



An Efficient Approach for Diagnosability and Diagnosis of DES Based on Labeled Petri Nets, Untimed and Timed Contexts

Baisi Liu

► To cite this version:

Baisi Liu. An Efficient Approach for Diagnosability and Diagnosis of DES Based on Labeled Petri Nets, Untimed and Timed Contexts. Signal and Image processing. Ecole Centrale de Lille, 2014. English. NNT: . tel-01064597v1

HAL Id: tel-01064597

<https://theses.hal.science/tel-01064597v1>

Submitted on 16 Sep 2014 (v1), last revised 5 Feb 2016 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre:

2	4	3
---	---	---

ÉCOLE CENTRALE DE LILLE

THÈSE

présentée en vue d'obtenir le grade de

DOCTEUR

en

Spécialité : Automatique, Génie informatique, traitement du signal et des images

par

Baisi LIU

DOCTORAT DÉLIVRÉ PAR L'ÉCOLE CENTRALE DE LILLE

Titre de la thèse :

**An Efficient Approach for Diagnosability and Diagnosis of DES
Based on Labeled Petri Nets – Untimed and Timed Contexts**

**Une approche efficace pour l'étude de la diagnosticabilité et le
diagnostic des SED modélisés par Réseaux de Petri labellisés –
contextes atemporel et temporel**

Soutenance prévue le 17 avril 2014 devant le jury d'examen :

Président

Rapporteurs	Prof. Jean-Marc FAURE	SUPMECA - LURPA
Rapporteurs	Prof. Michel COMBACAU	Université de Toulouse 3 - UPS - LAAS
Examineur	DR Joaquin RODRIGUEZ	IFSTTAR - Villeneuve d'Ascq
Examineur	MCF Alexandre PHILIPPOT	CreSTIC - Université de Reims
Directeur de thèse	Prof. Armand TOGUYÉNI	École Centrale de Lille - LAGIS
Encadrant	CR Mohamed GHAZEL	IFSTTAR - Villeneuve d'Ascq

Thèse préparée dans le Laboratoire d'Automatique, Génie Informatique et Signal

LAGIS, CNRS UMR 8219 - École Centrale de Lille

École Doctorale Sciences pour l'ingénieur ED 072

PRES Université Lille Nord de France

To my parents

Acknowledgements

This thesis reports on work performed while the author was under the supervision of Professor Armand Toguyéni and Dr. Mohamed Ghazel. The author wishes to acknowledge the financial support for this thesis by China Scholarship Council, École Centrale de Lille and IFSTTAR.

Many thanks to my supervisors Prof. Armand Toguyéni and Dr. Mohamed Ghazel for their detailed and constructive comments, encouragements, and guidance in the development of my research work. Thanks to Dr. Thomas Bourdeaud'huy for his insightful discussions about the research.

I also thank those who agreed to be the referees of this thesis and allocated their valuable time in order to evaluate the quality of this work: Prof. Jean-Marc Faure, Prof. Michel Combacau, DR Joaquin Rodriguez and MCF Alexandre Philippot for their examination of the report and their valuable comments.

I wish to express my special appreciation to Prof. Hélène Catsiapis, my French teacher. In her courses, I have learned not only the language but also the culture and history.

I also acknowledge the personnel of the laboratory LAGIS, École Centrale de Lille, École Doctorale Sciences pour l'ingénieur ED 072, the IFSTTAR institute.

Many thanks go to my friends who helped me a lot during the past three years. I mention Yongliang, Pengfei, Ramla, Rahma, Manel, Hongchang, Laurent, Ahmed, Abderaouf and many others whose names only by lack of memory I failed to include in this list.

Thanks to my parents, Lihua and Dejiang, for living in my heart and mind despite being on the other side of the ocean. Especial thanks to my girl friend Qian for her support.

Abstract

This PhD thesis deals with fault diagnosis of discrete event systems in both untimed and timed contexts using Petri net models. Some *on-the-fly* and *incremental* techniques are developed to reduce the state explosion problem while analyzing diagnosability. In the untimed context, an algebraic representation for labeled Petri nets (LPNs) is developed to feature the system behavior. The diagnosability of LPN models is tackled by analyzing a series of K -diagnosability problems, where K is increased progressively. Two models called respectively FM-graph and FM-set tree are developed and built on the fly to record the necessary information for diagnosability analysis and online diagnosis. Finally, a diagnoser is derived from the FM-set tree for online diagnosis. In the timed context, *time interval splitting* techniques are developed in order to generate a state representation of labeled time Petri net (LTPN) models, for which techniques from the untimed context can be used to analyze diagnosability and perform online diagnosis. Based on this, necessary and sufficient conditions for the diagnosability of LTPN models are determined. Moreover, we provide the solution for the minimum delay Δ that ensures diagnosability. From a practical point of view, diagnosability analysis is performed on the basis of on-the-fly building of a structure that we call ASG and which holds fault information about the LTPN states. Generally, using on-the-fly analysis and incremental techniques makes it possible to build and investigate only a part of the state space. Analysis results obtained on some chosen benchmarks show the efficiency in terms of time and memory compared with the traditional approaches based on state enumeration.

Keywords: Fault diagnosis, Discrete event systems, Labeled Petri

nets, Labeled time Petri nets, On-the-fly analysis, Incremental approach, Time interval splitting.

Résumé

Cette thèse s'intéresse à l'étude des problèmes de diagnostic des fautes sur les systèmes à événements discrets dans des contextes atemporel et temporel sur la base de modèles réseau de Petri. Des techniques d'exploration incrémentale et à-la-volée sont développées pour combattre le problème de l'explosion de l'espace d'état. Dans le contexte atemporel, une représentation algébrique pour les réseaux de Petri labellisés (RdP-L) a été développée pour caractériser le comportement du système. La diagnosticabilité de modèles RdP-L est ensuite abordée par l'analyse d'une série de problèmes d'analyse de K -diagnosticabilité, où K peut être augmenté progressivement. Concrètement, l'analyse de la diagnosticabilité est effectuée sur la base de deux modèles nommés respectivement FM-graph et FM-set tree qui sont développés à-la-volée et qui contiennent les informations relatives aux fautes. Un diagnostiqueur peut facilement être dérivé à partir du FM-set tree pour le diagnostic en ligne. Dans le contexte temporel, une technique de fractionnement des intervalles de temps a été élaborée pour développer une représentation de l'espace d'état des réseaux de Petri labellisés et temporels (RdP-LT) pour laquelle des techniques d'analyse de la diagnosticabilité du contexte atemporel, peuvent être exploitées. Sur cette base, les conditions nécessaires et suffisantes pour la diagnosticabilité de RdP-LT ont été déterminées, et nous présentons la solution pour le délai minimum Δ qui assure la diagnosticabilité. En pratique, l'analyse de la diagnosticabilité est effectuée sur la base de la construction à-la-volée d'une structure que l'on appelle ASG et qui contient des informations relatives à l'occurrence de fautes sur les états du RdP-LT. D'une manière générale, l'analyse effectuée sur la base des techniques à-la-volée et incrémentale per-

met de construire et explorer seulement une partie de l'espace d'état. Les résultats des analyses effectuées sur certains benchmarks montrent l'efficacité des techniques que nous avons développées en termes de temps et de mémoire par rapport aux approches traditionnelles basées sur l'énumération des états.

Mots clés : Diagnostic des fautes, Systèmes à événements discrets, Réseaux de Petri labellisés, Réseaux de Petri labellisés et temporels, Analyse à-la-volée, Approche incrémentale, Fractionnement d'intervalles temporels.

Contents

Contents	vii
List of Algorithms	xiii
List of Figures	xv
List of Tables	xix
1 Introduction	1
1.1 Background	2
1.1.1 Fault Diagnosis	2
1.1.2 Discrete Event Systems (DES)	3
1.1.3 Problems	5
1.2 Objectives	7
1.3 Contributions	8
1.4 Organization	9
2 Modeling Formalisms for Diagnosability Analysis of DES	11
2.1 Untimed Modeling Formalisms of DES	11
2.1.1 Automata	11
2.1.1.1 Finite State Automaton	12
2.1.1.2 Automata Language	13
2.1.1.3 Observer Automaton	14
2.1.1.4 Diagnoser Automaton	14
2.1.1.5 Diagnosability of Automata Models	15
2.1.2 Petri Nets (PNs)	17

CONTENTS

2.1.2.1	Definition of PN	18
2.1.2.2	Dynamics of PN	19
2.1.2.3	Properties of PNs	20
2.1.3	PN Language	21
2.1.4	Labeled Petri Nets (LPNs)	22
2.1.5	Diagnosability of LPNs	23
2.2	Timed Modeling Formalisms of DES	23
2.2.1	Timed Transition Systems (TTS)	24
2.2.2	Timed Automata (TA)	24
2.2.2.1	Definition of TA	25
2.2.2.2	TA Semantics	26
2.2.3	Time Petri Nets (TPNs)	27
2.2.3.1	Definition of TPN	27
2.2.3.2	Dynamics of TPN	28
2.2.3.3	TPN Semantics	28
2.2.3.4	State Class	29
2.2.4	Labeled Time Petri Nets (LTPNs)	31
2.2.4.1	Definition of LTPN	31
2.2.4.2	Timed Language for LTPNs	32
2.2.4.3	Diagnosability of LTPN	33
2.3	Hypothesis	34
2.4	Conclusion	35
3	Literature Review	37
3.1	Overview	37
3.2	Literature on Diagnosability of Discrete Event Systems (DES) . .	39
3.2.1	Classical Diagnosability	39
3.2.2	K -diagnosability of Untimed discrete event systems (DESs) .	41
3.2.3	Diagnosability and Δ -Diagnosability of timed discrete event system (TDES)	43
3.3	Literature on Diagnosis Techniques	44
3.3.1	Diagnoser Approach	44
3.3.2	Verifier (Twin plant) Approach	44

3.3.3	Decentralized/Distributed/Modular Approaches	45
3.3.4	Linear Programming Approach	45
3.3.5	Unfolding Technique	46
3.3.6	Model-Checking Based Techniques	47
3.4	Literature on Diagnosis Software Tools	48
3.5	Conclusion	49
4	Untimed PN-Based Diagnosis of DES	51
4.1	Motivation	52
4.1.1	On-the-Fly Analysis Technique	52
4.1.2	Incremental Analysis Technique	55
4.2	Mathematical Representation of LPNs to Check Diagnosability .	56
4.2.1	Extended Incidence Matrix	57
4.2.2	Event Marking	59
4.2.3	Extended State Equation	60
4.2.4	Homomorphic Structure for Event Markings: LPN with Event Counters	63
4.3	Verification of K -Diagnosability	66
4.3.1	Fault Marking (FM)	66
4.3.2	FM-Graph	68
4.3.3	FM-Set	70
4.3.4	FM-Set Tree	73
4.3.5	Conditions for K -Diagnosability	73
4.3.5.1	Equivalent Indeterminate Cycles in FM-Set Tree	73
4.3.5.2	Relation between Undiagnosability and Delay Val- ues	75
4.3.6	Algorithm for On-the-Fly Checking K -Diagnosability . . .	76
4.4	Verification of Diagnosability	82
4.4.1	Algorithm for Checking Diagnosability	82
4.4.2	Complexity and Effectiveness Analysis	84
4.5	Online Diagnosis	87
4.6	OF-PENDA Software Tool	89
4.6.1	Application to the WODES Diagnosis Benchmark	91

CONTENTS

4.6.1.1	WODES Diagnosis Benchmark	91
4.6.1.2	Comparative Simulation	91
4.6.1.3	Discussions	93
4.6.2	Application to the Level Crossing Benchmark	96
4.6.2.1	Level Crossing Benchmark	96
4.6.2.2	Comparative Simulation	97
4.6.2.3	Discussions	99
4.7	Conclusion	103
5	Time PN-Based Diagnosis of DES	105
5.1	Splitting Time Intervals Assigned to LTPN Transitions	107
5.1.1	Motivation	107
5.1.2	Semi-Interval	109
5.1.3	Basic Interval Set (BIS)	112
5.2	Reachability Analysis of LTPNs with Fault Information	118
5.2.1	Augmented State Class (ASC)	118
5.2.2	Sequence Duration	120
5.2.3	augmented state class (ASC)-Graph	122
5.2.4	ASC-Set	128
5.2.5	ASC-Set Graph (augmented state class set graph (ASG))	131
5.3	Checking Diagnosability	132
5.3.1	Conditions for Undiagnosability	133
5.3.2	On-the-Fly Checking of Diagnosability	137
5.4	Discussion on Bisimulation between ERA and ASG	138
5.5	Online Diagnosis	141
5.6	Conclusion	144
5.6.1	Summary	144
5.6.2	Perspectives	145
6	Conclusions and Perspectives	147
6.1	Conclusions	147
6.2	Perspectives	148
	Appendix A: Literature on DES-Based Diagnosis	151

CONTENTS

Appendix 2: Development of the Level Crossing Benchmark	155
References	167

CONTENTS

List of Algorithms

1	Algorithm for δ function	69
2	Checking K -diagnosability by on-the-fly building of fault marking graph (FM-graph) and fault marking set tree (FM-set tree) in parallel	76
3	Checking K -diagnosability by on-the-fly building of FM-graph and FM-set tree in parallel	78
4	NEXTFMSET(): a subfunction of Algorithm 3 for computing the next FM-set while building the FM-graph on the fly	79
5	UPDDELAY(): a subfunction of Algorithm 3 for updating delay values	79
6	Checking diagnosability	83
7	Fault detection using the diagnoser derived from FM-set tree . . .	89
8	Computation of BIS	114
9	Construction of the ASG	132
10	On-the-fly building of ASG, checking diagnosability and computing Δ_{min}	139
11	Fault detection using the LTD	143

LIST OF ALGORITHMS

List of Figures

1.1	Transient fault, intermittent fault and permanent fault	4
2.1	An example of automaton	12
2.2	The observer of the automaton in Figure 2.1	14
2.3	The diagnoser of the automaton in Figure 2.1	15
2.4	An automaton and its diagnoser without a indeterminate cycle .	17
2.5	An example of PN	20
2.6	An example of LPN	22
2.7	An example of timed automaton	25
2.8	An example of TPN	27
2.9	Relation between LTPN and PN, TPN, LPN	31
2.10	An example of LTPN	32
3.1	DES-based fault diagnosis	38
3.2	A diagnosable LPN with 2 minimal T-invariants with regard to (w.r.t) the same observable projection	41
3.3	An undiagnosable LPN with 2 minimal T-invariants w.r.t the same observable projection	41
4.1	An automaton G and its diagnoser $Diag(G)$	53
4.2	Three types of diagnosability analysis procedures	54
4.3	An unbounded Petri net (PN)	55
4.4	An LPN	58
4.5	A labeled Petri net with event counters (LPN-EC)	65

LIST OF FIGURES

4.6	The reached marking of the LPN-EC in Figure 4.5 by the firing of $t_1t_5t_9t_3t_7t_9$	66
4.7	The FM-graphs of the LPN in Figure 4.4	70
4.8	Fault propagation between fault marking sets (FM-sets)	72
4.9	Checking an indeterminate cycle	74
4.10	Updating the <i>delay</i> values while generating an FM-set tree	77
4.11	Solution process of K -diagnosability for Σ_{F_1}	81
4.12	Solution process of K -diagnosability for Σ_{F_2}	82
4.13	LPN N_L and its reachability graph G	85
4.14	The observer and diagnoser of G	86
4.15	The FM-graph and FM-set tree of N_L	86
4.16	LPN N_L	89
4.17	The FM-set tree of N_L	90
4.18	The diagnoser of N_L	90
4.19	The WODES diagnosis benchmark	92
4.20	Time cost for the diagnosability analysis on Σ_{F_1}	101
4.21	Time cost for the diagnosability analysis on Σ_{F_2}	101
5.1	The state class graph and the modified state class graph for LTPN N_{LT}	108
5.2	Time intervals in $BIS(A)$	113
5.3	The relation between β and γ'	117
5.4	The computation of BIS in Example 30	119
5.5	The LTPN graph for Example 30	121
5.6	The state classes following σ for Example 30	121
5.7	The LTPN graph for Example 32	123
5.8	The state graph of the LTPN in Figure 5.7	124
5.9	The construction of augmented state class graph (ASC-graph)	125
5.9	The construction of ASC-graph (Continued)	126
5.9	The construction of ASC-graph (Continued)	127
5.9	The construction of ASC-graph (Continued)	128
5.10	Computation of the reachable augmented state class set (ASC-set) of the initial ASC-set of LTPN in Figure 5.7	130

LIST OF FIGURES

5.11	The ASG for Example 35	133
5.12	Illustration of indeterminate cycle	134
5.13	A language equivalent ERA for the ASG in Figure 5.11	142
5.14	Diagnoser of the LTPN in Figure 5.7	144
5.15	The modified labeled timed diagnoser (MLTD) of the LTPN in Figure 5.7	145
1	The construction of a single-line track level crossing (LC) system .	156
2	The LPN model for a train passing an LC	157
3	The LPN model for LC controller	159
4	The PN model for an interlock	159
5	The LPN model for a barrier system	160
6	A single-line LC	161
7	A single-line LC with two classes of faults	162
8	n -line LC benchmark	164

LIST OF FIGURES

List of Tables

4.1	Event markings in Example 16	63
4.2	Fault markings corresponding to Σ_{F_1} in Example 19	68
4.3	Fault markings in Example 20	71
4.4	Definition of <i>delay</i> function for newly generated node	75
4.5	The markings in Figure 4.13(b)	85
4.6	The markings in Figure 4.15	87
4.7	Simulation results	94
4.8	Comparative simulation results based on n -line LC benchmark . .	100
5.1	Solution of $BIS(A)$ in Example 30	118
1	Literature on DES-based diagnosis	151
2	Some figures about the state space of the various LC models . . .	166

LIST OF TABLES

Chapter 1

Introduction

This thesis deals with Petri-net-based techniques for fault diagnosis of discrete event systems. The work is accomplished in the Fault Tolerant Systems' team (STF - Systèmes Tolérants aux Fautes) of the LAGIS laboratory (Laboratoire d'Automatique, Génie Informatique et Signal, UMR CNRS 8219), at the École Centrale de Lille (EC-Lille), in collaboration with the COSYS/ESTAS (Composants et systèmes / Évaluation des systèmes de transports automatisés et de leur sécurité) research team at IFSTTAR (Institut français des sciences et technologies des transports, de l'aménagement et des réseaux). This thesis was supervised by Prof. Armand TOGUYÉNI and Dr. Mohamed GHAZEL, senior researcher with IFSTTAR - COSYS/ESTAS.

The current chapter provides an overview of this thesis. Section 1.1 introduces the background of fault diagnosis problem in the context of discrete event systems. Section 1.2 presents our research objectives of this thesis. In Section 1.3, we state the contributions of this thesis. Finally, we give the organization of this thesis in Section 1.4. Note that the general hypothesis will be given in Section 2.3, after the introduction of some basic notations.

1. INTRODUCTION

1.1 Background

1.1.1 Fault Diagnosis

Systems around us are more functional as their structures are more complex. For complex and critical systems, e.g., aerospace systems, military systems, transportation systems, power systems, manufacturing and production systems, etc., the safe and continuous operation is imperative; therefore it is necessary to let any abnormal behavior be detected and identified as soon as possible, so that re-configurations can be made to prevent the system from dangerous consequences [Lin, 1994]. Fault diagnosis is such a field dealing with detecting any fault and its type, which ensures the safety and availability of systems.

Reviewing the systems mentioned above, we see that they have the following features:

1. *Large-scale and complex systems.* In order to fulfill complex functions, some system structures are designed to be larger and more complex, even though advanced design ideas have been used [Morel et al., 2007]. Complex structures also make potential failures more difficult to be detected and located, since the failure may be long-term underlying anywhere in the system.
2. *Safety-critical.* In safety-critical systems, a failure may cause serious material and/or human damages. In this context, it is important to ensure that any fault occurred is detected as soon as possible. Moreover, this issue should be well considered since the design stage.
3. *Continuous operation.* For large systems, the start-up, shutdown and alternation of operations often consume a lot of time and energy, thus they are expected and designed to offer high availability. From a practical point of view, it is not advisable to systematically halt or stop the system to make a thoroughly examination. This means the diagnostic procedure has to be performed online and needs to be efficient.

Now let us look at the features of faults. A fault results in a non-desired deviation of the system or of one of its components from its normal or intended

behavior. In fault diagnosis, a fault that we study on is often such an “*unobservable*”, “*undistinguishable*” or “*silent*” event (or state), which is difficult to be detected or distinguished directly, since an “*observable*” fault can be always detected by a direct sensor reading.

According to their lasting in time, faults are classified into the following three categories [D’Angelo et al., 1999; Dugan and Trivedi, 1989; Nelson, 1990], as shown in Figure 1.1:

1. A *transient* fault, which is often of finite duration shorter than the stability time interval of the affected signal, occurs only once and then disappears. It may be the result of external causes like interferences. Generally, transient faults are the most common, and they are also hard to detect since they may disappear after having produced errors.
2. An *intermittent* fault occurs and disappears repeatedly, making the system vibrate between normal and faulty states. It can be due to an unstable device operation.
3. A *permanent* fault occurs but does not disappear (such that the system remains in faulty state) until repairing measures are undertaken. Typically a permanent fault is caused by subsystem failures, physical damage or design error.

Diagnosis of transient [Schiller et al., 2001], intermittent [Contant et al., 2004; Jiang et al., 2003a; Soldani et al., 2007] and permanent faults have been studied. In this thesis, we only consider the diagnostic problems of permanent faults and a series of works on diagnosis of permanent faults will be reviewed in Section 3.

1.1.2 Discrete Event Systems (DES)

The diagnostic techniques we deal with in the framework of this thesis are model-based, which means that one has a behavioral model which depicts the system dynamics. Moreover, we assume that diagnosis analysis can be undertaken in an abstraction level such that the system behavior can be abstracted as a discrete event model [Lin, 1994].

1. INTRODUCTION

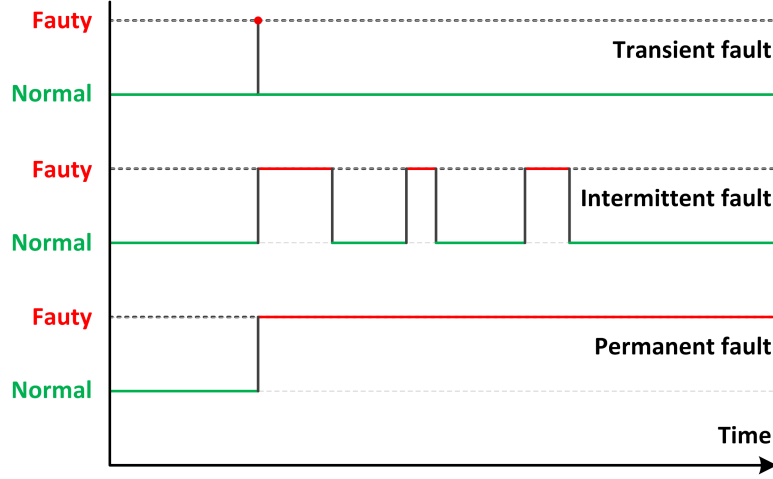


Figure 1.1: Transient fault, intermittent fault and permanent fault

A *DES* [Cassandras and Lafortune, 2007] is informally a *discrete-state* and *event-driven* system. More specifically, the state space of a DES is a *discrete* set, e.g., a traffic light has three states: RED, YELLOW and GREEN; a push button has two states: ON and OFF. The state transition mechanism of a DES is *event-driven*, i.e., any state change is driven by an event execution, e.g., the state of a traffic light changes from RED to GREEN after a permission signal is sent; the state of a push button changes from OFF to ON after being pressed.

We study the problem of fault diagnosis in the context of DES, since most industrial systems can be abstracted as a DES model to certain level of abstraction [Lin, 1994; Sampath et al., 1995], which can be untimed, timed or stochastic. Note that so many systems around us are *Computer-Controlled Systems*, whose state space and event-driven mechanism are based on binary logic, can be naturally treated as DES. This is why the control and diagnosis of DES has been extensively studied in recent years, as will be shown in the sequel.

In order to characterize DES, different modeling notations have been developed [Cassandras and Lafortune, 2007]: language and automata, PN theory, $(\max, +)$ algebra, Markov chains and queuing theory, discrete-event simulation, perturbation analysis, and concurrent estimation techniques. Among them, automata and PNs are the two most used models in DES-based diagnosis. In this thesis, we study the DES-based diagnosis both in untimed and timed context,

using the LPN and LTPN notation.

1.1.3 Problems

Fault diagnosis has been widely studied. Most existing methods can be classified as follows: fault-tree based methods, quantitative, analytical model-based methods, expert systems and other knowledge-based methods, model-based reasoning methods, and DES-based methods.

During the two past decades, DES-based methods have received a lot of attention in industry and academia, since more and more systems are controlled by computers, and most of them can be abstracted as DES models to a certain abstraction degree.

DES-based diagnosis is first studied in the framework of automata [Sampath et al., 1995], which brings the problem of “state explosion”. Afterwards, some research turns to the PN-based technique [Ushio et al., 1998], since PNs can provide solutions based on not only state enumeration but also structure analysis.

At this stage, we would like to point out that some notions pertaining to diagnosis issues can have various meaning according to whether they are used by continuous automatic community or discrete automatic community, e.g., detection, identification, localization, etc. Let us recall here that when using these concepts, we refer to the definitions as conventional within the discrete automatic community.

For DES-based diagnosis, we summarize the main issues as follows:

1. *Diagnosability* [Jiang et al., 2001; Lin, 1994; Sampath et al., 1995; Yoo and Lafortune, 2002a]. Informally, diagnosability refers to the ability to detect and identify any fault within a finite delay after its occurrence. The diagnosability is the basis of diagnosis, i.e., a fault can be diagnosed only if the system is diagnosable w.r.t this fault class. Diagnosability is a property that is analyzed offline, and most of the existing approaches use enumerative techniques to investigate diagnosability.
2. *K-diagnosability*. As an extension of diagnosability, *K*-diagnosability requires that a fault can be diagnosed in a given number of steps after its

1. INTRODUCTION

occurrence. This topic is studied as K -diagnosability in untimed context [Basile et al., 2010; Cabasino et al., 2012a; Yoo and Garcia, 2004] and as Δ -diagnosability in timed context [Tripakis, 2002], as will be presented in details in this dissertation. Note that, from different points of view, there are also some other extensions of diagnosability, which will be discussed in Chapter 3.

3. *Online diagnosis.* Diagnosis process, strictly speaking, refers to the online monitoring of the system which aims at detecting any abnormal behavior and identifying the possible faults behind that. Moreover, detection and identification need to be performed promptly and generally with as less interferences with the system as possible. Hence, the objective of online diagnosis is passively diagnosing a fault, without actively interrupting or alternating the operation of systems.
4. *Active diagnosis.* Compared with passive diagnosis wherein the objective is to observe system behavior and give verdict of potential faults, active diagnosis is an integrated approach to control and diagnosis [Sampath et al., 1998]:
 - In the early period, the design of systems and the development of diagnostic tools are decoupled. Fault diagnosis could not be well performed this way, since the system may be undiagnosable. Even if the system in question is diagnosable, the design and development of the corresponding diagnostic tool may be difficult, due to the practical limitations of space, environment, etc. The active diagnosis in the sense of [Sampath et al., 1998] aims at developing a diagnosable system by early integrating diagnosability analysis since the design phase. In distributed systems, [Ribot et al., 2007] determines the characteristics and the modifications which could be useful for designers to improve and to guarantee some diagnosability objectives.
 - For existing systems, one can turn certain sensors ON or OFF as necessary, such that the system can be monitored with a dynamic observer whose set of observable events varies according to external commands.

