DIAGNOSERS FOR DISCRETE EVENT SYSTEMS: IMPROVED REALIZATION AND EXAMPLES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

BORA ESER KART

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

FEBRUARY 2014

Approval of the thesis:

DIAGNOSERS FOR DISCRETE EVENT SYSTEMS: IMPROVED REALIZATION AND EXAMPLES

submitted by **BORA ESER KART** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Gönül Turhan Sayan
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Şenan Ece Schmidt
Supervisor, **Electrical and Electronics Eng. Dept., METU** _____

Assoc. Prof. Dr. Klaus Werner Schmidt
Co-supervisor, **Mechatronics Eng. Dept., Çankaya University** _____

Examining Committee Members:

Prof. Dr. Semih Bilgen
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Şenan Ece Schmidt
Electrical and Electronics Engineering Dept., METU _____

Prof. Dr. Kemal Leblebicioğlu
Electrical and Electronics Engineering Dept., METU _____

Assoc. Prof. Dr. Cüneyt Bazlamaçcı
Electrical and Electronics Engineering Dept., METU _____

Assist.Prof. Dr. Ulaş Beldek
Mechatronics Engineering Dept., Çankaya University _____

Date: **13.02.2014**

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: BORA ESER KART

Signature :

ABSTRACT

DIAGNOSERS FOR DISCRETE EVENT SYSTEMS: IMPROVED REALIZATION AND EXAMPLES

Kart, Bora Eser
M.S., Department of Electrical and Electronics Engineering
Supervisor : Assoc. Prof. Dr. Şenan Ece Schmidt
Co-Supervisor : Assoc. Prof. Dr. Klaus Werner Schmidt

February 2014, 97 pages

Many complex systems in different areas such as manufacturing, telecommunications or transportation can be modeled as Discrete Event Systems (DES). The task of fault detection and isolation is naturally desired for every system that has the possibility of any fault occurrences in it. To this end, a DES machine that can detect every modeled fault after a bounded number of event occurrence called diagnoser is used.

In this thesis, there are two diagnoser realizations corresponding to the notions of event and language diagnosability. The proposed diagnosers function as centralized diagnosers that run parallel to the given systems and perform online diagnosis. Differing from similar studies, we denote our diagnosers as improved diagnosers because they explicitly give a notification as soon as a faulty behavior is detected. This makes our diagnosers more useful in practice. In addition, our study simplifies the computation of the worst-case delay until a fault is detected. Moreover, we further enhance our improved diagnoser by applying an algorithm to remove unnecessary observations. As a result, fewer sensors are needed and the constructed diagnosers have a

smaller size. The merits of the proposed diagnoser approach and the applicability of our algorithmic implementation are demonstrated by a communication network system example.

Keywords: diagnoser, DES, failure diagnosis, online diagnosis, event diagnoser, language diagnoser

ÖZ

AYRIK OLAY SİSTEMLERİ İÇİN TANILAYICILAR : GELİŞTİRİLMİŞ GERÇEKLEŞTİRME VE ÖRNEKLER

Kart, Bora Eser
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi : Doç Dr. Şenan Ece Schmidt
Ortak Tez Yöneticisi : Doç. Dr. Klaus Werner Schmidt

Şubat 2014, 97 sayfa

Üretim, telekomünikasyon veya ulaştırma gibi farklı alanlardaki bir çok karmaşık sistem ayrık olaylı sistemler olarak modellenebilmektedir. İçerisinde hata olma olsalığı barındıran her sistemdeyse hata tanılama ve tecriti tabii olarak istenmektedir. Bu amaçla, tanılayıcı adı verilen, sınırlı sayıda olay meydana geldikten sonra, modellenmiş her hatayı tespit edebilen bir ayrık olaylı sistem makinesi kullanılır.

Bu tezde, olay ve dil tanılabilirliği kavramları üzerine iki adet tanılayıcı önerilecek ve bu tanılayıcıların her biri merkezi ve verilen sistemin paralelinde çalışan çevrimiçi bir tanılayıcı olarak görev yapacaktır. Benzer çalışmalardan farklı olarak kendi tanılayıcılarımızdan gelişmiş tanılayıcılar olarak bahsedeceğiz çünkü bu tanılayıcılar hata tespit edilir edil- mez açık bir şekilde uyarı vermekteler.Bu da bizim tanılayıcılarımız uygulamada daha kullanışlı yapmaktadır. Ek olarak, çalışmamız en kötü durumdaki gecikme hesaplamalarını sadeleştirmektedir. Bunların yanı sıra, gelişmiş tanılayıcımızı geliştir- diğimiz başka bir yöntem aracılığıyla gereksiz olayları gözlemleme ihtiyacından kurtararak daha da geliştireceğiz. Böylece daha az sensöre ihtiyaç duyulacak ve yapılan tanılayıcı daha küçük bir boyuta sahip olacaktır. Önerilen tanılayıcının yarar-

ları ve algoritmaların uygulanabilirliği bir iletişim ağı örneği ile gösterilmiştir.

Anahtar Kelimeler: tanılayıcı, ayrık olaylı sistemler, çevrimiçi tanılama, olay tanılayıcı, dil tanılayıcı

To My Beloved Family

ACKNOWLEDGMENTS

I would like to express my special thanks to my supervisor Assoc. Prof. Dr. Şenan Ece Schmidt and Assoc. Prof. Dr. Klaus Werner Schmidt not only for their guidance on my thesis but also for their great friendship and support in the course of this process.

I could not finish this thesis, even could not start this thesis without having their support. It was really an honor to work with them as a master student. Every graduate student should have supervisors like them to be encouraged and guided like me.

This thesis is dedicated to my family since they have never stopped encouraging me to complete my thesis work, not even a moment. Their love and guidance was also very important for me.

I would also like to express my appreciation to TÜBİTAK for their financial support to my thesis. It is impossible ignore their impact on my motivation to complete this thesis.

TABLE OF CONTENTS

APPENDICES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

DES	Discrete Event Systems
HVAC	Heating Ventilating Air Conditioning
SPEC	Specification Automaton
G	Overall System Behavior Automaton
G_{bd}	Basic Diagnoser Automaton
G_{md}	Modified Diagnoser Automaton
G_{rmd}	Reduced Modified Diagnoser Automaton
G_{spec}	Specification Automaton
K	Specification Language
L	Language
LC	Label Correction Function
LP	Label Propagation Function
M	Masking Function
q_d	Any Diagnoser State
q_0	Initial State
R	Range Function
s_b	Back Trace String
s_f	Forward Trace String
wcdd	Worst Case Detection Delay
x_0	Initial state
Δ_{dd}	Local Detection Delay
Σ_f	Faulty Event Set
Σ_o	Observable Event Set

Σ_{uo}	Unobservable Event Set

CHAPTER 1

INTRODUCTION

Fault detection and isolation by monitoring the system behavior has become a very important task for large and complex systems. There are a number of DES approaches on fault diagnosis for the systems that can be modeled as DES such as manufacturing systems [1], [2], [3], [4], telecommunications systems [5], [6], transportation systems [7], HVAC systems [8], [9] and document processing systems [10].

Fault diagnosis stands for the detection of any abnormal or undesired behavior for a system under partial observation. Furthermore, this notion requires the detection of the abnormal behavior within a finite time delay after the fault occurred in the system. In this framework, a DES is called *diagnosable* if every faulty behavior introduced to the system can be detected within a finite number of event observations. Since the fault diagnosis is a widely studied notion in the literature, there are several studies conducted to test if a DES is diagnosable, or not [8], [11], [12], [2], [13], [3], [14], [15], [4], [16].

The finite state automaton that is used for the task of fault diagnosis is called the *diagnoser*. There are different type of diagnosers according to the technique used for distinguishing faulty behavior. The diagnoser may perform its diagnosis task either by using *online diagnosis*, i.e. observing of the system behavior and evaluating the outputs generated by the system [8], [5] or by being used as a test bench for *offline diagnosis* [11], [2] to test if a given system is diagnosable or not by analyzing whether previously defined necessary and sufficient conditions are satisfied for the system. *Centralized diagnosis* [2], [8], [11], [13], [9], [3] is a diagnosis approach in which a single diagnosis unit is used for observing the whole behavior of the system

whilst *decentralized diagnosis* [17], [12], [18], [14], [19], [15], [4] is the diagnosis approach in which information is distributed to the several parts of the system and diagnosis units at different local sites are used to collect the distributed information and conclude fault occurrence.

This thesis focuses on centralized diagnosis and studies two types of diagnosers. An *event diagnoser* may recognize a faulty behavior by introduction of fault events [8], [2] and a *language diagnoser* introduces another automaton used for fault specification [12], [11]. An event diagnoser diagnoses the occurrence of a previously defined fault event in the system if all the possible faults can be predicted. However, some sequence of events can exist which are not desired to appear during the execution of a system even though none of the events in the sequence can be categorized as a fault event. For such systems, the normal behavior of the system is defined by another automaton by clearing all the undesired transitions from the possible behavior automaton of the system. Then, a language-diagnoser can be used to detect faults.

In this thesis work, firstly we introduce a modified event diagnoser and a modified language diagnoser together with the motivation for developing these improved diagnosers considering the previous work in the literature. We then present the construction algorithms for these diagnosers. In addition, we develop an algorithm for computing the worst-case delay until a fault can be detected and we realize an algorithm for the reduction of the set of observable events by removing unnecessary observations. All methods and algorithms are illustrated by academic examples and a communication system example demonstrates the applicability of the thesis work. The construction algorithm of the modified event-diagnoser is published in [20].

In summary, the contributions of the thesis are as follows:

- Implementation of the construction algorithms for the modified diagnosers using the software library libFAUDES [21], [22].
- Computation of the worst case detection delay for both diagnosers
- Implementation of an algorithm for the reduction of the observable event set [23]

- A detailed case study on a communication network example to illustrate the merits of the diagnosers and their implementation

After this introduction section, in Chapter 2 the required background information on discrete event systems is summarized. The construction of the modified event-diagnoser is described in Chapter 3 and the modified language-diagnoser is developed in Chapter 4 including the computation of the worst-case detection delay. After giving an algorithm to have an optimal set of observable events for the improved diagnoser in Chapter 5, our proposed improved language-diagnoser is applied to a communication network example in Chapter 6. Concluding remarks and ideas for future work are given in Chapter 7.

CHAPTER 2

BACKGROUND

This chapter gives a brief summary of the background information on discrete event systems (DES) and diagnosability. The basic notions and definitions provided in this chapter are required for a better understanding of the following chapters. We refer to [24] for further information.

2.1 Discrete Event Systems (DES)

A DES is described as an *event-driven* system having a *discrete* set of states. This kind of system performs asynchronous transitions between its states at asynchronous discrete points in time via the occurrence of *events*. That is, a DES has an event-driven state transition mechanism.

To be able to develop suitable models for studying DES and describe the behavior of the system with the goal of design, control or fault diagnosis, a DES can be formally modeled by *language*s and *automata*.

2.2 Languages

The usual way of starting to analyze a DES begins with defining the associated events, whereby the finite event set is denoted as the *alphabet* $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_m\}$. A sequence of events is called a *word* or *string*. From now on, we will use the term string for sequence of events in a DES. The empty string is writen as the symbol ε and is

described as a string consisting of no events. If s is a string, its *length* is denoted as $|s|$ (the number of events in the string).

Definition 1 (Language) *A set of finite-length strings that is formed from events in its event set* Σ *is defined as a language.*

The important thing to note here for a language is its possibility of having an infinite number of strings with arbitrary but finite length.

In the scope of thesis, there are some key operations on strings that must be known by the reader. Therefore, at first, we briefly introduce some terminology about strings and some of the most common string operations.

Consider the string $s = tuv$ with $t, u, v \in \Sigma^*$, then

- t is called a *prefix* of s,
- u is called a *substring* of s,
- v is called a *suffix* of s.

In the later parts of this work, we write s/t for the suffix of s after the prefix t.

- *Kleene Closure:* Let $L \subseteq \Sigma^*$ be a lanaguage. Then $L^* := \{\varepsilon\} \cup LL \cup LLL \ldots$

  For example, all finite strings of elements of Σ including the empty string ε are contained in Σ^*, which is the Kleene closure of Σ,

- Union, intersection, difference are applicable to languages as usual set operations since languages are also sets.

- *Concatenation:* Let $L_a, L_b \subseteq \Sigma^*$, then $L_a, L_b := \{ s \epsilon \Sigma^* | \ s = s_a s_b, \ s_a \epsilon \ L_a, \ s_b \epsilon L_b\}$.

  The identity element of the concatenation operation is the language $\{\varepsilon\}$ since $u\varepsilon = \varepsilon u = u$ for any u

- *Complement:* Let $L \subseteq \Sigma^*$, then the complement of L is defined as $L_c = \Sigma^*$ - L.

- *Prefix-Closure:* Let $L \subseteq \Sigma^*$, then we can define the prefix-closure $\overline{L}$ for language L as

$$\overline{L} := \{s \in \Sigma^* | \exists t \in \Sigma^* \text{ such that } st \in L\}.$$

L is said to be prefix-closed if $L = \overline{L}$.

Hence, in words, we can say that language L is prefix closed if any prefix of any string in L is also an element of L.

- *Post Language:* Let $L \subseteq \Sigma^*$ and $s \in L$

The post language of L after s is denoted by L/s and defined as the language

$$L/s := \{t \in \Sigma^* | st \in L\}.$$

By definition, $L/s = \emptyset$ if $s \notin L$

Definition 2 (Natural Projection) *: for an observation alphabet $\hat{\Sigma} \subseteq \Sigma$ we define natural projection as:*

$$P : \Sigma^* \to \hat{\Sigma}^*$$

where

$$\begin{aligned} &P(\varepsilon) := \varepsilon \\ &P(\sigma) := \begin{cases} \varepsilon \text{ if } \sigma \in \hat{\Sigma} \\ \sigma \text{ if } \sigma \notin \hat{\Sigma} \end{cases} \\ &P(s\sigma) := P(s)P(\sigma) \text{ for } s \in \Sigma^* \text{ and } \sigma \in \Sigma \end{aligned} \tag{2.1}$$

For better understanding, we can explain the projection operation as an operation that erases the events which do not belong to the smaller event set $\hat{\Sigma}$ from strings $s \in \Sigma^\star$.

There is an inverse operation called *inverse projection* P^{-1} by which we can obtain all the strings of the larger event set from a given smaller set.

After indicating that notation 2^A means the power set of A, i.e. the set of all subsets of A, we can formulate the definition of the inverse projection.

Definition 3 (Inverse Projection) *: Inverse projection* $P^{-1} : \hat{\Sigma}^* \rightarrow 2^{\Sigma^*}$ *is defined as*

$$P^{-1}(\hat{s}) := \{s \in \Sigma^* \mid p(s) = \hat{s}\} \tag{2.2}$$

for $\Sigma \subseteq \hat{\Sigma}^*$.

We can as well apply the natural projection to languages by applying the natural projection to each string of a languages. For $L \subseteq \Sigma^*$, natural projection is defined as

$$P(L) := \{t \in \hat{\Sigma}^* | \exists s \in L \text{ such that } p(s) = t\}$$

and for $L \subseteq \hat{\Sigma}^*$,

$$P^{-1}(L) := \{t \in \Sigma^* | \exists s \in L \text{ such that } p(s) = t\}$$

Therefore, in general $P^{-1}[P(L)] \neq L$ but rather $L \subseteq P^{-1}[P(L)]$ for $L \subseteq \Sigma^*$.

2.3 Automata

A convenient way of representing languages is the use of automata. In this section, we will formally define the notion of an automaton. First, we start with non-deterministic automata.

Definition 4 (Non-deterministic Automata) *:* A *non-deterministic automaton* is a denoted by G_{nd} and is a five-tuple defined as

$$G_{nd} = (X,\ \Sigma,\ \delta_{nd},\ X_0,\ X_m) \tag{2.3}$$

where the meanings of the elements of five-tuple are:

X	:	the set of *states*,
Σ	:	the finite set of *events* including the empty string ε,
δ_{nd}	:	$X \times \Sigma \rightarrow 2^X$ the *transition function*
X_0	:	the set of *initial states* and $X_0 \subseteq X$
X_m	:	the set of *marked states* and $X_m \subseteq X$

There is also an active event function $\Gamma : X \rightarrow 2^{\Sigma}$ that returns the set of all events σ for which $\delta_{nd}(x, \sigma)$ is defined for the state x of G_{nd} . It may take place in the definition but we are omitting it in the definition since it can easily be derived from δ_{nd}.

For applying δ_{nd} to a string u, we can extend its domain to $X \times \Sigma^*$.

$$\delta_{nd}(x, u\sigma) := \{\mathrm{z} \mid \mathrm{z} \in \delta_{nd}(y, \sigma) \text{ and } \mathrm{y} \in \delta_{nd}(x, u)\ \}$$

We will use the words *automaton* and *generator* for the object defined with this five-tuple. It can be thought that the word generator is the reason for which we use notation G for the object.

Since we have defined what an automaton is, now we are capable of defining what *generated language* is by considering all of the directed traces of a generator.

Definition 5 (Generated Language) *:* The *generated language* of G_{nd} is defined as

$$L(G_{nd}) = \{s \in \Sigma^* \mid \exists x \in x_o \textit{ and } \delta_{nd}(x, s) \textit{ is defined}\} \tag{2.4}$$

There is also another notion of automata called *deterministic automaton* and it is nothing but a frequently used special case case of non-deterministic automaton. The differences between deterministic automata and non-deterministic automata is given by the fact that deterministic automata have only one initial state and their transition function δ maps to a unique successor state for each predecessor state.

Definition 6 (Deterministic Automata) *:* A *deterministic automaton* is denoted by G and is a five-tuple defined as

$$G = (X,\ \Sigma,\ \delta,\ x_0,\ X_m) \tag{2.5}$$

where the elements of these five-tuple machine have the same definition as in the definition of non-deterministic automaton except the transition function δ and initial state x_0.

δ : X $\times\Sigma \rightarrow X$, the transition function causes a transition to a unique state in set X for each event $\delta \in \Sigma$ in a state $x \in X$.

The transition function δ can be extended from domain $X \times \Sigma$ to $X \times \Sigma^*$ to be able to apply to strings as well

$$\begin{aligned} \delta(x, \varepsilon) &:= x \\ \delta(x, s\delta) &:= \{\, \delta(\delta(x, s), \sigma) \mid s \in \Sigma^* \text{ and } \sigma \in \Sigma \,\} \end{aligned}$$

If the state space X of such an object is a finite set, we call it a *finite-state automaton*. Therefore, we use the term *deterministic finite-state automaton* for such G and *non-deterministic-finite state automaton* for the previously defined G_{nd}.

The generated language definition for the deterministic automaton G is similar to the one for G_{nd}.

$$L(G) = \{s \in \Sigma^* \mid \exists x \in x_o \text{ and } \delta_{nd}(x, s) \text{ is defined } \}$$

After completing the definition of generated language for deterministic automaton and what a finite-state automaton is, we will end this section by describing what a deadlock is since it will be used in our assumptions later.

If an automaton G reaches to a state x of its where $\Gamma(x) = \emptyset$ and $x \notin X_m$, this is called a *deadlock*. It is called a deadlock because no further event can be executed hereafter. So we can say that the system *blocks* the given string and enters a deadlock state.

Finally, we introduce the parallel (synchronous) composition $G_1 \parallel G_2$ for automata. In substance, it can be thought that $G_1 \parallel G_2$ represents an interconnection of systems G_1 and G_2. The parallel composition synchronizes events that are shared between G_1 and G_2, whereas the remaining events occur independently. Consider two automata

$$G_1 = (X_1, \Sigma_1, \sigma_1, x_{01}, X_{m1})$$

$$G_2 = (X_2, \Sigma_2, \sigma_2, x_{02}, X_{m2})$$

Definition 7 (Parallel Composition) *:* The *parallel composition* of G_1 and G_2 gives us the automaton

$$G_1 \parallel G_2 := Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, \delta, (x_{01}, x_{02}), X_{m1} \times X_{m2}) \tag{2.6}$$

where

$$\delta((x_1, x_2), \sigma) := \begin{cases} (\delta(x_1, \sigma), \delta(x_2, \sigma)) & \text{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (\delta(x_1, \sigma), x_2) & \text{if } \sigma \in \Gamma_1(x_1) \setminus \Sigma_2 \\ (x_1, \delta(x_2, \sigma)) & \text{if } \sigma \in \Gamma_1(x_2) \setminus \Sigma_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

as the operator Ac means that we only care about the accessible part of the automaton from the initial state. To be more explicit for the operation, an event in the set $\Sigma_1 \cap \Sigma_2$ is called a *shared event* and can only be executed on the composed system if two automata G_1 and G_2 can both execute it simultaneously. Hence we can say that two component automata are synchronized on the shared events.

From the parallel composition of automata, the parallel composition of languages can also be obtained.

$$L(G_1 \parallel G_2) \quad = \quad L(G_1) \parallel L(G_2)$$

2.4 Diagnosis

In this chapter, we will give a brief description of diagnosis and diagnosability, then introduce the reader with event diagnosability and language diagnosability.

2.4.1 The notion of Diagnosability

Diagnosability is one of the most studied properties of DES in the scope of analysis. In this part, we will present diagnosability of DES with respect to failure events. The basic purpose for studying diagnosability is to identify if any certain failure events occurred during execution of the system. There can happen observable or unobservable failure events in a system and our main concern for this thesis work is the unobservable failure events. The reason for dealing only with the unobservable failures is obvious since it will be a trivial case to deal with the observable failures taking into

account that they are already observable and detectable. To achieve the task of identifying the occurrence of certain unobservable events will be achieved by tracking all of the observable events that occur in the system and estimating the actual state of the system.

[8] and [11] conducted the initial studies on the diagnosability subject and introduce a systematic procedure for both detection and isolation of failure events. For this purpose, necessary and sufficient conditions for a language to be diagnosable are provided. These conditions are studied under existence of several restrictive assumptions such as liveness of the system and absence of unobservable event cycles. For now, we will only say that the verification of the diagnosability property of the system is performed off-line.

A diagnoser is used online for tracking system information about potential failure occurrences. If the diagnoser is able to detect the occurrence of a failure within a finite delay, it can be said that the system is diagnosable. The important thing that must be noted here is that a diagnoser works online to detect if a failure happened in the system while diagnosability verification is made with an offline analysis of the system.

We need to introduce some other notions such as unobservable events and liveness before defining the diagnosability according to [8]. Let us have a deterministic finite state automaton

$$G = (X, \Sigma, \delta, x_0, X_m) \tag{2.7}$$

There can be two types of events in the whole event set Σ. Some of these events are observable so that we can detect their occurrence in the system while the rest are unobservable. Therefore, we will split the event set Σ of the automaton G into two partitions as

$$\Sigma = \Sigma_o \cup \Sigma_{uo} \tag{2.8}$$

where Σ_o represents the set of observable events and Σ_{uo} represents the set of un-

observable events in the system. To be more explicit for the readers, we may call commands that a controller issued, sensor readings measured immediately after the execution of these controller commands and the changes recorded by sensor readings as observable events. Beside this, the unobservable events may be the failure events. Moreover they can be any other events that cause a change in the system but are not recorded by the sensors.

As mentioned formerly, we are dealing with the detection of the occurrence of failure events that happened in a running system. The set of failure events to be diagnosed is denoted by $\Sigma_f \subseteq \Sigma$. Since it is already discussed that it would be a trivial case to diagnose observable failures by only observing them as they are already observable events, we can assume that $\Sigma_f \subseteq \Sigma_{uo}$ without loss of any generality.

Studying diagnosis means that our main objective is to identify, if any, the occurrence of the failure events by observing only the events in Σ_o in the traces generated by the system. But it may sometimes be impossible to uniquely diagnose each fault occurred in the system due to inadequate instrumentation's rendering it impossible. Moreover, we may simply be interested in detecting the occurrence of a failure event in a set of failure events when the effects of this set of failure events on the system are recorded as same. Therefore, for this purpose the set of failure events Σ_f is partitioned into disjoint sets corresponding to different failure types exists in the system.

$$\Sigma_f = \Sigma_{f1} \cup \Sigma_{f2} \cup \ ... \ \cup \Sigma_{fm} \tag{2.9}$$

This partition is denoted by Π_f and $\Pi_f = \{1, ..., m\}$ denotes the index set enumerating the partitions for the failure types. Therefore, if we are talking about a failure of type F_i, indeed we mean some event from the set Σ_{fi}. It will be obvious that a failure of type F_i has occurred will mean that some event from that set has occurred.

We define some final preliminary notions before defining what diagnosability is for the reader. Let p_o denotes the operation of simply erasing the unobservable events in a given string for the system. Thus we will define the projection $p_o = \Sigma^* \rightarrow \Sigma_o^*$ as the the definition of projection that we explained formerly :

$$
\begin{aligned}
p_o(\varepsilon) &= \varepsilon \\
p_o(\sigma) &= \sigma \qquad if \sigma \in \Sigma_o \\
p_o(\sigma) &= \varepsilon \qquad if \sigma \in \Sigma_{uo} \\
p_o(s\sigma) &= p_o(s)p_o(\sigma) \quad if s \in \Sigma^*, \sigma \in \Sigma^*
\end{aligned}
\tag{2.10}
$$

And the inverse projection of p_o on the language L is denoted by p_L^{-1} and is defined as

$$
p_L^{-1}(y) = \{s \in L \,\|\, p_o(s) = y\} \tag{2.11}
$$

Since we will use it later, it is better to define here X_o as

$$
X_o = \{x_0\} \cup \{x \in X \,\|\, \text{ there is an observable transition ending in x}\} \tag{2.12}
$$

and $L_o(G, x)$ with $L_\sigma(G, x)$ as

$$
\begin{aligned}
L_o(G, x) &= \{s \in L(G, x) \,\|\, s = u\sigma, u \in \Sigma_{uo}^*, \sigma \in \Sigma_o\} \\
L_\sigma(G, x) &= \{s \in L_o(G, x) \,\|\, s_f = \sigma\}
\end{aligned}
\tag{2.13}
$$

to denote the set of all traces that originating from state x, ending at the first observable event and traces that ends with a particular observable event σ correspondingly where $L(G, x)$ denotes the set of all traces that originate from state x of system G.

