a structure named *Q-diagnoser* is proposed to verify the diagnosability of a net. An integer linear programming technique is developed to characterize the deadlocks. We prove that a plant is diagnosable if and only if there does not exist any indeterminate cycle in its Q-diagnoser. Finally, for an undiagnosable plant, we introduce a *diagnosability enforcing supervisor* to enforce the diagnosability by trimming a Q-diagnoser. Moreover, our approach guarantees that the closed-loop system cannot reach a dead marking unless a fault transition has fired.

(3) Asynchronous diagnosability enforcement problem in DESs

The asynchronous fault diagnosis is to detect the occurrence of faults in a plant under the condition that a diagnostic agent, i.e., a diagnoser, is activated asynchronously with the plant. Moreover, asynchronous diagnosability is a property of a plant such that any fault can be detected within finite future steps after its occurrence in the case of asynchronous activation of the diagnoser and the plant. In this thesis, we studies the asynchronous diagnosability enforcement problem in discrete event systems by using supervisory control theory, i.e., to design a supervisor for a plant such that the closed-loop system is asynchronous diagnosable. We first generalize the notion of asynchronous diagnosability to nonlive systems since the control actions may introduce deadlocks of a plant even if it is originally live. Then a structure called *asynchronous-quiescent diagnoser* is developed which is used both for online asynchronous diagnosis and asynchronous diagnosability verification. Finally, for a plant that is asynchronous undiagnosable, we propose a supervisor to enforce the asynchronous diagnosability based on its asynchronous-quiescent diagnoser.

6.2 Future Works

Based on the work proposed in this dissertation, there are some possible research directions, which are summarized as follows.

- Firstly, in the framework of automaton, we will study the active diagnosis problem in the distributed [93–96] and probabilistic [97–100] settings. As far as we know, there is no work to deal with active diagnosis problem in the framework of distributed or probabilistic systems.

- Secondly, as mentioned in Section 4.1, the observable quiescence q provides extra information about the current state estimation. In the future, we want to study a more general control problem (i.e., nonblocking control) by using the quiescence information. Once the observable quiescence is observed, the current state estimation can be updated. Accordingly, the control policy can be updated and some disabled events are re-enabled. As a result, the supervisor computed with quiescence can be more permissive than the traditional ones [101–104].

References

[1] CASSANDRAS C G, LAFORTUNE S. Introduction to discrete event systems[M]. [S.l.] : Springer Science & Business Media, 2009.

[2] ZAYTOON J, LAFORTUNE S. Overview of fault diagnosis methods for discrete event systems[J]. Annual Reviews in Control, 2013, 37(2) : 308 – 320.

[3] SAMPATH M, SENGUPTA R, LAFORTUNE S, et al. Diagnosability of Discrete-Event System[J]. IEEE Transactions on Automatic Control, 1995, 40(9) : 1555 – 1575.

[4] ZAD S H, KWONG R H, WONHAM W M. Fault diagnosis in discrete-event systems: framework and model reduction[J]. IEEE Transactions on Automatic Control, 2003, 48(7) : 1199 – 1212.

[5] WU Y, HADJICOSTIS C N. Algebraic approaches for fault identification in discrete-event systems[J]. IEEE Transactions on Automatic Control, 2005, 50(12) : 2048 – 2055.

[6] DEBOUK R, LAFORTUNE S, TENEKETZIS D. Coordinated decentralized protocols for failure diagnosis of discrete-event systems[J]. Mathematics and Statistics: Discrete Event Dynamic Systems, 2000, 10(1-2) : 33 – 86.

[7] QIU W, KUMAR R. Decentralized failure diagnosis of discrete event systems[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Syetems and Humans, 2006, 36(2) : 384 – 395.

[8] WANG Y, YOO T S, LAFORTUNE S. Diagnosis of discrete event systems using decentralized architectures[J]. Discrete Event Dynamic Systems, 2007, 17(2) : 233 – 263.

[9] KEROGLOU C, HADJICOSTIS C N. Distributed Fault Diagnosis in Discrete Event Systems via Set Intersection Refinements[J]. IEEE Transactions on Automatic Control, 2018, 63(10) : 3601 – 3607.

[10] CONG X, FANTI M P, MANGINI A M, et al. Decentralized Diagnosis by Petri Nets and Integer Linear Programming[J]. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2018, 48(10) : 1689 – 1700.