Diagnosis based on dynamic observers permits achieving cost savings [Cassez and Tripakis, 2008], since certain sensors only operate as necessary.

5. *Enhancement of diagnosability* [Wen et al., 2006]. Although the active diagnosis is a trend on the system design, existing undiagnosable systems have to be treated in another way. It is possible to let these undiagnosable systems become diagnosable, by the change of the number, type and placement of sensors. This has been further studied as the problem of sensor optimization [Cabasino et al., 2013b; Debouk et al., 2002; Jiang et al., 2003b; Ru and Hadjicostis, 2010].

Besides, there are also many other directions on diagnosis study, such as meta-diagnosis [Belard et al., 2011], robust diagnosis [Carvalho et al., 2012], etc. However, in this thesis, we will only deal with the first three problems, aiming at developing new techniques to improve the efficiency of fault diagnosis analysis on the basis of existing approaches. In the future, we are interested in active diagnosis and sensor optimization issues.

1.2 Objectives

As will be discussed in Chapter 3, some classic approaches [Cabasino et al., 2012a; Jiang et al., 2001; Sampath et al., 1995; Yoo and Lafortune, 2002b] for diagnosis analysis are based on *a priori* built state space, which suffer from the inherent state explosion problem. However, these works have developed necessary and sufficient conditions for diagnosability, and the corresponding results have been proved in a formal way. Our goal here is to develop new techniques, namely on-the-fly and incremental techniques, which are able to tackle the state explosion problem in the untimed context on the basis of existing proved results. In particular, in the timed context, we intend to develop a formulation of the diagnosability issue, with the help of the time interval splitting technique, in such a way to bring the techniques of the untimed context into play.

1.3 Contributions

This work will focus on fault diagnosis using the PN modeling formalism. The contributions discussed mainly in Chapters 4 and 5 are summarized as follows, while separating contributions in the untimed context and those dealing with timed diagnosis.

1. Fault diagnosis of untimed DESs:

- (a) Algebraic reformulation of diagnosability analysis:

The structure of a PN as well as its dynamics can be thoroughly described by classical mathematical representation with the help of markings, incidence matrix and state equation. This formulation, however, is not sufficient for featuring LPN, as there is no characterization of relations between transitions and events. To cope with the above shortcoming, we propose a novel mathematical representation for LPNs, based on some new notions that will be introduced, namely extended incidence matrix, event marking and extended state equation, to both make explicit the mapping relationship between transitions and events and record event occurrences.

- (b) On-the-fly and incremental analysis of K -diagnosability:

Based on several new concepts that we introduce, a tree-like structure holding both the markings and their related fault information is elaborated. This structure, called FM-set tree, is computed on the fly while checking K -diagnosability on the basis of a recursive algorithm we propose. Thanks to the on-the-fly investigation of diagnosability, building the whole FM-set tree is not necessarily required. This is a notable advantage compared with the existing approaches which first build the reachability graph [Ushio et al., 1998].

- (c) On-the-fly analysis of classical diagnosability based on K -diagnosability:

By extension, we solve the classical diagnosability by dealing with a series of K -diagnosability problems, where K increases progressively. Compared the existing approaches based on a marking graph or on a

diagnoser, generally only a part of the state space is generated and searched.

(d) Online diagnosis:

When the system is diagnosable (or K -diagnosable), the online diagnosis is performed on the basis of a diagnoser which is obtained from the FM-set tree in a straightforward way.

(e) Development of a software tool for K -diagnosability and diagnosability analysis:

In order to show the effectiveness of our method, we develop a diagnostic tool called On-the-Fly PEtri-Net-based Diagnosability Analyser (OF-PENDA), and compare our approach with other existing ones with the help of the Workshop on Discrete Event Systems (WODES) diagnostic benchmark [Giua, 2007] and our developed LC benchmark. We thus show that some big models are tractable using the on-the-fly technique, whereas some existing approaches fail to analyze them, due to memory limitations.

2. Fault diagnosis of TDES:

- (a) We deal with the diagnosability of timed discrete event systems. The model we use is the LTPN - an extension of time Petri net (TPN), wherein each transition is associated with an event which can be either observable or unobservable. We propose an approach to check diagnosability and provide the solution for the minimum delay Δ that ensures diagnosability. Diagnosability analysis is performed on the basis of on-the-fly building of a structure that we call ASG and which carries information about the state of the LTPN.
- (b) We develop a labeled timed diagnoser (LTD) for online diagnosis of LTPNs, on the basis of the developed ASG.

1.4 Organization

This manuscript is organized as follows:

1. INTRODUCTION

- In Chapter 2, we review some formalisms of DES and TDES, i.e. automata, Petri nets, timed automata and time Petri nets. In particular, we present two extensions of Petri nets that will be used in this thesis: LPNs and LTPNs.
- In Chapter 3, we review the literature on DES-based diagnosis. The existing works will be classified from different points of view.
- In Chapter 4, we discuss the problem of fault diagnosis of untimed DES. First, we introduce our mathematical representation of LPNs. Then we provide the algorithms to check K -diagnosability and classical diagnosability. We show how to perform online diagnosis of LPN models. Finally, we developed a diagnostic tool called OF-PENDA and perform a simulation with the help of WODES benchmark and the LC benchmark, to show the effectiveness of our technique.
- In Chapter 5, we study the fault diagnosis problem of TDES. We introduce a technique based on time interval splitting to reformulate the diagnosis problem of TDES in such a way to make it possible to apply the classical diagnosability analysis technique of the untimed context. Some notations are developed to characterize the features of LTPNs. We propose necessary and sufficient conditions for the diagnosability of TDES, and provide algorithms to check Δ -diagnosability and diagnosability for TDES.
- In Chapter 6, we give the concluding remarks and draw some future works.

Chapter 2

Modeling Formalisms for Diagnosability Analysis of DES

In order to discuss the fault diagnosis problem of DESs, this chapter reviews some main modeling formalisms of untimed DESs: automata, PNs and LPNs; and of TDESs: Timed automata, TPNs and LTPNs. Also, we address the diagnosability of DESs and TDESs while using these above notations.

2.1 Untimed Modeling Formalisms of DES

In the fault diagnosis field, DES-based methodology has been widely investigated and applied for high level analysis, since most considered systems can be abstracted as a DES model to a certain degree [Cassandras and Lafortune, 2007; Lin, 1994], using techniques such as language and automata, Petri net theory, $(\max, +)$ algebra, Markov chains and queuing theory, perturbation analysis, concurrent estimation, etc. In our study, we discuss fault diagnosis of DESs while considering automata and PNs models of the analyzed system.

2.1.1 Automata

An automaton is a graphical structure to describe state space and state transitions of a DES. In particular, one type of automata, called finite state automaton (or finite state machine) is very useful in the analysis of finite-state DESs.

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

2.1.1.1 Finite State Automaton

Definition 1 [*Cassandras and Lafortune, 2007*] A finite state automaton (FSA) is a 4-tuple $G = (X, \Sigma, \delta, x_0)$, where:

- X is a finite set of states;
- Σ is a finite set of events;
- $\delta : X \times \Sigma \rightarrow X$ is the partial transition function;
- x_0 is the initial state of the system.

An automaton can be represented by a graph, where a state is denoted by a circle, an event is denoted by an arrow from a source state to a destination state, the initial state is denoted by a circle with an arrow into it and a final state is denoted by a double circle.

Example 1 Let us look at automaton $G = (X, \Sigma, \delta, x_0)$, as shown in Figure 2.1,

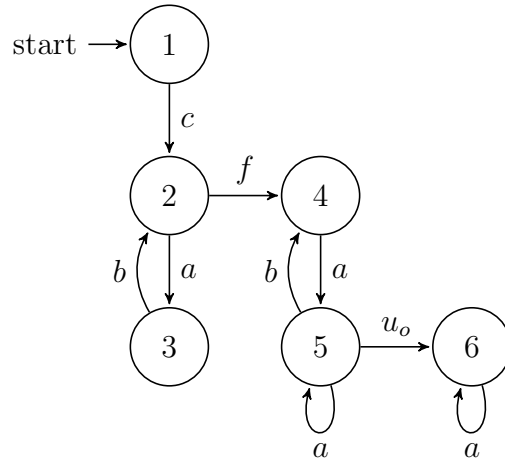


Figure 2.1: An example of automaton

- The set of states is $X = \{1, 2, 3, 4, 5, 6\}$;
- The set of events is $\Sigma = \{a, b, c, f, u_o\}$;
- The initial state is 1;
- $\delta(1, c) = 2$ and $\delta(5, a) = 5$.

2.1.1.2 Automata Language

The behavior of the system is described by the *prefix-closed language* $\mathcal{L}(G)$ generated by G . Henceforth, we shall denote $\mathcal{L}(G)$ by \mathcal{L} . \mathcal{L} is a subset of Σ^* , where Σ^* is the *Kleene closure* of set Σ .

In order to deal with the diagnosis problem of DESs in the framework of automata, a DES is first abstracted as an automaton model including both normal and faulty behavior. The set of events Σ is partitioned in two disjoint subsets as $\Sigma = \Sigma_o \uplus \Sigma_u$, where Σ_o is the set of observable events and Σ_u is the set of unobservable events. Let Σ_f denote the set of faulty events which are to be diagnosed. We assume that $\Sigma_f \subseteq \Sigma_u$, since it is straightforward to diagnose an observable faulty event. The set of fault events is partitioned into m disjoint subsets that represent the set of fault classes:

$$\Sigma_f = \Sigma_{F_1} \uplus \Sigma_{F_2} \uplus \cdots \uplus \Sigma_{F_m}$$

and this partition can be denoted by Π_f .

Let us define the projection operator $P_o : \Sigma^* \rightarrow \Sigma_o^*$ as

$$\begin{cases} P_o(\epsilon) = \epsilon \\ P_o(e) = e & \text{if } e \in \Sigma_o \\ P_o(e) = \epsilon & \text{if } e \in \Sigma_u \\ P_o(se) = P_o(s)P_o(e) & \text{with } s \in \Sigma^*, e \in \Sigma \end{cases} \quad (2.1)$$

In other words, given a sequence of events s , $P_o(s)$ filters all the unobservable and empty events, leaving a new sequence consisting of only the observable events in s .

The postlanguage of \mathcal{L} after s , denoted by \mathcal{L}/s , is defined by:

$$\mathcal{L}/s = \{s' \in \Sigma^* \mid ss' \in \mathcal{L}\}$$

Example 2 For automaton G in Figure 2.1, assume that $\Sigma_o = \{a, b, c\}$ and $\Sigma_u = \{u_o, f\}$. Given $s \in \mathcal{L}(G)$, $s = cfabau_oa$, $P_o(s) = cabaa$.

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

2.1.1.3 Observer Automaton

We say a system is “deterministic” in the sense that the next state after the occurrence of an event is unique. In this context, an automaton as introduced in Section 2.1.1.1 is not necessarily deterministic, since the function δ permits transitions from a state to two different states upon the same event. Moreover, ϵ -transitions are also allowed. However, a nondeterministic automaton G can be always transformed into a deterministic one $Obs(G)$, which is called observer automaton. This structure generates and marks the same languages as the original nondeterministic automaton. The algorithm of building an observer from a nondeterministic automaton is introduced in [Cassandras and Lafortune, 2007].

In other terms, an observer state is obtained by regrouping the automata states that are reached from a given observer state right after the occurrence of the same observable event. This provides a structure to estimate all possible states after the occurrence of a sequence of observable events. An observer automaton can be used as a basis to fault diagnosis.

Example 3 For the automaton in Figure 2.1, assume that the set of observable events is $\Sigma_o = \{a, b, c\}$, the set of unobservable events is $\Sigma_u = \{f, u_o\}$, the observer is built in Figure 2.2.

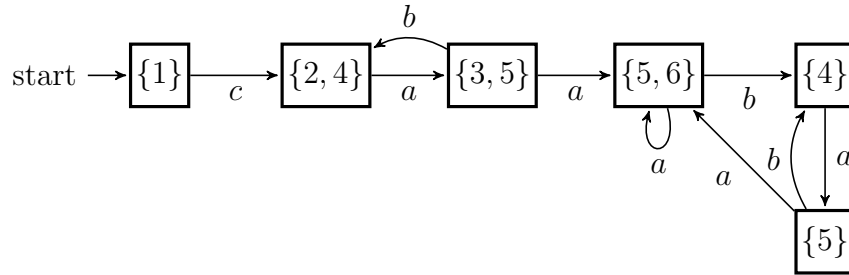


Figure 2.2: The observer of the automaton in Figure 2.1

2.1.1.4 Diagnoser Automaton

Diagnosis is the process consisting to assign to each observed string of events a diagnosis verdict, such as “normal”, “faulty” or “uncertain”. The uncertainty may be reduced by further observations. For G , a plant modeled by an automaton, this

inference can be done with the help of a diagnoser automaton called $Diag(G)$. A diagnoser automaton is actually a special observer such that each state is a subset of $X \times \{N, Y\}$, where N denotes that the state is reached after a sequence of events without fault, and Y denotes that the state is reached after a sequence of events holding a fault.

Given a state x of an automaton G having an entering observable transition, if it can be reached by two paths having the same observable projection and such that one path contains the fault f and the other one does not, then there will exist two pairs (x, N) and (x, F) in the states of $Diag(G)$.

This also means that the cardinality of $Diag(G)$ is always greater than or equal to the cardinality of $Obs(G)$.

Example 4 For the automaton in Figure 2.1, assume that the set of unobservable fault events is $\Sigma_f = \{f\}$, the diagnoser automaton is given in Figure 2.3.

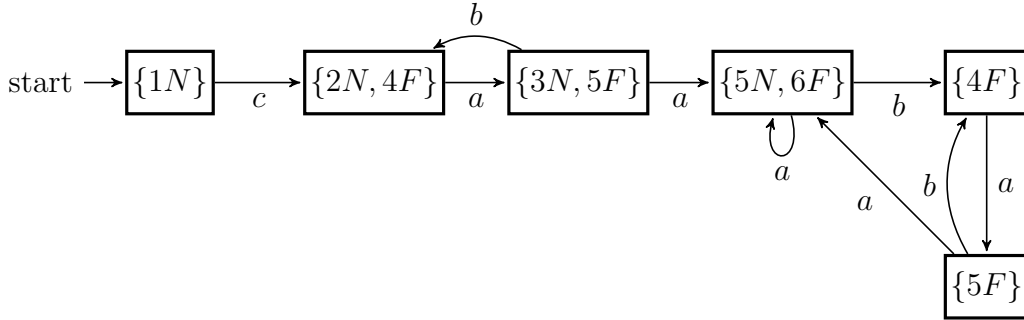


Figure 2.3: The diagnoser of the automaton in Figure 2.1

2.1.1.5 Diagnosability of Automata Models

The problem of diagnosability is to determine if the system is diagnosable, i.e., if once a fault has occurred, it can be detected and identified in a finite number of steps.

Definition 2 [Sampath et al., 1995] A prefix-closed and live language \mathcal{L} is said to be diagnosable w.r.t the projection P_o and w.r.t the partition Π_f on Σ_f if the following holds:

$$(\forall i \in \Pi_f)(\exists n_i \in \mathbb{N})[\forall s \in \Psi(\Sigma_{F_i})](\forall r \in \mathcal{L}/s)[|r| \geq n_i \Rightarrow D]$$

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

where:

- $\Psi(\Sigma_{F_i})$ denotes the set of all traces of \mathcal{L} that end in a faulty event belonging to the class Σ_{F_i} ;
- $|r|$ denotes the number of events in trace r ;
- the diagnosability condition D is

$$\omega \in P_o^{-1}[P_o(st)] \Rightarrow \Sigma_{F_i} \in \omega$$

In other words, diagnosability requires that each fault event leads to distinct observations, sufficient to allow the identification of the fault with a finite delay.

Let us now introduce the definition of indeterminate cycle that is fundamental to test the property of diagnosability in the diagnoser [Sampath et al., 1995].

Let us consider a system G and its diagnoser $Diag(G)$. We say that a cycle in $Diag(G)$ is an indeterminate cycle if it is composed exclusively of uncertain states for which there exist:

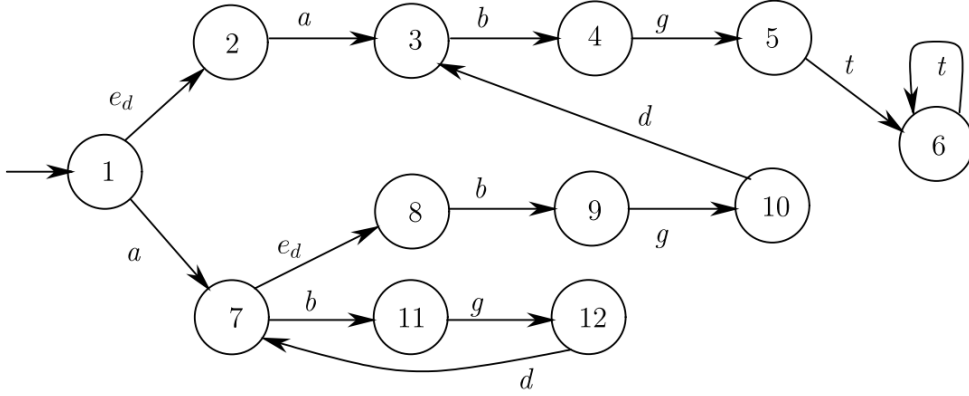
- a corresponding cycle in G involving only states that carry Y in their labels in the cycle in $Diag(G)$ and
- a corresponding cycle in G involving only states that carry N in their labels in the cycle in $Diag(G)$.

The notion of indeterminate cycle is very important because their analysis gives us necessary and sufficient conditions for diagnosability and gives a method to testify the property of diagnosability of the system.

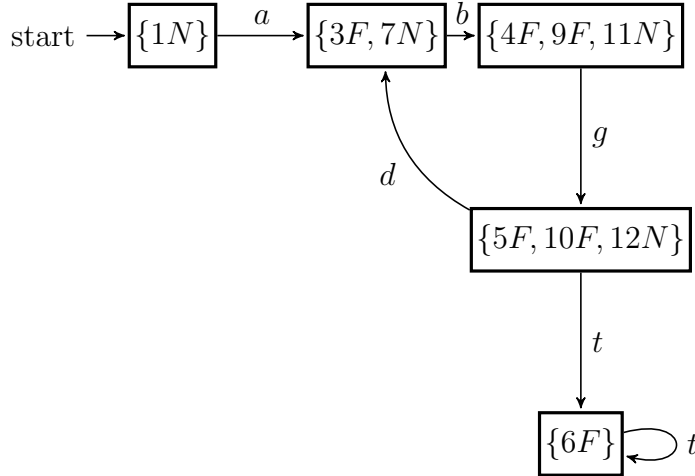
Proposition 1 [Sampath et al., 1995] *A language \mathcal{L} without multiple failures of the same type is diagnosable if and only if its diagnoser $Diag(G)$ has no indeterminate cycles w.r.t. each failure type Σ_{F_i} .*

It is important to emphasize that the presence of a cycle of uncertain states in the diagnoser does not necessarily imply undiagnosability. For example, in the diagnoser (cf. Figure 2.4(b)) of the automaton in Figure 2.4(a), there exists a cycle of uncertain states composed of $\{3F, 7N\}$, $\{4F, 9F, 11N\}$ and $\{5F, 10F, 12N\}$

w.r.t a feasible sequence $(bgd)^*$. Actually, this cycle is not an indeterminate cycle to imply undiagnosability, since there only exists a corresponding cycle of normal states (7,11 and 12) but no corresponding cycle of faulty states from $\{3, 4, 5, 9, 10\}$.



(a) An example of automaton [Cassandras and Lafortune, 2007]



(b) The diagnoser of the automaton in 2.4(a)

Figure 2.4: An automaton and its diagnoser without an indeterminate cycle

2.1.2 Petri Nets (PNs)

In this dissertation, we use PN (and their extensions) rather than automata to model DESs, since PN have the following advantages [Giua et al., 2007]:

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

- PNs give a graphical and mathematical representation of DESs. This helps to solve problems either by the development and the analysis of graphical structures or by mathematical calculations.
- PNs can well present concurrent processes, which is also one of the original objectives to develop such a notation.
- PNs models are quite convenient for composition and decomposition operations. Compared with automata which is a state transition graph, PNs describe more directly the natural structure of systems, including the relation between components and the distribution of resources in the system. In other words, the process of decomposing a modular system modeled by PN is more intuitive.
- Thanks to the mechanism of representing states with the distribution of tokens in places, a PN with a finite structure (with finite number of places and transitions) can represent an infinite state space.

Here, we review some basics of PNs. For details, please refer to [Murata, 1989].

2.1.2.1 Definition of PN

Petri nets, developed by Carl Adam Petri in the early 1960s, also named *Place/Transition nets* or *P/T nets*, are a graphical and mathematical modeling notation for DESs.

Definition 3 [Petri, 1962] *A PN is a tuple $N = (P, T, Pre, Post)$, where:*

- *P is a finite set of places (represented by circles in a PN graph);*
- *T is a finite set of transitions (represented by boxes or bars in a PN graph);*
- *$Pre: P \times T \rightarrow \mathbb{N}$ is the pre-incidence mapping that gives the arcs linking places to transitions in the net, as well as their corresponding weight;*
- *$Post: P \times T \rightarrow \mathbb{N}$ is the post-incidence mapping that gives the arcs linking transitions to places in the net, as well as their corresponding weight.*

A state of a PN is called “Marking”, presented by a distribution of tokens (dots inside the places of the PN graph) in the places of the net. A *marking* is a vector $M \in \mathbb{N}^{|P|}$ that assigns a non-negative integer to each place. We denote by \mathcal{M} the set of reachable markings.

A *marked PN* (N, M_0) is a PN N with an initial marking M_0 . For simplicity, we will use the term “PN” to refer to “marked PN” afterwards.

2.1.2.2 Dynamics of PN

The dynamics of a PN corresponds to a movement or redistribution of tokens according to some firing rules. A transition t_i is *enabled* at marking M if $M \geq \text{Pre}(\cdot, t_i)$, denoted by $M \llbracket t_i >$.

We denote by $\text{En}(M)$ the set of enabled transitions at M . Formally,

$$\text{En}(M) = \{t \mid t \in T, M \geq \text{Pre}(\cdot, t)\}. \quad (2.2)$$

A transition t_i enabled at a marking M can fire (here t_i is also said to be *firable*), yielding to a marking

$$M' = M + C \cdot \vec{t}_i \quad (2.3)$$

where $\vec{t}_i \in \{0, 1\}^{|T|}$ is a vector in which only the entry associated with transition t_i is equal to 1, and

$$C = \text{Post} - \text{Pre} \quad (2.4)$$

is called the *incidence matrix*.

Marking M' is then said to be *reachable* from marking M by firing transition t_i , also denoted by $M \llbracket t_i > M'$.

A sequence of transitions $\sigma = t_1 t_2 \dots t_k$ is *executable* (or *achievable*) at marking M , if $M \llbracket t_1 > M_1 \llbracket t_2 > \dots M_{k-1} \llbracket t_k >$, and we write it as $M \llbracket \sigma >$. The reached marking M' is computed by

$$M' = M + C \cdot \pi(\sigma) \quad (2.5)$$

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

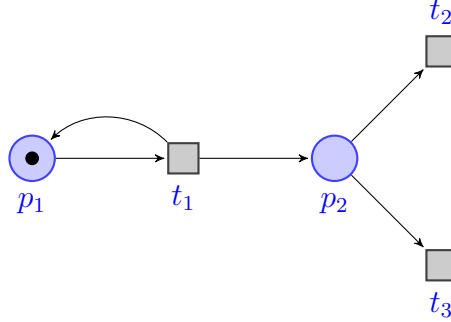


Figure 2.5: An example of PN

which is called *state equation* and denoted by $M [\sigma > M'$, where

$$\pi(\sigma) = \sum_{i=1}^k \vec{t}_i \quad (2.6)$$

is the *firing vector* relative to σ .

2.1.2.3 Properties of PNs

A PN (N, M_0) is said to be *bounded* if the number of tokens in each place does not exceed a finite number $m \in \mathbb{N}$, for any marking reachable from M_0 .

A PN (N, M_0) is said to be *live* if, no matter what marking has been reached from M_0 , it is possible to ultimately fire any transition of the net by progressing through some further firing sequence.

x is an *T-invariant* iff there exist a firing sequence σ and a marking M such that $M_0 [\sigma > M$ and $\pi(\sigma) = x$.

A Petri net is *acyclic* if there is no direct circuit of transitions in the graph.

Example 5 Consider the example of PN in Figure 3.3.

- The set of places is $P = \{p_1, p_2\}$;
- The set of transitions is $T = \{t_1, t_2, t_3\}$;

-
- *The initial marking is:*

$$M_0 = \begin{matrix} p_1 \\ p_2 \end{matrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix};$$

- *The pre-incidence mapping is:*

$$Pre = \begin{matrix} p_1 \\ p_2 \end{matrix} \begin{matrix} t_1 & t_2 & t_3 \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix} \end{matrix};$$

- *The post-incidence mapping is:*

$$Post = \begin{matrix} p_1 \\ p_2 \end{matrix} \begin{matrix} t_1 & t_2 & t_3 \\ \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix};$$

- *The incidence matrix is:*

$$C = Post - Pre = \begin{matrix} p_1 \\ p_2 \end{matrix} \begin{matrix} t_1 & t_2 & t_3 \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & -1 \end{bmatrix} \end{matrix};$$

- $En(M_0) = \{t_1\}$ *is the set of enabled transitions at marking M_0 ;*
- *The PN is without T-invariant, bounded, unlive and not acyclic.*

2.1.3 PN Language

A language over an event set Σ is a set of strings (or traces) formed from events in Σ . In order to represent a language by a PN, each transition of the PN is associated with an event by a labeling function. Hence, we speak about labeled Petri net (LPN).

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

2.1.4 Labeled Petri Nets (LPNs)

A *labeled Petri net (LPN)* is a quadruple $N_L = (N, M_0, \Sigma, \varphi)$, where

- (N, M_0) is a marked PN N with an initial marking M_0 ;
- Σ is a finite set of events for transition labeling;
- $\varphi: T \rightarrow \Sigma$ is the *transition labeling function*, φ is also extended to sequences of transitions, $\varphi: T^* \rightarrow \Sigma^*$.

We also define the inverse mapping of φ by $\varphi^{-1}: \Sigma \rightarrow 2^T$:

$$\varphi^{-1}(e) = \{t \mid t \in T, \varphi(t) = e\}$$

A LPN graph is presented as a PN graph in which each transition is labeled by an event in Σ .