Finally, we explain the use of s_f notation to denote the final event of a trace s and define the operation Π as

$$
\Pi(\Sigma_{fi}) = \{s\sigma_f \in L \,\|\, \sigma_f \in \Sigma_{fi}\} \tag{2.14}
$$

where $\Pi(\Sigma_{fi})$ defines the set of all traces of L that end with a failure event belonging to the class Σ_{fi}. With the slight abuse of the notation, to make it look similar to the notation $\sigma \in s$ which denotes the fact that σ is an event in the trace s, we may write

$\Sigma_{fi} \in s$ to denote the fact that $\sigma_f \in s$ for some $\sigma_f \in \Sigma_{fi}$. To be more formally, it can also be expressed as:

$$\overline{s} \cap \Pi(\Sigma_{fi}) \neq \emptyset$$

where $\overline{s}$ is the prefix-closure of s.

2.4.2 Event Diagnosability

The reader is introduced with the necessary preliminary definitions and notions formerly, now it seems possible to make a definition of diagnosability formally. We will use the formal definition of diagnosability presented in [8]. In order to highlight the objective of the kind of diagnosability and since, later on we will mention about other kinds of diagnosability, we call this notion of diagnosability as *event diagnosability*.

To start with, we need the system under investigation meet some important assumptions. These assumptions are liveness and having no cycles of unobservable events.

Assumption 1 *The language $L(G)$ is live. Hence, there is a transition defined at each state x in X and the system can not reach a deadlock state at which no transition event is possible.*

Assumption 2 *No cycles of unobservable events exist in the system G, i.e.*

$$\exists n_0 \in \mathbb{N} \text{ such that } \forall ust \in L, s \in \Sigma_{uo}^* \Rightarrow |s| \leq n_0 \tag{2.15}$$

In other words, since the diagnosing failures task is based on tracking observable events, system G does not generate any numbers of arbitrarily long sequences of unobservable events. Hence, we can say that the assumption in Equation 2.15 ensures the regularity of the occurrence of observations.

As indicated in the previous parts, by studying diagnosability, our main purpose is to detect the occurrence of a failure of any failure type. We can briefly say that a language L is diagnosable if detection of the fault is succeeded within a finite delay by tracking only the observable events generated by the system.

It mentioned in [8] that there are two important definitions of diagnosability survey. First definition is referred as diagnosability and the second one is referred as I-diagnosability.

Definition 8 (Diagnosability) *:* A prefix-closed and live language L is said to be diagnosable with respect to the projection operation p_o and with respect to the partition Π_f on Σ_f if the following proposition holds.

$$(\forall i \in \Pi_f)(\exists n_i \in \mathbb{N})[\forall s \in \Pi(\Sigma_{fi})](\forall t \in L \setminus s)[\|t\| \geq n_i \Rightarrow D] \tag{2.16}$$

where the diagnosability condition D is defined as:

$$D : \omega \in P_L^{-1}[p_o(st)] \Rightarrow \Sigma_{fi} \in \omega \tag{2.17}$$

For making the definition more clear to the reader, let us have a string $s \in L$ generated by the system and ending in a failure event f from the set Σ_{fi}. The diagnosability condition D for event-diagnosis indicates us that every trace belonging to the language L, i.e. $\omega \in L$, that produces the same record of observable events as the string st should contain in it should contain a failure from the event set Σ_{fi} for a sufficiently long continuation t of s. Therefore, along every continuation t of s, the occurrence of a failure of F_i can be detected with a finite delay, in at most n_i transitions generated in the system after the string s. To sum up, diagnosability property of a system requires that every failure event should lead to observations distinct enough to enable identification of that failure type uniquely.

If there are multiple failures from the same set of same failure type partition, F_i, occurring along a trace s of L, the definition that we have made recently does not require the detection of each of these failure occurrences separately. Concluding that a failure from the set Σ_{fi} has happened within finitely many events after the occurrence of the first failure in the trace s is sufficient.

To be able to help the reader understand the definition of diagnosability for event-diagnosis in a better way, we will use the example system given in Figure 2.1

Figure 2.1 shows a system to that we will demonstrate what event-diagnosis and diagnosability are in a more explicit way. Consider system G where observable events of

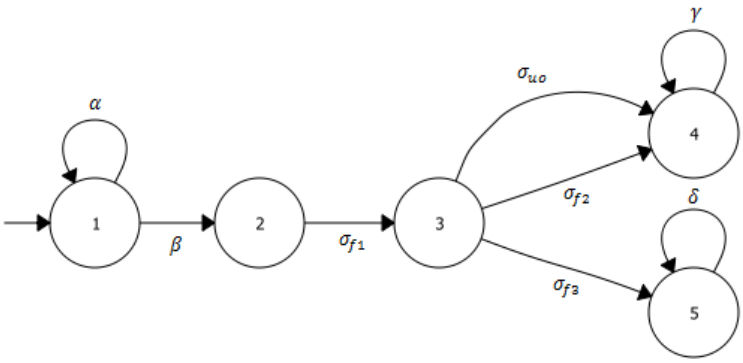


Figure 2.1: Example system for event-diagnosability

the system are $\Sigma_o = \{\alpha, \beta, \gamma, \delta\}$ while unobservable event set $\Sigma_{uo} = \{\sigma_{uo}, \sigma_{f1}, \sigma_{f2}, \sigma_{f3}\}$ includes only one non-failure event, i.e. σ_{uo} and σ_{f1}, σ_{f2} and σ_{f3} representing failure events. Let the initial state x_0 of the system be the state denoted with number 1. If the failure partition is chosen as $\Sigma_{f1} = \{\sigma_{f1}, \sigma_{f2}\}$ and $\Sigma_{f2} = \{\sigma_{f3}\}$, it is not required to distinguish between occurrences of failures σ_{f1} and σ_{f2}. With a quick analysis on the system G, it is easily seen that the system is diagnosable with this partition with finite delays $n_1 = 2$ and $n_2 = 1$.

On the other hand, if we select the failure partition as if it is $\Sigma_{f1} = \{\sigma_{f1}\}$, $\Sigma_{f2} = \{\sigma_{f2}\}$ and $\Sigma_{f3} = \{\sigma_{f3}\}$, the system becomes not diagnosable because then it is not possible to detect the occurrence of failure event σ_{f2} by tracking only the observable events generated by the system.

2.4.3 Language Diagnosability

In this chapter, after presenting the notion of event diagnosis and I-diagnosability, we will present the notion of language-diagnosability where abnormal behaviors of the system are specified with language differing from the case in event-diagnosis. Let us first begin this section with a formal definition of language-diagnosis.

Definition 9 (Language Diagnosability) *:*A prefix-closed language L is said to be *language-diagnosable* with respect to a prefix-closed language $L_n \subseteq L$ and a projec-

tion operation p_o over the events defined in L if the following holds.

$$(\exists n_d \in \mathbb{N})(\forall s \in L \setminus L_n)(\forall t \in L \setminus s)[(L \setminus L_n = \emptyset) \vee (\|t\| \geq n_d) \Rightarrow D_l] \tag{2.18}$$

where the diagnosability condition D_l is defined as:

$$D_l : P_L^{-1}[p_o(st)] \cap L_n \neq \emptyset \tag{2.19}$$

Moreover, the worst case failure detection delay of L with respect to L_n and p_o is defined as follows:

$$d_{dia} = min(n_d) \tag{2.20}$$

Thus, the language L provided with the worst-case detection delay parameter d_{dia} is called as a $d_{dia} - step\ language - diagnosable$ with respect to L_n and p_o. Beside this, L_n and L represent the normal behavior and the possible behavior of the system, respectively. Therefore, the abnormal behavior for language L is represented by the notation $L \setminus L_n := L \cap \overline{L_n}$.

In order that we can help the reader understand the notion of language diagnosability more clearly, let us illustrate this notion on an example system, its possible behavior and normal behavior.

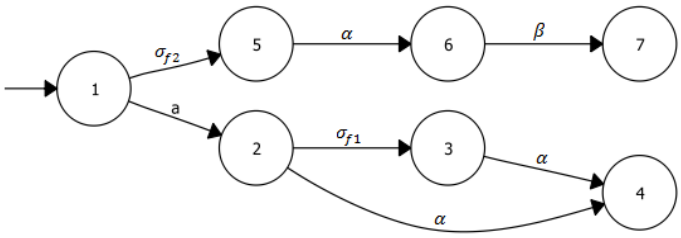


Figure 2.2: Example possible system G behavior for language-diagnosability

Figure 2.2 shows the possible behavior of our system and Figure 2.3 shows the desired normal behavior of its, which we call it spec of the system later on. It is easy to notice that the normal behavior of the system is exactly the system that is a part of the possible behavior cleared of the failure events and their corresponding paths.

In our system observable event set is composed of the letters belonging to the Greek alphabet, i.e. $\Sigma_o = \{\alpha, \beta\}$ while unobservable event set is composed of unobservable normal events which are indicated by the Latin alphabet and unobservable failure events denoted by σ_{f1} and σ_{f2}, that is $\sigma_{uo} = \{a, \sigma_{f1}, \sigma_{f2}\}$. All of the system behavior other than the normal behavior can be obtained by $G \setminus G_N$.

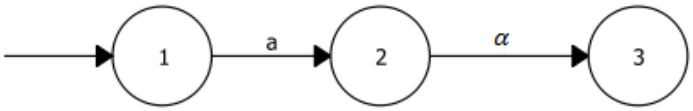


Figure 2.3: Example normal system behavior G_N for language-diagnosability

Unfortunately, diagnosability case is violated in our example. Even though an observer is able to uniquely diagnose the occurrence of failure σ_{f2}, which is not described in normal behavior G_N of system G, immediately after the occurrence of observable event β, the faulty string "$a\sigma_{f1}$" including the failure event σ_{f1} cannot be resolved by the observer. It cannot be resolved because there are non-faulty strings with the same observation in the set of all possible extensions of our faulty string "$a\sigma_{f1}$". Therefore, system G is not language-diagnosable with respect to normal behavior spec G_N and natural projection function p_o.

2.5 Discussion and Comparison with Related Work in Literature

In this section of the thesis work, we will briefly mention about the works in the literature related with the notion of diagnosability, as a superset notion both including event and language diagnosability. First of all, we will start by correlating two notions, language diagnosability and event diagnosability, then have a glance over other notions related with the diagnosability.

To begin with correlating the notions of language and event diagnosability, we note that language-diagnosability is more general than event-diagnosability. First, language-diagnosability does not require restrictive assumptions such as the absence of dead-

lock states and unobservable cycles. Second, any event-diagnosability problem can be converted into a language-diagnosability problem [11]. Hereby, the specification K for language-diagnosis simply needs to contain all strings without fault events.

Next, we point to some related work in the literature. It is very clear that our diagnosability notion is highly related with partial observation problem since the notion is with respect to a the observation function, i.e. natural projection. Although the diagnosability problem hasn't been studied in advance with high reputation, partial observation problems in DES's such as observability, observability with delay, invertibility etc. have been a recently popular investigation in the literature. The researches of [25], [26], [27], [28], [29], [30], [31] can be shown as examples of these studies. Nonetheless, diagnosability is a distinctly different concept even thought it seems to be related to formerly mentioned subjects. Partitioning the failure events, need to identify each failure type within a finite time, possibility of the case of multiple failures, possible unobservable events other than the failure events, no need of diagnosis during normal operation of the system can be counted for the differences of diagnosability notion. Now we will introduce the other notions to the reader some of which we may inspire later on and have a brief discussion on each other to show the differences and the similarities between these notions and diagnosability.

In his paper [32], Lin proposes the reader a state-based approach for studying diagnosability. A partial state information exists in the approach via an output function. The problems of off-line and on-line diagnosis is addressed in the research. By the off-line diagnosis term, analyzing the system to be diagnosed in a test-bed procedure can be thought. In this procedure, a sequence of test commands, observing the resulting outputs and drawing inferences on the set of possible states in which the system could be is involved. This can also be thought as the verification of diagnosability problem for the system. On the other hand, the system is assumed to be operating normally in on-line diagnosis. Identifying the state of the system is the aim of diagnosis process but differing from the off-line diagnosis case, possible occurrences of other uncontrollable events in the system must be accounted for diagnostic process. In [32], the author gives a guaranteed algorithm that is supposed to converge if the system is on-line diagnosable.

An extended off-line diagnosability research in which Bavishi considers testability of given DES is conducted in [33]. Moreover algorithms for determining both the optimal set of sensors to ensure testability and infimal partition of the state space is also presented in his study.

Language observability notion is introduced to the reader in Lin's study [27] as a supervisory control problem with the constraint of partial event observation. Thus, a language based definition of observability and state conditions for the existence of a solution to this control problem in terms of observability and controllability of the language is presented. In the problem, it is not necessary to explicitly determine the occurrence of unobservable events or identify the system state. We can conclude that diagnosability notion is different than the notion of observability introduced in this study.

The problem of state identification for DES is addressed in Ramadge's paper [31]. The system is modeled by a non-deterministic automaton with full event observability and partial state observability with an output map in his study. Reconstructing the state of the system after each event is the problem and there is an observer-state feedback approach for the synthesis of the controller. The work done in Ramadge's paper is also incomparable with the diagnosability problem in our case.

A slightly different approach for observability that is assuming a partial event observation model with no state observation directly is presented in the study of Özveren and Willsky [28]. They mark a system as observable if it is possible to determine the current state of it exactly using a record of observable events separated by a bounded number of events. Also they define a DES called an observer that is producing estimates of the state of the system after each observable event generated in the system. The notion of observability with delay is also defined in the study for a system in which it is possible to have perfect knowledge not of the current state of the system but of the state before some finite number of transitions.

Diagnosability is studied as an event detection problem in [8]. Diagnosability is a stronger notion than the observability if it is thought as a problem of state identification because of the necessity of identifying every failure state uniquely in contrast to [28]. A system is observable/observable with delay as long as there exists at least

one state which is uniquely identifiable at intermittent points in time whereas diagnosability only requires the identification of failure states with a finite delay. It only requires this identification for failure states because there is no similar need for the normal states. Therefore, a system can execute arbitrarily long sequences of events, in normal failure-less operation, with no single state being uniquely detectable even with some delay. Moreover, a system failing to be observable/observable with delay in the post-failure operation since there isn't any state that can be uniquely identifiable can still be diagnosable. The main reason behind this fact is our not requiring unique identification not of every failure state but only of every set of partition.

The state estimation problem for a partially observed automaton is also studied in Caine [25]. An input state-output automaton model with partial state information available through an output function is used for the system. Initial state and current state observability using two different type of observers is addressed in the study, classical dynamic observers and logic based dynamical observers. Classical dynamic observer is a finite state automaton which takes the observable system behavior, the sequence of input-output pairs as its input and generates a sequence of state estimates either of the initial state or the current state of the system. On the other hand, the logic based observers are kind of observers for which a logic based dynamical system is built in the framework of predicate calculus. This kind of observer generates a sequence of logic propositions describing the properties of the system. Their adaptability to changes in the system model is a very useful feature of these kind of observers. In [25], the author also assumes all future states in time is known, once the current state of the system is determined.

Invertibility is introduced in [29] by Özveren and Willsky as another notion which is closely related with the notion of diagnosability. An invertible language is defined as a language for which we can reconstruct full event sequence up to a finite bounded number of events by using the knowledge of tracked observable event sequence till that time. Indeed, invertibility is a stronger notion than diagnosability because we don't need reconstruction of entire event sequences for a system to be diagnosable and we are interested in identifying the occurrence of specific failure events only. Moreover, if the failure events are partitioned into sets, we are only interested in identifying the occurrence of one of a set of events. Additionally, if there is a case

of multiple failures from the same set of the failure partition, diagnosability does not require detection of every single occurrence of these failures and finds enough to be able to conclude that a failure event from the set has occurred at least once in the system. Thus, we can say that a diagnosable is system can be non-invertible.

This concludes the discussion and comparison of diagnosability notion with other related works that have appeared in the literature.

CHAPTER 3

MODIFIED EVENT DIAGNOSER

In this section we introduce a finite state machine that is built for checking if the given system has the properties mentioned formerly in Section 2.4.1. The machine is built from the system model G and is named as diagnoser since it simply performs the diagnosing task by observing the on-line behavior of G. This diagnoser machine realizes the test of event-diagnosability and it is the basis for this thesis work.

3.1 Basic Event-Diagnoser

We first present the basic diagnoser proposed by [8] and describe its construction. Afterwards, we will explain what kind of improvements are necessary for a better realization. This discussion leads to the construction of an improved diagnoser.

3.1.1 Construction of Basic Event-Diagnoser

First of all, we recall that all system observations are made through the natural projection p_o, whereby all unobservable events in Σ_{uo} cannot be seen. In particular, all fault events in Σ_f are unobservable.

Now we can start by defining a set of failure labels $\Delta_f = \{F_1, F_2, ..., F_m\}$ where $|\Pi_i| = m$. Next, we introduce the set of labels as follows.

$$\Delta = \{N\} \cup 2^{\{\Delta_f \cup \{A\}\}} \tag{3.1}$$

In this notation, N is a label that must be interpreted as having the meaning of *normal* while A must be interpreted as *ambiguous*. These labels will be used for defining the attributes of the states of diagnoser and they will be explained in more detail later on in this chapter. On the other hand, F_i will have a meaning that a failure from set of failure type F_i may occurred until the system reaches to that state where $i \in 1, ..., m$ Recalling the definition of X_o from Equation 2.12, we describe all the possible attributes of diagnoser states that may exist in a diagnoser as

$$Q_o = 2^{X_o \times \Delta} \tag{3.2}$$

The diagnoser for a given system now can be defined as the finite state machine

$$G_d = (Q_d, \Sigma_o, \delta_d, q_0) \tag{3.3}$$

Hereby, marked states need not be considered and all elements of the four-tuple have the usual interpretation. The initial state of our diagnoser is assumed to be normal (non-faulty) as

$$q_0 = \{(x_0, \{N\})\} \tag{3.4}$$

The state space of the diagnoser G_d is the resulting subset of Q_o, i.e. defined in Equation 3.2 composed only of the reachable states of the diagnoser from its initial state q_0 under transition function δ_d. δ_d construction will be explained after we mention about some preliminary functions necessary for the construction. A state q_d of G_d is of the form

$$q_d = \{(x_1, l_1), ..., (x_n, l_n)\} \tag{3.5}$$

since Q_d is a subset of Q_o where $x_i \in X_o$ and $l_i \in \Delta$, i.e. $l_i \subseteq \{\{N\}, \{A\}, \{F_{i_1}, F_{i_2}, ..., F_{i_m}\}$ and $\bigcup_j^k i_j \subseteq 1, 2, ..., m$.

As we mentioned formerly in Section 2.5, Özveren and Willsky proposed an observer for G in [28] for giving the estimation of the current state of the system after occur-

rence of each observable event generated by the system. Meanwhile, the diagnoser G_d can be thought as an extended observer in which a label of the form Δ is appended to state estimation. The labels attached to the state estimates carry failure occurrence information and failures are diagnosed by checking these labels.

We need to define the following three functions before giving the definition of the transition function δ_d of the diagnoser, i.e. *label propagation function* LP, *the range function* R, *label correction function* LC.

Definition 10 (Label Propagation Function) *:* Given that $x \in X_o, l \in \Delta, s \in L_o(G, x)$, label propagation function is $LP : X_o \times \Delta \times \Sigma^* \rightarrow \Delta$ defined as

$$LP\{(x, l, s)\} := \begin{cases} \{N\} & if\ l = \{N\} \cup \forall i[\Sigma_{fi} \notin s] \\ \{A\} & if\ l = \{A\} \cup \forall i[\Sigma_{fi} \notin s] \\ \{F_i : F_i \in l \vee \Sigma_{fi} \in s\} & otherwise \end{cases} \tag{3.6}$$

Definition 11 (Range Function) *:* The range function $R : Q_o \times \Sigma_o \rightarrow Q_o$ is defined as

$$R(q, \sigma) := \bigcup_{x,l \in q} \bigcup_{s \in L_\sigma(G,x)} \{(\delta(x, s), LP(x, l, s)\} \tag{3.7}$$

Definition 12 (Label Correction Function) *: The label correction function LC* : $Q_o \rightarrow Q_o$ *is defined as*

$$\begin{aligned} LC(q) \quad := \quad & \{(x, l) \in q \mid \ x \text{ appears only once in all the pairs in } q\} \cup \{(x, A \cup l_{i1} \cap l_{i2} \\ & \cap ... \cap l_{ik}) \text{ whenever } \exists \text{ two or more pairs } (x, l_{i1}), ..., l_{ik} \text{ in } q\} \end{aligned} \tag{3.8}$$

The label existing in any state x of the diagnoser along a trace s indicates if a failure of that type occurred or not when the system moves along trace s and generates transitions into state x. We can understand that the state x could have resulted from a failure event of a particular type, let's say F_i, or not if there exists two pairs such as $(x, l), (x, l')$ in return of function $R(q, \sigma)$ for some state q of our diagnoser. This condition requires us to attach label to x to indicate that there is an ambiguity for that state x. Therefore, the label A for x means that "*either* F_i *or not* F_i" happened. Please note

that we don't distinguish between cases "F_i *or* F_j", "F_j *or* F_k" or "N *or* F_i". We will use the label A instead in all these situations. It may cause an information loss, but label A will still be enough for diagnosing task, hence determination of diagnosability of the system.

Now we can define the transition function $\delta_d : Q_o \times \Sigma_o \rightarrow Q_o$ as

$$q_2 = \delta_d(q_1, \sigma) \Leftrightarrow q_2 = LC[R(q_1, \sigma)] \tag{3.9}$$

with $\sigma \in e_d(q_1)$ where $e_d(q_1)$ is defined as

$$e_d(q_1) = \bigcup_{(x,l) \in q_1} \{P(s) \mid s \in L_o(G, x)\} \tag{3.10}$$

In the equation, $e_d(q_1)$ denotes the set of all possible transitions of the diagnoser existing at the state q_1 that is set of active events in system G_d at this state. We can summarize the construction of the formerly mentioned G_d in three steps,where q_1 defines the current state of our diagnoser G which carries the state estimation information with corresponding labels, σ defines the next observed event in the system, q_2, is the next state into which the diagnoser will make transition is computed, as follows:

1) Obtain the reachable states of x for generated event σ by $S(x,\sigma) = \{\delta(x, s\sigma)$ where $s \in \Sigma_{uo}^*\}$ for each of the state estimate x in a state q_1 of the diagnoser,

2) Propagate label l corresponding to x to label l' with x' where $x' \in \delta(x,\sigma) = x'$ according to the given rules.

   i . the label l' is $\{N\}$ if $l = \{N\}$ and s contains no failure events
   ii . the label l' is $\{A\}$ if $l = \{A\}$ and s contains no failure events
   iii . the label l' is $\{F_i\}$ if $l = \{A, F_i\}$ and s contains no failure events
   iv . the label l' is $\{F_i, F_j\}$ if $l = \{A\}$ or $\{F_j\}$ and s contains failure events from the set Σ_{fi}, Σ_{fj}
   v . the label l' is $\{F_i, F_j, F_k\}$ if $l = \{F_i, F_j\}$ or $\{A, F_i, F_j\}$ and s contains failure events from the set Σ_{fk}

3) Replace all $(x', l'), (x', l'') \in q_2$ as F_i and F_j components of both l' and l'' by (x', A, F_i, F_j) where q_2 is the set of all (x', l') pairs computed according to step rules 1 and 2 for each $(x, l) \in q_2$. We associate all common components of the labels of estimate x' with x' itself if the same state estimate x' appears in q_2 with different labels. Additionally, we attach label A to x' in such a case with ambiguity.

It is important to note that the label A indicating the ambiguity is not propagated to the next state in case iii, iv and v. It does not cause a loss of information that will be necessary for determining the diagnosability attribute beside it does lead to a reduction in the state space of our diagnoser. Finally, we note that we assume the initial state of the system is known since the diagnoser conducts an on-line diagnosis task by running in parallel with the system from the start.

In order to illustrate the diagnoser construction, we provide the simple example in Figure 3.1. The system is stated as G whereas its corresponding diagnoser constructed according to the rules described formerly is stated as G_d and shown in Figure 3.2. The observable event set for the system in the example is denoted by greek letters, i.e. $\Sigma_o = \{\alpha, \beta, \gamma, \delta, \sigma\}$ and the unobservable event set Σ_{uo} is composed of elements

$\sigma_{uo}, \sigma_{f1}, \sigma_{f2}$ and σ_{f3} where $\Sigma = \Sigma_o \cup \Sigma_{uo}$. The fault events are σ_{f1}, σ_{f2} and σ_{f3}. Failure set is chosen to be partitioned into two sets such as $\Sigma_{f1} = \{\sigma_{f1}\}$ and $\Sigma_{f2} = \{\sigma_{f2}, \sigma_{f3}\}$.