[11] JIANG S, HUANG Z, CHANDRA V, et al. A Polynomial Algorithm for Testing Diagnosability of Discrete-Event Systems[J]. IEEE Transactions on Automatic Control, 2001, 46(8) : 1318 – 1321.

[12] YOO T-S, LAFORTUNE S. Polynomial-Time Verification of Diagnosability of Partially Observed Discrete-Event Systems[J]. IEEE Transactions on Automatic Control, 2002, 47(9) : 1491 – 1495.

[13] MOREIRA M V, JESUS T C, BASILIO J C. Polynomial time verification of decentralized diagnosability of discrete event systems[J]. IEEE Transactions on Automatic Control, 2011, 56(7) : 1679 – 1684.

[14] CARVALHO L K, BASILIO J C, MOREIRA M V. Robust diagnosis of discrete event systems against intermittent loss of observations[J]. Automatica, 2012, 48(9) : 2068 – 2078.

[15] YIN X, LAFORTUNE S. Codiagnosability and coobservability under dynamic observations: Transformation and verification[J]. Automatica, 2015, 61 : 241 – 252.

[16] CABASINO M P, GIUA A, SEATZU C. Diagnosability of bounded Petri nets[C] // In Proceedings of the 48th IEEE Conf. on Decision and Control. 2009 : 1254 – 1260.

[17] RU Y, CABASINO M P, GIUA A, et al. Supervisor synthesis for discrete event systems with arbitrary forbidden state specifications[C] // Proc. 47th IEEE Conference on Decision and Control. 2008 : 1048 – 1053.

[18] RU Y, CABASINO M P, GIUA A, et al. Supervisor synthesis for discrete event systems under partial observation and arbitrary forbidden state specifications[J]. Discrete Event Dynamic Systems, 2014, 24(3) : 275 – 307.

[19] GIUA A, SEATZU C. Fault detection for discrete event systems using Petri nets with unobservable transitions[C] // In Proceedings of the 44th Int. Conf. on Decision and Control and European Control Conference. 2005 : 6323 – 6328.

[20] LEFEBVRE D, DELHERM C. Diagnosis of DES with Petri net models[J]. IEEE Transactions on Automation Science and Engineering, 2007, 4(1) : 114 – 118.

[21] CABASINO M P, GIUA A, SEATZU C. Fault detection for discrete event systems using Petri nets with unobservable transitions[J]. Automatica, 2010, 46(9) : 1531 – 1539.

[22] CABASINO M P, GIUA A, POCCI M, et al. Discrete event diagnosis using labeled Petri nets: an application to manufacturing systems[J]. Control Engineering Practice, 2011, 19(9) : 989 – 1001.

[23] CABASINO M P, GIUA A, SEATZU C. Diagnosis Using Labeled Petri Nets With Silent or Undistinguishable Fault Events[J]. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2013, 43(2) : 345 – 355.

[24] DOTOLI M, FANTI M P, MANGINI A M, et al. On-line fault detection of discrete event systems by Petri nets and integer linear programming[J]. Automatica, 2009, 45(11) : 2665 – 2672.

[25] ZHU G, FENG L, LI Z, et al. An Efficient Fault Diagnosis Approach Based on Integer Linear Programming for Labeled Petri Nets[J]. IEEE Transactions on Automatic Control, 2020, 66(5) : 2393 – 2398.

[26] CABASINO M P, GIUA A, HADJICOSTIS C N, et al. Fault model identification and synthesis in Petri nets[J]. Discrete Event Dynamic Systems, 2015, 25(13) : 419 – 440.

[27] BASILE F, CHIACCHIO P, De Tommasi G. An efficient approach for online diagnosis of discrete event systems[J]. IEEE Transactions on Automatic Control, 2009, 54(4) : 748 – 759.

[28] RU Y, HADJICOSTIS C N. Fault diagnosis in discrete event systems modeled by partially

observed Petri nets[J]. Discrete Event Dynamic Systems, 2009, 19(4) : 551 – 575.

[29] BASILE F, CHIACCHIO P, TOMMASI G D. On K-diagnosability of Petri nets via integer linear programming[J]. Automatica, 2012, 48(9) : 2047 – 2058.