The *language* generated by LPN N_L is

$$\mathcal{L}(N_L) = \{\varphi(\sigma) \in \Sigma^* \mid \sigma \in T^*, M_0 [\sigma >\}$$

where mapping φ is extended to transition sequences.

Example 6 Let us consider the example of LPN $N_L = (N, M_0, \Sigma, \varphi)$ in Figure 2.6.

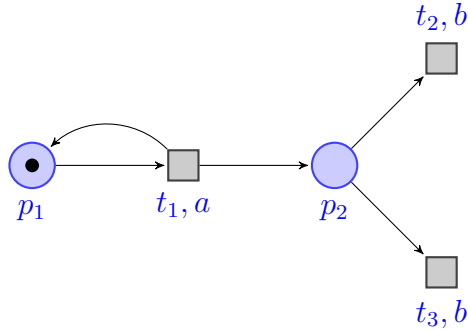


Figure 2.6: An example of LPN

- $\Sigma = \{a, b\}$ is the set of events;

-
- φ is the labeling function such that $\varphi(t_1) = a$, $\varphi(t_2) = \varphi(t_3) = b$, $\varphi^{-1}(a) = \{t_1\}$, $\varphi^{-1}(b) = \{t_2, t_3\}$.

2.1.5 Diagnosability of LPNs

As we have mentioned in Section 2.1.1.2, event set Σ is partitioned into two disjoint sets, i.e., $\Sigma = \Sigma_o \uplus \Sigma_u$ and the set of fault events is a subset of Σ_u ($\Sigma_f \subseteq \Sigma_u$). Accordingly, the set of transitions of a LPN is partitioned into the sets of observable and unobservable transitions,

$$T = T_o \uplus T_u,$$

and the set of faulty transitions is a subset of T_u ($T_f \subseteq T_u$).

We now give the definition of diagnosability of LPNs.

Definition 4 (*K-diagnosability of LPNs*) [*Liu et al., 2013*] Given a LPN N_L , $\forall e \in \Sigma_f$, e is diagnosable if $\forall u \in \mathcal{L}$, $u^{|u|} \in \Sigma_f$, $u^j \notin \Sigma_f \ \forall 1 \leq j \leq |u| - 1$ and $\forall v \in \mathcal{L}/u$, $\exists K \in \mathbb{N}$ such that if $|P_o(v)| \geq K$, then

$$r \in P_o^{-1}(P_o(uv)) \Rightarrow e \in r.$$

We also say here that N_L is K -diagnosable.

2.2 Timed Modeling Formalisms of DES

Untimed DES models are built when we consider only the logic features, i.e., the logical order of event occurrences. This is insufficient for the analysis of some systems whose behavior is based on quantitative temporal parameters. Therefore, the classical untimed models for DES have been extended with temporal features. As examples of such timed notations, one can cite timed transition systems (TTS), timed automata (TA) [*Alur and Dill, 1994*], timed Petri nets [*Ramchandani, 1974*], time Petri nets (TPNs) [*Merlin, 1974*] and labeled time Petri nets (LTPNs) [*Bérard et al., 2005*], etc.

In this section, we will review the background of TTS, TA, TPNs and LTPNs, which will be used afterwards. Some notations are inspired from [*Bérard et al.,*

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

2005; Diaz, 2001; Ghazel et al., 2009; Tripakis, 2002]. For more details, the reader can refer to these literatures.

2.2.1 Timed Transition Systems (TTS)

Let Σ be a finite set of events, and let $\mathbb{R}_{\geq 0}$ be the set of non-negative real numbers.

Definition 5 [Henzinger et al., 1992] A transition system S is a 4-tuple $(Q, q_0, \Sigma, \rightarrow)$ where:

- Q is the set of the states;
- $q_0 \in Q$ is the initial state;
- Σ is the set of events;
- $\rightarrow \subseteq Q \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times Q$ is the set of edges.

We use $q \xrightarrow{a} q'$ to denote $(q, a, q') \in \rightarrow$, which indicates that when the state of the system is q , it can change to q' upon $a \in \Sigma \cup \mathbb{R}_{\geq 0}$. The edges labeled with an event of Σ are called discrete edges and the edges labeled with a non-negative real number are called continuous edges. A path is a finite or infinite sequence of edges $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \dots$. A state $q' \in Q$ is reachable from a state q if there exists a finite sequence $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \dots \xrightarrow{a_{n-1}} q_n$, such that $q_0 = q$ and $q_n = q'$.

Definition 6 [Henzinger et al., 1992] A timed transition system (TTS) S is a 6-tuple $(Q, q_0, \Sigma, \rightarrow, l, u)$ where:

- $(Q, q_0, \Sigma, \rightarrow)$ is a transition system;
- $l_e \in \mathbb{Q}^+$ is a minimum delay for every transition $e \in \Sigma$;
- $u_e \in \mathbb{Q}^+ \cup \{\infty\}$ is a maximum delay for every transition $e \in \Sigma$.

2.2.2 Timed Automata (TA)

Timed automata are finite automata extended with real-valued variables called clocks to specify timing constraints between occurrences of events. For a detailed presentation of the fundamental results for timed automata, the reader can refer to the seminal paper of [Alur and Dill, 1994].

2.2.2.1 Definition of TA

Definition 7 [Alur et al., 1999] A timed automaton A is a 6-tuple $(L, L^0, \Sigma, X, I, E)$, where:

- L is a finite set of locations;
- $L^0 \subseteq L$ is a set of initial locations;
- Σ is a finite set of labels;
- X is a finite set of clocks;
- I is a mapping that labels each location s with some clock constraint in $\Phi(X)$, where the set $\Phi(X)$ of clock constraints φ is defined by the grammar

$$\varphi := x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2, c \in \mathbb{Q}^+;$$

- $E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L$ is a set of transitions. A transition $(s, a, \varphi, \lambda, s')$ represents an edge from location s to location s' on symbol a . φ is a clock constraint over X that specifies when the transition is enabled, and the set $\lambda \subseteq X$ gives the clocks to be reset while firing this transition.

Example 7 Figure 2.7 presents an example of timed automaton.

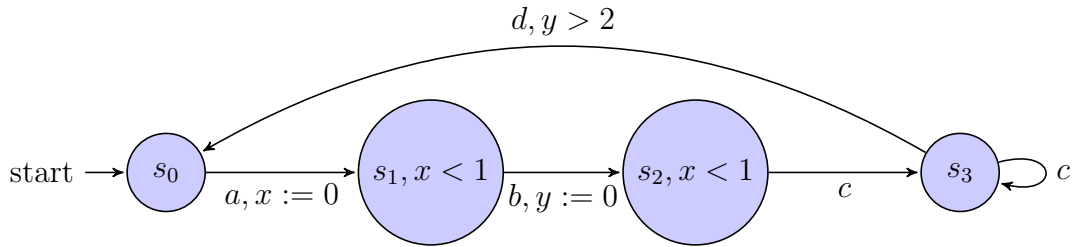


Figure 2.7: An example of timed automaton

- $L = \{s_0, s_1, s_2, s_3\}$;
- $L^0 = \{s_0\}$;

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

- $\Sigma = \{a, b, c, d\};$
- $X = \{x, y\};$
- $I(s_0) = I(s_3) = \emptyset, I(s_1) = I(s_2) = \{x < 1\};$

2.2.2.2 TA Semantics

The semantics of a timed automaton A is defined as a timed transition system S_A . A state of S_A is a pair (s, ν) such that s is a location of A and ν is a clock valuation for X such that ν satisfies the invariant $I(s)$. The set of all states of A is denoted Q_A . A state (s, ν) is an initial state if s is an initial location of A ($s \in L_0$) and $\nu(x) = 0$ for all clocks x . There are two types of transitions in S_A :

- Elapse of time: for a state (s, ν) and a real-valued time increment $\delta \geq 0$, $(s, \nu) \xrightarrow{\delta} (s, \nu + \delta)$ if for all $0 \leq \delta \leq \delta, \nu + \delta$ satisfy the invariant $I(s)$.
- Location transition: for a state (s, ν) and a transition $(s, a, \varphi, \lambda, s')$ such that ν satisfies φ , $(s, \nu) \xrightarrow{a} (s', \nu[\lambda := 0])$.

Thus, S_A is a transition system with label-set $\Sigma \cup \mathbb{R}_{\geq 0}$.

The time-additivity property for TA is defined as:

$$(q \xrightarrow{\delta} q') \wedge (q' \xrightarrow{\epsilon} q'') \Rightarrow (q \xrightarrow{\delta+\epsilon} q'')$$

with $\delta, \epsilon \in \mathbb{R}_{\geq 0}$.

Note that the executability constraints have been omitted here. First, when the invariant of a location is violated, some outgoing edge must be enabled. Second, from every reachable state the automaton should admit the possibility of time to diverge. For example, the automaton should not enforce infinitely many events in a finite interval of time. Automata satisfying this operational requirement are called nonZeno.

Example 8 For the timed automaton of Figure 2.7, the state-space of the associated transition system is $\{s_0, s_1, s_2, s_3\} \times \mathbb{R}^2$, the label-set is $\{a, b, c, d\} \cup \mathbb{R}_{\geq 0}$, and as an example of transition sequence: $(s_0, 0, 0) \xrightarrow{1.2} (s_0, 1.2, 1.2) \xrightarrow{a} (s_1, 0, 1.2) \xrightarrow{0.7} (s_1, 0.7, 1.9) \xrightarrow{b} (s_2, 0.7, 0)$.

2.2.3 Time Petri Nets (TPNs)

2.2.3.1 Definition of TPN

Definition 8 [*Merlin, 1974*] A TPN is a 6-tuple $(P, T, Pre, Post, M_0, SIM)$ where:

- $(P, T, Pre, Post, M_0)$ is a marked PN;
- $SIM: T \rightarrow \mathbb{Q}_{\geq 0} \times (\mathbb{Q}_{\geq 0} \cup \{\infty\})$ associates a static interval mapping with each transition, where $\mathbb{Q}_{\geq 0}$ is the set of non-negative rational numbers.

Example 9 Let us consider the example of TPN in Figure 2.8. Here SIM is the static interval mapping such that $SIM(t_1) = [1, 1]$, $SIM(t_2) = [0, 2]$ and $SIM(t_3) = [0, 3]$.

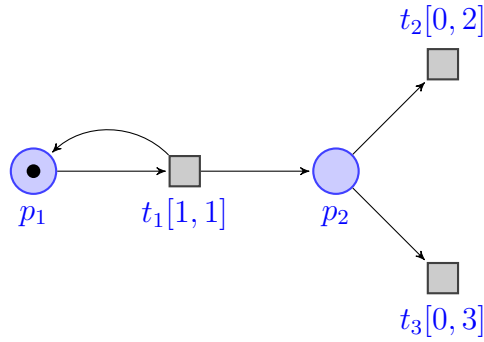


Figure 2.8: An example of TPN

The *state* of a TPN is a pair $E = (M, I)$, where M is the marking of the net, and I is the *firing interval mapping* which associates each transition with its firing interval.

The initial state is defined by $E_0 = (M_0, I_0)$, where M_0 is the initial marking, and I_0 is the mapping associating each transition enabled at M_0 with its static firing interval, and the empty interval for all other transitions. Formally, I_0 is defined by:

$$I_0(t_j) = \begin{cases} SIM(t_j) & \text{if } t_j \in En(M_0) \\ \emptyset & \text{otherwise} \end{cases} \quad (2.7)$$

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

2.2.3.2 Dynamics of TPN

Let us look at the state transition of the TPN from a state $E = (M, I)$ towards state $E' = (M', I')$ following the firing of a transition t with $I(t) = [\alpha_t, \beta_t]$ ¹, at a relative time θ_t . The following rules must be respected:

- $t \in En(M)$;
- $\theta_t \geq \alpha_t$;
- $\forall k \in En(M), \theta_t \leq \beta_k$.

We write $E \xrightarrow{(t, \theta_t)} E'$ to denote this state transition, and the new state E' is defined as follows:

- $M' = M - Pre(\cdot, t) + Post(\cdot, t)$;
- new firing intervals: $\forall k \in T$,
 - if $k \notin En(M')$, $I'(k) = \emptyset$;
 - if $k \neq t$ and $k \in En(M)$, and k is not in conflict with t , then:

$$I'(k) = \begin{cases} [max(0, \alpha_k - \theta_t), \beta_k - \theta_t] & \text{if } \beta_k \neq \infty \\ [max(0, \alpha_k - \theta_t), \infty[& \text{otherwise} \end{cases} \quad (2.8)$$

- $I'(k) = SIM(k)$, otherwise.

If a transition t remains enabled during its own firing (t is *multi-enabled* [Diaz, 2001]), then $I'(t) = SIM(t)$. That means t is considered as to be newly enabled.

2.2.3.3 TPN Semantics

The TTS $S_T = (Q, q_0, T, \rightarrow)$ associated with a TPN $N_T = (P, T, Pre, Post, M_0, SIM)$ is defined by $Q = \mathbb{N}^{|P|} \times (\mathbb{R}_{\geq 0})^n$ with $n \leq |T|$, $q_0 = (M_0, \vec{0})$, and $\rightarrow \subseteq Q \times (T \cup \mathbb{R}_{\geq 0}) \times Q$ is the set of edges defined by:

¹ An interval is denoted by $[\alpha_t, \beta_t]$ when $\beta_t \neq \infty$, and $[\alpha_t, \beta_t[$ when $\beta_t = \infty$.

-
1. The discrete edges (relative to transitions) are defined by, for all $t_i \in T$:

$$(M, v) \xrightarrow{t_i} (M', v') \Leftrightarrow \begin{cases} M \geq \text{Pre}(t_i) \wedge M' = M - \text{Pre}(t_i) + \text{Post}(t_i) \\ \alpha(t_i) \leq v_i \leq \beta(t_i) \\ v'_k = \begin{cases} 0 & \text{if } t_k \text{ is newly enabled after the firing of } t_i \text{ at } M \\ v_k & \text{otherwise} \end{cases} \end{cases} \quad (2.9)$$

2. The continuous edges (relative to time elapsing) are defined by, for all $\delta \in \mathbb{R}_{\geq 0}$: $(M, v) \xrightarrow{\delta} (M, v') \Leftrightarrow v' = v + \delta$, and $\forall k \in \{1, \dots, n\}, (M \geq \text{Pre}(t_k) \Rightarrow v'_k \leq \beta(t_k))$.

The last condition on continuous transitions ensures that the time that elapses in a marking cannot increase to a value which would disable transitions that are enabled by the current marking (strong semantics). For TPNs, as for TA, it is not possible to work directly on the TTS which represents the behavior of the TPN, because this TTS has infinitely many states (and infinitely many labels). Again, the use of abstraction methods permit the construction of a transition system where the labels expressing the passing of time are eliminated and where states are regrouped into classes on which the reachability analysis can be done. The state class graph [Berthomieu and Menasche, 1983] and the zone graph [Gardey et al., 2004] are examples of such approaches which gather the states that are equivalent up to a time elapsing in macro states. However, these methods do not always give a result because for a TPN the problems of reachability and boundedness are undecidable [Berthomieu and Diaz, 1991].

2.2.3.4 State Class

One can note that using notation (M, I) will lead to infinite set of states. State class has been introduced in order to gather the states which can be obtained from each other simply by time elapsing.

A *state class* of a TPN is associated to an achievable firing sequence of tran-

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

sitions from the initial state:

$$(M, D) = \{(M, i) \mid \exists(\sigma, u) \in \mathcal{D}, M_0 \llbracket \sigma > M, (M_0, i_0) \xrightarrow{(\sigma, u)} (M, i)\}.$$

D is called the *firing domain* and is the set of vector solutions of the τ_j -linear inequaties, where τ_j stands for the relative firing date of enabled transition t_j .

Given $t_j \in T$, t_j is firable starting from a given class $C = (M, D)$ iff:

- $M \geq \text{Pre}(\cdot, t_j)$;
- Inequaties in the firing domain D holds;
- $\forall j \neq k, \tau_j \leq \tau_k$. (**Condition A**)

Consider that from a given class $C = (M, D)$, the system reaches class $C' = (M', D')$ following the firing of transition t_j [Diaz, 2001], denoted by $C \xrightarrow{t_j} C'$. Here C' is defined by:

1. $M' = M - \text{Pre}(\cdot, t_j) + \text{Post}(\cdot, t_j)$;
2. the new firing domain D' is determined starting from the linear system associated with D , according to the following algorithm:
 - (a) **Condition A** is added to the linear system of C and denotes that transition t_j can be fired first among $\text{En}(M)$.
 - (b) All variables τ_k associated with transitions t_k in conflict with t_j are eliminated from the system.
 - (c) Each variable $\tau_l, l \neq j$ is replaced by the sum $\tau_j + \tau_l$. Then, τ_j is eliminated from the system.
 - (d) For each transition t_m newly enabled by M' , a new variable τ_m framed by the bounds of the static firing interval of t_m is introduced to the linear system.

The transition $C \xrightarrow{t_j} C'$ can be simply explained as follows: any state in C' can be reached from one of the states in C by firing transition t_j ; or there exists a subset of C such that any state in this subset can arrive at a state in C' by the firing of t_j .

Given two state classes C and C' , C' is said to be *reachable* from C if C' can be obtained by firing a sequence $\sigma \in T^*$ from C , and we denote it by $C \xrightarrow{\sigma} C'$.

Proposition 2 [*Diaz, 2001*] *A TPN is bounded iff the number of state classes of this net is finite.*

2.2.4 Labeled Time Petri Nets (LTPNs)

By associating with each transition of the TPN an event, this TPN is said labeled. We then speak about labeled time Petri net (LTPN) [*Bérard et al., 2005*], such that each firing of transition simultaneously produces the corresponding event.

2.2.4.1 Definition of LTPN

Definition 9 [*Bouyer et al., 2006*] *A LTPN is a tuple $(P, T, Pre, Post, M_0, SIM, \Sigma, \varphi)$, where:*

- $(P, T, Pre, Post, M_0, SIM)$ is a TPN;
- Σ is a finite set of events;
- $\varphi: T \rightarrow \Sigma$ is the transition labeling function as defined for LPNs.

Informally, we can treat a LTPN as a LPN with temporal constraints on its transitions, or a TPN whose transitions are labeled with an event. The relation between LTPN and PN, TPN, LPN is illustrated in Figure 2.9.

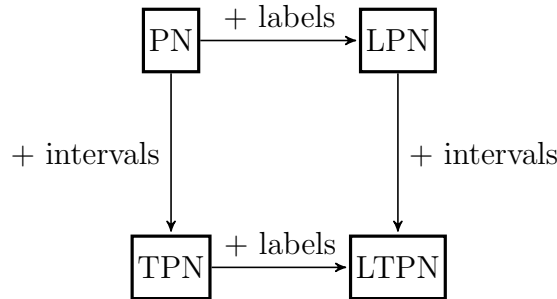


Figure 2.9: Relation between LTPN and PN, TPN, LPN

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

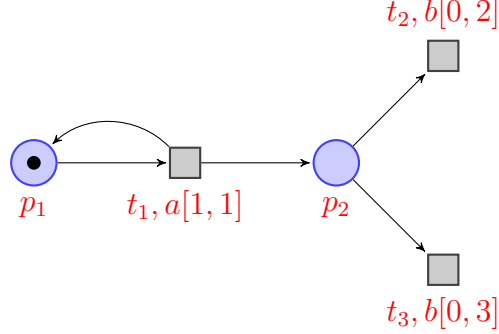


Figure 2.10: An example of LTPN

Example 10 Let us consider the example of LTPN in Figure 2.10. Here φ is the transition labeling function such that $\varphi(t_1) = a, \varphi(t_2) = \varphi(t_3) = b$.

A state change of a LTPN can be driven either by the firing of some transition or by time elapsing. Note that, in LTPNs, two different transitions can be labeled with the same event.

Definitions of *state*, *state class* and their corresponding transition mapping are the same as for TPNs. We denote by \mathcal{M} the set of states of LTPN. For $E, E' \in \mathcal{M}$, we write $E \xrightarrow{(t, \theta_t)} E'$ to denote this state transition from E to E' by the firing of transition t at the relative firing date θ_t . Besides, most analysis approaches of TPNs can be extended to LTPNs.

2.2.4.2 Timed Language for LTPNs

A *dated firing sequence (DFS)* [Diaz, 2001] is a pair (σ, u) , where $\sigma \in T^*$ is an achievable firing sequence, and u is the sequence of firing dates of the transitions in σ . The set of DFSs is denoted by \mathcal{D} .

Given a sequence of transition firings (or transition dates) w , we denote by w^j the j^{th} element in w , and $|w|$ the length (number of elements) of w . For $a \in T \times \mathbb{R}_{\geq 0}$ and $w \in (T \times \mathbb{R}_{\geq 0})^*$, we write $a \in w$ if there exists j such that $w^j = a$. We also write $w = w_1 w_2 \dots w_n$ to say that w is the concatenation of w_1, w_2, \dots, w_n , where w_1, w_2, \dots, w_n are sequences of transitions (or events, dates).

A state E' is said to be reachable from state E by the firing of a DFS (σ, u) ,

denoted by $E \xrightarrow{(\sigma, u)} E'$ with $|\sigma| = n$, if $\exists E_0, E_1, \dots, E_n$ such that $E = E_0, E_n = E'$ and $\forall 1 \leq j \leq n, E_{j-1} \xrightarrow{(\sigma^j, u^j)} E_j$.

Definition 10 A labeled dated firing sequence (LDFS) of DFS (σ, u) is defined by (s, u) , where $s = \varphi(\sigma)$, and φ is the extended form of the labeling function φ in the usual manner.

We write \mathcal{D}_l to denote the set of LDFSs.

Definition 11 The language generated by LTPN N_{LT} is defined by:

$$\mathcal{L}(N_{LT}) = \{(\varphi(\sigma), u) \mid \exists E \in \mathcal{M}, (\sigma, u) \in \mathcal{D}, s.t. E_0 \xrightarrow{(\sigma, u)} E\}.$$

For a given LTPN N_{LT} , we use \mathcal{L} to denote $\mathcal{L}(N_{LT})$ for short.

Let us define some projections for timed language. Given a LDFS p and a set of observable events Σ_o , $P_o(p)$ is the LDFS obtained by erasing from p all the unobservable events and summing up the relative delays to the delay of the very following observable event. Define the inverse projection operator P_o^{-1} as

$$P_o^{-1}(r) = \{p \in \mathcal{L} \mid P_o(p) = r\}$$

for $r \in (\Sigma_o \times \mathbb{R}_{\geq 0})^*$.

Given a language $\mathcal{L} \subseteq \mathcal{D}_l$ and a string $p \in \mathcal{L}$, the post-language of \mathcal{L} after p denoted by \mathcal{L}/p , is the language

$$\mathcal{L}/p = \{r \in \mathcal{D}_l \mid pr \in \mathcal{L}\}.$$

Example 11 Given $\Sigma_o = \{a, c\}, r_1, r_2 \in \mathcal{L}, s_1, s_2 \in \Sigma^*$ with $r_1 = (s_1, u_1), r_2 = (s_2, u_2), s_1 = abca, u_1 = (1, 2, 3, 2), s_2 = aba, u_2 = (1, 2, 0)$, then $P_o(r_1) = (s_3, u_3)$ wherein $s_3 = aca$ and $u_3 = (1, 5, 2)$. Finally, $(ca, (3, 2)), (a, 0) \in \mathcal{L}/(ab, (1, 2))$.

2.2.4.3 Diagnosability of LTPN

The diagnosability and Δ -diagnosability of TA has been introduced in [Tripakis, 2002]. Now we will discuss these issues in the framework of LTPNs. Without loss of generality, we consider only one class of faults.

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

Definition 12 [*Liu et al., 2013, 2014a*] Given a LTPN N_{LT} , we say G is diagnosable if $\exists \Delta \in \mathbb{Q}_{\geq 0}$ such that $\forall (s, u) \in \mathcal{L}$, $s^{|s|} \in \Sigma_f$, $s^j \notin \Sigma_f$ for $j < |s|$ and $\forall (w, z) \in \mathcal{L}/(s, u)$, $\sum_{j=1}^{|z|} z^j \geq \Delta$, then the following holds:

$$r \in P_o^{-1}(P_o(sw, uz)) \Rightarrow \exists e \in \Sigma_f \text{ s.t. } e \in r.$$

We also say here that G is Δ -diagnosable.

In simple terms, any fault in a diagnosable LTPN can be diagnosed upon a finite delay after its occurrence. Obviously, there exists Δ_{min} such that, G is Δ -diagnosable for any $\Delta \geq \Delta_{min}$, and G is not Δ -diagnosable for any $\Delta < \Delta_{min}$.

As we have analyzed, looking for the Δ_{min} of a diagnosable LTPN will be an interesting issue of practical significance since, in practice, we wish that the fault can be diagnosed as soon as possible and it is important to determine the minimum delay upon which we ensure the fault can be diagnosed.

2.3 Hypothesis

This work will deal with fault diagnosis of DESs in both untimed and timed contexts, on the basis of proved existing results. Thus, we followed the hypothesis for the classic diagnosis analysis. For clarity, we also make the following remarks:

- The system under consideration can be abstracted as an untimed or timed DES, which can be modeled by an LPN or an LTPN.
- In the used models LPNs and LTPNs, each transition is associated with either an observable or an unobservable event, and one event may assign to multiple transitions.
- No achievable cycle of unobservable transitions exists in the LPN or LTPN.
- During the analysis, the system structure does not change, and the system behavior does not respond to outside inferences (control commands).
- We consider only permanent faults, i.e., the system remains in faulty state after the occurrence of fault.

-
- The faults considered can be partitioned into multiple disjoint sets.

2.4 Conclusion

In the section, we have reviewed the modeling formalisms of DESs and TDESs, as will be used in this thesis, i.e., LPNs and LTPNs. We have also introduced some notations for languages, the definition of K -diagnosability for untimed DESs and Δ -diagnosability for TDESs. Finally, we give the basic hypothesis for the discussion in the sequel.

Before discussing the fault diagnosis issues, in the following chapter we will review the literature concerning existing diagnosis techniques using the DES models as introduced in the current chapter.

2. MODELING FORMALISMS FOR DIAGNOSABILITY ANALYSIS OF DES

Chapter 3

Literature Review

This chapter reviews the existing studies on DES-based fault diagnosis. Literature is summarized and classified by three parts: diagnosability and its extensions, diagnosis approaches, and diagnostic software tools.