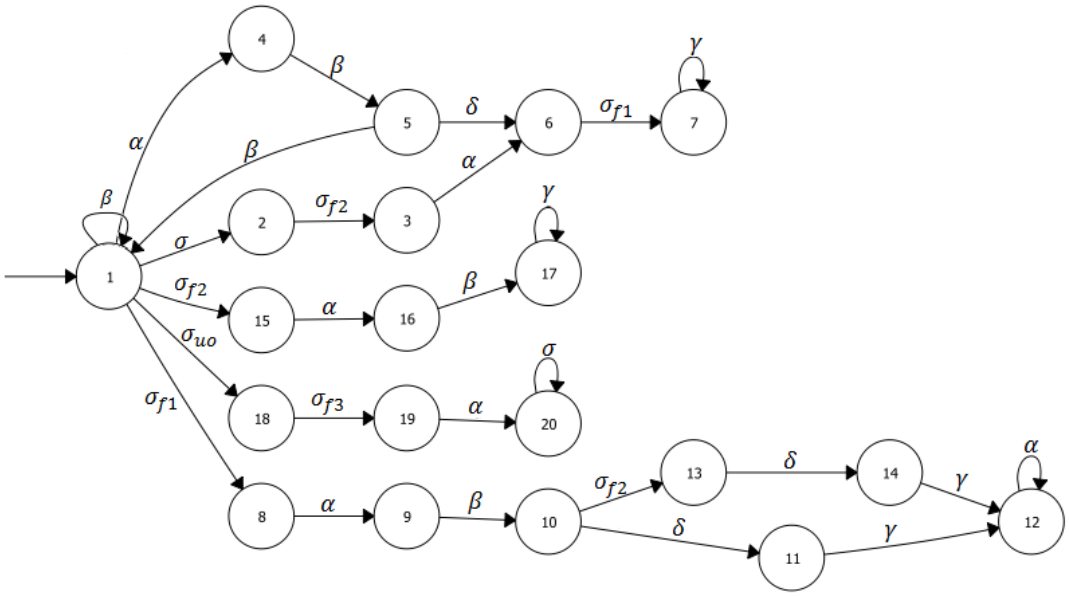


Figure 3.1: Example system model for illustrating event-diagnosability

3.1.2 Diagnosability Verification Using the Diagnoser

In order to investigate diagnosability based on the diagnoser G_d, we state several properties of the diagnoser that can be derived from its construction rules.

- Any $x_i \in X_o$ appears at most in one pair (x_i, l_i) of any state of Q_d as stated in the construction
- Let $q \in Q_d$,

  $$(x_1, l_1), (x_2, l_2) \in q \Leftrightarrow \exists s1, s2 \in L$$

  where

  $$s_{1f}, s_{2f} \in \Sigma_o, \delta(x_o, s_1) = x_1, \delta(x_o, s_2) = x_2 \text{ and } P(s_1) = P(s_2)$$

- Let $q_1, q_2 \in Q_d$ and $s \in \Sigma^*$ where $(x_1, l_1) \in q_1, (x_2, l_2) \in q_2, \delta(x_1, s_1) = x_2$ and $\delta_d[q_1, P(s)] = q_2$

  $$(F_i \notin l_2) \wedge (A \notin l_2) \rightarrow F_i - uncertain$$

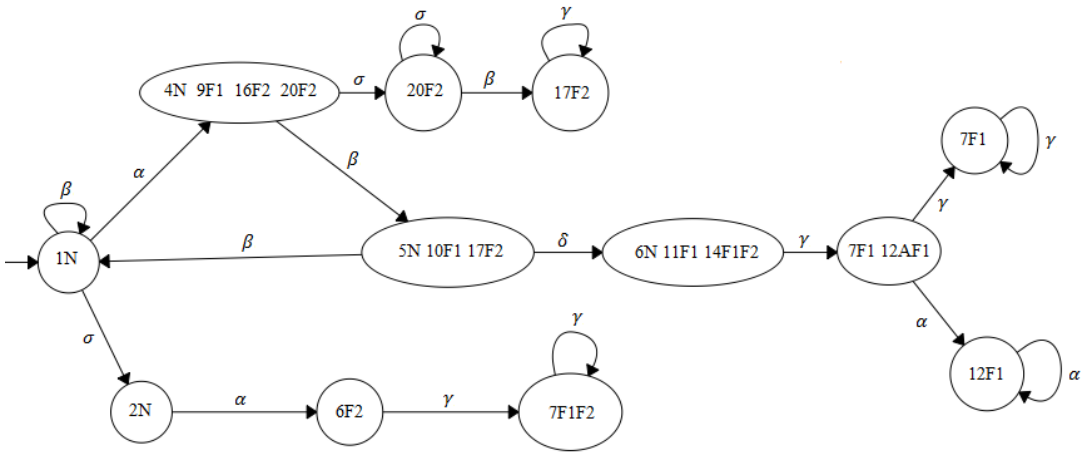


Figure 3.2: The Diagnoser G_d

The last property given indicates that the failure label F_i propagates from state to state if it is not replaced by label A due to the label correction function LC. Therefore, we can conclude that every successor x' carries the failure label F_i as its predecessor unless ambiguity is revealed during the construction transitions. Similarly, if along a trace $s \in L$, a state x carries the normal label N, so must all of its predecessors.

Definition 13 (F_i-certain state) *:* A state $q \in Q_d$ is called as an $F_i - certain$ state if $\forall(x, l) \in q, F_i \in l$.

Definition 14 (F_i-uncertain state) *:* A state $q \in Q_d$ is called as an $F_i - uncertain$ state if $(x, l), (y', l') \in q$ where $F_i \in l$ and $F_i \notin l'$.

Definition 15 (Ambiguous state) *:* A state $q \in Q_d$ is called as an *Ambiguous* state if $\exists(x, l) \in q$ where $A \in l$

It holds that $x \not\equiv y$ for the definition of an F_i-uncertain state. Moreover, if a state q of Q_d is not F_i-uncertain, it does not mean that q is F_i-certain. To be more clear, a state $q \in Q_d$ can be neither F_i-uncertain nor F_i-certain for $\forall(x, l) \in q$ and $F_i \notin l$. We arrive at the following conclusions from the construction method of the diagnoser.

i. If q is F_i-*certain* then $\Sigma_{fi} \in \omega, \forall\omega \in P_L^{-1}(u)$ where $\delta_d(q_0, u) = q$

ii. If q$\in Q_d$ is F_i-*uncertain* then $\forall s_1, s_2 \in L$ such that $\Sigma_{fi} \in s_1, \Sigma_{fi} \notin s_2, P(s_1) = P(s_2), \delta_d(q_0, P(s_1)) = q$ and $\delta(x_0, s_1) \neq \delta(x_0, s_2)$

iii. If q$\in Q_d$ is *ambiguous* then $\exists s_1, s_2 \in L$ and $\exists i \in \Pi_f$ such that $\Sigma_{fi} \in s_1, \Sigma_{fi} \notin s_2, P(s_1) = P(s_2), \delta_d(q_0, P(s_1)) = q$ and $\delta(x_0, s_1) = \delta(x_0, s_2)$

Now, it is obvious that a failure F_i has occurred in the system, if the current state of the diagnoser is F_i-certain regardless of what the current state of the given system G is. As indicated beforehand, this is exactly the type of diagnosis task that Sampath proposes to the reader in [8].

We give a more detailed analysis of the conditions F_i-uncertain and ambiguous. If there exists an F_i-uncertain state in G_d, it means that there are two strings s_1 and s_2 in $L(g)$ such that s_1 contains a failure event of failure type F_i while s_2 does not contain any event from that set besides their having the same record of observable events. Hence, it is usual to conclude that a failure type possibly occurred in the given system but we cannot be certain about this with the observed sequence of events. On the other hand, the presence of an ambiguous state in G_d of the diagnoser corresponds to the case where we have two strings s_1 and s_2 having the same record of observable events and generated by the system such that all possible continuations of these strings in $L(g)$ are the same. Since the system will end up in same state of G after any continuation, we call this type of traces as F_i-ambiguous traces.

Some final definitions including $F_i - indeterminatecycle$ that are needed for using the diagnoser to test diagnosability.

Definition 16 (Cycle of states) *:* A *cycle* is composed of a set of states $x_1, x_2, ..., x_n \in X$ in G if $\exists\ s \in L(G, x_1)$ such that $s = \sigma_1\sigma_2...\sigma_n$ and $\delta(x_l, \sigma_l) = x_{(l+1)modn}, l = 1, 2, ..., n$.

Considering the occurrence of possible cycles in both G_d and G', we will now define the notion of F_i-indeterminate cycle.

Definition 17 (F_i-indeterminate cycle) *:* A set of F_i-uncertain states $q_1, q_2, ..., q_n \in Q_d$ is said to form an F_i-*indeterminate cycle* if

1) States $q_1, q_2, ..., q_n$ forms a cycle in G_d with $\delta_d(q_l, \sigma_l) = q_{l+1}, l = 1, ..., n-1, \delta_d(q_n, \sigma_n) = q_1$ where $\sigma_l \in \Sigma_o, l = 1, ..., n$ and

2) $\exists\, ({x_l}^k, {l_l}^k), ({y_l}^r, \tilde{l}_l^r) \in q_l, l = 1, ..., n, k = 1, ..., m\ r = 1, ..., m'$ such that

   a) $F_i \in {l_k}^k, F_i \notin \tilde{l}_k^k$ for all l, k and r,

   b) The sequences of state ${x_l}^k, l = 1, ..., n,\ k = 1, ..., m$ and ${x_l}^k,\ l = 1, ..., n,\ r = 1, ..., m'$ form cycles in G' with

   $({x_l}^k, \sigma_l, {x_{l+1}}^k) \in \delta'_G, \quad l = 1, ..., n-1, k = 1, ..., m,$

   $({x_n}^k, \sigma_n, {x_1}^{k+1}) \in \delta'_G, \quad k = 1, ..., m-1$

   and

   $({x_n}^m, \sigma_n, {x_1}^1) \in \delta'_G$

   and

   $({y_l}^r, \sigma_l, {y_{l+1}}^r) \in \delta'_G, \quad l = 1, ..., n-1, r = 1, ..., m',$

   $({y_n}^r, \sigma_l, {y_{l+1}}^{r+1}) \in \delta'_G, \quad r = 1, ..., m'-1,$

   and

   $({y_m}^n, \sigma_n, {y_1}^1) \in \delta'_G$.

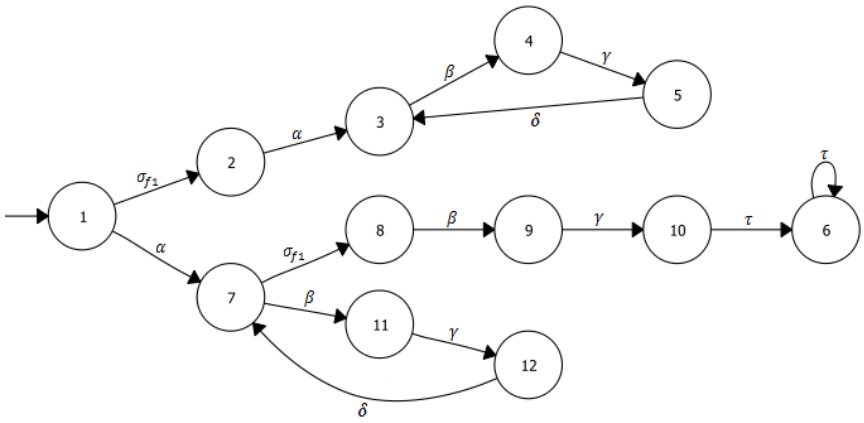
To be more explicit, an F_i-indeterminate cycle in G_d is a cycle composed of F_i-uncertain states including

- A corresponding cycle of observable events in G' which has only states carrying F_i in their labels in G_d (sequence of ${x_l}^k$) and
- A corresponding cycle of observable events in G' involving only states that do not carry failure label F_i (sequence of ${x_l}^k$)

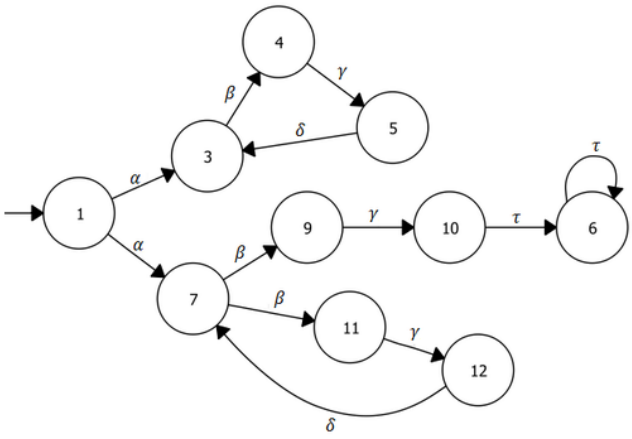
m and m' here are the number of times that the cycle $q_1, q_2, \ldots, q_n$ in G_d is completed before the cycle in G' is completed. Moreover, nm and nm' are the cycle lengths in ${x_l}^k$ and ${x_l}^k$, respectively.

An F_i-indeterminate cycle in G_d indicates the existence of two traces s_1 and s_2 in L with arbitrarily long sequence of events such that the record of observable events for both traces are identical while s_1 contains a failure of type F_i but s_2 does not contain such a failure of type F_i. We now present some examples in order to illustrate the aforementioned notions up to now.

The following figures depict 3 different systems with their corresponding observable state machines, G' and the diagnosers corresponding to these systems. Each system captures a distinct case for describing the given notions.



a) System behaviour G



b) Generator G' for the system G

Figure 3.3: Example of a system that will lead to a diagnoser G_d with an F_i-indeterminate cycle in it.

In Figure 3.3, we have a system G with observable event set $\Sigma_o = \{\alpha, \beta, \delta, \gamma, \tau\}$ and unobservable event set $\Sigma_{uo} = \{\sigma_{f1}\}$ which equals to the defined set of failures in the system G, i.e. Σ_f.

The corresponding diagnoser G_d is depicted in Figure 3.4. G_d has a cycle of F_1-uncertain states for the trace including "$\beta\gamma\delta$" after a single α is generated in the system. It next needs to be checked if it is an F_1-indeterminate cycle according to

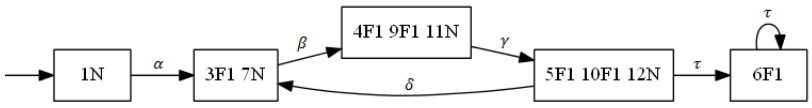


Figure 3.4: Corresponding diagnoser G_d.

the previous definitions. The first cycle in G' is composed of states $\{7, 11, 12\}$ and these states appear with label N in the states of diagnoser G_d. The other cycle in G' is composed of states $\{3, 4, 5\}$ and these states appear with an F_1 label in the corresponding states of G_d. Therefore, we can easily say that this pertains to the case that we described in notion of F_i-indeterminate cycle with $x_1^1 = 3$, $x_2^1 = 4$, $x_3^1 = 5$ and $y_1^1 = 7$, $y_2^1 = 11$, $y_3^1 = 12$ and $m = m' = 1$. Hence, the system is not diagnosable.

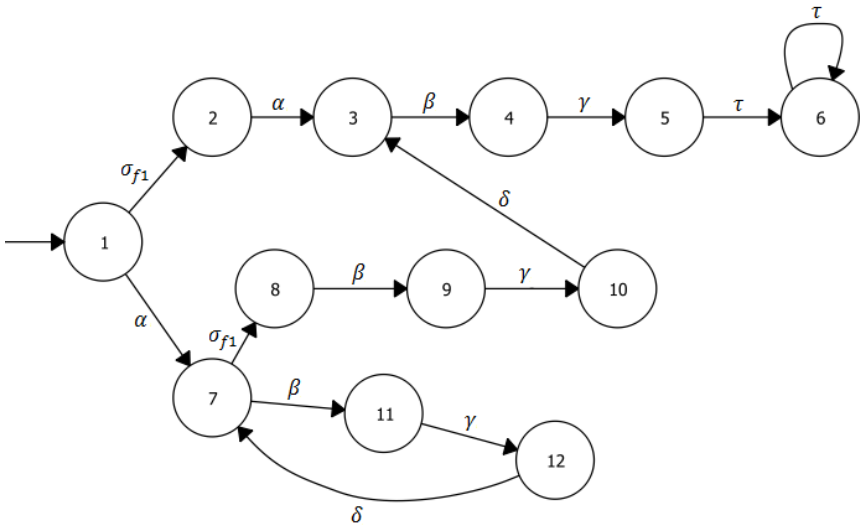


Figure 3.5: Example of a system having a diagnoser G_d with a cycle of F_i-uncertain states but don't have an F_i-indeterminate in it.

There is an another system is defined with the observable event set $\Sigma_o = \{\alpha, \beta, \delta, \gamma, \tau\}$ and the unobservable event set $\Sigma_{uo} = \{\sigma_{f1}\}$ as depicted with its generator G' and corresponding diagnoser G_d in Figure 3.5.

The corresponding diagnoser G_d can be seen in Figure 3.6.

The system G also has a cycle of F_1-uncertain states. As the previous example the cycle in G_d corresponds directly to a cycle in G' as expected. The loop of G' is

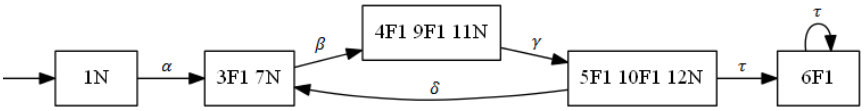


Figure 3.6: Corresponding diagnoser G_d.

completed with only one completion of the loop in G_d, i.e. $m = m' = 1$. This cycle corresponds to a cycle only composed of states with F_1 label. Therefore, the cycle in the diagnoser in G_d is not an F_1-indeterminate since it only includes a corresponding cycle in G_d with F_1 label and not with N labels even though the diagnoser of our last two examples are completely same. Hence, this system is diagnosable.

In our final example in Figure 3.7, we now have a system with an observable event set $\Sigma_o = \{\alpha, \beta, \gamma\}$ and unobservable event set $\Sigma_{uo} = \{\sigma_{uo}, \sigma_{f1}\}$ where failure events are $\Sigma_f = \{\Sigma_{f1}\}$. Again we have a cycle of F_1-uncertain states in its. It corresponds to two cycles in generator G'.

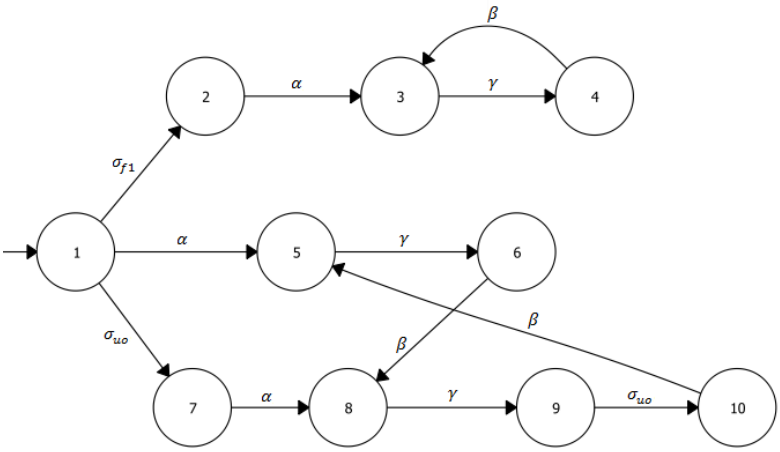


Figure 3.7: Another example of a system G with an F_i-indeterminate cycle in its corresponding G_d.

One of these cycles is composed of F_1 labeled states, i.e. the set x_l^k is $\{3, 4\}$ and the other cycle is composed of N labeled states, i.e. the set y_l^r is $\{5, 6, 8, 9\}$ or $\{8, 9, 5, 6\}$. It must also be noticed that the cycle after the failure occurred needs to be completed twice whilst the cycle that belongs to the non-faulty trace is completed only once. Therefore $m = 1$ and $m' = 2$.

After illustrating the notion of F_1-indeterminate cycle with three explanatory examples, we end this section by noting the necessary conditions stated in [8] for a diagnoser G_d to detect every possible failure.

Assuming that there is no multiple failure occurrence of same type during the system process, a language L that defines this system is *event-diagnosable* if and only if its diagnoser G_d satisfies the following two conditions

1) No indeterminate cycles exists in diagnoser G_d for all failure types of F_i

2) No ambiguous state q exists in Q_d of G_d assuming that there cannot happen multiple failures of the same type.

3.2 Motivation for Modified Event Diagnoser

In this section we will show that though the diagnoser G_d constructed in the previous section is enough for deciding whether its corresponding system G is diagnosable or not, it is not straightforward to obtain a diagnosis decision. Hence, we propose a more suitable way for online diagnosis.

For illustration, we consider a system very similar to the example system given in Figure 3.1 which is modeled by a prefix-closed, live language L, depicted in Figure 3.9. The failure events are $\Sigma_f = \Sigma_{f1} \cup \Sigma_{f2}$ is where $\Sigma_{f1} = \{\sigma_{fi}\}, \Sigma_{f2} = \{\sigma_{f2}, \sigma_{f3}\}$.

The constructed diagnoser for the system given in Figure 3.9 is shown in Figure 3.10. The diagnoser G_d in this figure is a good illustrative example since it is composed of all the possibilities of states allowed for a system to be diagnosable and defined in Section 3.1.1, i.e. [2N - Normal], [6N 11F1 14F1F2 F_i-uncertain], [12F1 F_1-certain]. By

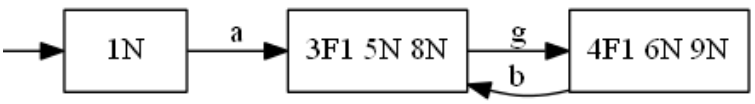


Figure 3.8: Diagnoser G_d for G.

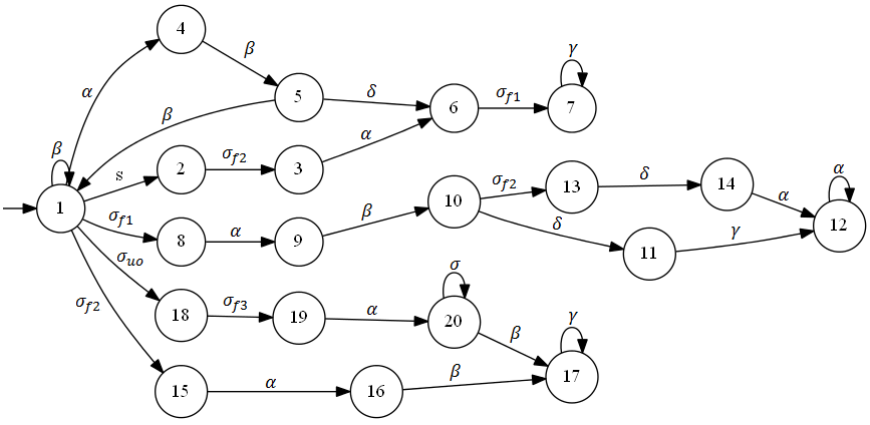


Figure 3.9: Motivating Diagnosable System Model G

such a diagnoser, we can understand that a failure event of type F_i occurred whenever we hit a F_i-certain state via tracking the observable events generated by the system. Hereby, we can conclude that the basic diagnoser proposed by [8] and constructed by the rules defined formerly is successfully performing its diagnosis task.

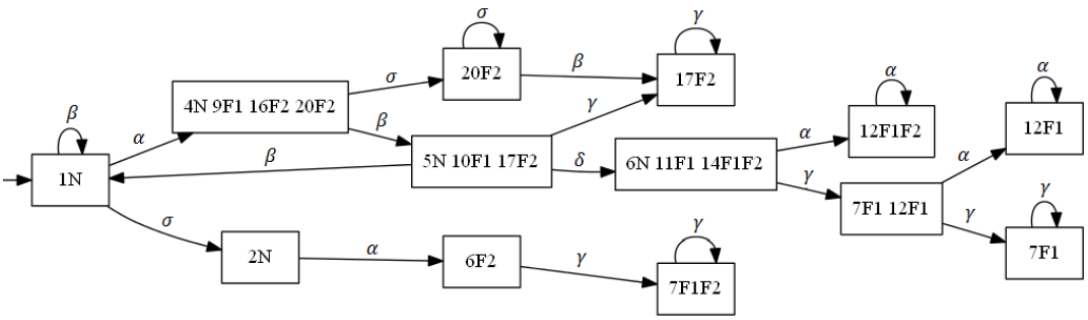


Figure 3.10: Basic Diagnoser G_{bd} for motivating system model G

Nevertheless, the suggested diagnoser model does not have states with observable state attributes and labels. Therefore, we need to track all the observable events generated by the system to determine the current state of the diagnoser from the start state of the given system. Moreover, we cannot determine that the current state of the diagnoser is an F_i-certain state, hence recognize that a failure event of type F_i is occurred although we track all the observable events and determine the current state of the diagnoser. What we need is analyzing the label of the current diagnoser state revealing its attribute and doing some other computations and deciding if that state is F_i-certain or not and this brings too much necessary calculations after each observed

transition.

One of the improvements that we propose in this thesis work is a more easily usable diagnoser which highlights the arrival at an F_i-certain state by triggering a fault-detection event. Such an event diagnoser for the same example system is shown in Figure 3.11.