[30] BASILE F, CHIACCHIO P, TOMMASI G D. Sufficient conditions for diagnosability of Petri nets[C] // In Proceedings of the 9th International Workshop on Discrete Event Systems. 2008 : 370 – 375.

[31] CABASINO M P, GIUA A, LAFORTUNE S, et al. A New Approach for Diagnosability Analysis of Petri Nets Using Verifier Nets[J]. IEEE Transactions on Automatic Control, 2012, 57(12) : 3104 – 3117.

[32] CABASINO M P, GIUA A, SEATZU C. Diagnosability of discrete-event systems using labeled Petri nets[J]. IEEE Transactions on Automation Science and Engineering, 2014, 11(1) : 144 – 153.

[33] JIROVEANU G, BOEL R K. The Diagnosability of Petri Net Models Using Minimal Explanations[J]. IEEE Transactions on Automatic Control, 2010, 55(7) : 1663 – 1668.

[34] RAMIREZ-TREVINO A, RUIZ-BELTRAN E, RIVERA-RANGEL I, et al. Online Fault Diagnosis of Discrete Event Systems: A Petri Net-Based Approach[J]. IEEE Transactions on Automation Science and Engineering, 2007, 4(1) : 31 – 39.

[35] RAMÍREZ-TREVIÑO A, RUIZ-BELTRáN E, ARáMBURO-LIZáRRAGA J, et al. Structural Diagnosability of DES and Design of Reduced Petri Net Diagnosers[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Syetems and Humans, 2012, 42(2) : 416 – 429.

[36] YIN X, LAFORTUNE S. On the Decidebility and Complexity of Diagnosability for Labeled Petri Nets[J]. IEEE Transactions on Automatic Control, 2017, 62(11) : 5931 – 5938.

[37] RAN N, SU H, GIUA A, et al. Codiagnosability Analysis of Bounded Petri Nets[J]. IEEE Transactions on Automatic Control, 2018, 63(4) : 1192 – 1199.

[38] RAMÍREZ-TREVIÑO A, RUIZ-BELTRáN E, RIVERA-RANGEL I, et al. Online Fault Diagnosis of Discrete Event Systems. A Petri Net-Based Approach[J]. IEEE Transactions on Automation Science and Engineering, 2007, 4(1) : 31 – 39.

[39] CASSEZ F, TRIPAKIS S. Fault diagnosis with static and dynamic observers[J]. Fundamenta Informaticae, 2008, 88(4) : 497 – 540.

[40] WANG W, LAFORTUNE S, GIRARD A R, et al. Optimal sensor activation for diagnosing discrete event systems[J]. Automatica, 2010, 46(7) : 1165 – 1175.

[41] THORSLEY D, TENEKETZIS D. Active acquisition of information for diagnosis and supervisory control of discrete event systems[J]. Discrete Event Dynamic Systems, 2007, 17(4) : 531 – 583.

[42] BASILE F, TOMMASI G D, STERLE C. Sensors selection for K-diagnosability of Petri nets via Integer Linear Programming[C] // Proc. 23rd Mediterranean Conference on Control and

Automation (MED). 2015 : 168 – 175.

[43] CABASINO M P, LAFORTUNE S, SEATZU C. Optimal sensor selection for ensuring diagnosability in labeled Petri nets[J]. Automatica, 2013, 49(8) : 2373 – 2383.

[44] RAN N, GIUA A, SEATZU C. Enforcement of diagnosability in labeled Petri nets via optimal sensor selection[J]. IEEE Transactions on Automatic Control, 2019, 64(7) : 2997 – 3004.

[45] SAMPATH M, LAFORTUNE S, TENEKETZIS D. Active diagnosis of discrete-event systems[J]. IEEE Transactions on Automatic Control, 1998, 43(7) : 908 – 929.

[46] CHANTHERY E, PENCOLÉ Y. Monitoring and active diagnosis for discrete-event systems[C] // Proc. 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes. 2009 : 1545 – 1550.

[47] HAAR S, HADDAD S, MELLITI T, et al. Optimal constructions for active diagnosis[C] // Proc. IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science. 2013 : 527 – 539.

[48] BÖHM S, HAAR S, HADDAD S, et al. Active diagnosis with observable quiescence[C] // Proc. 54th IEEE Conference on Decision and Control (CDC). 2015 : 1663 – 1668.