3.1 Overview

Fault diagnosis has been shown to play an essential role in the safe and reliable operation of industrial systems. This issue has received considerable attention in the context of DESs. DES diagnosis have been studied from different viewpoints with the application of various techniques, as shown in Figure 3.1:

- The two most discussed topics are *online diagnosis* and *diagnosability analysis* [Lin, 1994]. In simple terms, diagnosis is performed online to detect the occurrences of faults and to localize the cause of faults. Diagnosability refers to the ability to detect and locate any fault within a finite delay after its occurrence. Diagnosability analysis is performed offline. Logically, a fault can be eventually diagnosed if the system is diagnosable. Therefore, offline diagnosability analysis is the basis of online diagnosis.
- The two most used modeling formalisms are *automata* and *PNs*. DES-based diagnosis is first studied in the framework of regular languages and automata [Lin, 1994; Sampath et al., 1995]. These automata-based approaches are based on state enumeration, which can induce state explosion

3. LITERATURE REVIEW

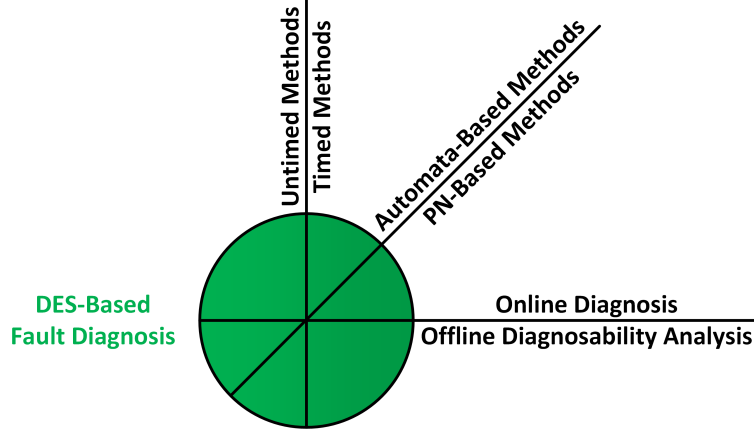


Figure 3.1: DES-based fault diagnosis

problems. In order to overcome this problem, Some subsequent automata-based approaches [Jiang et al., 2001; Yoo and Lafortune, 2002b] have been proposed for reducing the computational complexity, without the construction of a diagnoser automaton. Besides, a series of works [Aghasaryan and Fabre, 1998; Basile et al., 2010; Bouyer et al., 2005; Brandin and Wonham, 1994; Cabasino et al., 2010; Genc and Lafortune, 2007; Ghazel et al., 2009; Haar, 2009; Ru and Hadjicostis, 2009; Ushio et al., 1998; Wen et al., 2005] concerning the diagnosis and diagnosability of DESs turned to PN modeling, thus benefiting from the expressiveness and the well-developed theory of PNs.

- DES-based diagnosis are investigated in both *untimed* and *timed* contexts [Cassez, 2010; Ghazel et al., 2009; Holloway and Chand, 1994; Jiang et al., 2006; Liu et al., 2013; Tripakis, 2002; Wang et al., 2011, 2013; Zemouri and Faure, 2006]. Untimed discrete event models and timed discrete event models are two abstraction types of real systems. An untimed discrete event model characterizes the logical behavior of systems, i.e., only the ordering of events is considered, while a timed discrete event model makes it explicit the quantitative temporal constraints on the system behavior. Timing characterizes DESs in a new (time) dimension, so that the system behavior contains richer information. However, the complexity of dealing

with such systems is much higher.

We summarize and classify some main works on DES-based diagnosis, from the viewpoints of modeling formalism, subject investigated and technique used, as shown in Table 1 in Appendix 6.2.

3.2 Literature on Diagnosability of Discrete Event Systems (DES)

3.2.1 Classical Diagnosability

DES-based diagnosis has had a growing interest from both academia and industrial communities during the two past decades. [Sampath et al., 1995] is a pioneer work on this topic. The authors set a formal definition of diagnosability for untimed DESs, where the faults are treated as unobservable and classified into disjoint classes. They also give the necessary and sufficient conditions for diagnosability. A model called “diagnoser” is then introduced both to test diagnosability by examining indeterminate cycles and for online diagnosis by mapping the online observations on the diagnoser states. The diagnoser-based approach enumerates all the states and, consequently, suffers from state explosion problem. Generally, the diagnoser state space is exponential in the number of states of the original automaton.

Diagnosability of DESs is then introduced in the framework of PNs. In [Ushio et al., 1998], the language-based diagnosability in [Sampath et al., 1995] is extended to unbounded PNs, where the net marking is observable and all transitions are unobservable, and the faults are associated to transitions. A simple ω diagnoser and sufficient conditions for diagnosability of unbounded PNs are proposed.

For other literature on classical diagnosability, we review the following representative works.

In [Jiang et al., 2001], an algorithm based on the parallel composition of an automaton with itself is proposed. In this approach, no diagnoser is built and the complexity is polynomial of fourth order in the number of the system states and linear in the number of the failure types. In [Yoo and Lafortune, 2002b], a com-

3. LITERATURE REVIEW

parable polynomial-time algorithm for deciding diagnosability is presented. The approach is based on the construction of a non-deterministic automaton called “verifier”. Both methods are based on algorithms which investigate if the system is diagnosable by seeking some specific cycles. Hence the system is diagnosable if such cycles do not exist. Although these approaches are more efficient than the diagnoser approach in terms of time complexity, they are still based on *a priori* built state space and suffer from state explosion problem.

In [Xue and Zheng, 2004] and [Cabasino et al., 2012a], a composition net called verifier is constructed for the analysis of diagnosability for PNs. The practical verification condition is further given for unbounded PNs based on the coverability graph of the verifier. The diagnosability of a PN is then transformed as a reachability problem on the verifier model. The verifier net approach is more efficient compared with traditional diagnoser approach, however, it requires an exhaustive enumeration of the reachability set of the verifier net, which may be larger than the reachability set of the original PN. Note that the authors of [Cabasino et al., 2012a] also deal with the K -diagnosability problem and use the integer linear programming technique.

In [Wen et al., 2005], the authors propose a sufficient condition for testing diagnosability by checking the structure of subnet called T-components related to the T-invariants of the LPN without building a diagnoser. The indeterminate cycle in the sense of [Sampath et al., 1995] in the LPN is represented by the existence of two inequivalent T-invariants with the same observable projection. Thus the system is diagnosable if there exists no two such T-invariants. This ILP-based approach is of polynomial complexity in the number of nodes for computing a sufficient condition for diagnosability of the LPN. However, the solution of T-invariants are the firing count vectors which record the number of firing transitions but without their order. The approach is not suitable for analyzing the diagnosability of LPNs with the T-invariants of the same observable projection, e.g., no verdict can be made for the nets in Figure 3.2 and Figure 3.3 where events a, b are observable and f is the faulty transition.

Note that the above mentioned literature is on the diagnosability of permanent faults. There are also some works on the diagnosability of intermittent faults as in [Contant et al., 2004; Jiang et al., 2003a, 2006; Yoo and Garcia, 2004].

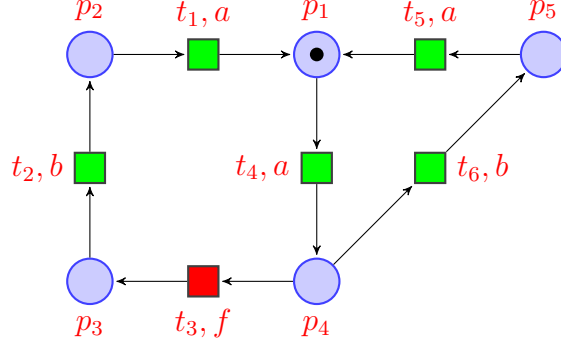


Figure 3.2: A diagnosable LPN with 2 minimal T-invariants w.r.t the same observable projection

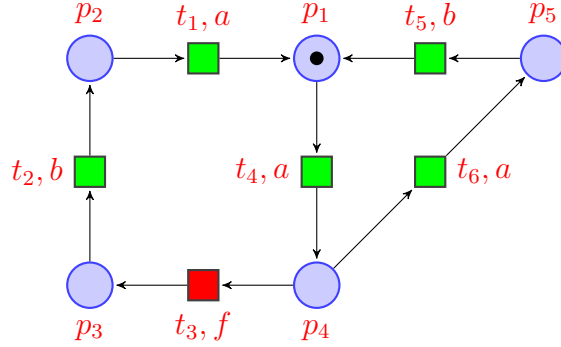


Figure 3.3: An undiagnosable LPN with 2 minimal T-invariants w.r.t the same observable projection

3.2.2 K -diagnosability of Untimed DESs

The classical diagnosability problem consists in qualitatively determining the existence of a finite delay upon which any fault (or class of faults) can be detected and identified. In practice, the diagnosability feature can be insufficient to ensure a safe operation of the system, namely when we deal with safety-critical systems. Indeed, this delay could be too long and faults may have dramatic consequences before being diagnosed and before some reconfiguration actions can be undertaken. Thus, some “quantitative” versions of diagnosability have been developed, namely K -diagnosability [Basile et al., 2010, 2012; Cabasino et al., 2012a] which ensure that any fault (or class of faults) can be determined within a finite delay

3. LITERATURE REVIEW

of K observable events upon its occurrence.

Generally speaking, there are two main problems on K -diagnosability. The first is to analyze K -diagnosability of a system under a given value K , i.e., whether or not any fault (or class of faults) can be detected and identified within K steps (observable events) after its occurrence. The second is to find the minimum K for a diagnosable system.

In [Basile et al., 2010], K -diagnosability is discussed in the framework of general PNs, where K refers to the number of both observable and unobservable transitions fired after the faulty transition. The K -diagnosability is solved as a linear programming problem. In [Basile et al., 2012], an extensive approach is proposed to check K -diagnosability for both unlabeled and labeled PNs. In these approaches, system behavior is represented by a series of linear equations. The diagnosability of the PN models can be verified only if the faulty behavior can be described by a finite number of equations. The approach is efficient when checking K -diagnosability, however, it is not suitable for classic diagnosability analysis, since in order to investigate classic diagnosability, new equation systems may have to be built for each K , and the existing state space cannot be sufficiently used.

In [Cabasino et al., 2012a], the authors provide necessary and sufficient conditions for classical diagnosability and diagnosability in K steps (K -diagnosability), and develop an approach to compute the bound K based on the analysis of the reachability/coverability graph of a structure called modified verifier net. In this work the value K refers to the number of observable transitions/events after the fault, which is a little different from that in [Basile et al., 2010, 2012].

Besides, there are many other studies on the other extensive versions of diagnosability for untimed DESs, such as $[1, K]$ -diagnosability in [Jiang et al., 2006; Yoo and Garcia, 2004], codiagnosability in [Wang et al., 2004], modular diagnosability in [Contant et al., 2004], etc. We do not review these works here, since the framework is different from the one considered in this thesis.

3.2.3 Diagnosability and Δ -Diagnosability of TDES

Compared with DES, TDES is an abstraction of the system behavior which considers both the ordering and the occurrence dates of events. With the help of the absolute occurrence dates of events or the relative duration between two states, two undistinguishable event sequences may become distinguishable in untimed context. Consequently, undiagnosable faults in untimed context may become diagnosable in a timed context, which means that time information can carry valuable knowledge w.r.t diagnosis issues. This is an important motivation for considering time information while dealing with these issues.

However, considering temporal information in the diagnosis framework often leads to much higher computation and memory complexity. Therefore, dealing with diagnosis in a timed context becomes even more challenging, especially given that one has to handle infinite state space in this case. In order to face this challenge, techniques seeking for finite representation of infinite state space of TDESs, and transforming timed problem into untimed one have been developed.

[Tripakis, 2002] provides algorithms to check Δ -diagnosability for TDESs, i.e., to diagnose a fault within a delay of at most Δ time units after its occurrence, and gives necessary and sufficient conditions of diagnosability for timed automata. The developed algorithms are based on standard reachability analysis of some accepting states or on searching non-zeno runs. [Xu et al., 2010] deals with diagnosis of TDESs modeled as timed automata. It is shown that the problem of diagnosability analysis and diagnosis of dense-time system is decidable by reducing this problem to the untimed setting. These approaches are on the basis of a known state space. For complex systems, computing the whole state space (time automata) in the timed context may be rather resource-consuming.

Diagnosis of TDESs are also considered in the framework of TPN. In [Pencolé, 2009], the authors describe faults that can occur during the execution of service workflows by means of chronicles, and propose a diagnosis algorithm based on chronicle recognition in the framework of TPNs. In [Jiroveanu et al., 2006], fault diagnosis of TPN model is discussed based on partial orders (unfoldings). The set of legal traces in the TPN is obtained solving a system of linear inequalities. Two methods based on Extended Linear Complementarity Problem and constraint

3. LITERATURE REVIEW

propagation are used for the solution. These approaches discuss only the diagnosis issue but not diagnosability. Besides, no analysis based on a more general model LTPN can be found.

3.3 Literature on Diagnosis Techniques

3.3.1 Diagnoser Approach

The classical diagnoser based approach [Sampath et al., 1995, 1996] is often referred as a pioneer work on the DES diagnosis topic. In this approach, the system behavior is characterized by states with the corresponding fault occurrence information. Accordingly, any state is labeled with a tag “Normal”, “F(Fault)-certain” or “F-uncertain”. “F-uncertain” state can lead to a “Normal”, “F-certain” after further observations. This approach is based on state enumeration, thus it is not suitable for unbounded system (the number of system states is infinite). Even for bounded model, it suffers from state space exploration. Therefore, some other automata or PN based approaches have been developed, as attempts to tackle this issue, even partially.

3.3.2 Verifier (Twin plant) Approach

Instead of building a diagnoser, in verifier based approach a new structure derived from the original model under consideration is constructed for diagnosability analysis. Then the diagnosability problem can be treated by analyzing the structure of the verifier. The verifier is constructed as the parallel composition of the plant model (possibly an automaton or a PN) with itself, the composite model is then called “twin plant”. The aim of building a verifier is indeed to perform the diagnosability analysis under a lower complexity than for the diagnoser-based approach.

In [Jiang et al., 2001; Yoo and Lafortune, 2002b], a verifier automaton, instead of a diagnoser automaton, is built by the parallel composition of the automaton with fault information and itself. The system is diagnosable if and only if there exists no such a cycle in the verifier that the fault occurs in one trace but not in

the other.

In [Cabasino et al., 2012a; Xue and Zheng, 2004], a verifier net obtained by the parallel composition of the PN and itself under some given rules is built. The diagnosability problem is then transformed as the reachability/coverability analysis of the verifier net.

3.3.3 Decentralized/Distributed/Modular Approaches

To overcome the problem of state explosion, approaches of decentralized, distributed and modular diagnosis [Benveniste et al., 2003; Cabasino et al., 2013a; Cassez, 2012; Contant et al., 2006; Debouk et al., 2000; Genc and Lafortune, 2007; Jiroveanu and Boel, 2005; Lafortune et al., 2005; Pencolé, 2009; Provan, 2002; Qiu et al., 2006] have also been discussed. Although We put these three literally similar methods together, there are some differences between the terminology [Zaytoon and Lafortune, 2013]. Generally speaking, decentralized approaches have a set of diagnosers, each with different observation capabilities, but all considering the global system model in their model-based inferencing. In distributed approaches, the individual diagnosers only use partial (local) system models as opposed to the global system model.

In [Genc and Lafortune, 2007], the authors develop a distributed (modular) approach for online diagnosis. The system under consideration is treated as a set of modules, for each of them a PN diagnoser is built to perform online diagnosis. Local diagnosis information can be shared between modules modeled by PNs with some common places, such that the global diagnosis information can be recovered.

3.3.4 Linear Programming Approach

The mathematical representation of PNs allows use of standard tools, such as integer linear programming (ILP), to solve DES diagnosis problems.

An early study on diagnosability of PNs models can be found in [Wen et al., 2005]. The authors provide an algorithm of polynomial complexity in the number of nodes for computing a sufficient condition for diagnosability of PN models. ILP technique is used to check a specific structure called T-component which is related

3. LITERATURE REVIEW

to minimum T-invariants. The proposed algorithm shows the efficiency compared with state enumeration approaches.

In [Basile et al., 2010, 2012], the authors propose ILP-based approaches to check K -diagnosability of DESs modeled by PNs and LPNs, which avoids using a diagnoser. The proposed approach does not require any specific assumption on the structure of the net induced by the unobservable transitions. Necessary and sufficient conditions are then given for diagnosability of bounded nets. The main drawback is that the characterization of a fireable sequence in terms of firing count vectors may require, in the worst case, a number of firing count vectors equal to the sequence length.

[Cabasino et al., 2012a] deals with diagnosability of LPN models using a twin plant called verifier net (VN). ILP technique is used to look for cycles associated with firable repetitive sequences (upon the firing of a fault transition) in the coverability graph of the VN.

3.3.5 Unfolding Technique

There are also some diagnosis analysis approaches using unfolding technique [Benveniste et al., 2003; Grabiec et al., 2010; Haar, 2009; Madalinski et al., 2010]. Unfolding is a well-established technique for verifying properties of Petri nets; its use for this purpose was initially proposed by McMillan [McMillan and Probst, 1995]. The unfolding of a (Petri) net is another net of acyclic structure that fully represents the state space (reachable markings) of the original net. Because unfoldings represent behavior by acyclic structures rather than by interleaved actions, they are often exponentially smaller than the state space of the net, and never larger than it.

[Grabiec et al., 2010] discusses the on-line diagnosis of distributed systems using TPN models. We propose to base the method on unfoldings. Given a partial observation, as a possibly structured set of actions, our method determines the causal relation between events in the model that explain the observation. It can also synthesize parametric constraints associated with these explanations. The method is implemented in the tool Romeo. We present its application to the diagnosis of the example of a cowshed with pigs.

[Madalinski et al., 2010] gives an approach to verify diagnosability in the framework of LPN unfoldings based on the twin plant method. The unfolding is infinite whenever the LPN N_L has an infinite run; however, if N_L has finitely many reachable states then the unfolding eventually starts to repeat itself and can be truncated without loss of information, yielding a finite representation. A verier, which compares pairs of paths from the initial model sharing the same observable behavior, is built check diagnosability.

3.3.6 Model-Checking Based Techniques

Model checking [Clarke et al., 1994] is a formal verification technique for assessing functional properties of systems, which are written in propositional temporal logic. The verification procedure is an exhaustive search of the state space to check whether or not the given model satisfies this property.

Model checking techniques are applied to fault diagnosis because it has the following advantages [Baier and Katoen, 2008]:

- It is a general verification approach for a wide range of applications such as embedded systems, software engineering, hardware design, etc.
- It supports partial verification, i.e., properties can be checked individually, thus allowing focus on the essential properties first. No complete requirement specification is needed.
- It provides diagnostic information and counterexamples in case a property is invalidated, which is useful for fault diagnosis.

In [Cimatti et al., 2003], the authors treat the diagnosability analysis as a model checking problem. A copy of the system is made to build a twin plant. The system is undiagnosable if there exist two same observable scenarios in the original and copy system respectively, such that the one brings the system to a faulty state and the other brings to the normal.

In [Grastien, 2009], the author presents a symbolic-based approach to test diagnosability. The search can be performed in a classical forward manner, or in a backward manner which potentially avoids exploring all the search space of

3. LITERATURE REVIEW

the DES. This approach can also be mixed with a decentralised computation, which allows early detection of diagnosability and reduction of the search space in general. Thus the approach shows its advantages when comparing with the existing enumerative approaches.

In [Huang et al., 2004], the authors study diagnosis of DESs modeled in the rule-based modeling formalism. An attractive feature of rule-based model is its compactness. A motivation for the work presented is to develop failure diagnosis techniques that are able to exploit this compactness. In this regard, they develop symbolic techniques for testing diagnosability and computing a diagnoser. Diagnosability test is shown to be an instance of ldt order temporal logic model-checking. An on-line algorithm for diagnoser synthesis is obtained by using predicates and predicate transformers.

[Peres and Ghazel, 2013] discusses the diagnosis issue using a unique logical framework called μ -calculus. Diagnosability analysis is performed through the computing of successive logical relations. The implementation consists of a DBMS-architecture (Database Management System) where system behavior is encoded as a set of relational tables and diagnosability investigation is performed through an ordered sequence of queries on these tables [Ghazel et al., 2012].

As an approach using exhaustive search, model-checking suffers from the state-space explosion problem, i.e., the number of states needed to model the system accurately may easily exceed the amount of available computer memory. Models of realistic systems may still be too large to fit in memory.

3.4 Literature on Diagnosis Software Tools

In this section, we review a few works on the development of diagnostic software tools for DESs, and the comparisons between them.

UMDES [Lafortune, 2000] is a library of C routines for the study of DESs modeled by finite-state automata (FSA). The tool provides manipulation of FSA, operations of supervisory control theory and failure diagnosis. The tool DESUMA [Ricker et al., 2006] is an integration of the UMDES library with the graphical environment GIDDES for visualizing discrete event systems. DESUMA allows the user to perform a variety of manipulations on DES modeled by finite state

automata, such as model editing, diagnosability analysis, verification, control under full and partial observation, and decentralized control.

In [Stamp et al., 2006], the authors develop a tool for Discrete-Event Control And Diagnosis Analysis, called DECADA. DECADA checks diagnosability of an automaton model under partial observation.

[Cabasino et al., 2011] develops a software platform for the integration of discrete event systems tools. The objective of this software platform is to integrate several tools dealing with Petri nets and automata. The purpose is twofold: first to allow for a rigorous comparison of the methods and algorithms developed by the DISC project partners, and second to provide a packaged tool which would facilitate transfer of these techniques to the end users. The interchange format is compliant to the ISO standard Petri Net Markup Language (PNML). The platform includes a series of plug-ins and adapters to manipulate/transform the different file formats supported by the platform.

It is worth noticing that in [Cabasino et al., 2012b], a comparison of three tools for checking diagnosability is performed: UMDES-LIB, PN_DIAG and PN_DIAG_UNBOUNDED.

3.5 Conclusion

This chapter has recalled the existing study works on fault diagnosis of DESs in both untimed and timed contexts. First, it is shown that the approaches based on existing automata suffer from the inherent state explosion problem. Thus, we will focus on coping with this problem in the PN framework using on-the-fly and incremental techniques. Secondly, in the timed context, the diagnosability of TPNs is still open, which will be discussed in this work.

3. LITERATURE REVIEW

Chapter 4

Untimed PN-Based Diagnosis of DES

In this chapter, we aim at developing *on-the-fly* and *incremental* techniques for fault diagnosis in order to cope with the state explosion problem when dealing with complex systems. As will be shown in this chapter, analyzing on the basis of a known automata or building the whole reachability graph is actually unnecessary in fault diagnosis analysis. Instead, our proposed approach can generally show more efficiency compared with existing approaches. The motivation for using these techniques will be illustrated in Section 4.1.

In Sections 4.2 – 4.6, we will discuss the fault diagnosis of DES modeled by LPNs, where faults correspond to unobservable transitions. Based on several new concepts that we introduce, a tree-like structure holding both the markings and their related information of fault occurrences is elaborated. This structure, called *FM-set tree*, is computed on the fly while checking K -diagnosability on the basis of a recursive algorithm that we propose [Liu et al., 2012]. Moreover, by extension, we transform the classic diagnosability problem into a series of K -diagnosability problems [Liu et al., 2014b], where K increases progressively. Additionally, when the system is K -diagnosable, the online diagnosis is performed on the basis of a diagnoser which is obtained from the FM-set tree in a straightforward way. Finally, comparative simulation is performed using the OF-PENDA software tool that we have developed to prove the correctness and the efficiency of our approach.

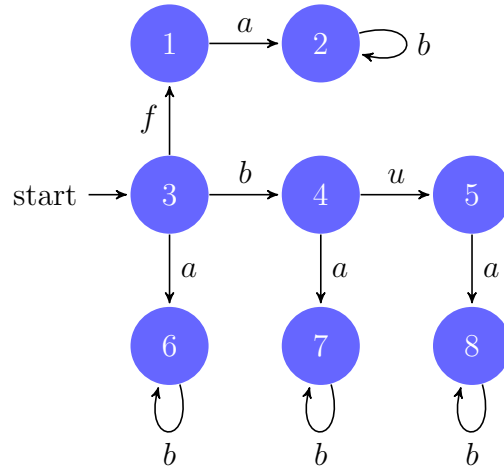
4.1 Motivation

4.1.1 On-the-Fly Analysis Technique

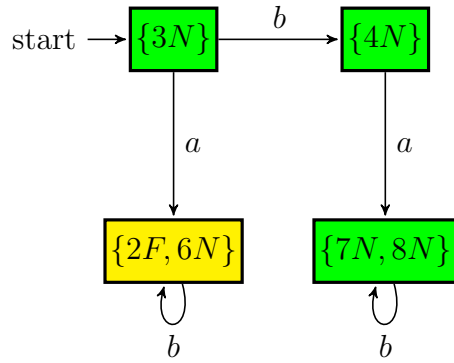
For most of the previous approaches reviewed, diagnosability analysis is composed of two stages. First, advanced models are developed for extracting the necessary information for diagnosability analysis from the original model, e.g., diagnoser automata [Sampath et al., 1995], verifier automata [Yoo and Lafortune, 2002b], verifier (Petri) nets [Cabasino et al., 2012a], linear inequalities [Basile et al., 2012], etc. Secondly, diagnosability analysis is tackled based on the structure analysis of the plant (or by solving mathematical models), e.g., through checking the existence of certain specific states or cycles, verifying the existence of linear inequalities solutions, etc. Traditionally, the two stages are proceeded independently and sequentially. The analysis of advanced models is performed after their state spaces have been completely generated. This presents the state explosion problem when dealing with large systems. Here, we will tackle this problem by using *on-the-fly* techniques [Schwoon and Esparza, 2005], since such techniques have the following advantages:

1. On-the-fly techniques can save memory resources. Generally, on-the-fly techniques permit us to generate and investigate only a part of the state space to find solutions, unlike the classic enumerative approaches which have to build the whole state space *a priori*. On-the-fly exploration techniques do not reduce the complexity of the original algorithms, but they do save memory resources in general, depending on the system structure and on the searching strategy. For example, using the classic diagnoser approach to analyze the diagnosability of the automaton in Figure 4.1(a) requires building *a priori* the corresponding diagnoser in Figure 4.1(b). Actually, generating a part of the diagnoser (cf. states $\{3N\}$ and $\{2F, 6N\}$ in Figure 4.1(b)) is sufficient to conclude the undiagnosability, since an indeterminate cycle (the self-loop on diagnoser state $\{2F, 6N\}$) is found.
2. On-the-fly techniques can save computing time. On the one hand, on-the-fly exploration terminates as soon as some specific features are found (cf. Figure 4.2(b)), which requires less time than investigating the whole state

space (cf. Figure 4.2(a)). On the other hand, for two-stage analysis, such as our approaches that will be given in the following sections, the advanced models can be derived and analyzed step by step as the on-the-fly building of the basic models (cf. Figure 4.2(c)), rather than being analyzed after building the whole basic models (cf. Figure 4.2(a) and 4.2(b)), which can save time from both analysis stages.