The modified diagnoser helps the observer by inserting an observable fault-detection event transitions into each F_i-certain state in which we can be sure about the occurrence of that failure type of unobservable failure for the first time along that trace. These fault-detection events are marked by "*FAILURE-1*" and "*FAILURE-2*" and correspond to unobservable failure event partitions Σ_{f1} and Σ_{f2}. Beside these, the first F_i-certain states along a trace are now marked states for illustration.

We can list the advantages of these fault-detection events as:

- There is no need to track the information about in which state the diagnoser is now and hence, the observer does not require tracking all the observable events generated by the system. Merely noticing the corresponding fault-detection event becomes enough to identify the occurrence of that failure.
- Even though the current state of the diagnoser is determined without tracking the generated observable events, an observer still needs to perform an analysis of each detected state to understand whether it is an Fi-certain state or not. There is no such need for an observer having an modified diagnoser.
- Since fault-detection events are generated whenever the first occurrence of an event of corresponding failure type is certainly noticed, a reaction to the fault is immediately possible.

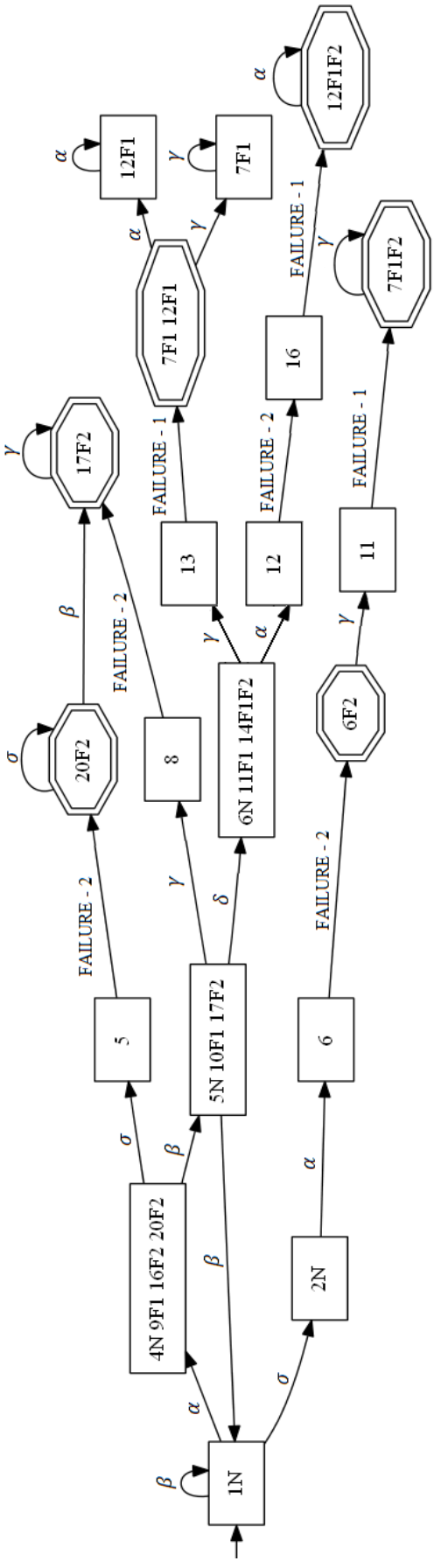


Figure 3.11: Modified Diagnoser for motivating system model G

3.3 Modified Event Diagnoser with Fault-Detection Events

In this section, we describe how the modified diagnoser is constructed by building on the former construction method of the basic diagnoser given in Section 3.1.1.

We use the deterministic finite state machine

$$G_{md} = (Q_{md}, \Sigma_{md}, \delta_{md}, q_{0md}, X_{md}) \tag{3.11}$$

We can describe the elements of the given five-tuple definition for the modified diagnoser by re-using the element descriptions of the basic diagnoser.

$$\begin{aligned} Q_{md} &= Q_{bd} + \bigcup Q_{transient\_states} \\ \Sigma_{md} &= \Sigma_{bd} + \bigcup(FAILURE - \Pi_i) \\ q_{0_{md}} &= q_{0_{bd}} \\ X_{md} &= \bigcup F_i\text{-certain states} \end{aligned} \tag{3.12}$$

where the terms having the subscript *md* are elements of the modified diagnoser and terms having *bd* subscript are elements of the basic diagnoser.

$Q_{transient\_states}$ is introduced for the new set of unlabeled states from where the modified diagnoser generates observable fault-detection events. F_i-certain states which occur for the first time along a trace corresponding to that failure event type are denoted as marked states for better illustration.

The only remaining element of the five-tuple representation which has not been described is the transition function δ_{md} of the modified diagnoser. This function is identical to δ_{bd} except for transitions that lead to an F_i-certain state for the first time along a generated trace. If δ_{bd} does so, then the transition function of the modified diagnoser δ_{md} enters to a transient state $q_{transient\_state} \in Q_{transient\_state}$ and then generates the corresponding observable fault-detection event with a transition that leads to the state reached by δ_{bd}. The explained procedure is summarized in the following pseudo code with the inputs rOrigGen (system model G), rAttrFTMap (failure type map Π), rDiagGen (modified diagnoser result G_{md}).

Algorithm 1 Pseudo Code for Modified Diagnoser

EVENTDIAGNOSER (*rOrigGen*, *rAttrFTMap*, *rDiagGen*)
Begin
if (!IsLive(G) or CycleOfUnobsEvents(G)) **then** // *Check assumptions*
  return;
end if
Obtain set of all failure partitions
possible_laterfailureTypes[q_0] $\longleftarrow$ Π_i //set of all failure partitions
while !*newDiagStates*.Empty() **do**
  currDiagState $\longleftarrow$ the next state of G_{md}
  for each $F_i \in$ possible_laterfailureTypes[*currDiagState*] **do**
    if is_fcertain(*currDiagState*) **then** // *do_fcertain(*currDiagState*)*
      Add fault-detection event *FAILURE*-Π_i to Σ
      Insert a *newstate* to Q_d
      Copy incoming transitions and l of *currDiagState* to *newstate* and delete
      Add transition via *FAILURE*-Π_i from *currDiagState* to *newstate* to δ_{md}
      Remove *FAILURE*-Π_i from possible_laterfailureTypes[*newstate*]
      currDiagState $\longleftarrow$ *newstate*
    end if
  end for
  for Each $x \in q_d$ where $q_d = \{(x_1, l_1), ..., (x_n, l_n)\}$ and q_d = *currDiagState* **do**
    Obtain set of reachable states $\forall\sigma \in \Gamma(x)$
  end for
  for Each $\sigma \in \Sigma$ corresponds to a *state* − *label* pair (q, l) **do**
    for Each element of label of the pair **do**
      Run *Label Propagation* on each element
    end for
    if New state-label pair (q, l) is not in the candidate pair set **then**
      Insert the new pair into the set
    end if
    Insert new *event* − (*state* − *labelpair*)*pair* for that event
  end for
  for Each event of *event* − (*state* − *labelpair*)*pair*s **do**
    Run *Label Correction* function on each state-label pair
    if New label l belongs to any existing q_d of G_{md} **then**
      nextstate $\longleftarrow$ q_d
    else if There exists no state with the new label l **then**
      Create a new $q \in Q_{md}$ and with l and *nextstate* $\longleftarrow$ q
    end if
    if Corresponding event $\sigma \notin \Sigma_{md}$ has not caused to a transition yet **then**
      Add σ to Σ_{md}
    end if
    Add the transition $\delta(currDiagState, \sigma)$ = *nextstate* to δ
  end for
  Erase *currDiagState* from *newDiagStates*
end while
End

To conclude, the most important advantage of the modified diagnoser is the detection of the first occurrence of any possible unobservable failure event and highlighting the detection moment by an observable fault-detection event. What the modified diagnoser needs for performing the detection of the first occurrence of an F_i-certain state along a generated trace can be summarized in three steps: 1. determining the arrival to an F_i-certain state; 2. noticing that this arrival occurs for the first time along the trace generated up to now since the start at the initial state; 3. if both of the previous two conditions are fulfilled, generating an observable corresponding fault-detection event Failure-Π_i in the modified diagnoser.

3.4 More Event Diagnoser Examples

In Figure 3.12, we see a system model G_1 including 12 states in it. Observable events of this system are denoted by $\Sigma_o = \{\alpha, \beta, \delta, \sigma, \gamma\}$ while unobservable event set Σ_{uo} equals to the set $\Sigma_f = \sigma_{f1}$. Please note here that failure event set has only one partition and hence we have only one type of failure, i.e. f_1 in this system. The corresponding basic diagnoser and modified diagnoser for this system are given in Figure 3.13 and Figure 3.14 respectively.

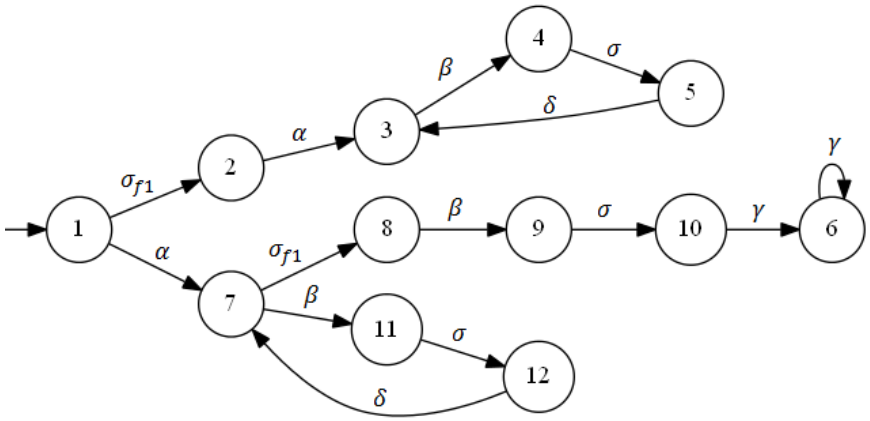


Figure 3.12: Example System G_1

For the basic diagnoser case, even though it seems to be enough for detection of a first time occurrence of any failure type, an observer still needs to track all observable events, make necessary analysis and calculations for each state.

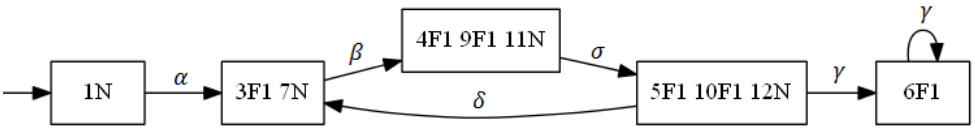


Figure 3.13: Basic diagnoser G_{md} for the system G_1

On the other hand, now with the help of the modified diagnoser, it is trivial to recognize where we are sure about a first time occurrence of any failure type along a generated trace. A generation of the observable fault-detection event *FAILURE-1* by the diagnoser machine is enough for this purpose.

Please note that though a modified diagnoser is obtained and injection of *FAILURE-1* event to the alphabet informs us about the first occurrence of Σ_{f1} type failure, this system is not diagnosable since it includes an F_i-indeterminate cycle in the diagnoser. In that sense, the modified diagnoser will indicate anyway the faults that can be diagnosed.

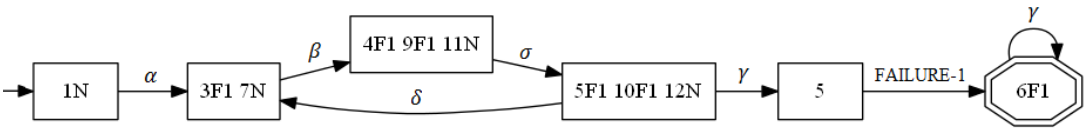


Figure 3.14: Modified diagnoser G_{md} for the system G_1

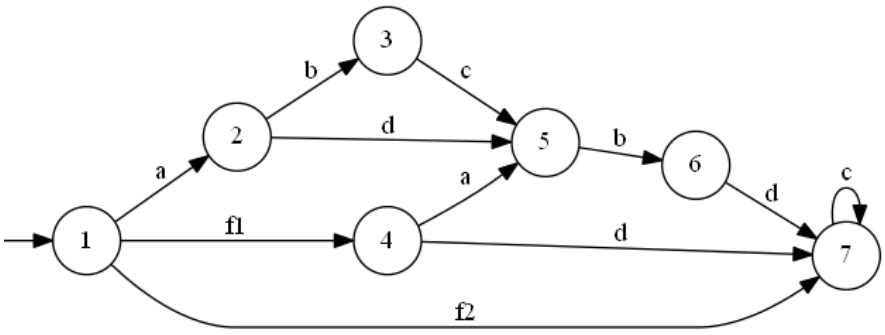


Figure 3.15: Another example system G_2

We have another system G_2 with failure event set $\Sigma_f = \Sigma_{uo} = \{f1, f2\}$ while observable

event set is defined as $\Sigma_o = \{a, b, c, d\}$. On the other hand, failure partition for this system is divided into two groups which are $\Sigma_{f1} = \{f1\}$ and $\Sigma_{f2} = \{f2\}$.

Corresponding basic diagnoser and modified diagnoser which are constructed from the given system G_2 according to formerly described algorithms are given Figure 3.16 and Figure 3.17 respectively.

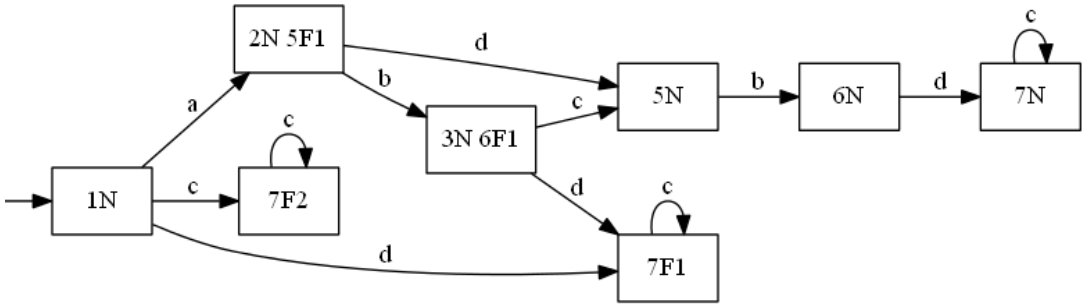


Figure 3.16: Basic diagnoser G_{bd} for the system G_2

Again in the basic diagnoser that is proposed in [8], there are F_i-certain states indicating that if the running system generates long enough traces, an observer will end up his trace in that F_i-certain state and understand that a failure event of corresponding failure type occurred certainly. Nonetheless, the observer needs computations for this result.

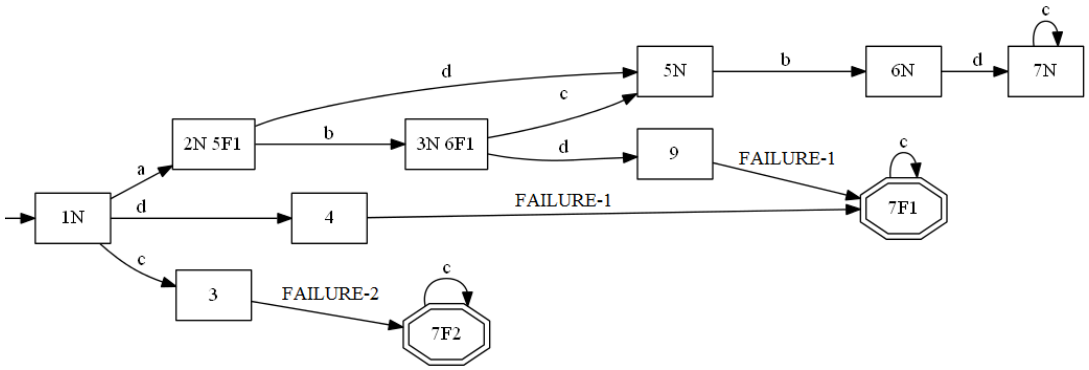


Figure 3.17: Modified diagnoser G_{md} for the system G_2

If we look at our modified diagnoser in Figure 3.17, we can see the desired fault-detection events "*FAILURE-1*" and "*FAILURE-2*" corresponding to unobservable failure events $f1$ and $f2$ are inserted in it. Therefore, an observer can immediately and

easily detect the state in which he is sure for the first time about the occurrence of any failure without requiring anything else by just observing these failure events.

CHAPTER 4

MODIFIED LANGUAGE DIAGNOSER

In the previous chapter, we introduced the modified event diagnoser. This diagnoser is able to detect and identify the occurrence of fault events by observing the observable system events whenever the system is diagnosable. Though this seems to be an acceptable method because unwanted failure events can be introduced when modeling the system, using failure events does not cover the case of unwanted sequences of observable/unobservable events. For instance, we may not want the event sequence "$\alpha\beta\gamma$" instead of "$\alpha\beta\sigma\gamma$" to be generated which means that the system skipped one necessary operation denoted by σ during its execution. Hereby, we can understand the relevance of language-diagnosability that generalizes event-diagnosability.

We now proceed similar to the previous chapter in order to determine a modified language-diagnoser. We first show how a language-diagnoser can be built and how it is used for verifying diagnosability. Finally, we present our constructon proccedure fot the modified language-diagnoser.

4.1 Basic Language Diagnoser

We consider language diagnosis as introduced in Section 2.4.3. That is, the system is composed of two behaviors which are called *normal system behavior* given by the specification $K \subseteq \Sigma^\star$ and the *possible system behavior* given by the plant automaton $G = (X, \Sigma, \delta, x_0)$.

Using language-diagnoss, each fault is represented by a specification langauge K.

That is, different from the case of event-diagnosis, always a single fault is considered per specification. If it is necessary to consider different faults, it is only required to formulate a separate specification for each fault. Since it is sufficient to consider a single specification, it is enough to mark all unspecified behavior with label F (faulty). Then, the set of all language-diagnoser state labels is given by

$$\Delta = \{N\} \cup \{F\} \tag{4.1}$$

Analogous to the construction of the event diagnoser, label N has the meaning of a normal operation while label A means that the state owning it is ambiguous. Furthermore label F indicates that the system may have generated a fault until reaching that state. By using this information and the previous definition of X_o, we describe all the possible diagnoser state attributes with Equation 4.2.

$$Q_o = 2^{X_o \times \Delta} \tag{4.2}$$

We again define the language-diagnoser as a four-tuple

$$G_d = (Q_d, \Sigma_o, \delta_d, q_0) \tag{4.3}$$

where Q_d is a subset of Q_o. The elements of this language diagnoser have the usual meaning of elements of a finite state machine.

The initial state of the language diagnoser is assumed to be normal at the beginning.

$$q_0 = \{(x_0, \{N\})\} \tag{4.4}$$

Moreover, any state q_d of the basic language diagnoser is in the form of

$$q_d = \{(x_1, l_1), ..., (x_n, l_n)\} \tag{4.5}$$

where

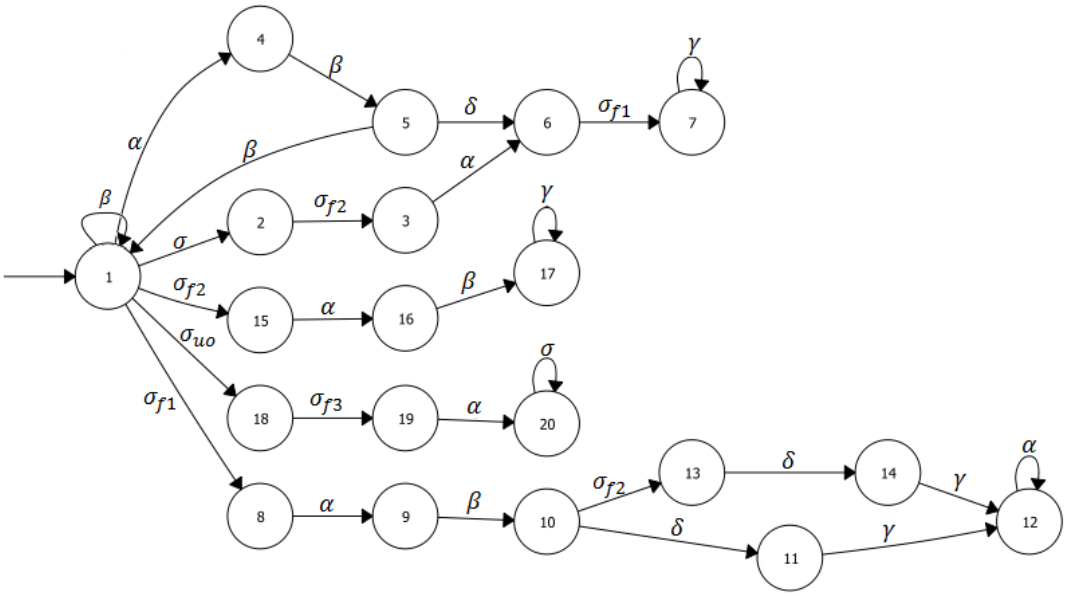


Figure 4.1: Example system model G for illustrating language-diagnosability

$$x_i \in X_o \; l_i \in \Delta, \text{ i.e. } l_i \subseteq \{\{N\}, \{F\}\}$$

since Q_d is a subset of Q_o.

We have completed defining the preliminaries for basic language diagnoser which show minor modifications compared to corresponding basic event diagnoser.

To help the reader understand the construction algorithm and steps of basic language diagnoser, we will illustrate this construction process by using the same example system that we used for illustrating the event diagnoser construction process. The system is modeled as in Figure 4.1.

The system G in Figure 4.1 is modeled exactly in the same way as in event diagnoser case. We have again the same observable event set $\Sigma_0 = \{\alpha, \beta, \gamma, \delta, \sigma\}$ and unobservable event set $\Sigma_{uo} = \{\sigma_{uo}, \sigma_{f1}, \sigma_{f2}\}$ where $\Sigma = \Sigma_o \cup \Sigma_{uo}$. Only difference between these models is the failure partition, as expected. We do not have any failure partitions in language diagnoser model since we will denote all abnormal, undesired events by label F, i.e. $\Sigma_f = \{\sigma_{f1}, \sigma_{f2}\}$.

Specification language model which includes only both observable and unobservable normal events and defines the *normal* behavior of G can be seen in Figure 4.2.

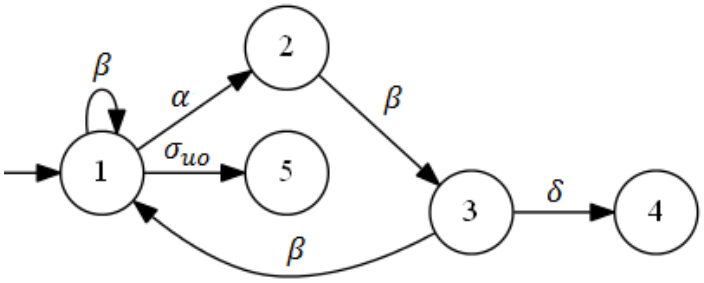


Figure 4.2: Language specification G_N for system model G

As stated earlier, we need to think cooperation of this language specification with all the possible behavior of the given system and obtain $G = G_P \parallel \overline{G_N}$ before starting to the construction of our basic language diagnoser.

To obtain such a language model, we can itemize the necessary steps which will be given in more detail later on as algorithm in pseudo code form.

i. Mark all the states of possible behavior language, i.e. whole system

ii. Mark all the states of specification language

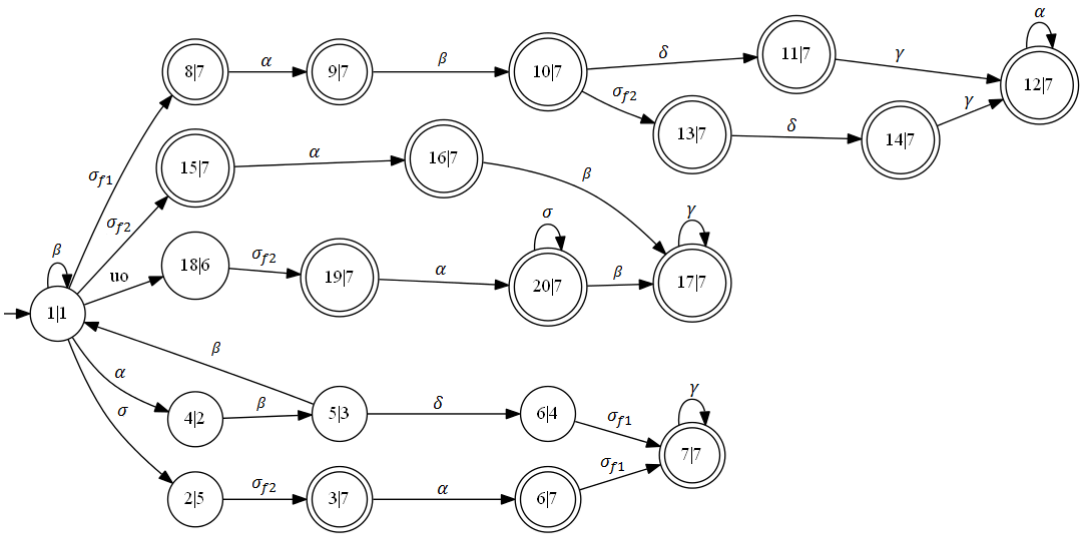
iii. Complement the marked specification language

iv. Obtain the parallel composition of possible and complement of spec automata

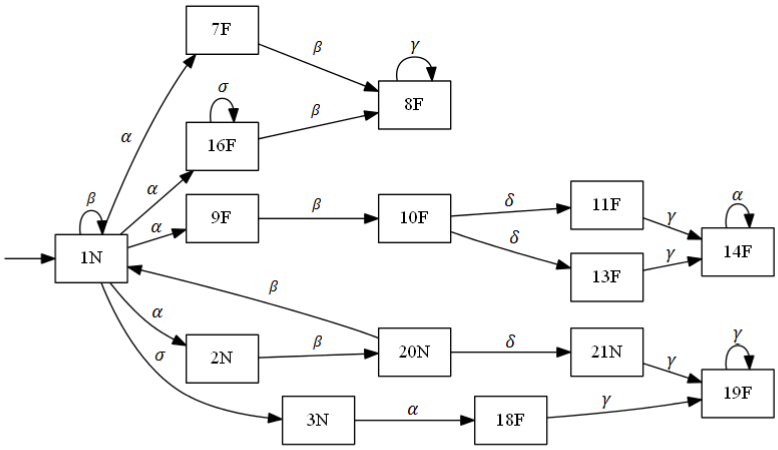
v. Remove the unobservable event transitions from the composed machine to obtain the non-deterministic basis for diagnoser

vi. Obtain the diagnoser by shrinking the non-deterministic machine to a deterministic one

The automata that are obtained at the intermediate steps of the construction process is illustrated in Figure 4.3. First automaton in this figure is obtained by applying parallel composition on marked versions of possible behavior and complement of normal behavior of the machine given in Figure 4.1. The second automaton is nothing but the same automaton that is cleared off the unobservable event transitions as described in step v. The reader can understand the process more clearly by analyzing the intermediate automata.



a) Resultant automata obtained at the end of step IV.



b) Non-deterministic basis for basic language diagnoser after step V.