[49] YIN X, LAFORTUNE S. A Uniform Approach for Synthesizing Property-Enforcing Supervisors for Partially-Observed Discrete-Event Systems[J]. IEEE Transactions on Automatic Control, 2016, 61(8) : 2140 – 2154.

[50] CHEN Z, LIN F, WANG C, et al. Active Diagnosability of Discrete Event Systems and its Application to Battery Fault Diagnosis[J]. IEEE Transactions on Control Syetems Technology, 2014, 22(5) : 1892 – 1898.

[51] HU Y, MA Z, LI Z. Active Diagnosis of Petri Nets Using Q-Diagnoser[C] // Proc. 15th IEEE Conference on Automation Science and Engineering. 2019 : 203 – 208.

[52] HU Y, MA Z, LI Z. Design of Supervisors for Active Diagnosis in Discrete Event Systems[J]. IEEE Transactions on Automatic Control, 2020, 65(12) : 5159 – 5172.

[53] HU Y, MA Z, LI Z, et al. Diagnosability enforcement in labeled Petri nets using supervisory control[J]. Automatica, 2021, DOI: https://doi.org/10.1016/j.automatica.2021.109776.

[54] SCHMIDT M, LUNZE J. Active diagnosis of deterministic I/O automata[C] // Proc. 4th IFAC Workshop on Dependable Control of Discrete Systems. 2013 : 79 – 84.

[55] BERTRAND N, FABRE É, HAAR S, et al. Active diagnosis for probabilistic systems[C] // Proc. International Conference on Foundations of Software Science and Computation Structures. 2014 : 29 – 42.

[56] WHITE A, KARIMODDINI A, SU R. Fault Diagnosis of Discrete Event Systems Under Unknown Initial Conditions[J]. IEEE Transactions on Automatic Control, 2019, 64(12) : 5246 – 5252.

[57] TONG Y, LI Z W, SEATZU C, et al. Verification of initial-state opacity in DES[C] //In Proceedings of the 54th IEEE Conf. on Decision and Control. 2015 : 344 – 349.

[58] LAN H, TONG Y, GUO J, et al. Verification of C-detectability Using Petri nets[J]. Information Sciences, 2020, 528 : 294 – 310.

[59] MURATA G T. Petri nets: properties, analysis and applications[J]. Proceedings of the IEEE, 1989, 77(4) : 541 – 580.

[60] CORONA D, GIUA A, SEATZU C. Marking estimation of Petri nets with silent transitions[J]. IEEE Transactions on Automatic Control, 2007, 52(9) : 1695 – 1699.

[61] MA Z, TONG Y, LI Z, et al. Basis Marking Representation of Petri Net Reachability Spaces and Its Application to the Reachability Problem[J]. IEEE Transactions on Automatic Control, 2017, 62(3) : 1078 – 1093.

[62] WONHAM W M, RAMADGE P J. On the supremal controllable sublanguage of a given language[J]. SIAM Journal on Control and Optimization, 1987, 25(3) : 637 – 659.

[63] RAMADGE P J G, WONHAM W M. The control of discrete event systems[J]. Proceedings of IEEE, 1989, 77(1) : 81 – 98.

[64] WONHAM W M. Supervisory Control of Discrete-Event Systems. (2014)[M]. [S.l.] : Dept. Elect. Comput. Eng., University of Toronto, Toronto, ON, Canada [Online]. Available at http://www.control.utoronto.ca/DES, 2015.

[65] LIN F, WONHAM W M. On observability of discrete-event systems[J]. Information Science, 1988, 44(3) : 173 – 198.

[66] HASHTRUDI S, KWONG R H, WONHAM W M. Fault diagnosis in discrete-event systems: framework and model reduction[J]. IEEE Transactions on Automatic Control, 2003, 48(7) : 1199 – 1212.

[67] TAKAI S, KODAMA S. M-controllable subpredicates arising in state feed back control of discrete event systems[J]. International Jonrnal of Control, 1997, 67(4) : 553 – 566.

[68] YIN X, LAFORTUNE S. Synthesis of maximally-permissive supervisors for the range control problem[J]. IEEE Transactions on Automatic Control, 2017, 62(8) : 3914 – 3929.