(a) G

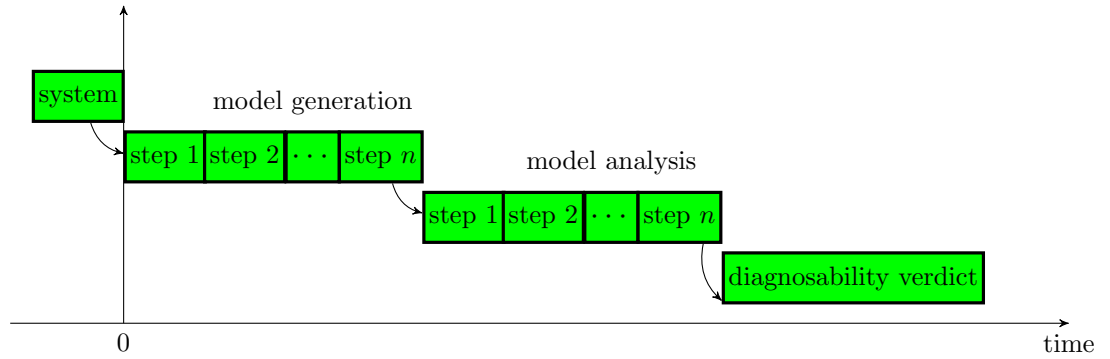


■ normal diagnoser state
■ F-uncertain diagnoser state

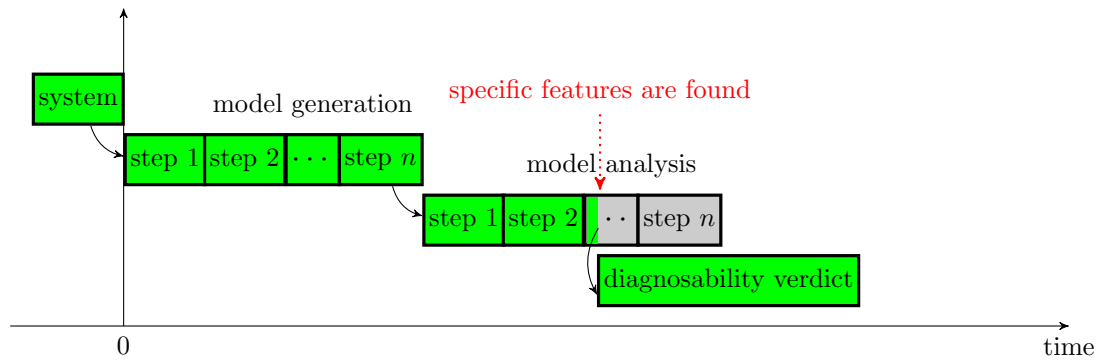
(b) $Diag(G)$

Figure 4.1: An automaton G and its diagnoser $Diag(G)$

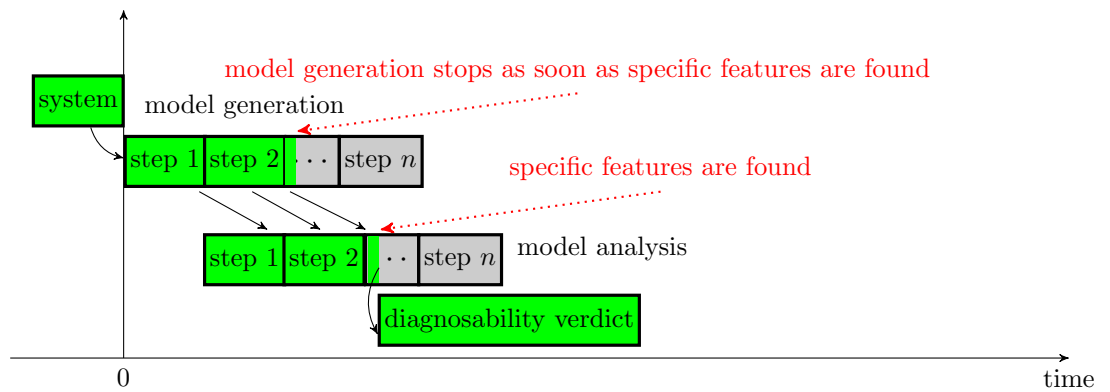
4. UNTIMED PN-BASED DIAGNOSIS OF DES



(a) Two-stage enumerative analysis



(b) Two-stage on-the-fly analysis



(c) Two-stage parallel and on-the-fly analysis

Figure 4.2: Three types of diagnosability analysis procedures

-
3. On-the-fly techniques can deal with some unbounded systems, since they return a verdict as soon as some specific features are found, instead of investigating the whole state space. Take the unbounded PN in Figure 4.3 (where t_1, t_2, t_4 and t_5 are observable transitions and t_3 is the only unobservable fault transition) for example, the PN is diagnosable, since the occurrence of event b starting from the initial marking proves the occurrence of fault f without further investing the component (t_5) which induces the unboundedness of the net.

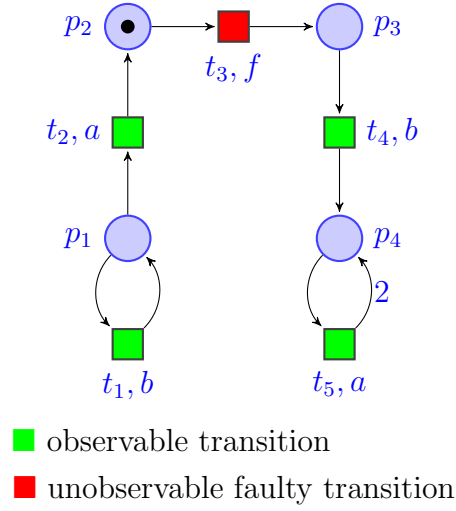


Figure 4.3: An unbounded PN

4.1.2 Incremental Analysis Technique

An *Incremental* method [Koenig et al., 2004] is a search technique that reuses the information from previous searches when some parameters change. Generally, it is faster than performing the search for each changed parameter from scratch. The analysis of the k^{th} step is based on the search result of the $(k - 1)^{th}$ step. Different from other speeding up searches, it can guarantee finding the shortest paths.

We will apply this technique in our diagnosability analysis, since it can bring the following advantages:

4. UNTIMED PN-BASED DIAGNOSIS OF DES

1. It is suitable for the step-by-step analysis in which the current analysis reuses the previous information. The classic diagnosability can be analyzed based on incrementally investigating K -diagnosability with increasing the value of K , and the K_{min} value which ensures the diagnosability will be eventually found (for diagnosable systems), as will be discussed in Section 4.4.1. Note that some ILP-based approaches [Basile et al., 2012; Wen et al., 2005] have to rebuild and solve the equation system (or inequalities) when seeking out K_{min} , without using the previous search results.
2. It is a skillful technique to speed up the search procedure. It should be used for bounded systems so that the search can terminate well. In particular, it can be used for unbounded systems when some conditions for terminating the search exist, as the analysis on unbounded PNs in Section 4.6.1.3.
3. Incremental techniques can be used with on-the-fly analysis to perform an efficient analysis. Both techniques do not change the computation complexity. However, they improve the searching efficiency when dealing with real systems, as will be shown in Section 4.6.

4.2 Mathematical Representation of LPNs to Check Diagnosability

We choose the LPN as the modeling notation in our study, since the LPN presents more general properties, besides those of the PN mentioned at the beginning of Section 2.1.2. As illustrated in Example 6, different transitions t_2 and t_3 can be associated with the same event b , which implies that, in reality, different behavior can be monitored using the same sensor. This will be of great significance when discussing sensor optimization problems in the future. Besides, LPNs can describe systems in the level of both structure (LPN graph) and state space (reachability graph). In the contrary, automata depict only the state space of systems, without the representation in the structure level.

In this section, the structure of a PN, as well as its dynamics, can be thoroughly described by classic mathematical representation with the help of *mark-*

ings, incidence matrix and state equation. This formulation, however, is insufficient for featuring LPNs, since there is no characterization of events. In this context, we propose a novel mathematical representation for LPNs, based on some new notions that we will introduce, namely *event-mapping matrix*, *event marking* and *extended state equation*, to both make the mapping relationship between transitions and events explicit, and record the event occurrences.

4.2.1 Extended Incidence Matrix

In order to characterize the mapping relationship between transitions and events for LPNs in a mathematical way, we first introduce the so-called event-mapping matrix.

Let $N_L = (N, M_0, \Sigma, \varphi)$ be an LPN, with set $\Sigma' \subseteq \Sigma$ containing only the events that we are interested in. For example, in event-based diagnosis Σ' contains all observable and fault events without considering harmless unobservable events, since all the harmless unobservable events are useless in the diagnosability analysis based on ϵ -reduction, as will be discussed in Section 4.2.3.

Definition 13 Let $\Sigma' = \{e_1, e_2, \dots, e_{|\Sigma'|}\}$. An event-mapping matrix is a $|\Sigma'| \times |T|$ matrix C_e , where $C_e(i, j) = 1$ if $\varphi(t_j) = e_i \in \Sigma'$, otherwise $C_e(i, j) = 0$.

When $\Sigma' = \Sigma$, matrices $C_e, Pre, Post$ and initial marking M_0 , provide a complete description of the LPN structure. That is, we can rebuild an LPN by employing the above matrices.

Example 12 Figure 4.4 is an LPN, where $\Sigma = \{a, b, u, f_1, f_2, f_3\}$, $\varphi(t_1) = f_1$, $\varphi(t_2) = f_2$, $\varphi(t_3) = f_3$, $\varphi(t_4) = u$, $\varphi(t_5) = \varphi(t_7) = \varphi(t_{10}) = a$, $\varphi(t_6) = \varphi(t_8) = \varphi(t_9) = b$.

We also denote by $\Sigma_o = \{a, b\}$, $\Sigma_u = \{u\} \cup \Sigma_f$, $\Sigma_f = \Sigma_{F_1} \cup \Sigma_{F_2}$, $\Sigma_{F_1} = \{f_1, f_3\}$, $\Sigma_{F_2} = \{f_2\}$, $\Sigma' = \Sigma_o \cup \Sigma_f$, which will be used in the sequel.

4. UNTIMED PN-BASED DIAGNOSIS OF DES

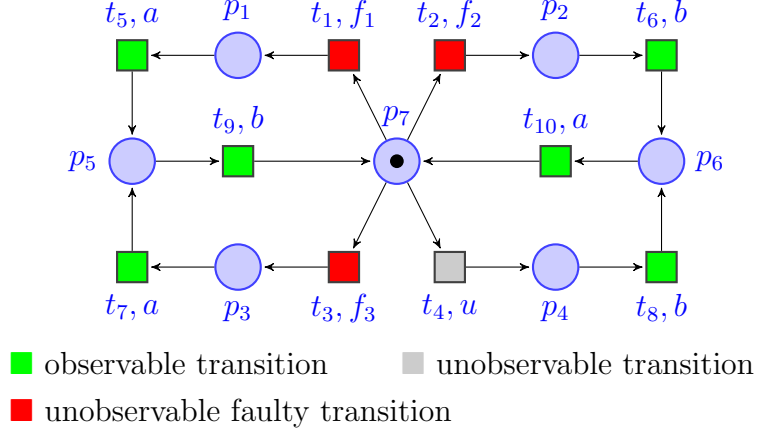


Figure 4.4: An LPN

The event-mapping matrix of this LPN is:

$$C_e = \begin{matrix} & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} \\ \begin{matrix} a \\ b \\ f_1 \\ f_2 \\ f_3 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{matrix} \quad (4.1)$$

Given an LPN, we can obtain a compact representation of both the incidence matrix and the event-mapping matrix.

Definition 14 An extended incidence matrix C_x is the orderly composition of the incidence matrix and the event-mapping matrix:

$$C_x = \begin{bmatrix} C \\ C_e \end{bmatrix} \quad (4.2)$$

Example 13 The extended incidence matrix of the LPN in Figure 4.4 is:

$$C_x = \begin{bmatrix} C \\ C_e \end{bmatrix} = \begin{array}{c|cccccccccc} & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} \\ \hline p_1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ p_2 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ p_3 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ p_4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ p_5 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 \\ p_6 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 \\ p_7 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline a & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ b & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ f_1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ f_2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ f_3 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \quad (4.3)$$

4.2.2 Event Marking

In the following definition, we assign to a given marking M a vector, called *event-marking*, which holds some event occurrences when the system state progresses from M_0 to M through a given sequence σ .

Definition 15 An event marking is a vector $EM \in \mathbb{N}^{|P|+|\Sigma'|}$ defined by:

$$EM = \begin{bmatrix} M \\ E \end{bmatrix} \quad (4.4)$$

where:

- $M \in \mathbb{N}^{|P|}$ is a marking such that $\exists \sigma \in T^*, M_0 \xrightarrow{\sigma} M$;
- $E \in \mathbb{N}^{|\Sigma'|}$ is an eventing vector $E = C_e \cdot \pi(\sigma)$.

The initial event marking is defined by:

$$EM_0 = \begin{bmatrix} M_0 \\ E_0 \end{bmatrix} = \begin{bmatrix} M_0 \\ \vec{0} \end{bmatrix} \quad (4.5)$$

4. UNTIMED PN-BASED DIAGNOSIS OF DES

We denote by \mathcal{E} the set of event markings.

An event marking records the occurrences of events (in Σ') from M_0 to M through some feasible sequence σ in its component E . Note that for a given marking M , E is not unique since M may be reached by different sequences, as will be shown in Example 16.

Example 14 *The initial event marking of the LPN in Figure 4.4 is:*¹

$$EM_0 = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 \mid 0\ 0\ 0\ 0]^T \quad (4.6)$$

Note that an LPN structure can be completely specified with Pre , $Post$, M_0 and the event-mapping matrix. In other words, an LPN can be rebuilt using the given event-mapping matrix and the initial marking, together with Pre and $Post$ matrices. Besides, these notations can be used to compute the dynamics of LPNs, as will be introduced in the following section.

4.2.3 Extended State Equation

The extended incidence matrix makes it possible to characterize the dynamics of an LPN, while giving both the distribution of tokens and the number of event occurrences.

We compute the successive event markings of EM by:

$$EM' = EM + C_x \cdot \pi(\sigma) \quad (4.7)$$

and we say that EM' is *reachable* from EM upon σ , written as $EM \mid \sigma > EM'$. We denote by $R(EM)$ the set of all event markings reachable from EM . Note that $R(EM)$ may be infinite for a live and bounded LPN, since the firing of any transition labeled with an event in Σ' adds 1 to the corresponding component in the event marking. Also, the set of event markings can be denoted by $\mathcal{E} = R(EM_0)$.

In particular, if we replace Σ' by T , then we describe the behavior of an unlabeled PN, where firing any transition does not generate events. In this case,

¹ M^T is the transpose of matrix M .

an event marking

$$EM = \begin{bmatrix} M \\ E \end{bmatrix}$$

records the firing number of each transition in the sequence σ having lead to M from M_0 .

In event-based diagnosis, we focus on the occurrences of observable and fault events that we are interested in, without considering the harmless unobservable events. If we take into account m fault classes $\Sigma_f = \biguplus_{j=1}^m \Sigma_{F_j}$, then $\Sigma' = \Sigma_o \uplus \Sigma_f$.

For the sake of clarity and convenience, let C_e be the orderly composition of matrices C_o and C_f ,

$$C_e = \begin{bmatrix} C_o \\ C_f \end{bmatrix} \quad (4.8)$$

where:

- C_o is a $|\Sigma_o| \times |T|$ matrix, $C_o(i, j) = 1$ if $[\varphi(t_j) = e_i] \wedge (e_i \in \Sigma_o)$, otherwise $C_o(i, j) = 0$;
- C_f is an $m \times |T|$ matrix, $C_f(i, j) = 1$ if $\varphi(t_j) \in \Sigma_{F_i}$, otherwise $C_f(i, j) = 0$.

Example 15 *The extended incidence matrix of the LPN in Figure 4.4, while considering two classes of faults $\Sigma_{F_1} = \{f_1, f_3\}$ and $\Sigma_{F_2} = \{f_2\}$, is then:*

$$C_x = \begin{bmatrix} C \\ C_o \\ C_f \end{bmatrix} = \begin{array}{c|cccccccccc} & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} \\ \hline p_1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ p_2 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ p_3 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ p_4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ p_5 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 \\ p_6 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 \\ p_7 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 1 & 1 \\ \hline a & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ b & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ \hline \Sigma_{F_1} & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \Sigma_{F_2} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \quad (4.9)$$

4. UNTIMED PN-BASED DIAGNOSIS OF DES

Note that Matrix 4.9 is different from Matrix 4.3, since we combine here f_1 and f_3 in the same fault class.

Similarly, we denote an event marking by the orderly composition,

$$EM = \begin{bmatrix} M \\ E \end{bmatrix} = \begin{bmatrix} mark(EM) \\ event(EM) \end{bmatrix} = \begin{bmatrix} mark(EM) \\ obs(EM) \\ fault(EM) \end{bmatrix} \quad (4.10)$$

where:

- $mark$ is a marking projection, $mark: \mathcal{E} \rightarrow \mathbb{N}^{|P|}$,

$$mark(EM) = [EM_1, \dots, EM_{|P|}]^\tau;$$

- $event$ is a projection relative to the considered events, $event: \mathcal{E} \rightarrow \mathbb{N}^{|\Sigma_o|+m}$,

$$event(EM) = [EM_{|P|+1}, \dots, EM_{|P|+|\Sigma_o|+m}]^\tau;$$

- obs is a projection relative to the observable events, $obs: \mathcal{E} \rightarrow \mathbb{N}^{|\Sigma_o|}$,

$$obs(EM) = [EM_{|P|+1}, \dots, EM_{|P|+|\Sigma_o|}]^\tau;$$

- $fault$ is a projection relative to the considered fault classes, $fault: \mathcal{E} \rightarrow \mathbb{N}^m$,

$$fault(EM) = [EM_{|P|+|\Sigma_o|+1}, \dots, EM_{|P|+|\Sigma_o|+m}]^\tau.$$

Example 16 Considering the LPN in Figure 4.4, for sequence $\sigma = t_1 t_5 t_9 t_3 t_7 t_9$, we have $EM_0 [t_1 > EM_1 [t_5 > EM_2 [t_9 > EM_3 [t_3 > EM_4 [t_7 > EM_5 [t_9 > EM_6$ and $EM_6 = EM_0 + C_x \cdot \pi(\sigma)$, where the event markings generated successively are given in Table 4.1.

$$mark(EM_0) = mark(EM_3) = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1]^\tau$$

$$obs(EM_0) = obs(EM_1) = [0 \ 0]^\tau$$

Table 4.1: Event markings in Example 16

i	EM_i
0	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^\tau$
1	$[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^\tau$
2	$[0\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0]^\tau$
3	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0]^\tau$
4	$[0\ 0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 2\ 0]^\tau$
5	$[0\ 0\ 0\ 0\ 1\ 0\ 0\ 2\ 1\ 2\ 0]^\tau$
6	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 2\ 2\ 2\ 0]^\tau$

$$obs(EM_2) = [1\ 0]^\tau$$

$$fault(EM_1) = fault(EM_2) = fault(EM_3) = [1\ 0]^\tau$$

Note that one can easily deduce if an observable event has been generated between two successive event markings, just by comparing their *obs* components.

Example 17 *By looking at the event components of the event markings generated successively in Table 4.1, we can state that a sequence of transitions $\sigma' = t'_1 t'_2 t'_3 t'_4 t'_5 t'_6$ exists, such that $EM_0 [t'_1 > EM_1 [t'_2 > EM_2 [t'_3 > EM_3 [t'_4 > EM_4 [t'_5 > EM_5 [t'_6 > EM_6$, $\varphi(t'_1), \varphi(t'_4) \in \Sigma_{F_1}$, $\varphi(t'_2) = \varphi(t'_5) = a$, $\varphi(t'_3) = \varphi(t'_6) = b$.*

As a side note, we can observe that an event marking also provides a suitable representation for diagnosis based on the observation of both places (markings) and transitions, as in [Ru and Hadjicostis, 2009].

4.2.4 Homomorphic Structure for Event Markings: LPN with Event Counters

As presented in the previous section, an event marking records both the marking and the corresponding event occurrence information. However, event markings are only an mathematical representation. Actually, an event marking can be visually presented by the structure that we call *event counter*.

4. UNTIMED PN-BASED DIAGNOSIS OF DES

Definition 16 *Given an LPN N_L , an event counter relative to event $e \in \Sigma$ is a place p_e with an input arc from all transitions labeled with e .*

In other terms, the number of occurrences of e can be directly read from the number of tokens in p_e .

Definition 17 *Given an LPN $N_L = (P, T, Pre, Post, M_0, \Sigma, \varphi)$ and a set of events $\Sigma' \subseteq \Sigma$, the corresponding LPN with event counters (LPN-EC) is a tuple $N_L^{\Sigma'} = (P', T, Pre, Post', M'_0, \Sigma, \varphi)$, where:*

- $P' = P \cup P^{\Sigma'}$ with $P^{\Sigma'}$ a finite set of places assigned to Σ' ;
- $Post' = Post \cup Post^{\Sigma'}$ with $Post^{\Sigma'} : T \times P' \rightarrow \{1\}$ is the mapping that gives the arcs linking transitions to the places in P' ;
- M'_0 is the initial marking such that $M'_0(p) = M_0(p)$ for $p \in P$ and $M'_0(p) = 0$ for $p \in P^{\Sigma'}$.

Example 18 *Consider the LPN-EC with event counters in Figure 4.5, where the yellow places are the event counters of the original LPN as shown in Figure 4.4. Initially, each event counter contains no tokens, since there is no occurrence of any event at the initial marking.*

The LPN-EC can reach a new marking upon the firing of a sequence of transitions $\sigma = t_1 t_5 t_9 t_3 t_7 t_9$, yielding to the redistribution of tokens in the event counters, as shown in Figure 4.6: $M(p_a) = M(p_b) = 2$ shows that there are 2 occurrences of events a and b respectively in the firing of σ . Likewise, there are 2 occurrences of faults in Σ_{F_1} and no occurrence of faults in Σ_{F_2} .

We now summarize the relation between event markings and event counters:

1. Event markings are the vectors to carry the information of both markings and event occurrences, in which some components indicate the number of tokens, and the others indicate the number of event occurrences. An event counter is a specific place added to the original LPN which translates the event occurrence information into the number of tokens that it holds. The marking and the event occurrence information of an LPN under consideration can then be depicted as the marking of the new LPN-EC.

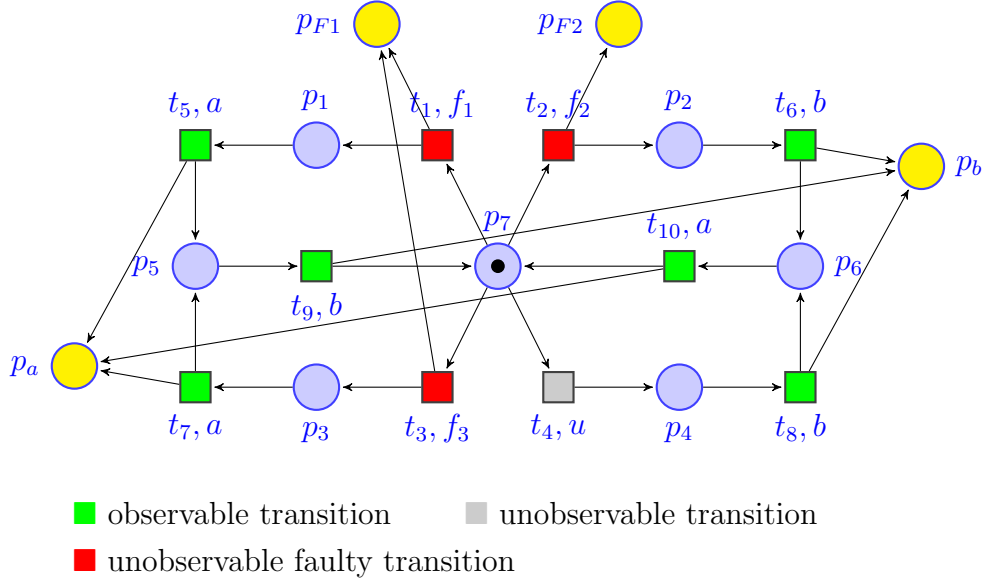


Figure 4.5: A LPN-EC

2. Using event counters requires introducing additional places and the necessary connecting arcs into the LPN under consideration. The liveness of the LPN will not change after the introduction of event counters. Moreover, a bounded LPN may become unbounded after the addition of event counters, as in Example 18.

Note that we can find, for any event marking of an LPN, the same vector (marking) in its corresponding LPN-EC, since they naturally carry the same information but in a different structure. Thus, we say that an LPN-EC is an extended LPN, whose markings represent both markings of the original LPN and the record of event occurrences. Both event markings and event counters can be employed for monitoring and diagnosing an LPN model as needed.

As a side note, a similar structure can be found in [Cabasino et al., 2012a]. The authors add a place linked with some transitions (not necessarily labeled with the same event) to an existing PN (verifier net) in order to record the steps after a fault for K -diagnosability analysis.

4. UNTIMED PN-BASED DIAGNOSIS OF DES

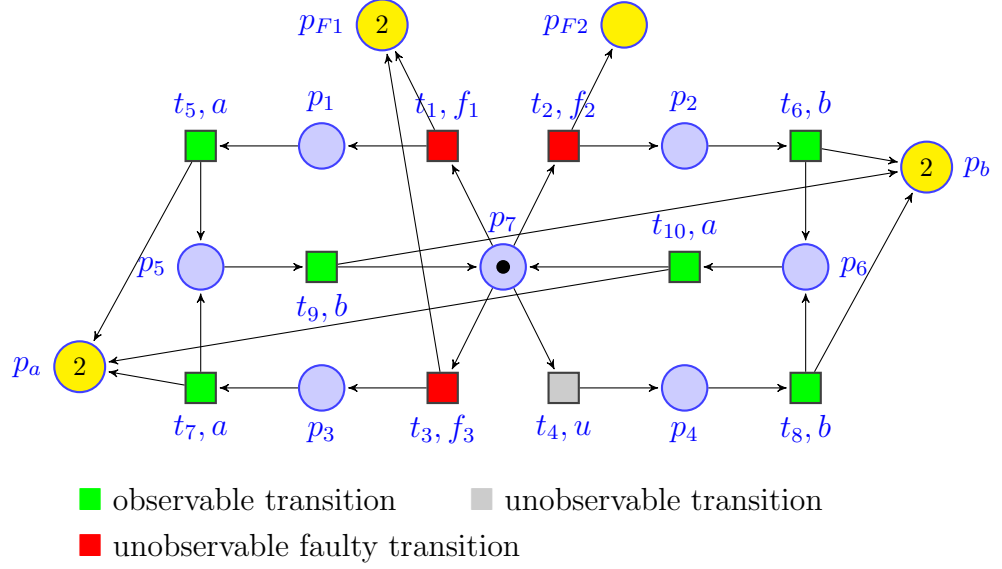


Figure 4.6: The reached marking of the LPN-EC in Figure 4.5 by the firing of $t_1 t_5 t_9 t_3 t_7 t_9$

4.3 Verification of K -Diagnosability

Before discussing the diagnosability issue, we make the following assumptions:

1. The LPN is live and bounded;
2. No infinite feasible sequence of unobservable transitions exists;
3. Faults are permanent, i.e., when a fault occurs the system remains definitively faulty.

4.3.1 Fault Marking (FM)

By extension, we introduce fault marking for a more compact representation to check diagnosability of a given class of faults.

Definition 18 A fault marking of event marking EM w.r.t a fault class Σ_{F_i} ($1 \leq$

$i \leq m$), is a vector $FM^i \in \mathbb{N}^{|P|+1}$, defined as follows:

$$FM^i(EM) = \begin{bmatrix} mark(FM^i(EM)) \\ fault(FM^i(EM)) \end{bmatrix}$$

where:

- $mark(FM^i(EM)) = mark(EM)$;
- $fault(FM^i(EM)) = \begin{cases} 1 & \text{if } [fault(EM)]^i > 0 \\ 0 & \text{otherwise} \end{cases}$

where for a given vector $\vec{x} = [x_1, \dots, x_{|\vec{x}|}]^\tau$, $[\vec{x}]^k$ denotes the k^{th} entry x_k .