Figure 4.3: Intermediate-step automata for basic language diagnoser construction

Finally, the basic diagnoser which is being completely constructed after applying six formerly described steps in order is given in Figure 4.4.

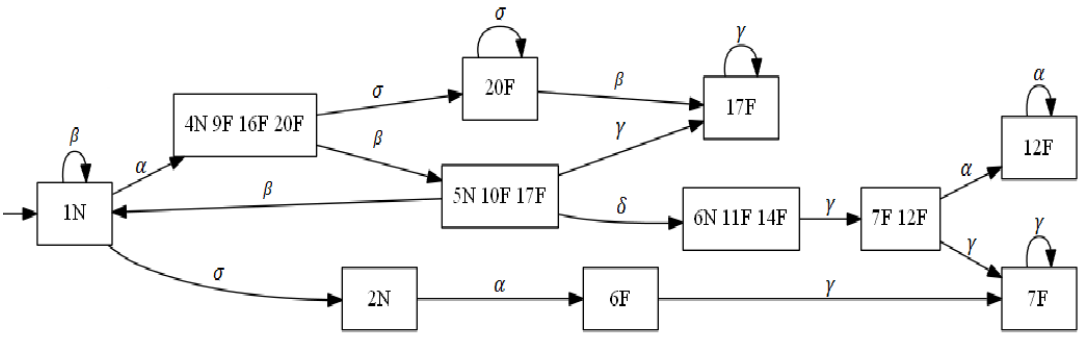


Figure 4.4: Basic language diagnoser G_{bd} corresponding to system G

Remark 1 *: We have recently informed the reader sufficiently about the construction process of basic language diagnoser, furthermore mentioned about the similarities and the differences between the construction process of basic event and language diagnosers. On the other hand, the necessary conditions that a basic language diagnoser needs to have for being able to help an observer detect all abnormal cases in a language model is nothing but the similar conditions as defined in basic event diagnoser, i.e. in Section 3.1.2. The definitions such as F_i-certain state, F_i-uncertain state, ambiguous state, F_i-indeterminite cycle etc. have also the same meanings as defined in the same section. Please refer to Section 3.1.1 for more details.*

Remark 2 *: It is clear that the proposed language diagnoser here by [8] marks all the undesired behavior as the same type and does not distinguish between them. To be able to detect more than one type of failures, we need a specification model for each type of failure. Then, we can use each specification language for obtaining separate diagnosers corresponding to each specification model, hence each failure type. Running these separate diagnosers in parallel will be enough for online detection of any corresponding type of failure occurred in the system.*

4.2 Modified Language Diagnoser with Fault-Detection Events

We propose to use the same idea as in Section 3.2 of introducing observable fault-detection events in order to simplify language-diagnosis. The idea is recalled in the following example. The system model is given in Figure 4.2.

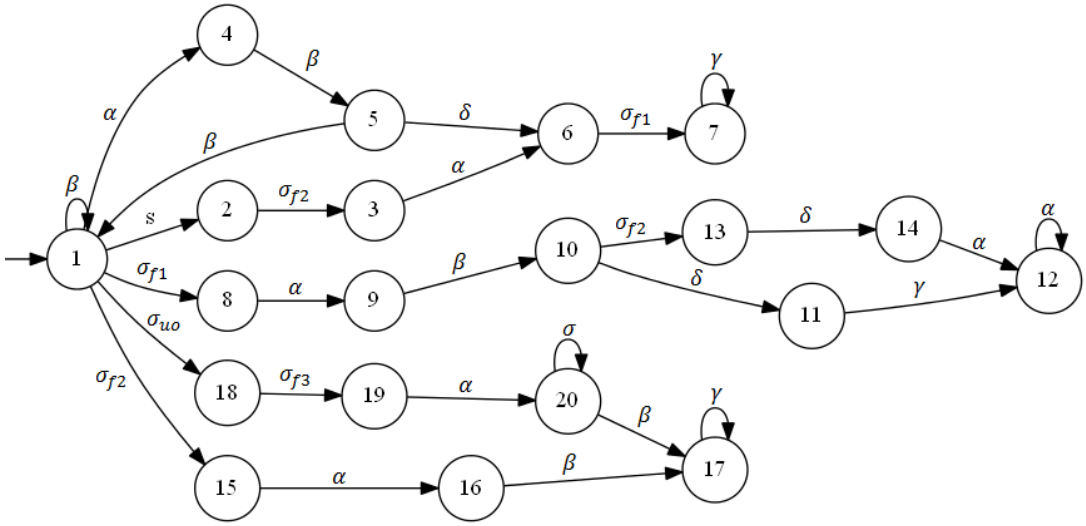


Figure 4.5: Motivating system model G

To describe the given system G briefly, we can point to its set of event partitions. Observable events of system model G are $\Sigma_o = \{\alpha, \beta, \gamma, \delta, \sigma\}$ whereas unobservable events are denoted by $\Sigma_{uo} = \{\sigma_{uo}, \sigma_{f1}, \sigma_{f2}, \sigma_{f3}\}$. Note that no failure events are introduced to the system as expected. All the faulty behavior is modeled by a specification language. Furthermore, being observable or not observable is not an issue for the alphabet of the specification language. The specification for this example is given in Figure 4.6.

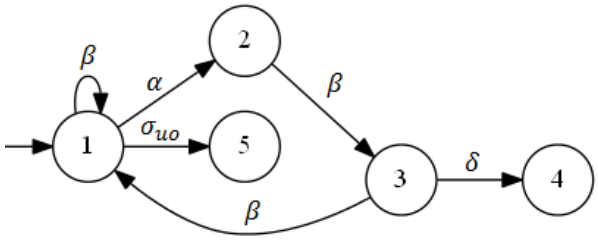


Figure 4.6: Specification language automaton G_{spec} for the system G

The basic language diagnoser G_{bd} for this example system is shown in Figure 4.7. We can easily notice that there are F-certain states in G_{bd}. Therefore, if an abnormal event occurred in the system according to specification language G_{spec}, an observer can understand it whenever he recognizes an arrival to such F-certain state assuming that the system has executed enough transitions.

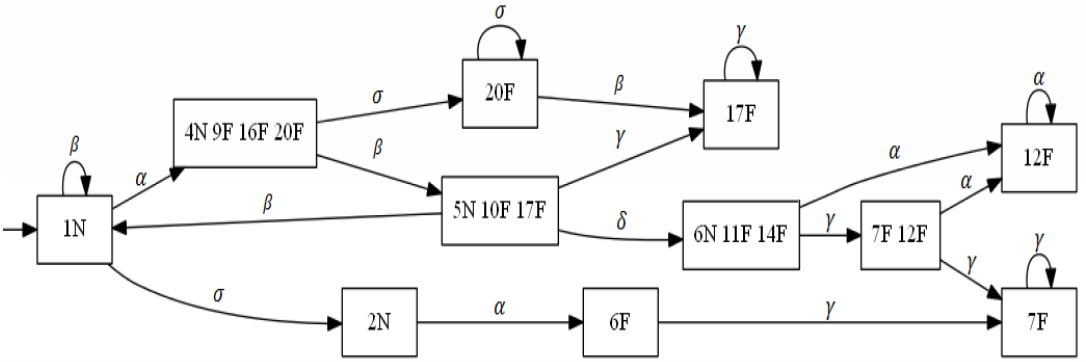


Figure 4.7: Basic language diagnoser G_{bd} for the system G with language specification G_{spec}

The modified language diagnoser which directly highlights every arrival to an F-certain state by an observable fault-detection event is shown in Figure 4.8.

It can be directly seen that G_{md} is a more practical language diagnoser realization compared to G_{bd} due to insertion of fault-detection event *FAILURE* to the language diagnoser alphabet. The fault-detection event *FAILURE* occurs immediately whenever the modified language diagnoser G_{md} is sure about the first time violation of the specification along a generated event sequence. For illustration, states in which the modified language diagnoser is sure about the occurrence of any specification violation for the first time are marked. Considering that the idea of the modified language-diagnoser is analogous to the idea of the modified event-diagnoser, it comes with the same advantages as listed in Section 3.2.

In order to explain the construction of the modified language-diagnoser, we use the analogous notation to Section 3.3. The modified language-diagnoser is a state machine

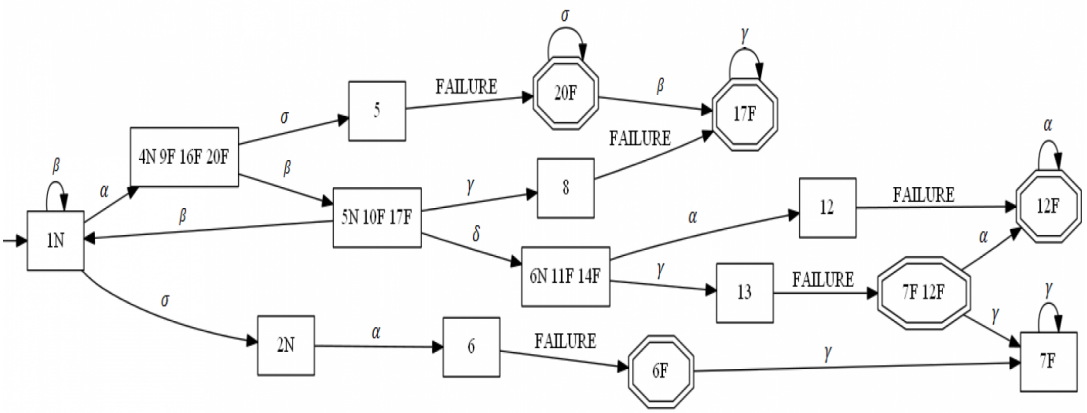


Figure 4.8: Modified language diagnoser G_{md} for the system G with language specification G_{spec}

$$G_{md} = (Q_{md}, \Sigma_{md}, \delta_{md}, q_{0md}, X_{md}) \tag{4.6}$$

where the elements of this five-tuple have the meanings of

$$\begin{aligned} Q_{md} &= Q_{bd} + \bigcup Q_{transient\_states} \\ \Sigma_{md} &= \Sigma_{bd} + FAILURE \\ q_{0_{md}} &= q_{0_{bd}} \\ X_{md} &= \bigcup F\text{-certain states} \end{aligned} \tag{4.7}$$

Again, the terms having the subscript *md* are elements of the modified diagnoser and terms having the subscript *bd* are elements of the basic diagnoser.

Inputs : rGen - System Model (Possible Behavior), G
rSpec - Specification Language G_{spec}
rDiagGen - Modified Diagnoser G_{md}

Outputs : void

It is obvious that the elements of G_{md} are nothing but slightly modified versions of the corresponding G_{bd} elements as mentioned during the introduction phase of the

Algorithm 2 Pseudo Code for Modified Language Diagnoser

LANGUAGEDIAGNOSER (*rGen*, *rSpec*, *rDiagGen*)
Begin
$G_{spec\_complement} \longleftarrow \overline{G_{spec}}$
$G_{par\_comp} \longleftarrow G \parallel G_{spec\_complement}$ and obtain composition map *cmap*
Remove unobs. transitions of G_{par_comp} to have a nondeterministic basis G_{obs} for G_{md}
Remove unobs. transitions from *cmap*
Create $q_{0_{md}}$ and assign $l_{q_{0_{md}}} \longleftarrow N$
Start matching G_{obs} states with G_{md} states
while !*dstates*.Empty() **do**
    dstate $\longleftarrow$ the next state of G_{md}
    if is_fcertain(*dstate*) and firstfailureonpath[dstate]=false **then**
        Do necessary changes on diagnoser for insertion FAILURE events
    end if
    Obtain the set of reachable states for G_{obs} correspondent of *dstate*
    for Each corresponding state $q \in G_{obs}$ of *cmap* **do**
        Collect transition states of q
    end for
    Clear l of *dstate*
    for Each transition event collected via corresponding G_{obs} state // parsing Eventwise **do**
        Add Arg1 of corresponding *cmap* element of corresponding G_{obs} of current diagnoser state with the label of G_{obs} state
        if Not end of transitions **then**
            Continue with next transition
        end if
        for $\forall\ q_d \in Q_d$ of G_md **do**
            if l of *dstate* is **not** equal to l' of q_d **then**
                continue
            end if
            if $q_d \in X_{md}$ and firstfailureonpath[*dstate*]=false **then**
                $\delta_{md} \longleftarrow \delta(dstate, FAILURE)=q_d$
            else
                $\delta_{md} \longleftarrow \delta(dstate, parsedevent)=q_d$
            end if
        end for
        if $!(\exists\ l'=l)$ **then**
            Create $q_d \in Q_d$ with corresponding l
            $\delta_{md} \longleftarrow \delta(dstate, parsedevent)=q_d$
        end if
        Clear l of *dstate*
    end for
end while
End

modified language diagnoser. Since the fact behind the idea of enhancement is the introduction of fault-detection events to the diagnoser automaton whenever the generated event sequence is traced enough to detect any abnormal event occurrence, we can see that automaton terms having *md* and *bd* as subscripts differ only by $Q_{transient\_states}$, $FAILURE$ event and marking of F-certain states as expected. To make them more clear for the reader, we can say that $Q_{transient\_states}$ are the states into which the modified diagnoser will make transition immediately after the last event helps it understand an F-certain state will be reached first time along that trace, hence just before generating *FAILURE* event and reaching into corresponding Fs-certain state. *FAILURE* events are the fault-detection event to indicate that something not expected according to given specification language G_{spec} is happened in the system. Final term X_{md} is the same F-certain states as in the case of basic language diagnoser, but now they are marked for better illustration.

Now it should be trivial to obtain δ_{md}. We can split the behavior of δ_{md} into two. First of them is the case in which transition function of basic language diagnoser δ_{bd} returns an F-certain state that an observer meets for the first time along the corresponding state. In this case, δ_{md} just throws the aforementioned *FAILURE* event and enters to a $q_{transient\_state} \in Q_{transient\_states}$ and then passes to the same resultant state with the same input event. Otherwise, δ_{md} does exactly the same transition as δ_{bd}.

On the other hand, please note that all of these elements are also similar to the elements of modified event diagnoser which is mentioned in Section 3.3 since the logic behind both of the modified diagnosers are same. The only difference between them is the existence of only one type of observable *FAILURE* event for every kind of abnormal behavior instead of *FAILURE*-Π_i corresponding to each type of failure.

More detailed explanation on the algorithm for construction of such an modified language diagnoser is given as in the form of a pseudo code. The ones who want to get involved in the subject deeper can inspect the pseudo code of the software implementation of modified language diagnoser.

We will end this section by some illustrative example systems in order to let the reader make comparisons for corresponding basic and modified language diagnosers and understand the notion in a better way.

4.3 Computation of Worst Case Detection Delay

In Section A we have introduced the algorithm proposed by [Yoo] which can be used to test the existence of a worst case detection delay for language diagnosability. Existence of a worst case detection delay was important since it makes us understand that the system is language diagnosable.

On the other hand, in this section we introduce a method for the practical computation of the exact worst case detection delay after verifying its existence. This approach differs from the other studies in the literature which only consider the existence of a theoretical upper bound for the detection delay. We determine an exact number based on the constructed modified language diagnoser.

To mention briefly about how *wcdd* is computed using the diagnoser, we will use the modified diagnoser G_{md} and the marked states X_{md} of this diagnoser which are known to be the marked F-certain states that are met first time along a generated sequence since the automaton starts to perform its operation. We will backtrace from each F-certain state to any normal labeled $q_{i_{md}}$ of G_{md} and. For each arrival to a normal state $q_{j_{md}}$, a forward trace on G up to each x_i having label F of $q_{i_{md}}$ will be searched from the x_j having label N corresponding to normal state $q_{j_{md}}$ by using the trace obtained from the backtrace. After each traversal, if wanted states are reached by consuming the traces, detection delays are stored and evaluated at the end.

Algorithm 3 Pseudo Code for Worst Case Detection Delay

Begin
Obtain the set of F-certain states into which there exists an fault-detection transition
for Each F-certain state in the set **do**
Go back to state from which $FAILURE$ transition stems from to enter the right path
Backward DFS by accumulating the events from the parsed F-certain until a normal state is reached
Obtain the set of indices corresponding to generator states in the label of parsed state
for Each index d corresponding to a generator state in the index set **do**
for Each state index g in the normal diagnoser state **do**
Forward DFS with accumulated trace from parsed g until reaching d
if any abnormal behavior not existing in G_{spec} is met first time along the accumulated trace **then**
Record its depth
end if
if target state d is reached by consuming the accumulated trace **then**
Record the depth of recorded abnormal behavior in wcdd pool
end if
end for
end for
end for
Output the greatest number in wcdd pool as wcdd
End

4.4 More Examples on Language Diagnoser

We will start this section by a system given in Figure 4.9 . The system is very similar to the system of the first example in Section 3.4 and only difference is that the transition from state 5 to state 3 is realized via event τ instead of δ in order to make the system language diagnosable.

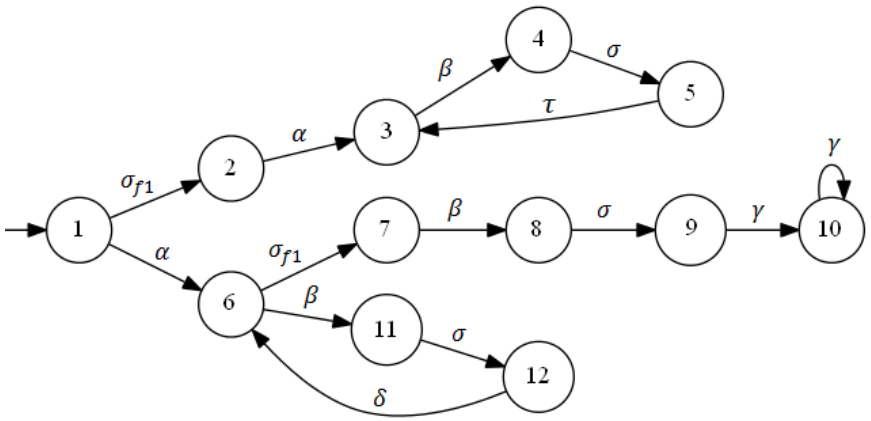


Figure 4.9: Example System G_1

Observable event set of the system in Figure 4.9 is $\Sigma_o = \{\alpha, \beta, \delta, \sigma, \gamma\}$ and unobservable event set is $\Sigma_{uo} = \{\sigma_{f1}\}$ as in the event diagnoser example. Don't forget to note that no failure partition exists for the given system. All of the abnormal behavior is defined according to the specification language which is obtained only by clearing the undesired events from the possible behavior automaton and given in Figure 4.10.

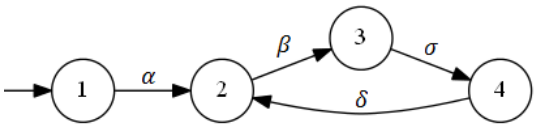


Figure 4.10: Specification Language Automaton G_{1spec} for Example System G_1

Corresponding basic and modified language diagnosers can be seen in Figure 4.11 and Figure 4.12 respectively.

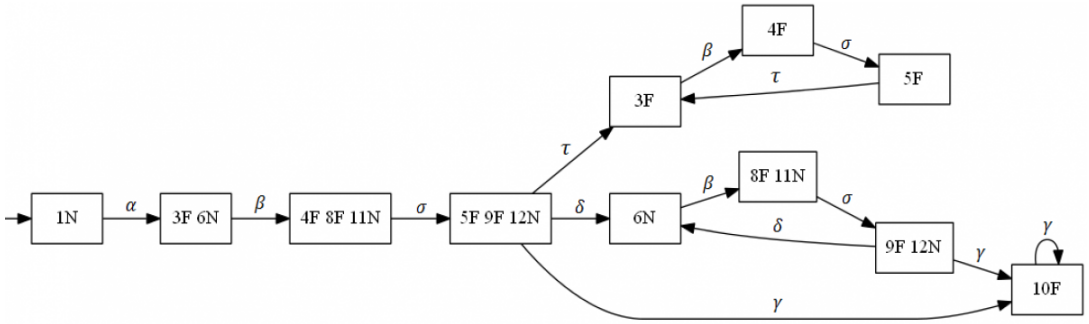


Figure 4.11: Basic Diagnoser G_{1bd} w.r.t G_{1spec} & G_1

Even though the basic diagnoser, G_{bd}, of the given system seems to have loops, i.e. {3F-4F-5F} and {6N-8F11N-9F2N}, it does not violate the necessary condition for being diagnosable since none of the loops are F_i-indeterminate. Moreover there are no ambiguous states in the diagnoser, therefore an observer can understand that there is something abnormal happened in the system using the diagnoser through additional computations and analysis.

On the other hand, in our modified diagnoser for the same system, it is easy to notice the *FAILURE* events which are inserted in order to indicate every abnormal behavior occurred in the system according to given language specification. These *FAILURE* events will save us from all the addidional computations to be able to detect abnormal behaviors of the system.

Finally, we will compute the worst case detection delay of our modified diagnoser G_{1md} for this system. Even though, we obtain the result directly from the software program by using the algorithm described in Section 4.3, we will still describe the algorithm steps in more detail only for this example system since it is the first time the reader meets its application on a system. There are three backtraces to normal states from marked F-certain states of G_{1md}. We can list these traces ignoring FAILURE as

s_1: $\gamma - \sigma - \beta - \alpha$ from "10F" to "1N"

s_2: $\gamma - \sigma - \beta$ from "10F" to "6N"

s_3: $\tau - \sigma - \beta - \alpha$ from "3F" to "1N"

Now, we will reverse these traces on G_1 by starting from the state they have ended and compute their detection delays. In fact, the traces obtained from G_{1md} and corresponding detection delays Δ_{dd} on the traces were respectively found out to be as

$$\begin{aligned} s_1 &\longrightarrow \alpha - \sigma_{f1} - \beta - \sigma - \gamma, \quad \Delta_{dd_{s1}} = 3 \\ s_2 &\longrightarrow \sigma_{f1} - \beta - \sigma - \gamma, \quad \Delta_{dd_{s2}} = 3 \\ s_3 &\longrightarrow \sigma_{f1} - \alpha - \beta - \sigma - \gamma, \quad \Delta_{dd_{s3}} = 4 \end{aligned}$$

By evaluating all the information supplied, it is obvious that $wcdd_{G_{md1}} = max(\Delta_{dd_{s1}}, \Delta_{dd_{s2}}, \Delta_{dd_{s3}}) = 4$. Hereby, we can conclude that our modified diagnoser will notify the user for the occurrence of faulty behavior after at most 4 event generations of the plant.

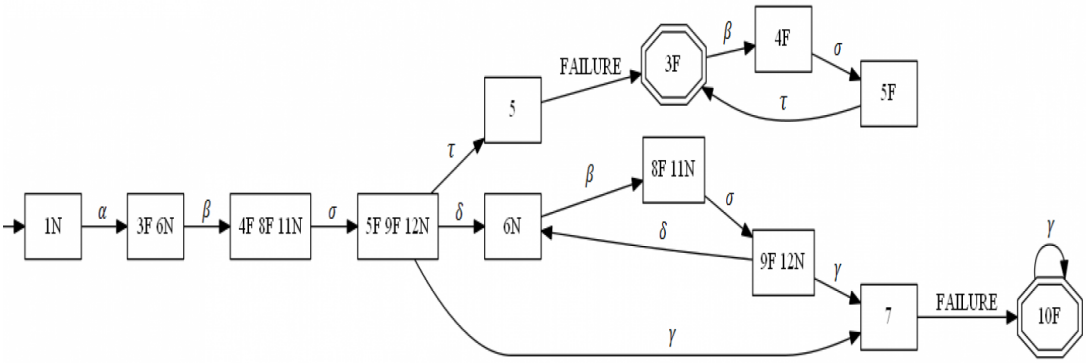


Figure 4.12: Modified Diagnoser G_{1md} w.r.t G_{1spec} & G_1

There is another system automaton G_2 is defined in Figure 4.13. Observable event set of the system is $\Sigma_o = \{\alpha, \beta, \gamma, \delta\}$ and unobservable event set is $\Sigma_{uo} = \{\sigma_{f1}, \sigma_{f2}\}$. We do not have any failure partition set in the system as expected. Additionally, it is easily noticable that normal execution of the system is realized through states $1 - 2 - 4 - 5$ and it returns to the start state 1 via transition with event d after any failure occured.