[69] CORMEN T H, LEISERSON C E, RIVEST R L, et al. Introduction to Algorithms[M]. [S.l.] : MIT Press, 2009.

[70] HERNÁNDEZ-RUEDA K, MEDA-CAMPAÑA M E, ARÁMBURO-LIZÁRRAGA J. Enforcing Diagnosability in Interpreted Petri Nets[J]. IFAC-PapersOnline, 2015, 48(7) : 56 – 63.

[71] MA Z, YIN X, LI Z. Marking Diagnosability Verification in Labeled Petri Nets[J]. Automatica, 2021, in press.

[72] MA Z, YIN X, LI Z. Marking Predictability and Prediction in Labeled Petri Nets[J]. IEEE

Transactions on Automatic Control, 2020, DOI: 10.1109/TAC.2020.3024270.

[73] MA Z, ZHU G, LI Z. Marking Estimation in Petri Nets Using Hierarchical Basis Reachability Graphs[J]. IEEE Transactions on Automatic Control, 2020, 66(2) : 810 – 817.

[74] TONG Y, LI Z W, SEATZU C, et al. Verification of State-Based Opacity Using Petri Nets[J]. IEEE Transactions on Automatic Control, 2017, 62(6) : 2823 – 2837.

[75] GIUA A. Supervisory control of Petri nets with language specifications[G] // SEATZU C, SILVA M, van SCHUPPEN J. Control of Discrete-Event Systems : Vol 433. London, U.K. : Springer, 2013 : 235 – 255.

[76] BASILE F, TOMMASI G D, STERLE C. Non-Interference Enforcement in Bounded Petri Nets[C] // Proc. 2018 IEEE Conference on Decision and Control. 2018 : 4827 – 4832.

[77] BASILE F, TOMMASI G D, STERLE C. Non-interference enforcement via supervisory control in bounded Petri Nets[J]. IEEE Transactions on Automatic Control, 2020, DOI: 10.1109/TAC.2020.3024274.

[78] GIUA A, LAFORTUNE S, SEATZU C. Divergence properties of labeled Petri nets and their relevance for diagnosability analysis[J]. IEEE Transactions on Automatic Control, 2020, 65(7) : 3092 – 3097.

[79] CABASINO M P, GIUA A, SEATZU C. Identification of Petri nets from knowledge of their language[J]. Discrete Event Dynamic Systems, 2007, 17(4) : 447 – 474.

[80] LI Z, ZHOU M, WU N. A Survey and Comparison of Petri Net-Based Deadlock Prevention Policies for Flexible Manufacturing Systems[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2008, 38(2) : 173 – 188.

[81] LI Z W, ZHOU M C. Deadlock Resolution in Automated Manufacturing Systems: A Novel Petri Net Approach[M]. London : Springer, 2009.

[82] LI Z W, WU N Q, ZHOU M C. Deadlock control of automated manufacturing systems based on Petri nets – A literature review[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part C, 2012, 42(4) : 437 – 462.

[83] GIUA A, SEATZU C, BASILE F. Observer-based state-feedback control of timed Petri nets with deadlock recovery[J]. IEEE Transactions on Automatic Control, 2004, 49(1) : 17 – 29.

[84] ZHU G, LI Z, WU N. Model-based fault identification of discrete event systems using partially observed Petri nets[J]. Automatica, 2018, 96 : 201 – 212.

[85] WANG Y, YOO T-S, LAFORTUNE S. Diagnosis of discrete event systems using decentralized architectures[J]. Discrete Event Dynamic Systems, 2007, 17(2) : 233 – 263.

[86] KEROGLOU C, HADJICOSTIS C N. Distributed Fault Diagnosis in Discrete Event Systems via Set Intersection Refinements[J]. IEEE Transactions on Automatic Control, 2018, 63(10) : 3601 –

3607.

[87] WHITE A, KARIMODDINI A. Asynchronous fault diagnosis of discrete event systems[C] // Proc. 2017 American Control Conference (ACC). 2017 : 3224 – 3229.

[88] YIN X, LAFORTUNE S. Synthesis of maximally permissive supervisors for partially observed discrete event systems[J]. IEEE Transactions on Automatic Control, 2016, 61(5) : 1239 – 1254.