In other words, $fault(FM^i)$ is the tag relative to the occurrence of fault events belonging to Σ_{F_i} from M_0 to marking $mark(EM)$: “0” indicates that no fault in Σ_{F_i} has occurred and “1” indicates that a fault in Σ_{F_i} has occurred at least once. The initial fault marking is:

$$FM_0^i = \begin{bmatrix} M_0 \\ 0 \end{bmatrix} \quad (4.11)$$

We denote by \mathcal{Q}^i the set of fault markings corresponding to Σ_{F_i} .

Example 19 For the LPN in Figure 4.4, the fault markings corresponding to fault class Σ_{F_1} are given in Table 4.2.

Compared with event markings, a fault marking carries the following information: a marking with its relative occurrence information for a given class of faults, without containing information relative to the occurrence of observable events. Therefore, two event markings with different *obs* components will correspond to the same fault marking, if their *mark* and *fault* components are the same. This implies that a fault marking may be reached from M_0 by different observable sequences. For event markings EM_1 and EM_2 satisfying $EM_1 \mid \sigma > EM_2$, $\sigma \in T^*$, we write $FM_1^i \mid \sigma > FM_2^i$, where FM_1^i and FM_2^i are their fault markings relative to Σ_{F_i} respectively.

Proposition 3 For a bounded LPN, given a fault class Σ_{F_i} , the FM^i set is finite.

4. UNTIMED PN-BASED DIAGNOSIS OF DES

Table 4.2: Fault markings corresponding to Σ_{F_1} in Example 19

j	FM_j^1
0	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^\tau$
1	$[1\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^\tau$
2	$[0\ 0\ 0\ 0\ 1\ 0\ 0\ 1]^\tau$
3	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 1]^\tau$
4	$[0\ 0\ 1\ 0\ 0\ 0\ 0\ 1]^\tau$
5	$[0\ 0\ 0\ 0\ 1\ 0\ 0\ 1]^\tau$
6	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 1]^\tau$

Proof. A bounded LPN N_L has a finite marking graph. Assuming the number of markings of N_L is p , then N_L has at most $2p$ fault markings, since a fault marking relative to Σ_{F_i} is composed of a marking combined with the fault information corresponding to Σ_{F_i} , namely 0 or 1. \square

One can directly guess the utility of introducing fault markings compared to event markings: to obtain a finite representation suitable for diagnosability analysis.

4.3.2 FM-Graph

For the purpose of diagnosis, we develop a fault marking graph (hereafter FM-graph) to record certain specific markings with their respective fault occurrence information, while dealing with single given class of faults.

Definition 19 *The FM-graph relative to fault class Σ_{F_i} and called FM^i -graph is a tuple $(\mathcal{N}, \Sigma_o, \delta, FM_0)$, where:*

- $\mathcal{N} \subseteq \mathcal{Q}^i$ is a set of FM^i nodes (fault markings);
- Σ_o is the set of observable events;
- $FM_0 = \begin{bmatrix} M_0 \\ 0 \end{bmatrix}$ is the initial node;

-
- $\delta: \mathcal{Q}^i \times \Sigma_o \rightarrow 2^{\mathcal{Q}^i}$ is the transition function of fault markings: given $FM_1^i \in \mathcal{Q}^i$ and $e \in \Sigma_o$, $\delta(FM_1^i, e) = \{FM_2^i \mid \exists \sigma \in T^* \text{ s.t. } P_o(\varphi(\sigma)) = \varphi(\sigma^{|\sigma|}) = e, \text{mark}(FM_1^i) \sqsubseteq \sigma \succ \text{mark}(FM_2^i), \text{fault}(FM_2^i) = 1 \text{ iff } [(\text{fault}(FM_1^i) = 1) \vee (\exists k, (\varphi(\sigma))^k \in \Sigma_{F_i})]\}$, as shown in Algorithm 1.

Algorithm 1 Algorithm for δ function

```

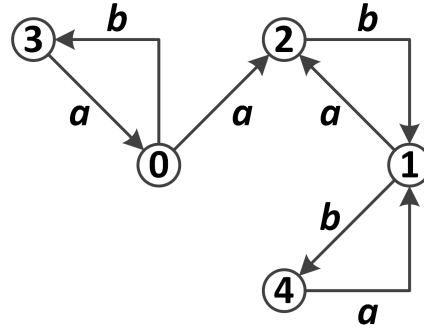
1: Input: a fault marking  $FM$  and an observable event  $e$ ;
2: Output:  $\mathcal{F} = \delta(FM, e)$ ;
3: function  $\delta(FM, e)$ 
4:    $\mathcal{F}_{con} \leftarrow \{FM\}$ ;  $\triangleright \mathcal{F}_{con}$  is the set of fault markings under consideration.
5:    $\mathcal{F} \leftarrow \emptyset$ ;  $\triangleright \mathcal{F}$  is the set of fault markings reached from  $FM$  immediately
   after the occurrence of  $e$ .
6:   for all  $y \in \mathcal{F}_{con}$  do
7:     if  $\text{mark}(y) \sqsubseteq t \succ$  then  $\triangleright t$  is enabled at marking  $\text{mark}(y)$ .
8:       if  $t \in T_o$  then
9:         if  $\varphi(t) = e$  then
10:           $z \leftarrow y \sqsubseteq t \succ$ ;  $\triangleright z$  is reached from  $y$  by the firing of  $t$ .
11:           $\mathcal{F} \leftarrow \mathcal{F} \cup \{z\}$ ;
12:        else  $\triangleright$  If  $t$  is an unobservable transition.
13:           $z \leftarrow y \sqsubseteq t \succ$ ;
14:           $\mathcal{F}_{con} \leftarrow \mathcal{F}_{con} \cup \{z\}$ ;
15:   return  $\mathcal{F}$ ;
```

An FM^i -graph is a directed non-deterministic graph. Each node indicates a given fault marking and each arc indicates an observable event. Note that an arc from one node to itself is permitted. For a given observable event e and two nodes FM_1^i and FM_2^i , at most one arc labeled with e may link FM_1^i to FM_2^i . Actually, an FM^i -graph can be treated as an ϵ -reduced observer automaton with fault tag. We take the same idea as in [Sampath et al., 1995] but in a different formulation (using fault marking vectors), in order to use mathematical tools to solve the problem.

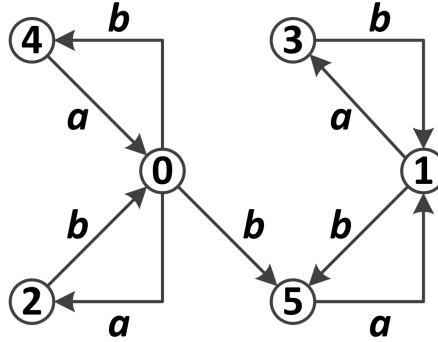
For a bounded LPN, the complete FM^i -graph w.r.t Σ_{F_i} can be built by a finite number of δ functions from the initial fault marking. However, in order to perform diagnosis analysis efficiently, we do not build the whole FM^i -graph in our approach. Instead, we build the FM^i -graph and the FM-set tree on the fly in parallel, as will be discussed in Section 4.3.6.

4. UNTIMED PN-BASED DIAGNOSIS OF DES

Example 20 Consider the LPN in Figure 4.4, the FM^1 -graph (resp. FM^2 -graph) is given in Figure 4.7(a) (resp. Figure 4.7(b)). The corresponding nodes of these graphs are given in Table 4.3.



(a) FM^1 -graph



(b) FM^2 -graph

Figure 4.7: The FM-graphs of the LPN in Figure 4.4

4.3.3 FM-Set

The formal definition of K -diagnosability is introduced in Section 2.1.5. Without loss of generality, we first discuss the K -diagnosability for one class of faults Σ_{F_i} . In the sequel, we will always reason about Σ_{F_i} . Thereby, for the sake of clarity, we will dismiss index i relative to the considered fault class. The generalization

Table 4.3: Fault markings in Example 20

j	FM_j^1	FM_j^2
0	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^\tau$	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^\tau$
1	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 1]^\tau$	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 1]^\tau$
2	$[0\ 0\ 0\ 0\ 1\ 0\ 0\ 1]^\tau$	$[0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]^\tau$
3	$[0\ 0\ 0\ 0\ 0\ 1\ 0\ 0]^\tau$	$[0\ 0\ 0\ 0\ 1\ 0\ 0\ 1]^\tau$
4	$[0\ 0\ 0\ 0\ 0\ 1\ 0\ 1]^\tau$	$[0\ 0\ 0\ 0\ 0\ 1\ 0\ 0]^\tau$
5	—	$[0\ 0\ 0\ 0\ 0\ 1\ 0\ 1]^\tau$

of our approach can be obtained just by repeating the same process for each class Σ_{F_i} .

We now introduce the following notations to help describe the K -diagnosability problem.

Let the FM-set power set be $\mathcal{X} = 2^\Omega$ and the initial FM-set $x_0 = \{FM_0\}$.

Definition 20 *The FM-set transition mapping $\lambda: \mathcal{X} \times \Sigma_o \rightarrow \mathcal{X}$ is defined as follows: given an FM-set $x \in \mathcal{X}$ and an observable event $e \in \Sigma_o$,*

$$\lambda(x, e) = \{FM' \mid \exists FM \in x, \sigma \in T^* \text{ s.t. } P_o(\varphi(\sigma)) = \varphi(\sigma^{| \sigma |}) = e, FM \sqsubset \sigma > FM'\}$$

Here, $\lambda(x, e)$ is the FM-set whose fault markings are reachable from those of x by executing a sequence of unobservable events (possibly empty) followed by observable event e . We extend this definition to the set of observable events Σ_o by defining the mapping $\Lambda: \mathcal{X} \rightarrow 2^\mathcal{X}$,

$$\Lambda(x) = \cup_{e \in \Sigma_o} \{\lambda(x, e)\}$$

For simplicity, we write $x \rightsquigarrow x'$ if $x' \in \Lambda(x)$.

Proposition 4 *A bounded LPN has a finite number of FM-sets.*

Proof. For a bounded LPN with p markings, the upper bound of its fault marking number is $2p$ (cf. Proposition 3). The maximum number of FM-sets will

4. UNTIMED PN-BASED DIAGNOSIS OF DES

not exceed the number of all fault marking combinations:

$$\begin{aligned}
 & C_{2p}^1 + C_{2p}^2 + \cdots + C_{2p}^{2p} \\
 &= (C_{2p}^0 + C_{2p}^1 + C_{2p}^2 + \cdots + C_{2p}^{2p}) - C_{2p}^0 \\
 &= 2^{2p} - 1
 \end{aligned}$$

□

Definition 21 *The tagging function $tag: \mathcal{X} \rightarrow \{F, N, U\}$ is defined as follows:*

$$tag(x) = \begin{cases} N & \text{if } \forall FM \in x, fault(FM) = 0 \\ F & \text{if } \forall FM \in x, fault(FM) = 1 \\ U & \text{otherwise} \end{cases}$$

An FM-set x is also said to be normal (resp. F -certain, F -uncertain) if $tag(x) = N$ (resp. F , U). Here, we use the same idea as in [Sampath et al., 1995].

The propagation of tags between FM-sets is shown in Figure 4.8, where an arrow indicates an observable event. For $x \rightsquigarrow x'$, if $tag(x) \in \{N, U\}$, we may have $tag(x') \in \{F, N, U\}$; whereas if $tag(x) = F$ then $tag(x') = F$, as faults are assumed to be permanent and, therefore, the F -certain tag is propagated to all the successive FM-sets. A similar idea has been presented as the fault propagation function of the diagnoser automata in [Sampath et al., 1995].

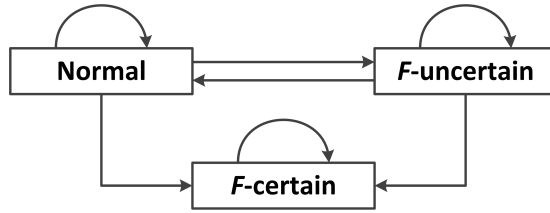


Figure 4.8: Fault propagation between FM-sets

Example 21 *Considering the LPN in Figure 4.4, for Σ_{F_1} , we have*

$$\lambda(x_0, a) = x_1, \lambda(x_0, b) = x_2$$

and then

$$\Lambda(x_0) = \{x_1, x_2\}$$

where, referring to Table 4.3:

$$x_0 = \{FM_0^1\}, x_1 = \{FM_3^1\}, x_2 = \{FM_4^1\}$$

4.3.4 FM-Set Tree

In order to represent both the reachability and fault propagation of an LPN model, we introduce a structure called *FM-set tree*, which is the basis of the diagnosability analysis.

An FM-set tree is a tree-like structure, where:

- the root node is the initial FM-set $x_0 = \{FM_0\}$;
- the subsequent nodes are the FM-sets reachable from x_0 by a finite number of operations of λ function;
- for a given node x , the set of its child nodes (direct successors) is $\Lambda(x)$.

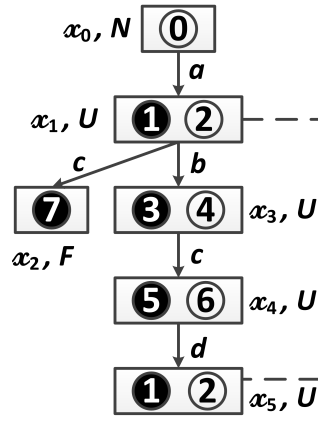
4.3.5 Conditions for K -Diagnosability

4.3.5.1 Equivalent Indeterminate Cycles in FM-Set Tree

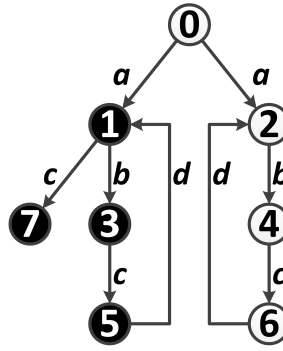
Let us recall that the condition for undiagnosability of an automaton is the existence of an F_i -indeterminate cycle in its diagnoser model [Sampath et al., 1995]. We will use this condition in the following diagnosability analysis.

While building the FM-set tree on the fly, as shown in Figure 4.9(a) (where faulty fault markings are indicated by black circles and the normal ones by white circles), a newly-generated F -uncertain node (here x_5) may be equal to one of its predecessors (here x_1). In this case, if two fault markings 1 and 2 exist in such a node, such that 1 is faulty, 2 is normal, and either of them has a path to itself in the FM-graph, i.e., 1, 3, 5 and 2, 4, 6 as shown in Figure 4.9(b), then we determine the existence of an indeterminate cycle and, therefore, the system is undiagnosable; otherwise, there is no indeterminate cycle, as shown in Figure 4.9(c), since a normal cycle 2, 4, 6 exists, but no faulty cycle exists.

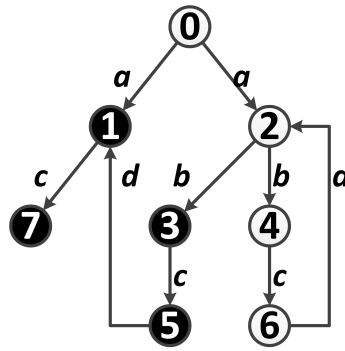
4. UNTIMED PN-BASED DIAGNOSIS OF DES



(a) FM-set tree



(b) FM-graph



(c) FM-graph

Figure 4.9: Checking an indeterminate cycle

4.3.5.2 Relation between Undiagnosability and Delay Values

For the sake of analyzing K -diagnosability, we introduce a *delay* function to record the number of successive F -uncertain FM-sets as the FM-set tree is processed.

Definition 22 *The function $delay: \mathcal{X} \rightarrow \mathbb{N}$ is defined as follows:*

- $delay(x_0) = 0$;
- for any newly-generated node $x' \in \Lambda(x)$, define $delay(x')$ as shown in Table 4.4;

Table 4.4: Definition of *delay* function for newly generated node

$tag(x)$	$tag(x')$	$delay(x')$
N	N or F	0
N	U	1
U	N or F	0
U	U	$delay(x) + 1$

- in particular, if a newly generated F -uncertain node x' has been given a delay value according to Table 4.4, and it also satisfies the following condition: $(x' = x'') \wedge [delay(x') > delay(x'')]$, where x'' is an existing node, then let $d = delay(x') - delay(x'')$. For x'' and each of its successor F -uncertain FM-sets y , update $delay(y)$ with $delay(y) + d$. This will be illustrated in the sequel in Figure 4.10 (cf. nodes x_1, x_3, x_4 and x_5).

We do not consider the case where $tag(x) = F$, since, as will be shown later when building the FM-set tree, the processing of a branch is stopped as soon as a faulty node (an FM-set tagged F) is obtained since, obviously, the subsequent states will all be faulty (permanent faults) and, online, a diagnosis verdict can immediately be given.

In simple terms, $delay(x)$ denotes the maximum number of successive F -uncertain FM-sets until x . In the on-the-fly construction of the FM-set tree, $delay(x)$ is dynamic, as it is updated while building the FM-set tree (in the case

4. UNTIMED PN-BASED DIAGNOSIS OF DES

where two equivalent uncertain nodes are met in the same branch). Indeed, when meeting the relation $x = x'$, we have to compare $\text{delay}(x)$ and $\text{delay}(x')$, and update delay values of relevant FM-sets.

Example 22 *Let us consider the FM-set tree under construction shown in Figure 4.10(a), where each arrow indicates an observable event. x_2 (resp. x_8) is the second F -uncertain node after a fault that may have occurred between x_0 and x_1 (resp. x_6 and x_7). When x_9 , the child node of x_8 , is generated, and as $x_9 = x_2$ and $\text{delay}(x_9) > \text{delay}(x_2)$, we update the delay value of x_2 and x_3 as shown in Figure 4.10(b).*

4.3.6 Algorithm for On-the-Fly Checking K -Diagnosability

We now develop our algorithm for checking K -diagnosability based on on-the-fly building of the FM-set tree.

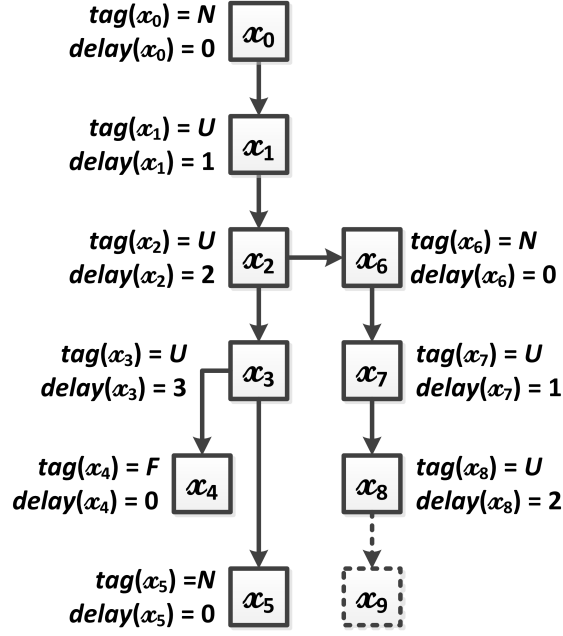
Algorithm 2 Checking K -diagnosability by on-the-fly building of FM-graph and FM-set tree in parallel

```

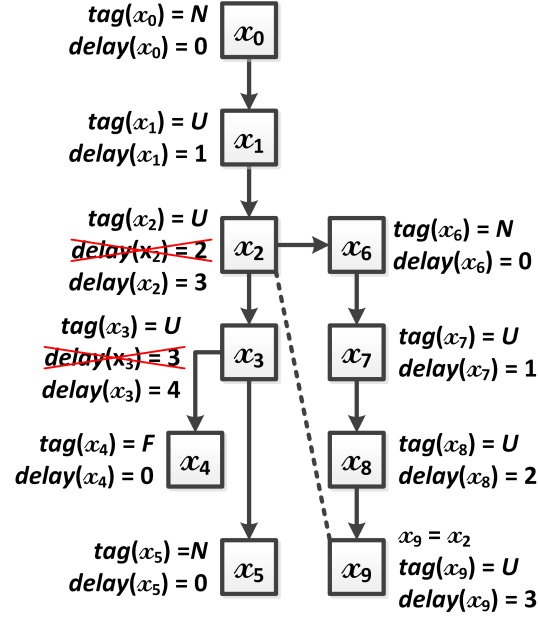
1: Input:  $K, N_L = (P, T, Pre, Post, \Sigma, \varphi, M_0), T_o, T_u, T_f$ .
2: Output:  $K$ -diagnosability of  $N_L$ ;
3:  $\mathcal{N} \leftarrow \emptyset$ ; ▷  $\mathcal{N}$  is the set of the FM-graph nodes.
4:  $\mathcal{A} \leftarrow \emptyset$ ; ▷  $\mathcal{A}$  is the set of the FM-graph arcs.
5:  $\mathcal{X}_v \leftarrow \{x_0\}$ ; ▷  $\mathcal{X}_v$  is the set of the FM-sets nodes.
6:  $\mathcal{A}_v \leftarrow \emptyset$ ; ▷  $\mathcal{A}_v$  is the set of the FM-sets arcs.
7:  $x \leftarrow x_0$ ;
8:  $\mathcal{T} \leftarrow$  the initial FM-set tree which contains only  $x_0$ ;
9:  $(\mathcal{T}', y, n) \leftarrow \text{KDIAG}(\mathcal{T}, x, K)$ ; ▷ cf. Algorithm 3
10: switch  $n$  do
11:   case 1
12:     assert( $N_L$  is  $K$ -diagnosable;);
13:   case 0
14:     assert( $N_L$  is not  $K$ -diagnosable;); ▷  $N_L$  may be  $K'$ -diagnosable for
        $K' > K$ .
15:   case -1
16:     assert( $N_L$  is not diagnosable;); ▷ An indeterminate cycle is found.

```

Proposition 5 *For a bounded LPN, KDIAG function in Algorithm 3 terminates and its diagnosability verdict is correct.*



(a) The FM-set tree after generating node x_8



(b) The FM-set tree after generating node x_9

Figure 4.10: Updating the *delay* values while generating an FM-set tree

4. UNTIMED PN-BASED DIAGNOSIS OF DES

Algorithm 3 Checking K -diagnosability by on-the-fly building of FM-graph and FM-set tree in parallel

```

1: Input: an FM-set  $x$  in the FM-set tree  $\mathcal{T}$ , and  $K$  relative to  $K$ -diagnosability;
2: Output: a triple  $(\mathcal{T}', y, n)$ , where  $\mathcal{T}'$  is the FM-set tree generated after running KDIAG function,  $y$  is the FM-set where the generation of an FM-set tree branch stops and  $n$  is the  $K$ -diagnosability verdict;
3: function KDIAG( $\mathcal{T}, x, K$ )
4:   for all  $e \in \Sigma_o$  do
5:      $x' \leftarrow \text{NEXTFMSET}(x, e)$  ;  $\triangleright x'$  is the child node of  $x$ .
6:     update  $\mathcal{T}$  to  $\mathcal{T}'$  by adding  $x'$  from  $x$  upon  $e$ ;
7:     if  $(x' \neq \emptyset) \wedge [\text{tag}(x') = N]$  then
8:       if  $(\forall x'' \in \mathcal{X}_v)(x' \neq x'')$  then  $\triangleright$  No equivalent node already exists.
9:          $\mathcal{X}_v \leftarrow \mathcal{X}_v \cup \{x'\}$ ;
10:         $(\mathcal{T}'', y, n) \leftarrow \text{KDIAG}(\mathcal{T}', x', K)$ ;
11:        if  $n \neq 1$  then
12:          return  $(\mathcal{T}'', y, n)$ ;
13:     if  $(x' \neq \emptyset) \wedge [\text{tag}(x') = U]$  then
14:       if  $\text{delay}(x') = K$  then
15:         return  $(\mathcal{T}', x', 0)$ ;  $\triangleright 0$  denotes that  $N_L$  is not  $K$ -diagnosable.
16:          $\triangleright$  However, it may be  $K'$ -diagnosable for  $K' > K$ .
17:       else
18:         if  $(\exists x'' \in \mathcal{X}_v)(x'' = x')$  then
19:           if  $x'$  is in an indeterminate cycle then  $\triangleright$  Use of function
20:              $\triangleright \text{path\_exists}$  from the library digraph [Rushton, 2012].
21:           return  $(\mathcal{T}', x', -1)$ ;  $\triangleright -1$  denotes  $N_L$  is not
22:            $\triangleright (K)$ -diagnosable due to the indeterminate cycle.
23:         else if  $\text{delay}(x') > \text{delay}(x'')$  then
24:            $d \leftarrow \text{delay}(x') - \text{delay}(x'')$ ;
25:           if  $\text{UPDDelay}(x'', d, K) = \text{FALSE}$  then
26:             return  $(\mathcal{T}', x', 0)$ ;  $\triangleright$  cf. Algorithm 5.
27:         else
28:            $\mathcal{X}_v \leftarrow \mathcal{X}_v \cup \{x'\}$ ;
29:            $(\mathcal{T}'', y, n) \leftarrow \text{KDIAG}(\mathcal{T}', x', K - 1)$ ;
30:           if  $n \neq 1$  then return  $(\mathcal{T}'', y, n)$ ;
31:   return  $(\mathcal{T}', x', 1)$ ;  $\triangleright 1$  denotes that  $N_L$  is  $K$ -diagnosable.

```

Proof. In Algorithm 3, we proceed, step by step with NEXTFMSET function, to build the FM-set tree in parallel with the FM-graph, on the fly, for solving K -diagnosability.

Algorithm 4 NEXTFMSET(): a subfunction of Algorithm 3 for computing the next FM-set while building the FM-graph on the fly

```

1: Input: an FM-set  $x$  and an observable event  $e$ ;
2: Output: FM-set  $x'$  which is reached from  $x$  immediately after  $e$ ;
3: function NEXTFMSET( $x, e$ )
4:    $x' \leftarrow \emptyset$ ;
5:   for all  $y \in x$  do
6:      $x' \leftarrow x' \cup \delta(y, e)$ ;            $\triangleright$  Function  $\delta$  is given in Algorithm 1.
7:      $\mathcal{N} \leftarrow \mathcal{N} \cup x'$ ;            $\triangleright \mathcal{N}$  is the set of the FM-graph nodes.
8:     for all  $z \in \delta(y, e)$  do
9:        $\mathcal{A} \leftarrow \mathcal{A} \cup \{(y, e, z)\}$ ;    $\triangleright \mathcal{A}$  is the set of the FM-graph arcs.
10:  return  $x'$ ;

```

Algorithm 5 UPDDelay(): a subfunction of Algorithm 3 for updating delay values

```

1: Input: an FM-set  $x$ , an integer  $d$  and  $K$  relative to  $K$ -diagnosability;
2: Output: FALSE if  $N_L$  is not  $K$ -diagnosable or TRUE if further investigation
   is needed;
3: function UPDDelay( $x, d, K$ )
4:   for all node  $z$  s.t.  $x \rightsquigarrow z$  do
5:     if  $\text{tag}(z) = U$  then
6:        $\text{delay}(z) \leftarrow \text{delay}(z) + d$ ;
7:       if  $\text{delay}(z) \geq K$  then
8:         return FALSE;            $\triangleright N_L$  is not  $K$ -diagnosable.
9:       else
10:        return UPDDelay( $z, d, K$ );
11:  return TRUE;

```

First, we will prove that the algorithm terminates for a bounded LPN. The investigation of a branch of FM-set tree is stopped, when:

1. An F -certain FM-set is generated;
2. An F -uncertain FM-set, whose delay value is $\geq K$, is generated (Line 14 – 16 and 23 – 26 of Algorithm 3);
3. A new normal FM-set is equal to a previous one (Line 8 of Algorithm 3);
4. A new F -uncertain FM-set is equal to a previous one (Line 18 – 26 of Algorithm 3).