Therefore, specification automaton G_{2spec} of system G_2 that is the possible system behavior without any abnormal behavior is given in Figure 4.14.

Corresponding basic language diagnoser and modified language diagnosers with fault-detection event, i.e. *FAILURE* are given in Figure 4.15 and Figure 4.16, respectively.

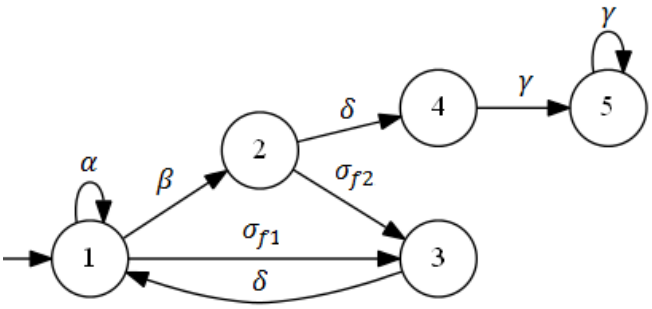


Figure 4.13: Example System G_2

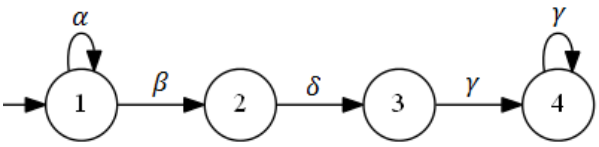


Figure 4.14: Specification Language G_{2spec} for Example System G_2

We will end analyzing this example by computing its worst case detection delay. The worst case detection algorithm given in Section 4.3 found out that $wcdd_{G_{2md}} = 3$ by evaluating 4 traces discovered by backtracing from marked F-certain states of G_{2md} to first normal state on the way. Since, local detection delays for these traces were computed as $\Delta_{dd_{s1}} = 1$, $\Delta_{dd_{s2}} = 3$, $\Delta_{dd_{s3}} = 2$ and $\Delta_{dd_{s4}} = 2$, $wcdd_{G_{2md}}$ equals 3. Therefore, we can say conclude that this system is capable of detecting an abnormal behavior in at most 3 events generated by the system.

The aforesaid traces are :

$$\begin{aligned}
s_1 &\longrightarrow \sigma_{f1} - \delta, & \Delta_{dd_{s1}} &= 1 \\
s_2 &\longrightarrow \sigma_{f2} - \delta - \sigma_{f1} - \delta, & \Delta_{dd_{s2}} &= 3 \\
s_3 &\longrightarrow \sigma_{f1} - \delta - \alpha, & \Delta_{dd_{s3}} &= 2 \\
s_4 &\longrightarrow \sigma_{f1} - \delta - \beta, & \Delta_{dd_{s4}} &= 2
\end{aligned}$$

Remark 3 *We conclude the modified language diagnoser chapter by highlighting that the algorithms proposed for constructing a modified diagnoser in Section 4.2 and*

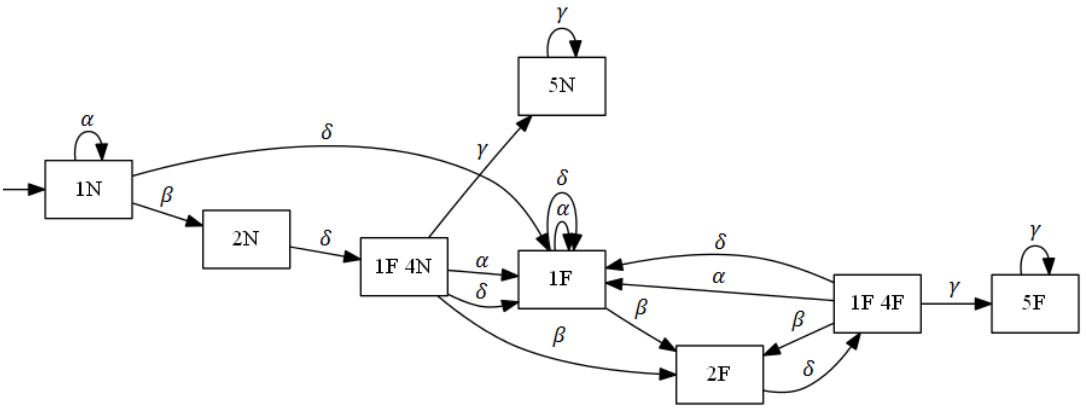


Figure 4.15: Basic Diagnoser G_{2bd} w.r.t G_{2spec} & G_2

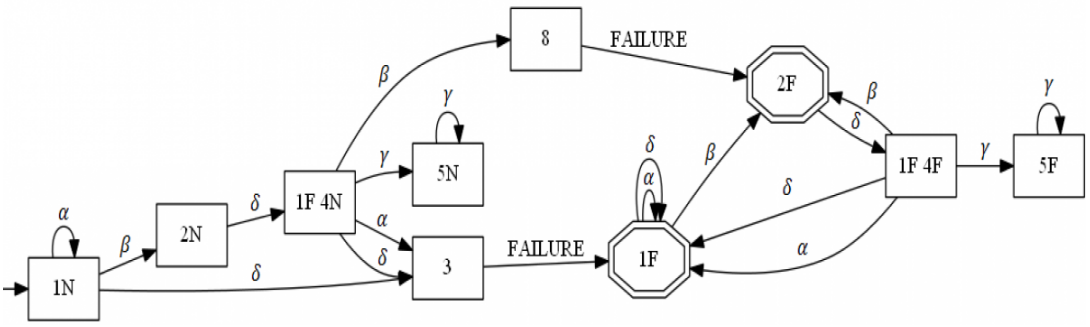


Figure 4.16: Modified Diagnoser G_{2id} w.r.t G_{2spec} & G_2

computing its worst case detection delay in Section 4.3 is applicable to any existing system that can be modeled by a DES with a specification automaton of normal behavior. Even though the given system is not language diagnosable with respect to the normal behavior automaton and masking function, algorithms will anyway produce a modified language diagnoser alerting all of the failure occurrences that are possible to be diagnosed and compute their corresponding worst case detection delays.

Moreover, since both of the algorithms are based on DFS algorithm and traversing the automata components corresponding to given system according to DFS, the resultant diagnoser automaton and wcdd computations are unique and any other outputs are impossible.

CHAPTER 5

REDUCTION OF THE OBSERVABLE EVENT SET

In the former part of the thesis work, we discussed the necessity, construction algorithm and advantages of the modified language diagnoser realization for notions of event and language diagnosability. To make our modified diagnoser further enhanced, we will look for the possibility of reducing the observable event set of the diagnoser alphabet in this chapter.

5.1 Reduction of the Observable Event Set

For this purpose, we analyze the problem of selecting an optimal set of observable events which can still make us accomplish the pre-assigned task of diagnosability. Having a reduced and optimal set of observable events without losing the diagnosability attribute of the system is important because each observable event requires its own detection by a sensor and also usually causes a larger size of the related diagnoser. By this study, we will have the opportunity of both ignoring some events by treating them as if they are unobservable even when they are indeed observable and hence, require fewer sensors for detection of any failure occurrence.

We will implement an algorithm that is based on the study Jiang [23] for the reduction of observable event set. Two methods are proposed to obtain an optimal set of sensors, or equivalently an optimal observation mask for mask-monotonic systems. Also in [23], Jiang states that since diagnosability is preserved under an increase in the information of event observation and it is mask monotonic. Moreover, and proves that mask these two methods One of them is called bottom-up method and the other

is called top-down method.

Consider a plant G with alphabet Σ and a set of observable events $\Sigma_o \subseteq \Sigma$. The non-faulty behavior of the system is given by a specification $K \subseteq L(G)$ and we assume that K is language-diagnosable for G and $p : \Sigma^\star \to \Sigma_o^\star$. We are interested in subsets $\Sigma_s \subseteq \Sigma_o$ such that still K is language-diagnosable for G and the reduced observation $p_s : \Sigma^\star \to \Sigma_s^\star$. In the best case, we find a set Σ_s such that it is not possible to remove and observations from Σ_s because otherwise diagnosability is violated. Such set Σ_s is called optimal.

The approach for finding an optimal Σ_s is based on a property of diagnosability that is found in [23]. Diagnosability is mask-monotonic. This means that diagnosability can only be violated by removing event observations. In contrast, a system will always stay diagnosable if new observations are added.

In this thesis, we realize the top-down method as proposed in [23].

Top-down method : The method is based on recursively finding a $\Sigma_s \subseteq \Sigma_o$ by removing events from Σ_o and testing if diagnosability is violated. The algorithms stops if no more observable events can be removed.

In [23], it is verified in detail with corresponding proofs, the top-down algorithm for finding a minimum partition in the set of all partitions satisfying the desired attribute results in a minimum partition set without violating the conditions of that attribute. Of course, it generates a partition only if there exists such a minimum partition in the set of all partitions that does not violate our mask-monotonic property, i.e. diagnosability. At this point, we will adapt the proposed algorithm for which existence of the solution is guaranteed by [23].

The last important thing to be noted for the top-down algorithm is its not assuring uniqueness of the resultant optimal set although existence of such solution is guaranteed. If there exists more than one optimal sensor sets, we may obtain any of them after applying the algorithm depending on the order in which we remove the events from the set Σ_o.

After explaining the algorithm in more detail by giving the pseudo code of it, we will

end this section by applying the algorithm on the previous illustrative systems given in Section 4.4.

The algorithms is implemented with the inputs rGen (plant model) and rSpec (specification language).

Algorithm 4 Pseudo Code for Reduction of the Observable Event Set

```
OPTIMIZEDIAGEVENTS (rGen, rSpec)
Begin
Deduce all the observable event system of the copied system
for Each observable event do
    Set parsed event as unobservable
    if The system is not language diagnosable w.r.t given rSpec then
        Reset the event as observable
    else
        Erase the event from the observable event set
    end if
end for
End
```

5.2 Reduced Modified Language Diagnosers for Previous Examples

We will first try to optimize the modified diagnoser G_{1md} which is previously constructed for the system given in Figure 4.9. Without losing the property of language diagnosability our algorithm shrank the observable event set $\Sigma_{o_{1md}} = \{\alpha, \beta, \sigma, \gamma, \tau\}$ of G_{1md} to $\Sigma_{o_{1rmd}} = \{\gamma, \tau\}$. The corresponding reduced modified language diagnoser is constructed as in Figure 5.1.

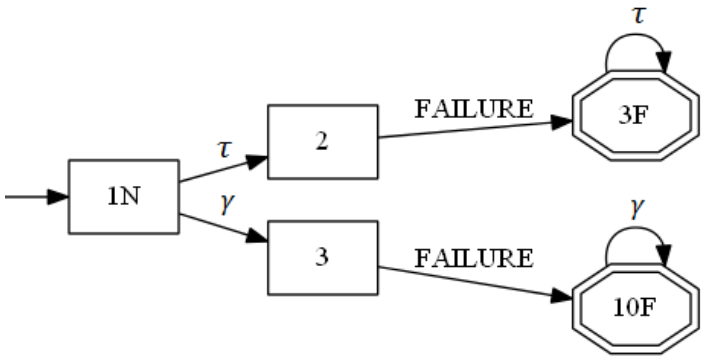


Figure 5.1: Reduced Modified Language Diagnoser $G_{1_{rmd}}$

The decrease in the size of required observations for language diagnosis is noticeable. Formerly $\left|\Sigma_{o_{1md}}\right| = 5$ but now $\left|\Sigma_{o_{1rmd}}\right| = 2$. Even though our previous language diagnoser model G_{1md} was saving an observer from lots of computation, analysis and observations, it seems that requiring lots of unnecessary event observation is a drawback for it. It is obvious that by using algorithm for reduction of observable event set, our modified language diagnoser will also save huge amount of cost by removing the necessity of observing unnecessary events, hence removing the usage of unnecessary sensors in a real system to observe these event occurrences.

We end this example by computing the *wcdd* delay by using the algorithm given in Section 4.3. The backtraces obtained by backward DFS are :

$$s_{b1}: \quad \tau \quad \text{from} \quad \text{"3F" to "1N"}$$
$$s_{b2}: \quad \gamma \quad \text{from} \quad \text{"10F" to "1N"}$$

And their corresponding forward traces obtained by forward DFS are:

$$s_{f1}: \quad \sigma_{f1} - \alpha - \beta - \sigma - \tau, \quad \Delta_{dd_{sf1}} = 4$$
$$s_{f2}: \quad \alpha - \sigma_{f1} - \beta - \sigma - \gamma, \quad \Delta_{dd_{sf2}} = 3$$

With such local detection delays of $\Delta_{dd_{sf1}}$ and $\Delta_{dd_{sf2}}$, it is obvious that $wcdd = 4$. It was also computed as 4 for the modified diagnoser without reduction algorithm. So, we can say that we have reduced the number of required observation set without an increase in *wcdd* for this example system.

There was another example system G_2 and its corresponding specification language G_{2spec} as shown in Figure 4.13 and Figure 4.14 respectively, in Section 4.4 and we will end this section by comparing our modified language diagnoser and the reduced version of it. Although our modified diagnoser was able to successfully inform the user by *FAILURE* events after a failure happened in the system, it needed 4 observable events to run smoothly. The necessary observable event set was $\Sigma_{o_{2md}}=\{\alpha, \beta, \delta, \gamma\}$ and $\left|\Sigma_{o_{2md}}\right|= 4$.

The reduced modified diagnoser $G_{2_{rmd}}$ for this illustrative system show us that it is also possible to detect any faulty behavior happened in the system within a finite event transition according to the specification automaton G_{2spec} by using only 2 event

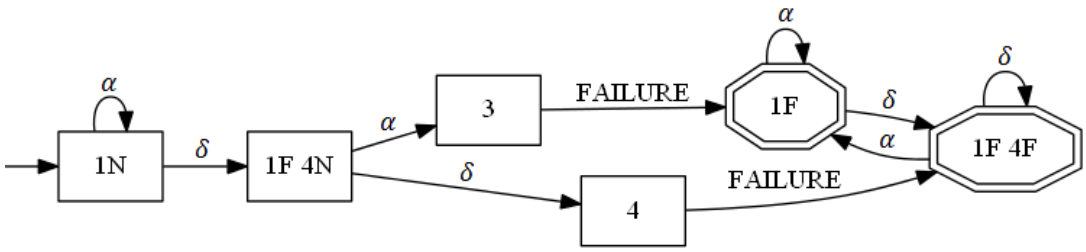


Figure 5.2: Reduced Modified Language Diagnoser $G_{2_{rmd}}$

observations, i.e. α, γ. The reduction in size of required observations from 4 to 2, again will save us from extra cost for detecting the other 2 unnecessary event generations, i.e. β, δ in the system.

As in the previous reduced modified diagnoser example, we want to make a comparison between *wcdd*s of reduced modified and default modified diagnosers of this example. The backtraces obtaine by backward DFS are :

$$s_{b1}: \quad \alpha - \delta \quad \text{from} \quad \text{"1F" to "1N"}$$
$$s_{b2}: \quad \delta - \delta \quad \text{from} \quad \text{"1F4F" to "1N"}$$

Forward traces corresponds to s_{b1} and s_{b1} which are obtained by forward DFS are:

$$\begin{aligned}
s_{f1}: &\quad \beta - \sigma_{f2} - \delta - \alpha, && \Delta_{dd_{sf1}} = 2 \\
s_{f2}: &\quad \sigma_{f1} - \delta - \alpha, && \Delta_{dd_{sf2}} = 2 \\
s_{f3}: &\quad \beta - \sigma_{f2} - \delta - \beta - \sigma_{f2} - \delta, && \Delta_{dd_{sf2}} = 4 \\
s_{f4}: &\quad \beta - \sigma_{f2} - \delta - \sigma_{f1} - \delta, && \Delta_{dd_{sf2}} = 3 \\
s_{f5}: &\quad \sigma_{f1} - \delta - \beta - \sigma_{f2} - \delta, && \Delta_{dd_{sf2}} = 4 \\
s_{f6}: &\quad \sigma_{f1} - \delta - \sigma_{f1} - \delta, && \Delta_{dd_{sf2}} = 3 \\
s_{f7}: &\quad \beta - \sigma_{f2} - \delta - \beta - \delta, && \Delta_{dd_{sf2}} = 3 \\
s_{f8}: &\quad \sigma_{f1} - \delta - \beta - \delta, && \Delta_{dd_{sf2}} = 3
\end{aligned}$$

, respectively. In section 4.3, for the default modified language diagnoser case, we found out that *wcdd* was 3 for the modified diagnoser of same example. Even though, there is a slight increase in *wcdd* and it equals to 4 in the case of reduced modified diagnoser, there may be up to 50% reduction in size and cost of sensors by applying the optimal sensor selection algorithm to this system.

Remark 4 *By applying optimal sensor selection algorithm to a given system, obviously there becomes a reduction in the information of observation of the system. Although it seems useful for reducing the number of required observations and their corresponding sensors in order to have a smaller modified diagnoser in size and accordingly a cheaper one in cost, we may encounter with more delay for the detection of abnormal behavior occurrences. Necessary observations may happen much later since there aren't as much as observable events in reduced modified diagnoser when compared to the default modified diagnoser.*

Even though it is not certain and to encounter with such delays, it must be carefully computed and wcdds must be carefully taken into account before constructing a real modified diagnoser in order not to meet with unfavourable wcdds.

CHAPTER 6

APPLICATION EXAMPLE : A COMMUNICATION NETWORK

In this chapter, we will show the applicability of our modified diagnoser proposal on an illustrative communication system. In sequence, we will give a brief description about the overall system, corresponding automaton models for each active part in the system including normal and faulty behaviors will be introduced. At the end, our modified diagnoser will be constructed for several cases to be able to detect the possible failure occurrences happened in the system in terms of the notion of language diagnosability.

6.1 Communication Network Example

We will use a sample communication network consisted of three separate nodes which are connected by a communication link. The nodes are connected to communication link via connection points AB, CD and EF respectively. Overall view of the system is given Figure 6.1

The general operation of the system can be summarized as follows:

- Node-1 functions as a master in the communication network,
- Node-2 & Node-3 function as slave in the communication network,
- Node-1 runs a query in order to confirm that Nodes 2, 3 and the communication links are available for sending a command,

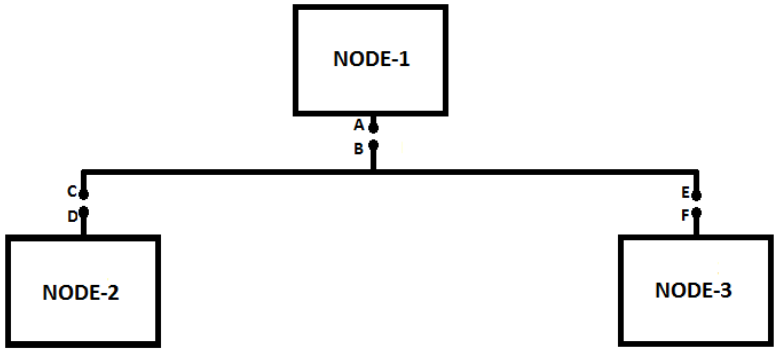


Figure 6.1: Overall view of the communication network

- Node-1 sends the command as soon as it receives the confirmations from the other nodes.

The system includes some problematic cases in which communication link AB and/or communication link CD may possibly break any time during the normal operation of the network and unfortunately these break events, i.e. *breakA-B* and *breakC-D* can not directly be noticed by an observer. Hereby, we will end this section by presenting the default masking function P_m and hence the events that can be sensed by an observer.

$$
\begin{aligned}
&P_m(\varepsilon) := \varepsilon \\
&P_m(\sigma) := \begin{cases} \varepsilon \text{ if } \sigma \in \{breakA\text{-}B,\ breakC\text{-}D\} \\ \sigma \text{ if } \sigma \notin \{breakA\text{-}B,\ breakC\text{-}D\} \end{cases} \\
&P_m(s\sigma) := P_m(s)P_m(\sigma) \text{ for s} \in \Sigma^* \text{ and } \sigma \in \Sigma
\end{aligned} \tag{6.1}
$$

6.2 Models of the System Components

The system is assumed to be composed of mainly 5 components, i.e. Node-1, Node-2, Node-3, Link-1, Link-2. In this section we will present their DFA models in sequence.

In Figure 6.2, the finite state automaton model for the possible behavior of Node-1 which is the master node of the communication network is given. To mention it briefly, its normal behavior can be understood by following the marked states starting from the initial state. As it is aforementioned, the normal operation of Node-1 starts

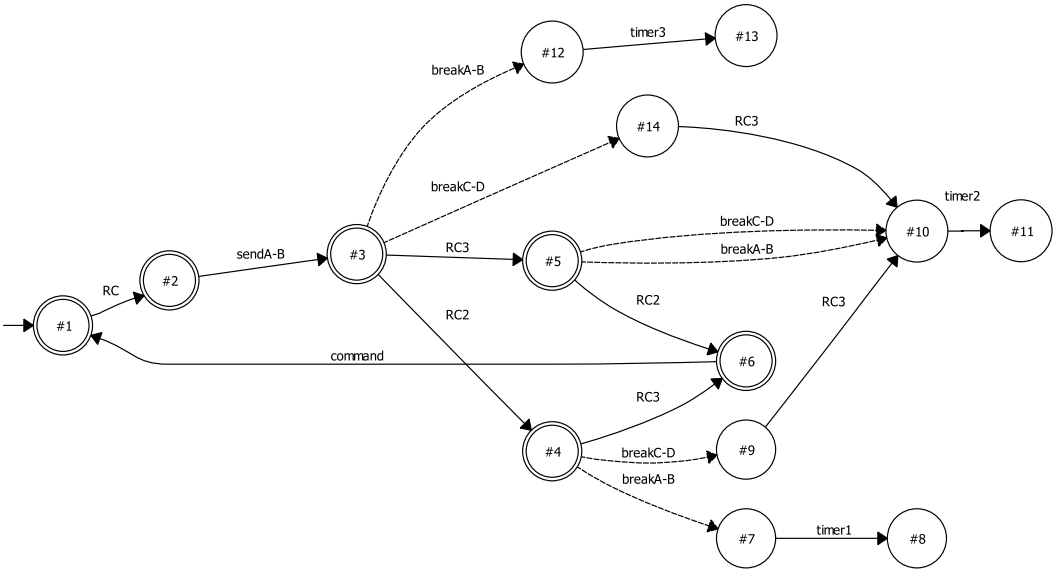


Figure 6.2: Possible Behavior Model of Node-1, the Master Node

with a request, i.e. *RC*, that is generated for the inquiry of Node-2 and Node-3 and then the request is passed to the communication link via *sendA-B* event and later on Node-1 starts to wait for receiving the responses of Node-2 and Node-3 and generates *RC2* and *RC3* for their arrivals. At the end of normal operation, Node-1 sends the command to the nodes to complete transaction. In spite of this, there may exist some faulty behavior in the possible behavior of Node-1 and connections on the links AB and CD may be broken, i.e. *breakA-B* and *breakC-D*, any time during this interaction and this break down cannot directly sensed by sensors according to given masking function P_m. Additionally, we can see in the model that Node-1 generates related timer events whenever too much time will elapse after a break down.

Possible behavior model of Node-2 can be seen Figure 6.3 and it is naturally in correlation with the behavior of Node-1. As it can also be discovered by again following the transitions between marked states, in the normal execution, if no failure such as *breakA-B* or *breakC-D* occurs in the system, it waits until the request from Node-1 arrives to itself over the communication link, i.e. *sendC-D*, then generates the respective response for it by event *respC-D* and later on, waits for Node-1 to send the

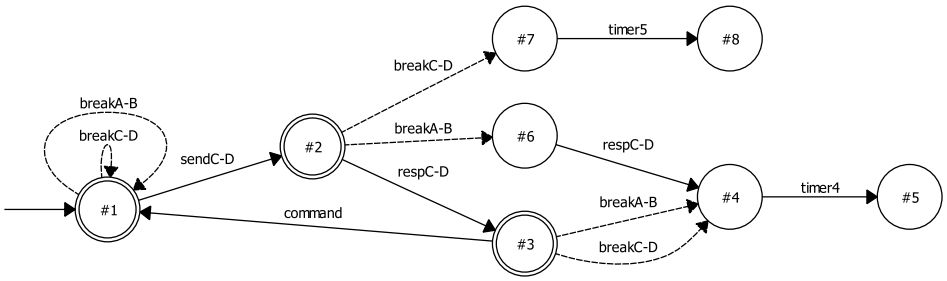


Figure 6.3: Possible Behavior Model of Node-2

command to complete the transaction. On the other hand, the possible faulty behavior on the communication links can also affect the possible behavior of Node-2 and we can see the corresponding *breakA-B* and *breakC-D* events triggering the related timer events after too much time elapses following a breakdown.