[89] YIN X, LAFORTUNE S. On maximal permissiveness in partially observed discrete event systems: Verification and synthesis[C] // Proc. 13th International Workshop on Discrete Event Systems. 2016 : 1 – 7.

[90] TRETMANS J. Model based testing with labelled transition systems[G] // HIERONS R M, BOWEN J P, HARMAN M. Formal Methods and Testing : Vol 4949. Berlin, Heidelberg : Springer, 2008 : 1 – 38.

[91] TRETMANS J. Test Generation with Inputs, Outputs and Repetitive Quiescence[J]. Software Concepts and Tools, 1996, 17(3) : 103 – 120.

[92] GIUA A, SEATZU C, BASILE F. Petri net control using event observers and timing information[C] // Proc. 41st IEEE Conference on Decision and Control. 2002 : 787 – 792.

[93] SU R, WONHAM W M. Global and local consistencies in distributed fault diagnosis for discrete-event systems[J]. IEEE Transactions on Automatic Control, 2005, 50(12) : 1923 – 1935.

[94] QIU W, KUMAR R. Distributed diagnosis under bounded-delay communication of immediately forwarded local observations[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part A: Syetems and Humans, 2008, 38(3) : 628 – 643.

[95] KEROGLOU C, HADJICOSTIS C N. Distributed diagnosis using predetermined synchronization strategies[C] // Proc. 53rd IEEE Conf. Decis. Control Eur. Control Conf.. 2014 : 5955 – 5960.

[96] KEROGLOU C, HADJICOSTIS C N. Distributed diagnosis using predetermined synchronization strategies in the presence of communication constraints[C] // Proc. IEEE International Conference on Automation Science and Engineering. 2015 : 831 – 836.

[97] ATHANASOPOULOU E, LI L, HADJICOSTIS C N. Maximum likelihood failure diagnosis in finite state machines under unreliable observations[J]. IEEE Transactions on Automatic Control, 2010, 55(3) : 579 – 593.

[98] BAZILLE H, FABRE E, GENES B. Diagnosability degree of stochastic discrete event systems[C] // Proc. 56th IEEE Conference on Decision and Control (CDC). 2017 : 5726 – 5731.

[99] CHEN J, KUMAR R. Failure Detection Framework for Stochastic Discrete Event Systems With Guaranteed Error Bounds[J]. IEEE Transactions on Automatic Control, 2015, 60(6) : 1542 – 1553.

[100] KEROGLOU C, HADJICOSTIS C N. Detectability in stochastic discrete event systems[J]. Systems Control Letters, 2015, 84 : 21 – 26.

[101] LIN F, WONHAM W M. On observability of discrete-event systems[J]. Information Sciences, 1988, 44(3) : 173 – 198.

[102] TAKAI S, USHIO T. Effective computation of an L_m(G)-closed, controllable, and observable sublanguage arising in supervisory control[J]. Systems Control Letters, 2003, 49(3) : 191 – 200.

[103] CIESLAK R, DESCLAUX C, FAWAZ A, et al. Supervisory control of discrete-event processes with partial observations[J]. IEEE Transactions on Automatic Control, 1988, 33(3) : 249 – 260.

[104] CAI K, ZHANG R, WONHAM W M. Relative observability of discrete-event systems and its supremal sublanguages[J]. IEEE Transactions on Automatic Control, 2015, 60(3) : 659 – 670.

Acknowledgement

I would like to express my sincere gratitude to all those people who gave me support during my PhD period.

Firstly, I would like to express my appreciation to my supervisors Prof. Zhiwu Li and Prof. Alessandro Giua for their thoughtful support and guidance. Prof. Li has gave me instructive advise in doing research and writing papers. He provided me precious opportunities of following and working with the remarkable professors in the field of Discrete Event Systems. Without his recommendation, I would not have the opportunity to study in Italy. Prof. Giua taught me how to think creatively and critically. His broad theoretical foundation and knowledge in Discrete Event Systems inspired me a lot. I appreciate him for his professional guidance on my research work and help on my life in Italy. I felt so lucky to work with them during my PhD period.

Secondly, I would like to show my sincere gratitude to Dr. Ziyue Ma for his insightful suggestions and great help on my research. All my research work in this thesis is completed with his kindly guidance. I benefited a lot, more than academically, from the experience of working with him.