4. UNTIMED PN-BASED DIAGNOSIS OF DES

In the on-the-fly construction of an FM-set tree, for any branch, one of the above conditions will be met sooner or later, since we consider live and bounded LPNs here, which means that the algorithm terminates well.

Secondly, we prove that this algorithm covers all the cases while constructing the FM-set tree on the fly. For any newly-generated node x' ,

1. If x' is F -certain, it is not necessary to consider its child nodes, as all of them will be F -certain according to the fault propagation relation illustrated in Figure 4.8. A fault will be definitively determined when the system is in such a state. In this case we stop proceeding the FM-set tree along this branch. Note that this case was consequently omitted in the above algorithm.
2. If x' is normal and
 - (a) If there is already an existing node x'' (in \mathcal{X}_v) such that $x' = x''$, then we stop investigating this branch, since x' would have the same branches as x'' , which has already been considered;
 - (b) Otherwise, we continue investigating this branch (Lines 8 – 12).
3. If x' is F -uncertain, there are many cases to be handled (Lines 13 – 30).
 - (a) If $\text{delay}(x') = K$, it means that until x' there are already K successive F -uncertain nodes, and then the fault cannot be detected after K observable events upon its occurrence. Thus, N_L is not K -diagnosable (Lines 14 – 16 of Algorithm 3);
 - (b) If $\text{delay}(x') < K$,
 - i. If there is an existing node x'' (in \mathcal{X}_v) such that $x' = x''$, (Lines 18 – 26 of Algorithm 3)
 - A. If x' is in an F_i -indeterminate cycle, therefore N_L is not (K -) diagnosable (Lines 20 – 22 of Algorithm 3);
 - B. Otherwise, we update the delay value of the related nodes, and check if one of these nodes x''' satisfying $\text{delay}(x''') \geq K$ exists (by function UpdDelay). If so, N_L is not K -diagnosable (Lines 23 – 26 of Algorithm 3);

-
- ii. Otherwise, we continue investigating this branch by recalling KDIAG function on the current node (Lines 27 – 30 of Algorithm 3).

Since all the possible cases are considered, and since the number of FM-sets is finite (cf. Proposition 4), we can be sure that our algorithm terminates well.

□

Example 23 *Let us analyze the K -diagnosability of LPN N_L in Figure 4.4. As illustrated through the solution process of Figure 4.11, we can conclude that N_L is 1-diagnosable w.r.t Σ_{F_1} . Similarly, according to Figure 4.12, N_L is not 1,2,3-diagnosable w.r.t Σ_{F_2} .*

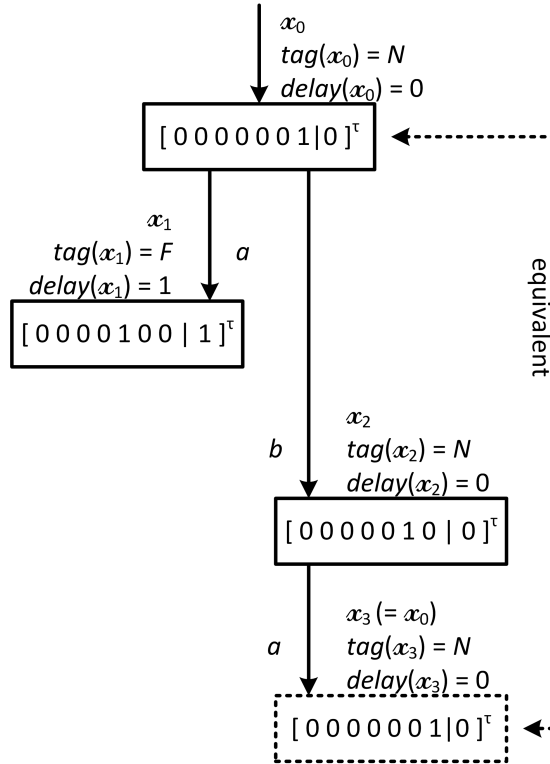
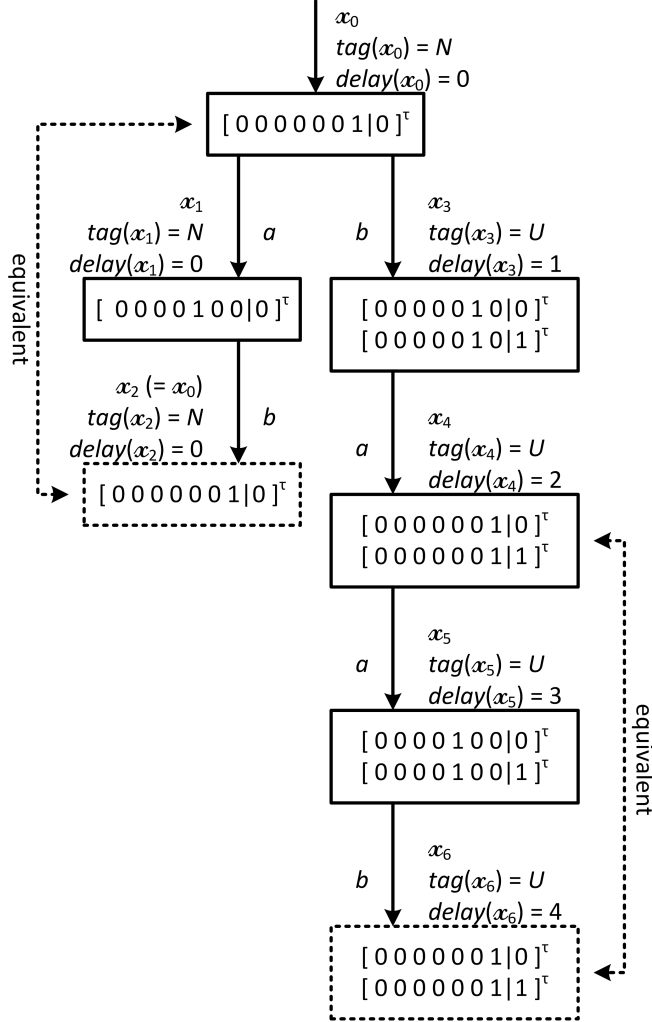


Figure 4.11: Solution process of K -diagnosability for Σ_{F_1}


 Figure 4.12: Solution process of K -diagnosability for Σ_{F_2}

4.4 Verification of Diagnosability

4.4.1 Algorithm for Checking Diagnosability

In this section, we show how to investigate classic diagnosability on the basis of K -diagnosability analysis.

For a K -diagnosable system A , it is obvious that: 1) A is K' -diagnosable for any $K' \geq K$, and 2) $K_{min} \in \mathbb{N}$ exists such that A is K_{min} -diagnosable and for all $K' < K_{min}$, A is not K' -diagnosable. In other words, K_{min} is the minimum

value to ensure diagnosability, i.e., any fault can be diagnosed within at most K_{min} steps (observable events) after its occurrence. Thus, finding K_{min} is of great significance while studying diagnosability.

We extend Algorithm 3 to solve the classic diagnosability problem (cf. Algorithm 6). Our goal here is twofold: first to determine if the system is diagnosable, and, if so, what the minimum value of K is for which the system is K -diagnosable and return the last FM-set tree for building the diagnoser in the future (cf. Section 4.5).

Algorithm 6 Checking diagnosability

```

1: Input:  $N_L = (P, T, Pre, Post, M_0, \Sigma, \varphi), T_o, T_f$ ;
2: Output: diagnosability verdict and  $(\mathcal{T}', K_{min})$  if  $N_L$  is diagnosable;
3:  $y \leftarrow x_0$ ;
4:  $K \leftarrow 0$ ;
5:  $n \leftarrow 0$ ;
6:  $\mathcal{T}' \leftarrow$  the initial FM-set tree which contains only  $x_0$ ;
7: while  $n = 0$  do ▷ If  $N_L$  is not  $K$ -diagnosable.
8:    $K \leftarrow K + 1$ ;
9:    $x \leftarrow y$ ;
10:   $\mathcal{T} \leftarrow \mathcal{T}'$ ;
11:   $(\mathcal{T}', y, n) \leftarrow \text{KDIAG}(\mathcal{T}, x, K)$ ; ▷ cf. Algorithm 3.
12: if  $n = 1$  then
13:   return  $(\mathcal{T}', K)$ ; ▷  $N_L$  is  $K_{min}$ -diagnosable where  $K_{min} = K$ .
14: ▷  $\mathcal{T}'$  is used for building the diagnoser, cf. Section 4.5.
15: else ▷ If  $n = -1$ .
16:   return 0; ▷  $N_L$  is not diagnosable.

```

As shown in Algorithm 6, checking diagnosability is performed on the fly while building the FM-set tree and the FM-graph. A notable advantage of this method is that, when checking $(K + 1)$ -diagnosability, the models (FM-graph and FM-set tree) generated while investigating K -diagnosability are reused, instead of completely restarting from scratch.

It should be noted that in Algorithm 6 we do not set an upper bound for K . It seems that K could be increased to infinity in the incremental research. Actually, the algorithm can terminate well, since the system under consideration is bounded and K cannot exceed the number of FM-set tree states.

4. UNTIMED PN-BASED DIAGNOSIS OF DES

Example 24 *Using Algorithm 6, we have analyzed the diagnosability of LPN N_L in Figure 4.4 w.r.t Σ_{F_2} . We conclude that N_L is not diagnosable for Σ_{F_2} since an F_2 -indeterminate cycle has been found when 4-diagnosability is investigated, after N_L has been concluded to be not 1, 2 or 3-diagnosable.*

4.4.2 Complexity and Effectiveness Analysis

Let us analyze the proposed algorithm in terms of memory complexity. According to Proposition 1 and 2, for a given class of faults, if the number of markings of the considered LPN is p (which is also the number of automaton states if we consider the reachability graph as an automaton), the number of nodes in the FM-graph will be $\leq 2p$, and the number of nodes in the FM-set tree will be $\leq 2^{2p}$ (cf. Proposition 3). In the worst case, when all the possible FM-sets are generated and investigated, i.e., when the FM-set tree states are enumerated, the complexity in terms of memory is equal to the diagnoser approach of [Sampath et al., 1995]. The number of the FM-set tree states is exponential to the number of LPN markings p .

However, in general, we do not build the whole FM-graph and FM-set tree, since we build them on the fly and keep only necessary nodes for diagnosis. In particular, an undiagnosable PN will be identified immediately after an indeterminate cycle is found, rather than continuing generating other FM-sets. Moreover, as soon as an F -certain node is met, the investigation of the current branch is stopped, since faults are permanent and, consequently, all the subsequent states will be faulty as well. The following example will show the difference in memory complexity between the diagnoser approach and ours.

Example 25 *Consider LPN N_L in Figure 4.13(a), whose reachability graph can be treated as automaton G in Figure 4.13(b), where the states in G are given in Table 4.5.*

Using the traditional diagnoser approach, diagnoser $\text{Diag}(G)$ (cf. Figure 4.14(b)) can be built based on observer $\text{Obs}(G)$ (cf. Figure 4.14(a)) and the system is determined to be undiagnosable due to the existence of an indeterminate cycle (the self loop on diagnoser state $\{2F, 6N\}$ in Figure 4.14(b)).

Table 4.5: The markings in Figure 4.13(b)

State of G	Marking of N_L
1	$[1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$
2	$[0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]^T$
3	$[0\ 0\ 1\ 0\ 0\ 0\ 0\ 0]^T$
4	$[0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T$
5	$[0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]^T$
6	$[0\ 0\ 0\ 0\ 0\ 1\ 0\ 0]^T$
7	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0]^T$
8	$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^T$

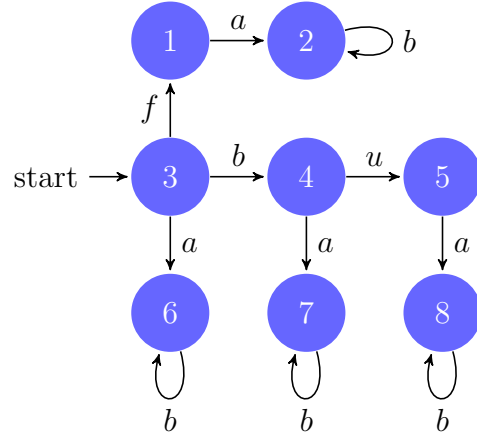
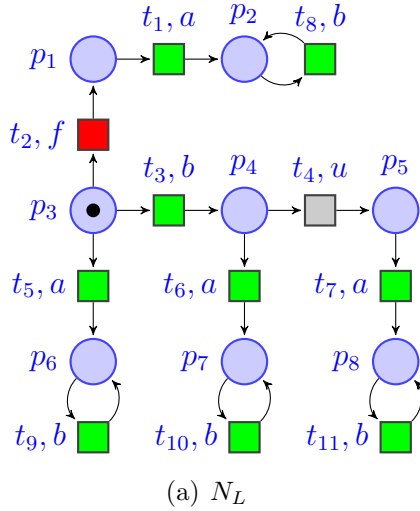


Figure 4.13: LPN N_L and its reachability graph G

The same verdict can be obtained when using our on-the-fly approach. The on-the-fly analysis requires building the FM-set tree (cf. Figure 4.15(b)) on the basis of the FM-graph (cf. Figure 4.15(a)), where the fault markings are given in Table 4.6. Here, we assume that the branches investigated first in the FM-graph (and accordingly in the FM-set tree) are the ones labeled by a . It is worth

4. UNTIMED PN-BASED DIAGNOSIS OF DES

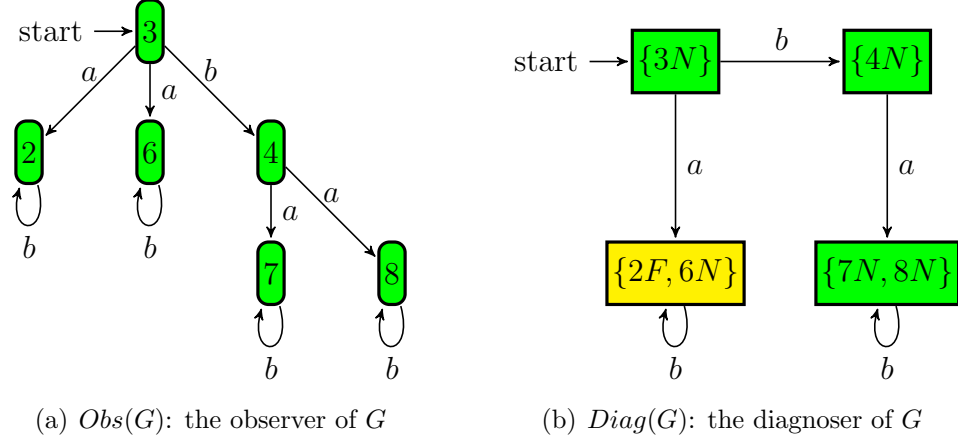


Figure 4.14: The observer and diagnoser of G

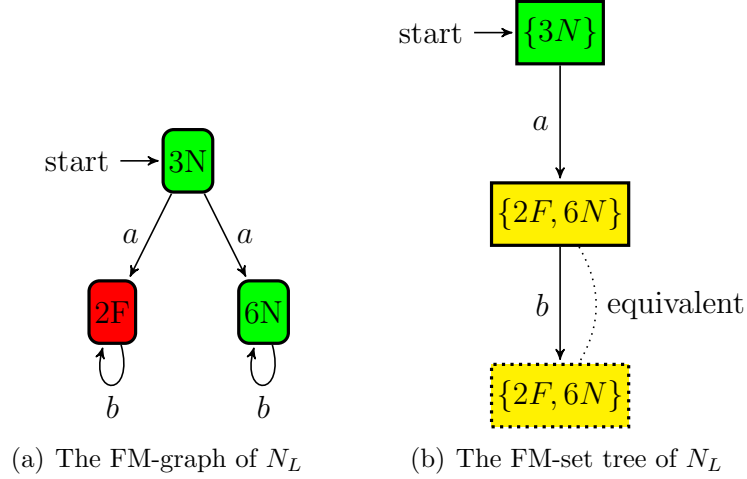


Figure 4.15: The FM-graph and FM-set tree of N_L

noting here that the order of branches investigation will be discussed later on in the manuscript.

Compare the two methods, the diagnoser approach generates 6 observer states and 4 diagnoser states, whereas the on-the-fly approach generates 3 FM-graph states and 2 FM-set tree states. The difference will be shown again in the comparative simulation results on the WODES benchmark (cf. Section 4.6.1.2).

Generally speaking, the goal behind using an on-the-fly approach is to avoid,

Table 4.6: The markings in Figure 4.15

State of FM-graph	Fault markings
$2F$	$[0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1]^\tau$
$3N$	$[0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^\tau$
$6N$	$[0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0]^\tau$

as much as possible, building the whole state space. However, from an analytical point of view, we are not able to determine, analytically and in general, the gain of our on-the-fly approach comparatively to the existing approaches, which are based on *a priori* building of intermediate exhaustive models (reachability graph, diagnoser, etc.). Indeed, this gain largely depends on the analyzed model and there is no actual worse case on which the complexity computation can be based.

It is worth noticing that our algorithm based on a depth-first search, does not define priorities in the investigation of branches. We need to define some heuristics in terms of priority between the branches to be investigated, to make our algorithm more efficient. This needs to make numerous experiments on different benchmarks to validate the strategy to be adopted.

Moreover, in order to solve diagnosability for each class of faults Σ_{F_i} , it is sufficient to perform our algorithm for each Σ_{F_i} respectively. Thus, the computational complexity will be linear with the number of fault classes (the same as in the diagnoser and the verifier-based approaches).

4.5 Online Diagnosis

In this section, we develop an approach for online diagnosis of diagnosable LPNs on the basis of a diagnoser, which is obtained from the FM-set tree in a straightforward way. The objective is to determine, from a sequence of observable events, whether the system is faulty and if so to which class the fault belongs.

An FM-set tree can be used for online diagnosis, because:

1. The node tag indicates the occurrence of fault: “normal” if no fault has

4. UNTIMED PN-BASED DIAGNOSIS OF DES

occurred, “ F -uncertain” if a fault has probably occurred, or “ F -certain” if a fault has occurred;

2. Each branch corresponds to one of the possible sequences of observable events from the initial marking and ends at a node that is F -certain or is equal to another existing node.

However, the marking component of a fault marking is useless for diagnosis, as we perform diagnosis based on sequences of observable events and not on the basis of markings. In order to perform online diagnosis efficiently, we generate the diagnosis graph (diagnoser) from the FM-set tree (cf. the return value \mathcal{T} in Algorithm 6), by:

1. Merging the equivalent nodes;
2. For each node, keep only the fault tag information.

Actually, a diagnoser is a deterministic graph where each node carries a fault tag and each arc is tagged with an observable event. Thus for any sequence of observable events, by following the corresponding path in the diagnosis graph from the initial node, one can determine the occurrence of a fault, online, by looking at the tag of the reached node.

The above algorithm considers a single fault type, however in order to deal with several fault classes, the nodes fault tags must be extended accordingly.

Example 26 Consider the LPN in Figure 4.16 with observable events $\Sigma_o = \{a, b\}$. The diagnosis graph for $\Sigma_f = \{f\}$ (cf. Figure 4.18) is generated from the FM-set tree of Figure 4.17. Letter N (resp. U, F) indicates “normal” (resp. “ F -uncertain”, “ F -certain”).

By looking at this graph, upon the observation of trace “aaa” from the initial state node “ N ” is reached. One can state that no fault from Σ_f has occurred. Upon “abb” one can conclude that one fault from Σ_f has occurred, since in the diagnoser “ F ” is the only reachable node after “ab” and the last “b” is unnecessary for giving diagnosis verdict. If “a” is observed, node “ U ” will be reached, meaning that a fault has possibly happened and further observation is needed to determine the system state.

Algorithm 7 Fault detection using the diagnoser derived from FM-set tree

```
1: Input: A sequence composed of observable event  $e_j$ ;  
2: Output: The fault occurrence information of the current system state;  
3:  $q_0 \leftarrow N$ ; ▷ The system is normal ( $N$ ) after the initialization.  
4:  $j \leftarrow 1$ ;  
5: while the system is in operation do  
6:   Wait for the input observable event  $e_j$ ;  
7:    $q_j \leftarrow$  the state from  $q_{j-1}$  upon  $e_j$ ;  
8:   switch  $q_j$  do  
9:     case  $N$   
10:      assert(No fault belonging to  $\Sigma_{F_i}$  has happened;)  
11:     case  $U$   
12:      assert(A fault belonging to  $\Sigma_{F_i}$  has probably happened;)  
13:     case  $F$   
14:      assert(A fault belonging to  $\Sigma_{F_i}$  has happened;)  
15:     case  $\emptyset$  ▷ The system arrives at a blocked faulty state “ $F$ ”.  
16:      assert(A fault belonging to  $\Sigma_{F_i}$  has happened;)  
17:    $j \leftarrow j + 1$ ;
```

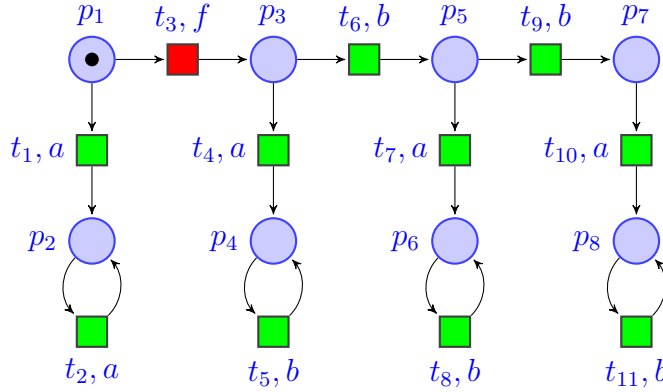


Figure 4.16: LPN N_L

4.6 OF-PENDA Software Tool

We have developed a tool implementing our various algorithms in C++, which we called *OF-PENDA*. In order to show the effectiveness of our method, the WODES diagnosis benchmark [Giua, 2007] and the LC benchmark are used in the com-

4. UNTIMED PN-BASED DIAGNOSIS OF DES

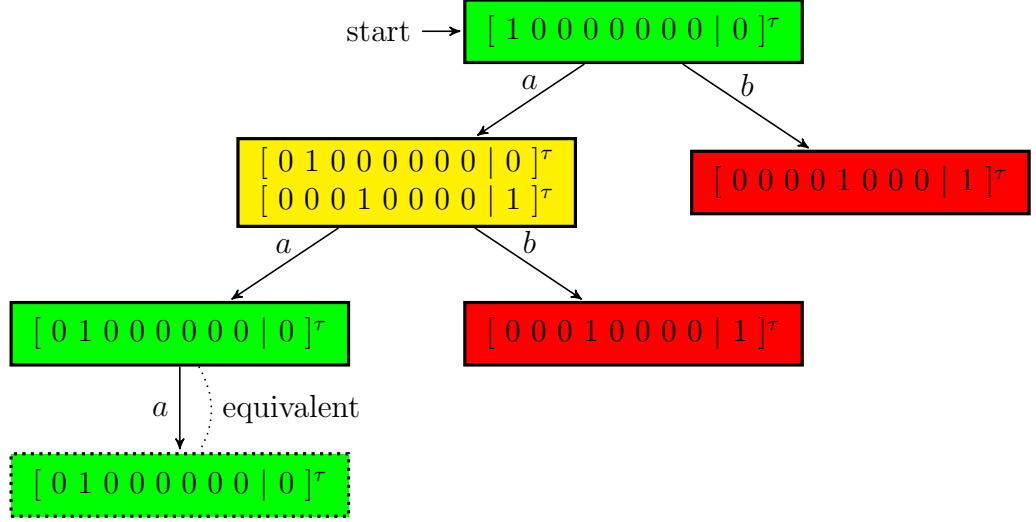


Figure 4.17: The FM-set tree of N_L

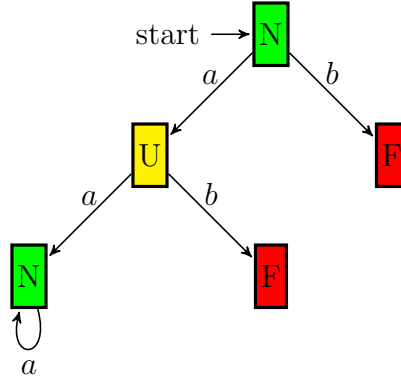


Figure 4.18: The diagnoser of N_L

parative simulation using OF-PENDA and the UMDES library [Lafortune, 2000], with the help of TINA tool [Berthomieu et al., 2004]. Based on the simulation results, we will point out the similarities and differences between our approach and some existing DES-based diagnosis approaches.

4.6.1 Application to the WODES Diagnosis Benchmark

4.6.1.1 WODES Diagnosis Benchmark

The WODES diagnosis benchmark is shown in Figure 4.19 and describes a manufacturing system characterized by three parameters: n, m and k , where:

- n is the number of production lines;
- m is the number of units of the final product that can be simultaneously produced. Each unit of product is composed of n parts;
- k is the number of operations that each part must undergo in each line.

The observable transitions are indicated by white boxes, and the unobservable transitions by black boxes. For more details on the benchmark the reader can refer to [Giua, 2007].

4.6.1.2 Comparative Simulation

We now analyze the diagnosability upon the fault class $\Sigma_{F_1} = \{f_1, \dots, f_{n-1}\}$. In other terms, only one class of faults is considered here. In order to perform a comparative simulation with OF-PENDA and the UMDES library, some preparations are necessary. The UMDES library deals with automata models by importing a “.fsm” file. Thus, we first generate the reachability graph of the considered PN with the help of TINA yielding to a “.aut” file, which is then transformed into a “.fsm” file by a script integrated in OF-PENDA that we have developed. This ensures that the comparative simulation is performed with the same input.

The simulation has been performed on a PC Intel with a clock of 2.26 GHz and the results are shown in Table 4.7.

- The first 3 columns titled “ m ”, “ n ” and “ k ” are the basic structural parameters of the WODES diagnosis benchmark.
- The 4th column is the stopping value of “ K ” obtained with Algorithm 6. Note that for a diagnosable case (cf. Column 9), this “ K ” value is equal to “ K_{min} ”.