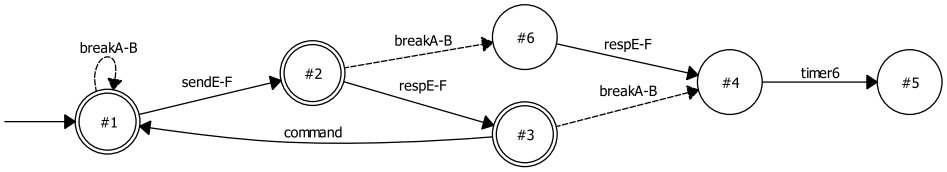


Figure 6.4: Possible Behavior Model of Node-3

The model of Node-3 is shown in Figure 6.4. As expected, its possible behavior is very similar to the possible behavior of Node-2 except for having no event *breakC-D*. We do not expect it to have that break event since the link CD is on the other side of the network but there might be a break down on link EF and we could denote it by an event *breakE-F*. We do not give place to such event since diagnosing a break down on communication link EF will require exactly the same construction steps as diagnosing a break down on link CD. We make this assumption to keep our overall system compact as possible for now. Apart from these the remaining possible behavior of Node-3 can be described by replacing C-D including events.

We model the behavior of communication links in 4 parts, i.e. Link-AD, Link-AF, Link-DA and Link-FA. Modelization of the system will end after giving their respective automata and brief descriptions about their process.

In Figure 6.5 the model for Link-AD is shown. In the normal operation mode, it forwards the inquiry to Node-2 by generating a *sendC-D*event after the arrival of *sendA-B* command.

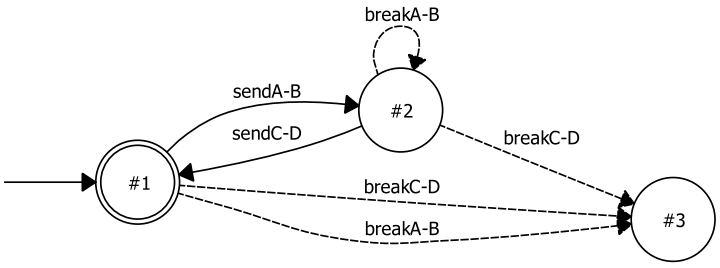


Figure 6.5: Possible Behavior Model of Link-AD

The behavior of Link-AF is modeled in a similar manner to Link-AD. The link forwards the confirmation request *sendA-B* to Node-3 by generating a *sendE-F* event in the normal operation mode. This behavior model is given in Figure 6.6

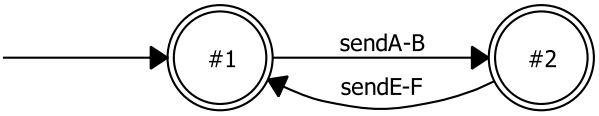


Figure 6.6: Possible Behavior Model of Link-AD

The remaining models belongs to Link-DA and Link-FA. Since their normal operation resembles each other and work in a very similar way we will describe them together.

As a normal behavior, these links generate *RC2* and *RC3* events after the respective arrival of responses from Node-2 and Node-3 for confirming the request of Node-1. Additionaly there can happen a *breakC-D* event on Link-DA during the transaction.

Finally, at the end of this section we will obtain an overall communication network automaton by synchronizing all the models of the components in the network. For this purpose, an automaton model G representing the whole communication network is obtained by synchronous composition of all elements.

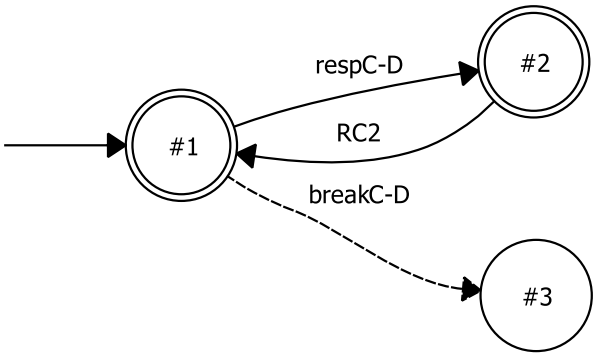


Figure 6.7: Possible Behavior Model of Link-DA

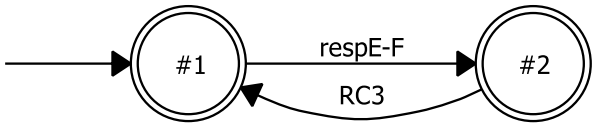


Figure 6.8: Possible Behavior Model of Link-FA

$$G = G_{Node-1} \parallel G_{Node-2} \parallel G_{Node-3} \parallel G_{Link-AD} \parallel G_{Link-AF} \parallel G_{Link-DA} \parallel G_{Link-FA} \quad (6.2)$$

6.3 Diagnosis of BreakA-B Event

We will first try our proposal of improved language diagnoser for the purpose of diagnosing occurrences of unobservable failure event *breakA-B* in the communication network. For being able to diagnose such event in terms of language diagnosability notion, what we need is a corresponding specification automaton $G_{breakAB}$. We will obtain this specification automaton by erasing the transitions via *breakA-B* event from G and then using the accessible part of the remaining automaton. Resultant specification automaton $G_{breakAB}$ is given in Figure 6.10.

As mentioned formerly, our modified language diagnoser is supposed to perform online and centralized diagnosis. For this application example, we assume that our

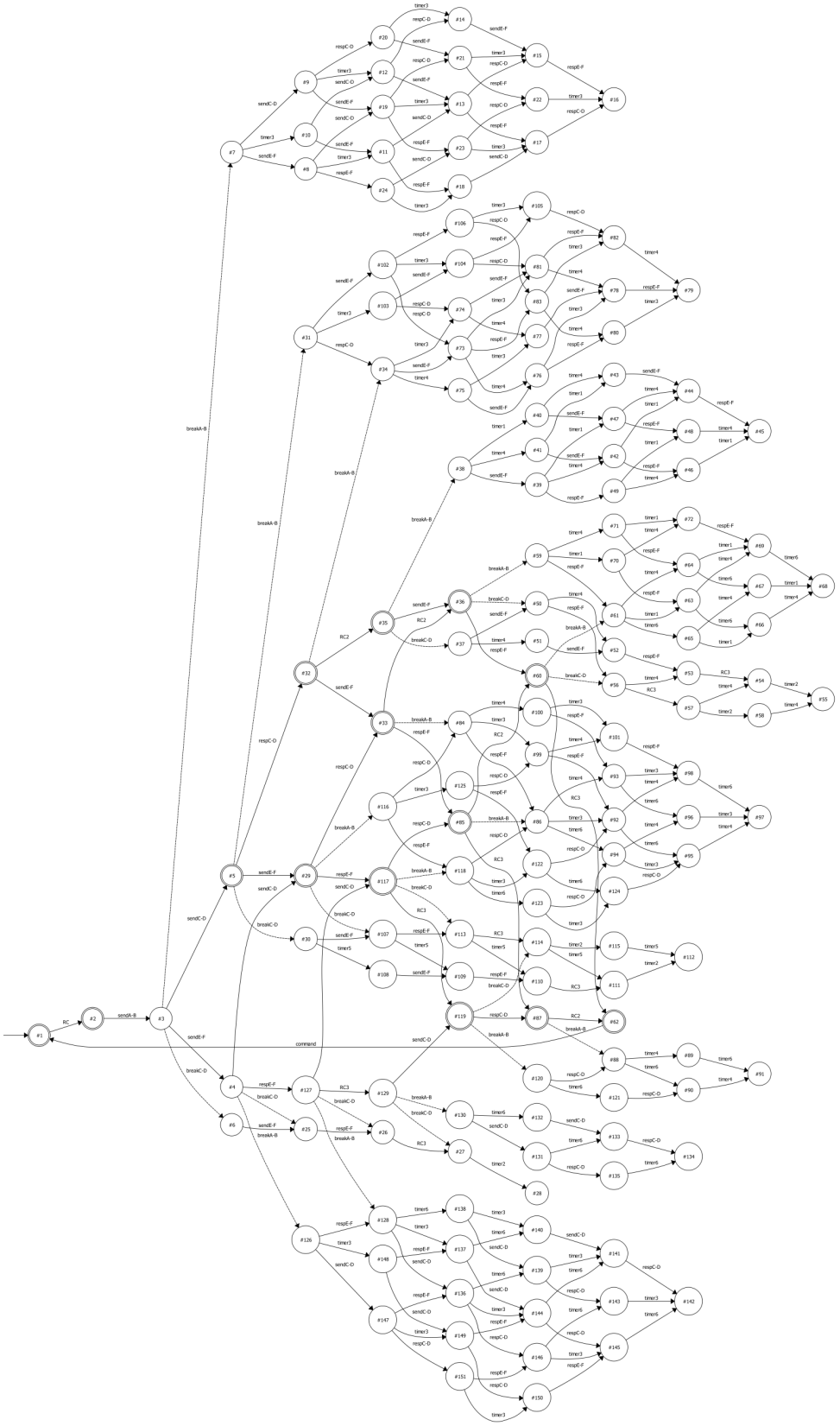


Figure 6.9: Possible Behavior Model for Overall Network

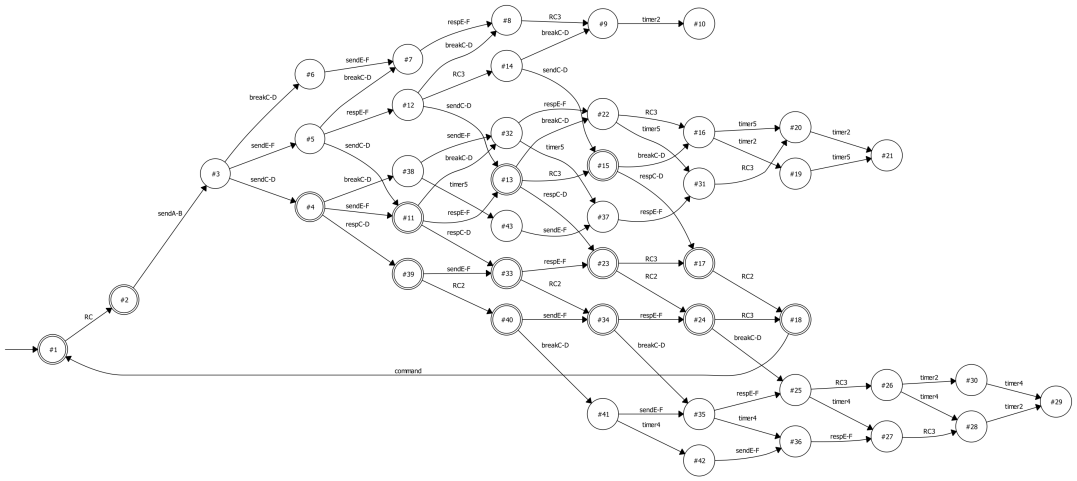


Figure 6.10: Specification Automaton $G_{breakAB}$

central diagnoser is observing the overall communication network and evaluating the gathered observations according to the given masking function in Equation 6.1.

With the supplied observable event set, Σ_{o_md}={$sendA-B$, $sendC-D$, $sendE-F$, RC, $RC2$, $RC3$, $respC-D$, $respE-F$, $timer1$, $timer2$, $timer3$, $timer4$, $timer5$, $timer6$}, the constructed modified language diagnoser is shown in Figure 6.11.

The default construction method for our modified language diagnoser results in this kind of huge automaton. Despite it is a large and complex automaton, only important thing to do by an observer is to note the *FAILURE* events inserted into G_{md} indicating the first time arrivals to F-certain states along a generated event sequence because G_{md} does all of the necessary computations, follows the generated observable event sequence and alerts for the occurrence of any faulty behavior happened in the system according to the specification automaton $G_{breakAB}$.

The worst case detection delay of the modified diagnoser that is obtained by *wcdd* algorithm that is proposed in Section 4.3 equals to 5. To be more explicit, we will reveal which paths are indicated to be the worst case paths according to algorithm. Since there are totally hundreds of paths to be analyzed for such a huge system, we will only reveal the ones causing a worst case detection.

Back trace(s) obtained from G_{md} automaton in Figure 6.11 and causing a *wcdd* are:

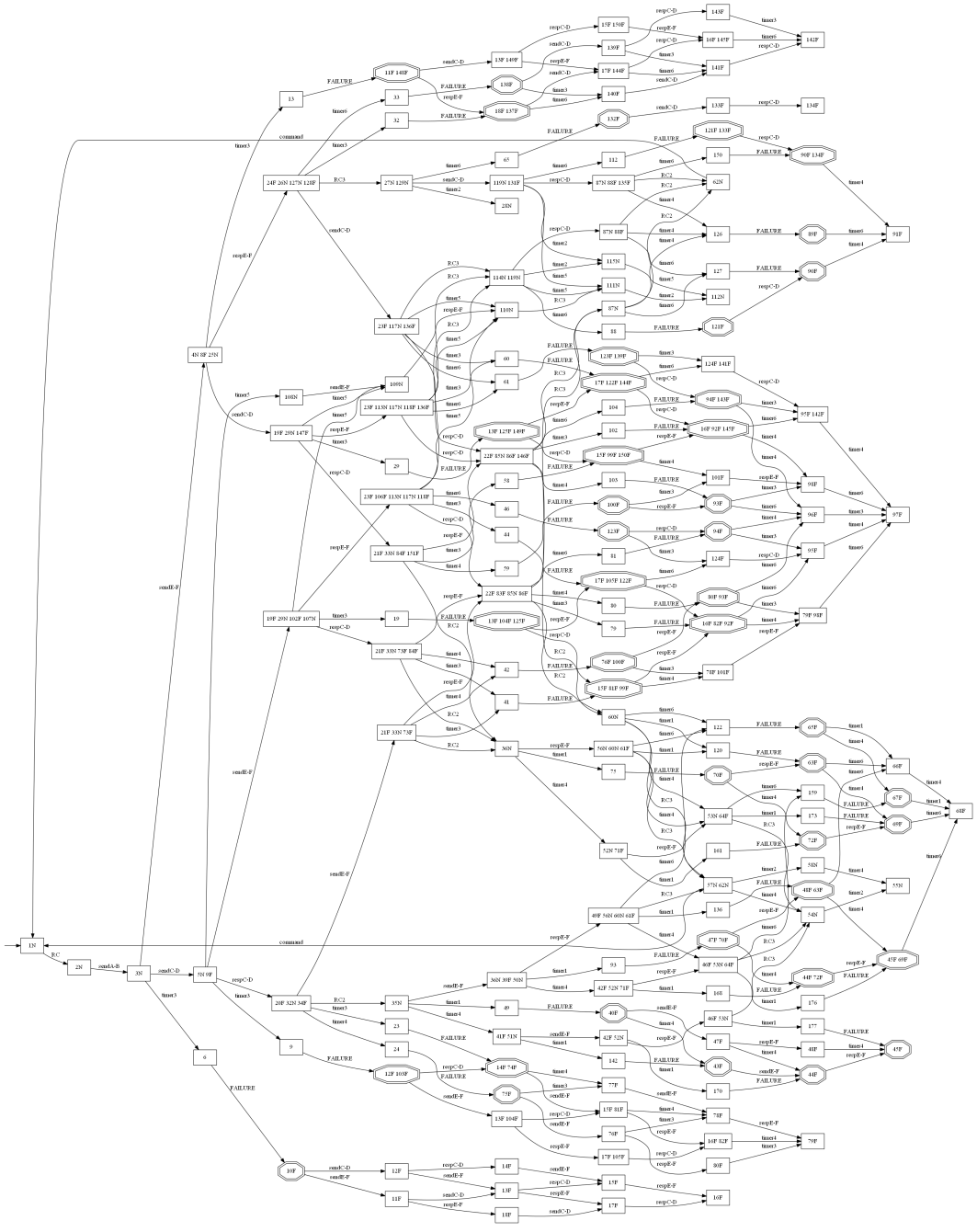


Figure 6.11: Modified Language Diagnoser G_{md} for Diagnosing $breakA - B$

s_{b1}: *timer*3-*respEF*-*respCD*-*sendEF*-*sendCD* from "16F 82F 92F" to "3N"
s_{b2}: *timer*3-*respEF*-*respCD*-*sendCD*-*sendEF* from "16F 92 145" to "3N"

and their corresponding forward traces causing to *wcdd* obtained by forward DFS are:

s_{f1}: *breakAB*-*sendCD*-*sendEF*-*respCD*-*respEF*-*timer*3, $\Delta_{dd_{sf1}} = 5$
s_{f2}: *breakAB*-*sendEF*-*sendCD*-*respCD*-*respEF*-*timer*3, $\Delta_{dd_{sf2}} = 5$

We will end this section by presenting the further modified diagnoser, i.e. the reduced modified language diagnoser with the optimal observable event set required for detection of *breakA*-*B* in terms of the notion of language diagnosability. The algorithm described in Section 5 reduces the number of necessary observations to only $|\Sigma_{o\_rmd}|$=3, which was $|\Sigma_{o\_md}|$=14 formerly, by making the observable event set equal to Σ_{o_rmd}={*timer*1, *timer*3, *timer*6} and our modified diagnoser is much more lucid. Corresponding reduced modified language diagnoser G_{rmd} is given in Figure 6.12.

To be able to make a comparison between the modified and reduced modified diagnosers of this communication system for the diagnosis of *breakA*-*B* event, we will compute *wcdd* for the reduced modified diagnoser, i.e. G_{rmd}.

Back trace for *wcdd* G_{rmd} is:

s_{b1}: *timer*3 from "10F 11F 12F 13F 14F 15F 16F 17F 18F 74F 77F 78F 79F
81F 82F 92F 98F 99F 101F 103F 104F 105F 122F 125F
137F 144F 145F 148F 149F 150F" to "1N"

and its corresponding forward traces obtained by forward DFS are:

s_{f1}: *RC*-*sendAB*-*breakAB*-*sendEF*-*sendCD*-*respCD*-*respEF*-*timer*3,
$\Delta_{dd_{sf1}} = 5$
s_{f2}: *RC*-*sendAB*-*sendCD*-*breakAB*-*respCD*-*sendEF*-*timer*4-*respEF*-*timer*3,
$\Delta_{dd_{sf2}} = 5$

Thus, we can conclude by noticing that there is not an increase in *wcdd* for diagnosis of *breakA*-*B* event although we reduce the size and cost of modified diagnoser significantly with further enhancement by applying the optimal sensor selection algorithm.

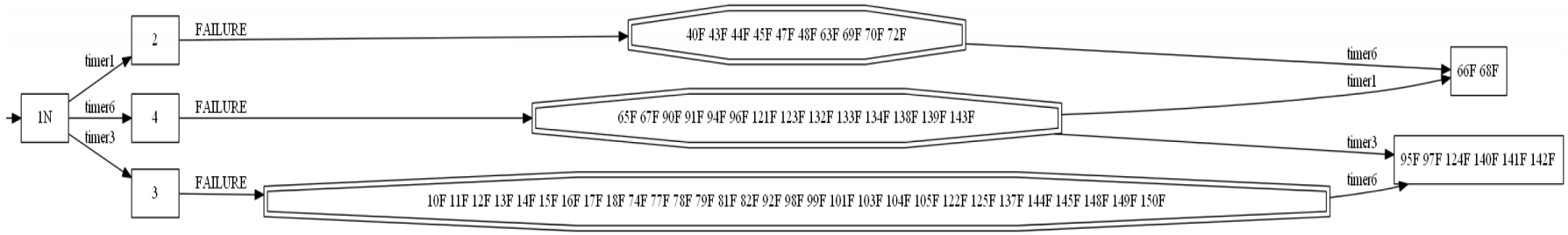


Figure 6.12: Reduced Modified Language Diagnoser G_{rmd}

6.4 Diagnosis of BreakC-D Event

In this section, we will focus our attention to diagnosis of the other unobservable failure event *breakC-D*. As in the case of diagnosis of *breakA-B* event, again we will need a specification automaton for the detection of *breakC-D* in terms of language diagnosability.

An automaton for such purpose is obtained by removing all the existing *breakC-D* transitions from the overall network model G and using the accessible part of the remaining automaton. Such specification automaton $G_{breakCD}$ obtained by using the described method is given in Figure 6.13.

We will assume that our diagnoser is observing the overall communication network and therefore, in the default setting, its observable event set is Σ_{o_md}={$sendA - B$, $sendC - D$, $sendE - F$, RC, $RC2$, $RC3$, $respC - D$, $respE - F$, $timer1$, $timer2$, $timer3$, $timer4$, $timer5$, $timer6$}.

Using these instructions with overall network model G and specification automaton $G_{breakCD}$, we will have our modified language diagnoser G_{md} constructed as it is seen in Figure 6.14.

Default modified language diagnoser G_{md} has a similar large and complex structure with the G_{md} of diagnosis of *breakA-B* case. Unfortunately, such diagnoser size is common in central diagnosers of complex systems since they are observing the overall system and evaluating it. But thanks to our proposal of modified language diagnoser for saving us dealing with such complex automata, tracking so many possible event generations and make us only track the generations of *FAILURE* event.

For the task of *breakC-D* event diagnosis , the worst case detection delay of the modified diagnoser equals to 4 according to *wcdd* algorithm that is proposed in Section 4.3. We will again only reveal the paths are causing to the worst case detection paths becasue there are too many paths to be analyzed for such a huge system.

By using the after the first step of *wcdd* computation algorithm, i.e. backward DFS, we obtain lots of back trace(s). The back trace(s) obtained from G_{md} automaton in Figure 6.11 and causing a *wcdd* are:

s_{b1}: *timer*2-*RC*3-*respEF*-*sendEF* from "24F" to "3N"

s_{b2}: *timer*2-*RC*3-*respEF*-*sendEF* from "115F" to "5N 9N"

s_{b3}: *timer*5-*RC*3-*respEF*-*sendEF* from "111F" to "5N 9N"

s_{b4}: *RC*3-*timer*4-*respEF*-*sendEF* from "54F" to "35N"

s_{b5}: *RC*3-*respEF*-*sendEF*-*timer*4 from "54F" to "35N"

and their corresponding forward traces which are resulting in a *wcdd* and obtained by forward DFS are:

s_{f1}: *breakCD*-*sendEF*-*respEF*-*RC*3-*timer*2, $\Delta_{dd_{sf1}} = 4$

s_{f2}: *breakCD*-*sendEF*-*respEF*-*RC*3-*timer*2, $\Delta_{dd_{sf2}} = 4$

s_{f3}: *breakCD*-*sendEF*-*respEF*-*RC*3-*timer*5, $\Delta_{dd_{sf3}} = 4$

s_{f4}: *breakCD*-*sendEF*-*respEF*-*timer*4-*RC*3, $\Delta_{dd_{sf3}} = 4$

s_{f5}: *breakCD*-*timer*4-*sendEF*-*respEF*-*RC*3, $\Delta_{dd_{sf3}} = 4$

Finally, we will look for any possible reduction in the size of required observable events for the diagnosis of breakC-D event by using our *optimizediagevent* algorithm given in Section 5. The algorithm again reduces the number of required observations significantly for the diagnosis of *breakC-D* event and save our modified diagnosers from tracking unnecessary observable events. After applying the algorithm reduced modified language diagnoser G_{rmd} finds out that it needs only detecting the generation of *timer2* event for the same diagnosis task. It is obvious that the final reduced and modified language diagnoser algorithm saves the cost of sensing 13 unnecessary observations since $|\Sigma_{md}|$=14 and $|\Sigma_{rmd}|$=1. Corresponding G_{rmd} is given in Figure 6.15.

At the end of this section, we will again compute *wcdd* for the reduced modified diagnoser, i.e. G_{rmd} in order to make a comparison between the modified and reduced modified diagnosers of this communication system for the diagnosis of *breakC-D* event.

Back traces from G_{rmd} causing to *wcdd* are:

s_{b1}: *timer*2 from "28F 55F 58F 112F 115F" to "1N"

and their corresponding forward traces causing to *wcdd* obtained by forward DFS are:

s_{f1}: *RC-sendAB-sendCD-respCD-RC2-breakCD-sendEF-timer4-respEF-RC3-timer2*, $\Delta_{dd_{sf1}} = 5$

s_{f2}: *RC-sendAB-sendCD-breakCD-sendEF-timer5-respEF-RC3-timer2*, $\Delta_{dd_{sf2}} = 5$

By comparing the worst case detection delays of modified and reduced modified diagnoser, we can conclude by noticing that there is a slight increase of *wcdd* from 4 to 5 for diagnosis of *breakC-D* event even though we reduce the size and cost of modified diagnoser significantly with further enhancement by applying the optimal sensor selection algorithm.