I am also grateful to all members of the System Control & Automation Group in Xidian University. It is my pleasure to work with them for five years. Particular among them are Dr. Ding Liu, Dr. Yufeng Chen, Dr. Anrong Wang, Dr. Gaiyun Liu, Dr. Jiafeng Zhang, Dr. Xubin Ping, Dr. Lan Yang, Dr. Wang Xi, Mr. JunjunYang, Mr. Qinrui Chen, Mr. Wei Duan, Mr. Peng Nie, Mr. Cong Wang, Mr. Yanjun Zhou, Mr. Tailong Jing, Mr. Dajiang Sun, Ms. Ye Liang, Mr. Jun Li, Mr. Jiazhong Zhou, Mr. Xiaoyan Li, Mr. Xiaoyu Han, Mr. Ziliang Zhang, Mr. Shaopeng Hu, Ms. Yao Lu, Ms. Menghuan Hu, Mr. Kuangze Wang, Mr. Tenglong Kang, Mr. Tianyu Liu, Ms. Wenjie Zhao, Ms. Yulin Zhao, Mr. Kun Peng, Ms. Dan Zhao, Ms. Yanan Zhang, Ms. Yuling Zhang. Thanks for their encouragement and the leisure time we spent together in Xidian University.

In particular, I would like to thank Mr. Nan Du, Mr. Hao Lan, Mr. Chao Gu, Ms. Chao Gao and Ms. Tong Yin for their help and support when I studied in the University of Cagliari. With them I have never felt lonely during my life in Italy.

Last but not the least, I will give the deepest gratitude to my parents, Shuangbo Hu and Lifang Fan, for their constant love and support through my life. Finally, I would like to thank my beloved girlfriend Ms. Chenyang Meng. When I felt frustrated, she always gave me the strongest encouragement and support.

Yihui Hu

August 2021

Biography

1. Basic Information

Yihui Hu (male) was born in Hancheng, Shaanxi Province, China, in July 1994. He received Bachelor Degree in Electrical Engineering and Automation from Xi'an University of Architecture and Technology, Xi'an, China, in 2016. From September 2016 to July 2017 he was a master student of School of Electro-Mechanical Engineering of Xidian University majored in Control Theory and Control Engineering. Since September 2017 he has been a Ph.D student of School of Electro-Mechanical Engineering of Xidian University majored in Control Theory and Control Engineering. During September 2019 to the present, he has been in the program of joint Ph.D degree between Xidian University and University of Cagliari, Italy.

2. Educational Background

2012.09 ～ 2016.07, Xi'an University of Architecture and Technology, China, B.A. in Electrical Engineering and Automation

2016.09 ～ 2017.07, Xidian University, China, M.A. student in Control Theory and Control Engineering (Master-doctor postgraduate student program)

2017.09 ～ present, Xidian University, China, Ph.D student in Control Theory and Control Engineering (Master-doctor postgraduate student program)

2019.09 ～ present, University of Cagliari, Italy, Ph.D student in Electronic and Computer Engineering

3. Academic Publications

Journal Publications

[1] Y. Hu, Z. Ma, and Z. Li, "Design of Supervisors for Active Diagnosis in Discrete Event Systems," *IEEE Transactions on Automatic Control*, vol. 65, no. 12, pp. 5159–5172, 2020. (SCI Journal, JCR Q1)

[2] Y. Hu, Z. Ma, Z. Li, and A. Giua, "Diagnosability Enforcement in Labeled Petri Nets Using Supervisory Control," *Automatica*, vol. 131, 2021. Early access available online. DOI: https://doi.org/10.1016/j.automatica.2021.109776. (SCI

Journal, JCR Q1)

Conference Publications

[1] Y. Hu, Z. Ma, and Z. Li, “Active Diagnosis of Petri Nets Using Q-diagnoser,” In *proceedings of the 15th IEEE International Conference on Automation Science and Engineering*, 2019:203–208. (EI: 20194107517745)

4. Participation in Academic Programs

[1] National Natural Science Foundation of China under Grant No. 61873342 and entitled “Deadlock control and analysis of labeled Petri nets”, 2018.1-2021.12.

[2] National Natural Science Foundation of China under Grant No. 61703321 and entitled “Design of state and behavioral supervisors in Petri nets”, 2018.1-2020.12.