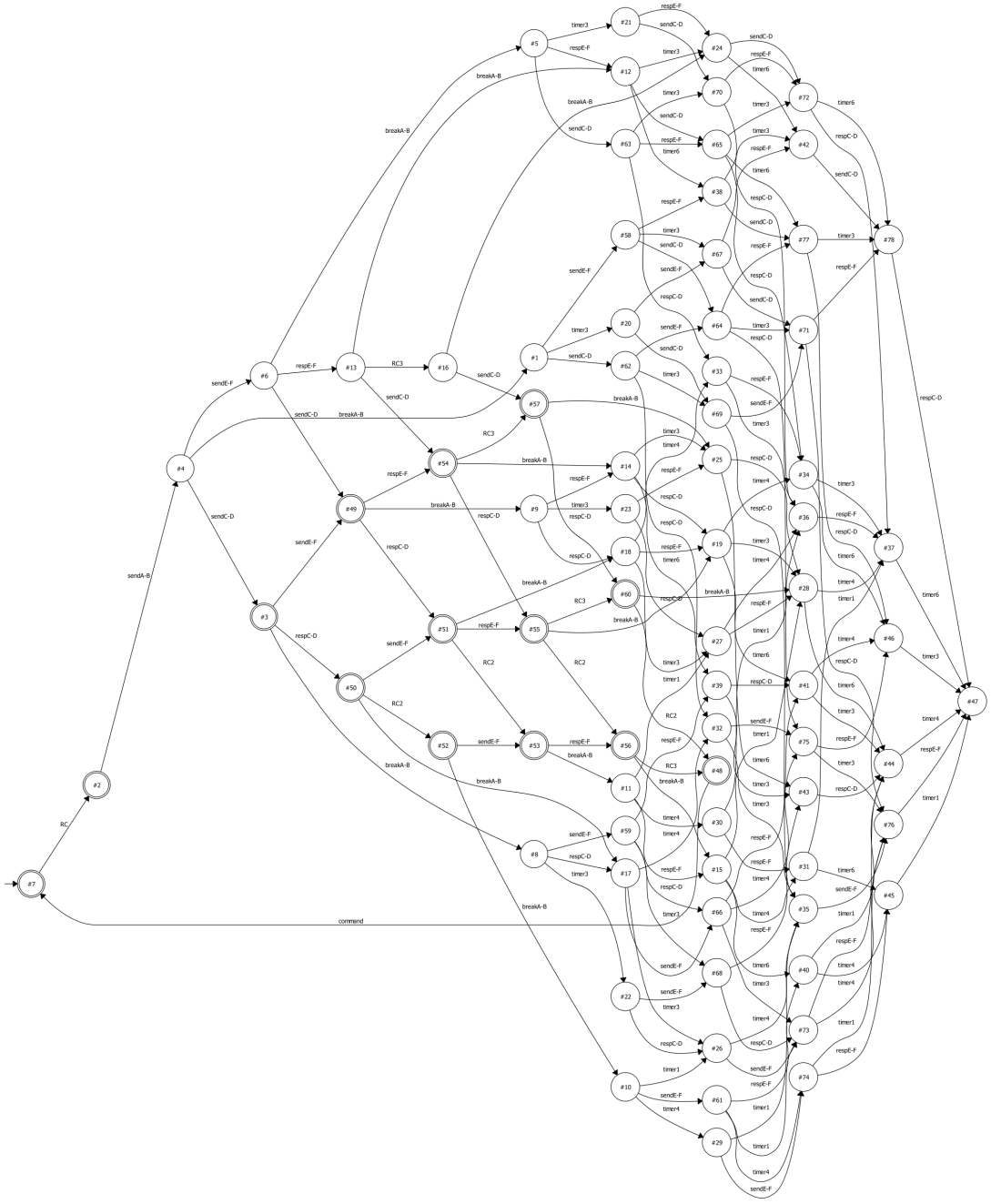


Figure 6.13: Specification Automaton $G_{breakCD}$

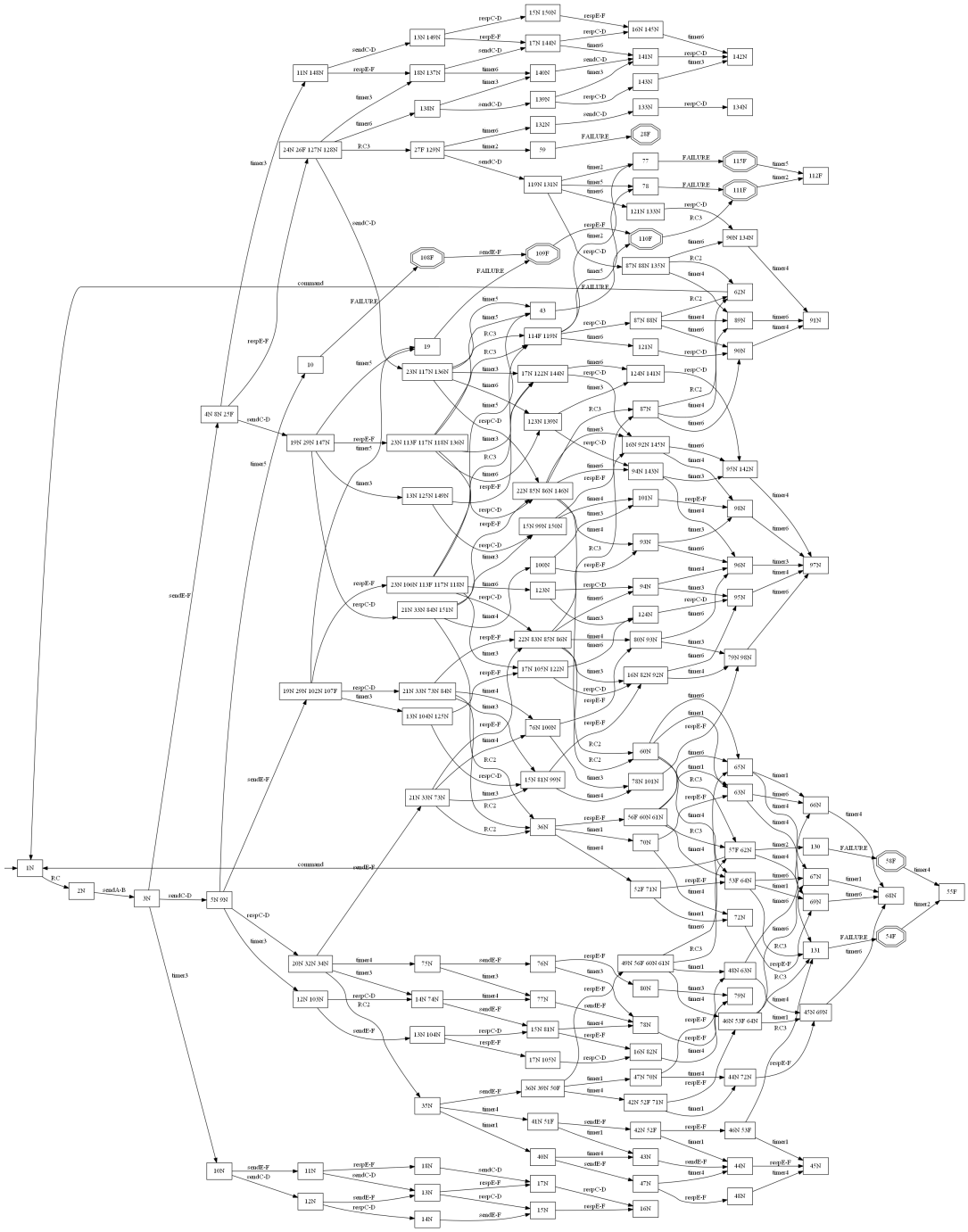


Figure 6.14: Modified Language Diagnoser G_{md} for Diagnosing $breakC - D$

Figure 6.15: Reduced Modified Language Diagnoser G_{rmd}

CHAPTER 7

CONCLUSION AND FUTURE WORK

In this thesis work, we conducted our studies generally on the notion of diagnosability. As well as already existing DES approaches in the literature, the diagnosability notion was inspected in terms of both event diagnosability [8], [2], i.e. each faulty behavior is introduced to the system as failure events and language diagnosability [11], [12], i.e. each abnormal behavior other than the normal execution of the system is introduced by a special specification automaton.

As the first contribution, this thesis improves the applicability of basic event/language diagnosers proposed in [8] by the development of modified event/language diagnosers. The main advantage of these modified diagnosers are directly signalling the detection of a fault and do not require any additional computation. Hence, they can be conveniently used in online diagnosis. As the second contribution, an algorithm for the exact worst-case detection delay computation is developed for our modified language-diagnoser. This is different than other studies that only show the existence of such worst-case detection delay. Moreover, both of these contributions are generalized and they can be applied to any existing DES modeled system for the detection of failure occurrences and compute their worst case detection delays even though the given system needs to be diagnosable for letting the proposed diagnoers detect all of the faulty behavior occurences and computing their worst case detection delays, respectively. The third contribution of the thesis is providing an algorithm for reducing the required observations for diagnosis if there exists such a possible reduction without losing the property of diagnosability. Though the resultant required observation set may not be unique and there can exist more than one optimal sensor set with

respect to the given system, the algorithm is guaranteed to output one of the existing optimal observation sets without violating diagnosability.

As a result, this thesis offers a modified and, if possible, reduced diagnoser which has a computed worst case detection delay for each diagnoable failure whenever all of the methods being proposed in the thesis is applied to a given system. The methodology is generalized and applicable to all DES modeled systems. Existance of such a modified diagnoser plus its wcdd computation is guaranteed by the algorithms in the thesis but there may not be a unique outcome because of the optimal sensor selection algorithm even though the algorithms for modified diagnoser and worst case detection computation produces a unique modified diagnoser.

All algorithms are implemented in the software library libFAUDES [19], [15] and all of the offered methods and algorithms are applied to a communication network example after being described in detail and their functionality is demonstrated.

Both of the diagnosers that we proposed in this thesis fall into category of centralized diagnosis in the literature since all the information obtained from the system is gathered and evaluated by a single diagnosis unit. From this point of view, this study can be extended for decentralized diagnosis for reaching a more extensive scope in the notion of diagnosability as a future work.

Additionally, the case in which repeated failures can happen along a trace is not taken care by our modified diagnosers and only the occurrence of corresponding permanent type of faulty behavior is reported to an observer. In such a case of repeatedly occurring failures of same type along a trace, our diagnosers will be able to inform the user only about the occurence of corresponding type of failure, i.e. not about each happening of that type of failure. As another future work, a more advanced enhancement on our approach for the modified diagnosers may be used for repeatedly diagnosing and repairing for the case that faults repeatedly happen along a trace.

REFERENCES

[1] S. Das and L. Holloway, "Characterizing a confidence space for discrete event timings for fault monitoring using discrete sensing and actuation signals," *Systems, Man and Cybernetics*, vol. 30, no. 1, pp. 52–66, 2000.

[2] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial algorithm for testing diagnosability of discrete-event systems," *Automatic Control*, vol. 46, no. 8, pp. 1318–1321, 2001.

[3] K. Schmidt, "Abstraction-based failure diagnosis for discrete event systems," *System and Control Letters*, vol. 59, pp. 42–47, 2010.

[4] ——, "Abstraction-based verification of codiagnosability for discrete event systems," *Automatica*, vol. 46, pp. 1489–1494, 2010.

[5] A. Bouloutas, G. Hart, and M. Schwartz, "Simple finite-state fault detectors for communication networks," *Communications*, vol. 40, no. 3, pp. 477–479, 1992.

[6] A. Benveniste, E. Fabre, S. Haar, and C. Jard, "Diagnosis of asynchronous discrete-event systems: a net unfolding approach," *Automatic Control*, vol. 48, no. 5, pp. 714–727, 2003.

[7] D. Godbole, J. Lygeros, E. Singh, A. Deshpande, and A. Lindsey, "Communication protocols for a fault-tolerant automated highway system," *Control Systems Technology*, vol. 8, no. 5, pp. 787–800, 2000.

[8] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete-event systems," *Automatic Control, IEEE Transactions on*, vol. 40, no. 9, pp. 1555–1575, 1995.

[9] S. H. Zad, R. Kwong, and W. Wonham, "Fault diagnosis in discrete-event systems: framework and model reduction," *Automatic Control*, vol. 48, no. 7, pp. 1199–1212, 2003.

[10] M. Sampath, "A hybrid approach to failure diagnosis of industrial systems," in *American Control Conference*, vol. 3, 2001, pp. 2077–2082.

[11] T. Yoo and H.E.Garcia, "Diagnosis of behaviors of interest in partially-observed discrete-event systems," *System Control Letters*, vol. 57, no. 12, pp. 1023–1029, 2008.

[12] W. Qui and R.Kumar, "Decentralized failure diagnosis of discrete event systems," *Systems, Man and Cybernetics*, vol. 36, no. 2, pp. 384–395, 2006.

[13] S. Yoo and S. Lafortune, "Polynomial time verification of diagnosability of partially observed discrete-event systems," *Automatic Control*, vol. 47, no. 9, pp. 1491–1495, 2002.

[14] Y. Wang, S. Yoo, and S. Lafortune, "Diagnosis of discrete event systems using decentralized architectures," *Discrete Event Dynamic Systems*, vol. 17, no. 2, pp. 233–263, 2007.

[15] Y. Pencolé and M. O. Cordier, "A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks," *Artificial Intelligence*, vol. 164, no. 1-2, pp. 121–170, 2005.

[16] K. Schmidt, "Verification of modular diagnosability with local specifications for discrete-event systems," *Systems, Man, and Cybernetics: Systems, IEEE Transactions on*, vol. 43, no. 5, pp. 1130–1140, 2013.

[17] R. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[18] R. Debouk, S. Lafortune, and D. Teneketzis, "Coordinated decentralized protocols for failure diagnosis of discrete-event systems," *Discrete Event Systems: Theory and Applications*, vol. 10, pp. 33–86, 2000.

[19] R. Suu and M. Wonham, "Global and local consistencies in distributed fault diagnosis for discrete-event systems," *Automatic Control*, vol. 50, no. 12, pp. 1923–1935, 2005.

[20] B. E. Kart and K. W. Schmidt, "Ayrık olaylı sistemler İçin geliştirilmiş bir tanılayıcı," in *Mühendislik ve Teknoloji Sempozyumu, Çankaya Üniversitesi*, 2013.

[21] T. Moor, K. Schmidt, and S. Perk, "libfaudes - an open source c++ library for discrete event systems," *Proceedings of the 9th International Workshop on Discrete Event Systems Goteborg, Sweden*, 2008.

[22] libFAUDES. (2006–2011) libFAUDES software library for discrete event systems. [Online]. Available: www.rt.eei.uni-erlangen.de/FGdes/faudes

[23] S. Jiang, R. R. Kumar, and H. E. Garcia, "Optimal sensor selection for discrete-event systems with partial observation," *Automatic Control, IEEE Transactions on*, vol. 48, no. 3, pp. 369–381, Mar. 2003.

[24] C. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Springer, 2008.

[25] R. G. P. Caines and S. Wang, "Classical and logic based dynamic observers for finite automata," *IMA J. Math. Contr. Inform.*, vol. 8, pp. 45–80, 1991.

[26] A. F. R. Cieslak, D. Desclaux and P. Varaiya, "Supervisory control of discrete-event processes with partial observations," *IEEE Trans. Automat. Contr.*, vol. 33, pp. 249–260, 1988.

[27] F. Lin and W. M. Wonham, "On observability of discrete-event systems," *Inform. Sci.*, vol. 44, pp. 173–198, 1988.

[28] C. M. Özveren and A. S. Willsky, "Observability of discrete-event dynamic systems," *IEEE Trans. Automat. Contr*, vol. 35, pp. 797–806, 1990.

[29] ——, "Invertibility of discrete-event dynamic systems," *Math. Contr. Signals Syst.*, vol. 5, pp. 365–390, 1992.

[30] Y. Park and E. Chong, "On the eventual invertibility of discrete-event systems and its application," *Proc. 32nd Conf. Decis. Contr.*, pp. 680–685, 1993.

[31] P. J. Ramadge, "Observability of discrete-event systems," *Proc. 25th Conf. Decis. Contr.*, pp. 1108–1112, 1986.

[32] F. Lin, "Diagnosability of discrete-event systems and its applications," *J. DEDS*, vol. 4, no. 2, pp. 197–212, 1994.

[33] S. Bavishi and E. Chong, "Automated fault diagnosis using a discrete event systems framework," *IMA J. Math. Contr. Inform.*, vol. 8, pp. 213–218, 1994.

APPENDIX A

Testing Language Diagnosability

In this section we will introduce the method which Yoo suggested in his paper [11] to the reader for being able to check the existence of finite detection delay. Since it is used in our thesis work with some simple modification, we had better make the reader familiar with the algorithm here. Via this proposed method, it can be understood that if our system has the formerly mentioned system property, language diagnosability. Moreover, building upon the results of this section, another algorithm for computing the worst case detection delay, d_{dia}, is mentioned in [11] for the ones who want to probe the subject.

Our purpose is to build a weighted and directed graph in order to verify the existence of finite detection delay. We will denote the formerly mentioned two notions, the normal system behavior and the possible system behavior by N and P, respectively and our masking function M will be the natural projection function p_o.

The weighted, directed graph G and its element, the set of vertices V are described as

$$G(N, P, M) = (V(N, P), E(N, P, M)) \tag{A.1}$$

where a weight function w is defined for the weighted and directed graph $G(N, P, M)$ as

$$w : E \rightarrow \{-1, 0\} \tag{A.2}$$

After completely giving the description of graph G, we will explain the implication

of it in more detail. The reader must know about the following transition notation before proceeding into description of how edges are assigned in the graph.

$$\begin{aligned} \delta^N(q_1, \sigma') &= q_1' \\ \delta^N(q_2, \sigma') &= q_2' \\ \delta^N(q_3, \sigma') &= q_3' \end{aligned} \tag{A.3}$$

Please note here that, event σ is used to define q_2' and q_3' while event σ' is used to define q_1'. Since no other constraint is defined for the events in the graph, it can be thought that events $\sigma$and σ' can be identical.

The notation $p \xrightarrow{i} q$ implies that there is an edge $(p, q) \in E$ with candidate $i \in \{-1, 0\}$. The weight of the edge is determined by choosing the minimum of weight candidates defined over the edge $(p, q) \in E$. The function defines the edges of the graph has the following rules :

For $\sigma', \sigma \in \Sigma_P$ such that $M(\sigma') = M(\sigma) = \varepsilon$

$$\begin{aligned} (q_1, q_2, q_3, normal) &\xrightarrow{0} (q_1', q_2, q_3, normal) && \text{if } q_1' \text{ is defined} \\ (q_1, q_2, q_3, normal) &\xrightarrow{0} (q_1, q_2', q_3', normal) && \text{if } q_2' \text{ and } q_3' \text{ are defined} \\ (q_1, q_2, q_3, normal) &\xrightarrow{-1} (q_1, q_2, q_3', confused) && \text{if } q_2' \text{ is not but } q_3' \text{ is defined} \\ (q_1, q_2, q_3, confused) &\xrightarrow{0} (q_1', q_2, q_3, confused) && \text{if } q_1' \text{ is defined} \\ (q_1, q_2, q_3, confused) &\xrightarrow{-1} (q_1, q_2, q_3', confused) && \text{if } q_1' \text{ is defined} \end{aligned} \tag{A.4}$$

For $\sigma', \sigma \in \Sigma_P$ such that $M(\sigma') = M(\sigma) \neq \varepsilon$

$$\begin{aligned} (q_1, q_2, q_3, normal) &\xrightarrow{0} (q_1', q_2', q_3', normal) && \text{if } q_1', q_2' \text{ and } q_3' \text{ are defined} \\ (q_1, q_2, q_3, normal) &\xrightarrow{-1} (q_1', q_2, q_3', confused) && \text{if } q_1', q_3' \text{ are defined but } q_2' \text{ is not} \\ (q_1, q_2, q_3, confused) &\xrightarrow{-1} (q_1', q_2, q_3', confused) && \text{if } q_1' \text{ and } q_3' \text{ are defined} \end{aligned} \tag{A.5}$$

Finally, the edges to Block vertex are defined as

For $\sigma', \sigma \in \Sigma_P$ such that $M(\sigma') = M(\sigma) \neq \varepsilon$

$$(q_1, q_2, q_3, confused) \xrightarrow{-1} Block \quad \text{if } q_3' \text{ is not defined} \tag{A.6}$$

It is obvious that an edge existing in the graph can have two possible values as weight candidates -1 and 0 according to the rules of the function. In case both of the candidates are available for an edge, minimum candidate, i.e. -1, is chosen as the weight of the edge. With the algorithm described recently, we have our weighted, directed graph G. From so now, we only consider the accessible part of our graph G. Now the readers must be ready for understanding the implication of directed and weighted graph G. The graph G is designed to track traces $s' \in L(N)$ and $s \in L(P)$ from the starting vertex $(q_0^N, q_0^N, q_0^P, normal)$. To be more specific, the vertex space and edge relation are defined to track the traces in the following manner:

$$\underbrace{Q^N}_{\text{s'}} \times \underbrace{Q^N \times Q^N}_{\text{s}} \times \{normal, confused\} \tag{A.7}$$

In [13] Yoo & Garcia studied a transition relation of F-verifier similar to the structure of this edge definition. The indicator set normal, confused informs us about the trace's being in normal behavior $L(P)$ or abnormal behavior $L(P) \setminus L(N)$. Since q_2' and q_3' is defined by the same event, the second term Q^N and the third term Q^P in a vertex label track the trace s simultaneously as long as $s \in L(N)$. The indicator remains at *normal* during this tracking operation. It changes to *confused* when q_2' is not defined but q_3' is defined. This situation happens when s becomes abnormal, i.e. $s \in L(P) \setminus L(N)$. Afterwards, we do not need to update the term $q_2 \in Q^N$ within a edge from vertices having indicator confused.

Moreover, it is important to note that the weight candidate -1 is assigned if and only if q_3' is defined and the vertex reached by that edge has the indicator confused. Finally, we can have an end for the implication of graph G part by a conclusion which states that the edges with weight -1 are for updating abnormal traces while the edges with 0 are for updating normal traces.

All the explanation given for this algorithm is for being able to test the given systems in terms of language diagnosability as stated at the beginning of the section. After stating the properties of the algorithm, we will give the final conclusion theorem of the algorithm and an explanatory example for the reader.

Theorem 1 *Given two automata N and P as stating normal behavior and possible*

behavior of the language respectively and the natural projection function p_o, $L(P)$ *is not language diagnosable with respect to* $L(N)$ *and* p_0 *iff there is a cycle that has an edge with negative weight or the block vertex is reachable from the start state* $(q_0^N, q_0^N, q_0^P, normal)$ *in graph* $G(N, P, M)$

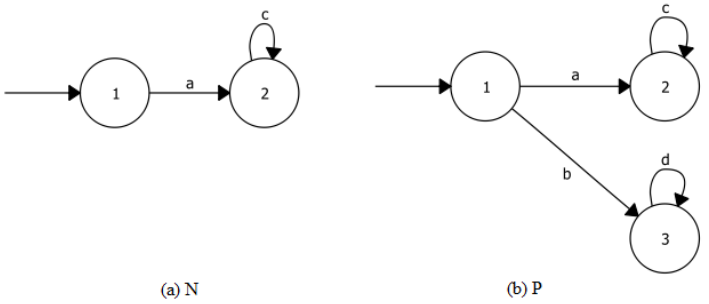


Figure A.1: Normal and possible system behavior

Let us look at Figure A.1. The normal behavior $L(N)$ of the system is generated by automaton N depicted in Figure A.1.a and the possible behavior $L(P)$ is generated by automaton P depicted in A.1.b. Note that $L(N) \subseteq L(P)$.

Our mask function will be the projection function P_1 and it is defined as :

$$P_1(a) = P_1(b) = \varepsilon \text{ and}$$

$$P_1(c) = P_1(d) = e$$

The algorithm for generating the directed, weighted graph G produces the graph G_1 in Figure A.2 with the given projection function P_1. It is easily seen that the graph has a that includes an edge with the negative weight which means that this language is not language diagnosable with respect to given specification automaton N and projection function P_1.For instance, the traces $s_1 = bd^n \in L(P) \setminus L(N)$ and $s_2 = ac^n \in L(N)$ cannot be distinguished with given specification language and the mask function since $P_1(s_1) = P_1(s_2)$ for all $n \in N$.

On the other hand, since the notion of language diagnosability depends on normal

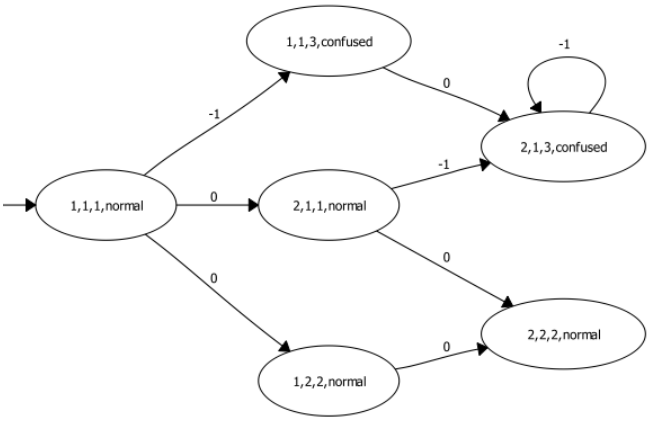


Figure A.2: $G_1(N, P, P_1)$

behavior definition, i.e. the specification automaton N, we can make the same system language diagnosable if we choose a more suitable mask function P_2 as such

$$P_2(a) = P_2(b) = P_2(c) = \varepsilon \text{ and } P_2(d) = e$$

With the same specification automaton defining the normal events in the language, the algorithm generates the weighted, directed graph G_2 in Figure A.3 associated with this masking function P_2. It is clear that the graph G_2 does not include any cycle with a negative weighted edge or blocking states which makes us draw the conclusion of the language's being language diagnosable with respect to given specification automaton N and mask function P_2.

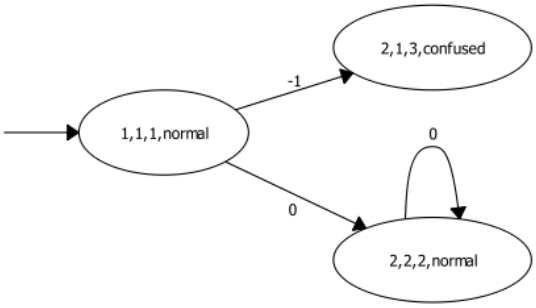


Figure A.3: $G_2(N, P, P_2)$

The proof and more explanation can be found in [11] for more interested readers.