4. UNTIMED PN-BASED DIAGNOSIS OF DES

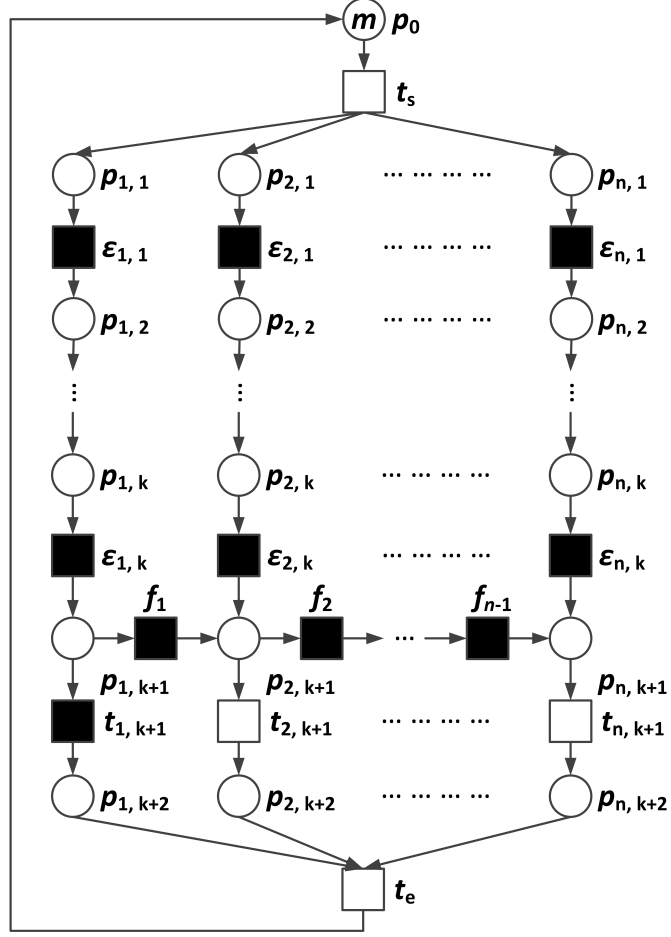


Figure 4.19: The WODES diagnosis benchmark

- The 5th column titled “ $|R|$ ” is the number of nodes in the marking graph computed by TINA, which is equal to the number of states of the automaton for building the diagnoser by the UMDES library.
- The 6th column titled “ $|\mathcal{N}|$ ” is the number of the FM-graph nodes.
- The 7th column titled “ $|\mathcal{X}_v|$ ” is the number of the FM-set tree nodes.
- The 8th column titled “ $|Diag|$ ” is the number of the diagnoser automaton states generated by UMDES.
- The 9th column titled “ \mathcal{D}_O ” is the diagnosability verdict returned by OF-

PENDA, where “Yes” indicates that the system is diagnosable and “No” indicates undiagnosable.

All the results are obtained under a simulation time of less than 6 hours. “o.t.” (out of time) means the result cannot be computed within 6 hours.

The discussion relative to this comparative study is given in the following section.

4.6.1.3 Discussions

A Finer Version of Diagnosability Both the OF-PENDA and the UMDES library allow us to check diagnosability for bounded DES. In particular, thanks to our incremental investigation of diagnosability, our tool also gives K_{min} for diagnosable systems, while the UMDES library [Lafortune, 2000] does not.

It is worthwhile to recall that the K -diagnosability is a finer version of diagnosability with the following two main features:

Online, the value of K_{min} gives us a valuable piece of information indicating the minimum number of steps necessary to detect and identify faults for a diagnosable system. Note that in [Basile et al., 2012], K is considered to be the number of both observable and unobservable transitions after a faulty transition. While here, as well as in [Cabasino et al., 2012a] K is the value relative to only observable events. Modifying our algorithm to compute both observable and unobservable transitions can be done easily.

Secondly, K -diagnosability allows a meticulous description of the diagnosability for multiple faults. It is advisable to discuss the traditional diagnosability as a series of K -diagnosability problems for each class of faults $\Sigma_{F_i} \subseteq \Sigma_f$. Further analysis on K could help to enhance the diagnosability. In order to solve K -diagnosability for each Σ_{F_i} , it is sufficient to perform our algorithm for each class of faults Σ_{F_i} iteratively. Thus, the computational complexity will be linear with the number of fault classes.

An On-the-fly and Incremental Method The UMDES library deals with diagnosability of systems modeled by automata based on the construction of an observer and a diagnoser automaton, which requires an exhaustive enumeration of the state space.

4. UNTIMED PN-BASED DIAGNOSIS OF DES

Table 4.7: Simulation results

m	n	k	$ R $	$ \mathcal{N} $	$ Diag $	$ \mathcal{X}_v $	\mathcal{D}_D	\mathcal{D}_O	K
1	2	1	15	8	4	3	Yes	Yes	2
1	2	2	24	10	4	3	Yes	Yes	2
1	2	3	35	12	4	3	Yes	Yes	2
1	2	4	48	14	4	3	Yes	Yes	2
1	3	1	80	52	10	6	Yes	Yes	3
1	3	2	159	90	10	6	Yes	Yes	3
1	3	3	274	138	10	6	Yes	Yes	3
1	3	4	431	196	10	6	Yes	Yes	3
1	4	1	495	367	29	17	Yes	Yes	4
1	4	2	1200	822	29	17	Yes	Yes	4
1	4	3	2415	1533	o.t.	17	o.t.	Yes	4
1	4	4	4320	2554	o.t.	17	o.t.	Yes	4
1	5	1	3295	2607	o.t.	66	o.t.	Yes	5
1	5	2	9691	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.
2	2	1	96	68	20	9	No	No	8
2	2	2	237	137	o.t.	9	o.t.	No	8
2	3	1	1484	801	20	12	No	No	11
2	3	2	5949	2746	o.t.	12	o.t.	No	11
2	4	1	28203	8795	o.t.	15	o.t.	No	14
2	4	2	180918	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.
3	2	1	377	290	66	12	No	No	11
3	3	1	12048	5165	o.t.	16	o.t.	No	15
3	4	1	484841	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.

Our approach can solve the diagnosability for both labeled and unlabeled PNs. We solve the classic diagnosability by handling a series of K -diagnosability problems, where K increases progressively. In other words, we reuse the state space

generated while analyzing K -diagnosability to deal with $(K + 1)$ -diagnosability. This is totally different from the approach in [Basile et al., 2012], since for each value of K a new system of equations is generated in their technique based on linear programming. Additionally, by progressively increasing K , it is certain to find K_{min} if the system is diagnosable. A similar idea can be found in the solution of Δ -diagnosability for timed automata in [Tripakis, 2002].

Thanks to the on-the-fly investigation of diagnosability, building the whole FM-set tree is not necessary. Actually, for undiagnosable systems, the FM-set tree building as well as its exploration are stopped as soon as an indeterminate cycle is found. Moreover, for diagnosable systems, while generating a branch in the FM-set tree, we stop as soon as an F -certain or an existing node is obtained. This is a notable advantage compared with the existing approaches [Cabasino et al., 2010; Sampath et al., 1995] which first build an exhaustive diagnoser or a reachability graph. The test result using the WODES diagnosis benchmark has shown this point since we were able to investigate diagnosability on models which are not tractable using UMDES library.

Relation between Diagnosability and Boundedness of PNs

The UMDES library solves the diagnosability problem based on exhaustive enumeration of the state space. Therefore, it can only deal with bounded systems modeled by finite state automata.

It is worth noting that there exist two PN-based approaches for the diagnosability analysis of unbounded models. In [Cabasino et al., 2012a], the authors check the diagnosability of unbounded PN models by analyzing the structure of the generated verifier net reachability graph. This approach requires an exhaustive enumeration of the reachability set of the verifier net, which may be larger than the reachability set of the original PN. In [Basile et al., 2012], the proposed approach is based on linear programming technique. System behavior is represented by a series of linear equations. The diagnosability of the unbounded PN models can be verified only if the faulty behavior can be described by a finite number of equations.

With the help of the on-the-fly computation, our algorithm can determine undiagnosability as soon as an F_i -indeterminate cycle is detected. Hence, this approach can be applied to some unbounded PNs with an F_i -indeterminate cycle.

4. UNTIMED PN-BASED DIAGNOSIS OF DES

Although an unbounded PN has infinite fault markings, which may result in an infinite number of FM-sets, the construction of an FM-set tree terminates once an F_i -indeterminate cycle is detected and a negative diagnosability verdict is emitted. From a practical point of view, some thresholds need to be used if our algorithm is used to deal with unbounded LPNs.

Case of Unlive PNs

Note that we have extended our algorithm to also deal with unlive systems, while considering the definition of diagnosability relative to unlive DES, given in [Sampath et al., 1998]. Besides, the WODES diagnosis benchmark we dealt with is unlive when $n \geq 2$, which is against the assumption of liveness in [Sampath et al., 1995]. Concretely, as the computation of the FM-set tree is performed on the fly, we have added an additional stopping condition: when some F -uncertain FM-set containing a deadlock state is obtained. Indeed, in this case, the system may stay indefinitely in this uncertain state, and no diagnosis verdict can be emitted.

4.6.2 Application to the Level Crossing Benchmark

In the previous section, we have proved the efficiency of our developed techniques using WODES benchmark, which is a classic diagnosis benchmark often used in diagnosis analysis. However, it is not quite suitable for testing diagnosability analysis approach, since

- The benchmark is live only if $n = 1$.
- The benchmark is diagnosable only if $m = 1$.

In order to obtain more general results, we will perform another group of analysis based on our developed LC benchmark.

4.6.2.1 Level Crossing Benchmark

In this section, we develop an n -line LC benchmark based on the single-line LC model in [Leveson and Stolzy, 1985]. Different from the original model of [Leveson

and Stolzy, 1985], our benchmark is live and integrated with the operation principles for multi-line LC, which can be sufficiently complex for testing diagnosability analysis approaches.

Figure 7 describes a global LPN model for a single-line LC with a unidirectional track. Based on this model, a more general model is given in Figure 8 – involving n railway tracks, which can be obtained from the single-line LC model while fulfilling the following controlling rules under a nominal situation:

- The LC must be closed if any approaching train is detected in any line;
- The LC can be reopened if there is no train in the “within” or “before” sections in any line.

In other terms, the above rules eliminate all the possibilities that the collision between railway and road traffic may take place.

For more about the background of the LC systems and development of the n -line LC benchmark, the reader can refer to Appendix 6.2.

4.6.2.2 Comparative Simulation

In this section, we will analyze the diagnosability of the LC model while considering various values of n . Here, we will consider two classes of faults:

- For $T_{F1} = \cup_{i=1}^n \{t_{i,5}\}$, we consider all the faults $t_{i,5}$ in each track as belonging to the same class. Recall that such faults express the fact that trains can enter the crossing zone while the barriers are not ensured to be down.
- For $T_{F2} = \{t_6\}$, this fault class depicts an early lowering of the barriers, i.e., before all the trains are ensured to have left the LC.

A comparative simulation with UMDES library is performed on an Intel PC (CPU: 2.50 GHz, RAM: 3.16 GB) and the results are given in Table 4.8, Figure 4.20 and Figure 4.21, where:

- n is the number of railway tracks;
- Σ_{Fi} is the considered fault class (Σ_{F1} or Σ_{F2});

4. UNTIMED PN-BASED DIAGNOSIS OF DES

- $|P|$ and $|T|$ are the number of places and transitions of the PN model respectively;
- $|A|$ and $|R|$, which are the number of the arcs and the nodes of the reachability graph respectively, give the scale of the PN reachability graph computed by TINA and which serves as the input automaton states for UMDES;
- $|\mathcal{N}|$ is the number of (on-the-fly) generated fault markings by OF-PENDA when the diagnosability verdict is issued;
- $|Diag|$ is the number of diagnoser states generated by UMDES (UMDES) (the diagnoser approach) which were needed to give the diagnosability verdict.
- $|\mathcal{X}_v|$ is the number of (on-the-fly) generated FM-sets by OF-PENDA when the diagnosability verdict is issued, and corresponds also to the number of nodes of the diagnoser derived from this FM-set tree when the model is diagnosable;
- \mathcal{D}_D , \mathcal{D}_V and \mathcal{D}_O are the diagnosability verdicts obtained by diagnoser approach, verifier approach of UMDES and OF-PENDA respectively, where “Yes” indicates that the system is diagnosable and “No” indicates undiagnosable;
- K is the minimum value ensuring diagnosability computed by OF-PENDA, if the system is diagnosable; otherwise, it is the last value under which K -diagnosability is investigated before concluding that the system is undiagnosable;
- \mathcal{T}_T is the time needed to generate the PN reachability graph computed by TINA, i.e., the time used for preparing the input automaton needed for UMDES.
- \mathcal{T}_D , \mathcal{T}_V and \mathcal{T}_O are the times needed to obtain the diagnosability verdict by *dcycle.exe* (diagnoser approach), *verifer_dia.exe* (verifier approach) of UMDES and OF-PENDA (on-the-fly approach), respectively.

4.6.2.3 Discussions

For the comparative simulation results, we give the following remarks:

1. The UMDES library has been integrated in the DESUMA framework which also integrates GIDDES for graphical facilities. However, we directly use the command lines in UMDES library rather than the interface framework (DESUMA), since the DESUMA takes so much time to load large automaton models.
2. The scale of the PN reachability graph grows quickly (cf. the columns titled with $|A|$ and $|R|$ in Table 4.8) as n increases. The capability of TINA to calculate the reachability graph considerably depends on the memory of the computer, since TINA needs to refer to the already built part of the reachability graph all along the computation. One can observe that the generated *.aut* files become considerable starting from $n = 7$. For instance, the size of the *.aut* files for 1, 2, 3, 4, 5, 6, 7-line LC benchmark is 655b, 11KB, 128KB, 1.2MB, 9.9MB, 74.8MB, 518.6MB, respectively. A Mac with a RAM of 16GB cannot store the *.aut* file for 8-line benchmark even if it can be calculated eventually. For the case of $n = 9$, the calculation stops with an error of “out of memory”. Thus, the simulation for the cases with $n \geq 9$ cannot be performed using UMDES.
3. The simulation using UMDES and OF-PENDA are performed on a Windows PC. For Σ_{F1} , some “accidental quits” happen during the running of *dcycle.exe* (diagnoser approach in UMDES) for some cases. However, the diagnosability verdict can be obtained by using the verifier technique (command *verifier_dia.exe*) in UMDES library. We also meet the same problem when dealing with fault class Σ_{F2} under $n = 4$. Note that the relative results given by OF-PENDA when an accidental quit happens are the last outputs before the program’s exit. We do not know exactly the reason for this problem. The improvement of the source code and using other operation systems may eliminate the problems.

Table 4.8: Comparative simulation results based on n -line LC benchmark

n	Σ_{Fi}	$ P $	$ T $	$ A $	$ R $	$ N $	$ Diag $	$ X_v $	\mathcal{D}_D	\mathcal{D}_V	\mathcal{D}_O	K	\mathcal{T}_T	\mathcal{T}_D	\mathcal{T}_V	\mathcal{T}_O
1	Σ_{F1}	13	11	28	24	24	26	15	YES	YES	YES	3	<1s	<1s	<1s	<1ms
2		17	16	540	216	116	262	18	a.q.	NO	NO	11	<1s	11s	<1s	15ms
3		21	21	6256	1632	173	1924	18	a.q.	NO	NO	11	<1s	12s	7s	46ms
4		25	26	56704	11008	230	12504	18	a.q.	NO	NO	11	<1s	32s	21m22s	62ms
5		29	31	442880	68608	287	75722	18	a.q.	o.t.	NO	11	2s	28m34s	o.t.	78ms
6		33	36	3126272	403456	344	a.q.	18	o.t.	o.t.	NO	11	11s	1h36m	o.t.	125ms
7		37	41	20500480	2269184	401	o.t.	18	o.t.	o.t.	NO	11	140s	o.t.	o.t.	156ms
8		41	46	127074304	12320768	458	o.t.	18	o.t.	o.t.	NO	11	29m	o.t.	o.t.	203ms
9		45	51	o.m.	o.m.	515	-	18	-	-	NO	11	o.m.	-	-	249ms
10		49	56	o.m.	o.m.	572	-	18	-	-	NO	11	o.m.	-	-	296ms
20		89	106	o.m.	o.m.	1142	-	18	-	-	NO	11	o.m.	-	-	1467ms
40		169	206	o.m.	o.m.	2282	-	18	-	-	NO	11	o.m.	-	-	5460ms
1	Σ_{F2}	13	11	28	24	29	26	15	YES	YES	YES	7	<1s	<1s	<1s	<1ms
2		17	16	540	216	277	262	207	YES	YES	YES	10	<1s	<1s	<1s	453ms
3		21	21	6256	1632	2109	1924	1842	YES	YES	YES	17	<1s	<1s	5s	109s
4		25	26	56704	11008	13353	12504	5670	YES	YES	a.q.	18	<1s	20s	16m19s	2h4m
5		29	31	442880	68608	o.t.	75722	o.t.	YES	o.t.	o.t.	o.t.	2s	27m12s	o.t.	o.t.
6		33	36	3126272	403456	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	11s	o.t.	o.t.	o.t.
7		37	41	20500480	2269184	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	140s	o.t.	o.t.	o.t.
8		41	46	127074304	12320768	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	o.t.	29m	o.t.	o.t.	o.t.
9		45	51	o.m.	o.m.	o.t.	-	o.t.	-	-	o.t.	o.t.	o.m.	-	-	o.t.

Note: o.m. = out of memory

o.t. = out of time (> 6h)

a.q.= accidental quit

■ results obtained by TINA

■ results obtained by *diag_UR.exe* and *dcycle.exe* of UMDES■ results obtained by *verifier_dia.exe* of UMDES

■ results obtained by OF-PENDA

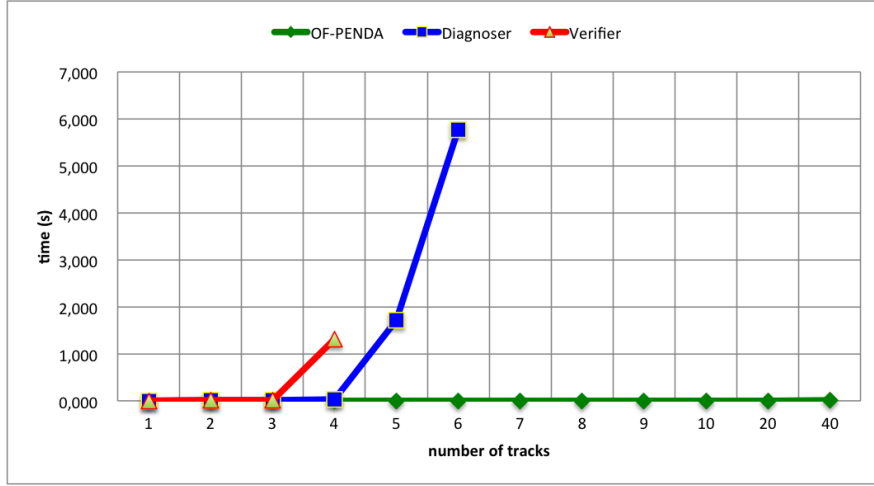


Figure 4.20: Time cost for the diagnosability analysis on Σ_{F1}

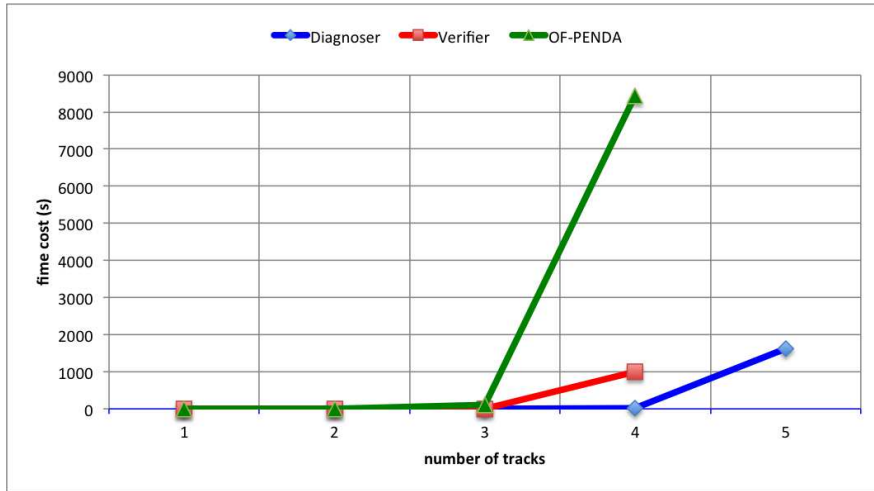


Figure 4.21: Time cost for the diagnosability analysis on Σ_{F2}

4. In order to compare the efficiency in terms of time, we make a record of the computing time for each case. The time indicated in the columns titled with \mathcal{T}_T , \mathcal{T}_D and \mathcal{T}_V are obtained by an external timer, since TINA and UMDES do not output this information. Thus, there is an error margin of less than 1s. For the OF-PENDA, an automatic timer has been integrated with an error margin of less than 1ms.

4. UNTIMED PN-BASED DIAGNOSIS OF DES

5. Besides the diagnosability verdict, OF-PENDA also gives the minimum value of K to ensure diagnosability, which cannot be obtained by UMDES.
6. In these cases, the number of FM-sets generated by OF-PENDA is lower than the number of diagnoser states given by UMDES. This shows the advantage offered by our on-the-fly technique. Besides, it is worthwhile to mention that the FM-sets can be far fewer than the diagnoser states when an indeterminate cycle exists and is detected early (if the model is undiagnosable).
7. Our approach spends more computing time, because we generate the FM-set tree directly from the PN, while the UMDES takes an automaton as an input which is equivalent to the reachability graph of the PN. In other words, the inputs for UMDES and OF-PENDA are not the same and we had to compute the PN reachability graph *a priori* before performing analysis via UMDES. On the contrary, the reachable markings are computed on the fly while investigating diagnosability by OF-PENDA.
8. In different cases, such as the WODES benchmark and the n -line LC benchmark, our approach shows a different efficiency level in terms of memory. In some undiagnosable cases, depending on the model structure, our approach may save memory resources considerably compared to the enumerative approaches, as has been shown in Section 4.6.1.2.
9. The results (cf. column \mathcal{T}_D and \mathcal{T}_V) show that the diagnoser approach is more efficient than the verifier approach while dealing with the n -line LC benchmark. This does not violate the claim that the verifier approach is more efficient in terms of time complexity (polynomial for the verifier approach vs exponential for the diagnoser approach), since the theoretical complexity is always computed while considering the worst case.

Here, we will give the transition sequences which help to obtain the diagnosability verdict easily, such that the simulation results can be compared with this theoretical analysis to evaluate their correctness:

-
- For the case $n = 1$ and for Σ_{F1} , the system is normal if in any transition sequence $t_{1,2}$ and $t_{1,3}$ appear alternatively. Otherwise $t_{1,5}$ must have occurred. Therefore, the system can be diagnosable according to the order of appearance of $t_{1,2}$ and $t_{1,3}$.
 - For Σ_{F1} in the cases of n -line LC ($n \geq 2$), fault transition $t_{1,5}$ can fire or not right after $t_{1,1}$ firing. For both cases, a possible followed firing sequence $t_1 t_4 t_{2,1} t_{2,2} t_{2,3} t_{2,4} t_5$ exists, which corresponds to an indeterminate cycle. Therefore, the system is undiagnosable.
 - For Σ_{F2} in the cases of n -line LC ($n \geq 1$), t_6 can fire or not right after the firing of sequence $t_{i,1} t_1 t_4$. Then the system can remain F_2 -uncertain for as long as 6 steps during the firing of sequence $t_{i,2} t_{i,3} t_{i,4} t_{i,1} t_2 t_1$. The 7th observable transition firing will terminate the sequence of F_2 -uncertain states, i.e., the system is normal if t_4 fires; otherwise the firing of t_3 implies that fault t_6 has occurred.

Compared with the diagnoser and verifier approaches, our on-the-fly approach can deal with some quite complex models that UMDES cannot deal with (e.g., for the cases Σ_{F1} when $n \geq 7$), especially for some undiagnosable systems, and shows better efficiency in terms of time and memory. However, for some diagnosable cases (e.g., for the cases Σ_{F2}), we spend less memory but more time, which is the limitation of our approach. This shows, again, that the computing efficiency depends on model structures.

4.7 Conclusion

In this chapter, we have studied fault diagnosis of DES using on-the-fly and incremental techniques to cope with state explosion problems. Algorithms for checking K -diagnosability, diagnosability and online diagnosis have been developed. The obtained results through the WODES benchmark are encouraging.

In Section 4.1, the on-the-fly and incremental techniques have been introduced. The on-the-fly techniques allow generating and investigating only a part of state space to find the solutions. The incremental techniques can reuse historical information to avoid recomputing works. Compared with traditional state

4. UNTIMED PN-BASED DIAGNOSIS OF DES

enumerative approaches, they have shown the advantages in improving efficiency of analyzing diagnosis problems.

In Section 4.2, we have developed mathematical representations for LPNs, namely the extended incidence matrix, event marking and extended state equation, such that the behavior relative to events can be well presented.

In Section 4.3, notions such as fault marking, FM-graph and the FM-set tree are introduced to describe the state evolution with fault propagation of the system. We have developed an approach to check K -diagnosability based on the on-the-fly building of FM-graph and the FM-set tree.

In Section 4.4, classic diagnosability was discussed by solving a series of K -diagnosability. The incremental technique has been used to analyze diagnosability and seek out K_{min} to ensure diagnosability.

In Section 4.5, the diagnoser derived from the FM-set tree has been developed. Comparisons between the traditional enumerative approaches and ours were performed, showing that our diagnoser consumes less memory.

In Section 4.6, a group of comparative simulations on the basis of the WODES and the LC benchmark have been performed. The obtained results showed that the on-the-fly and incremental techniques can largely improve the efficiency in terms of time and memory.

Chapter 5

Time PN-Based Diagnosis of DES

In the previous chapter, we have discussed fault diagnosis issue in the untimed context. Principles in untimed analysis are simple, however, they sometimes require much resources. Actually, by considering time information, some untimed analysis can be rather easy. In [Tripakis, 2002], fault diagnosis of timed DESs has been presented in the framework of timed automata. Necessary and sufficient conditions for diagnosability of timed automata have been given. For the analysis based on TPNs, it is possible to first transform TPN models into timed automata using existing techniques, and then to reuse the results of [Tripakis, 2002]. However, the transformation is not practical work, since:

- The TPN is the extension of the PN, thus it inherits the advantages of PNs compared with automata.
- TPN is more expressive than timed automata and more suitable for modeling a real system. In the contrary, generating the whole state space (timed automata) of real complex timed systems is rather burdensome work.

This chapter deals with fault diagnosis of DESs in a timed context. Besides the relative ordering of events considered in untimed context, the information about the event occurrence dates are also used for system modeling and fault diagnosis analysis.

5. TIME PN-BASED DIAGNOSIS OF DES

The model that will be used here is LTPN – an extension of TPN, in which each transition is associated with an event which can be either observable or not. A given label (event) may be assigned to various transitions.

Thanks to a skillful splitting of the time intervals assigned to the LTPN transitions, a deterministic on-the-fly-built structure called ASG can be built for LTPN models. The observer is enriched with information about fault occurrence, such that it is sufficient to monitor the system behavior and detect faults reacting to observable events and their respective occurrence dates. An ASG carries both the marking reachability and the fault propagation. Based on the ASG which is built on the fly, the timed diagnosability problem is transformed and analyzed as in the untimed context. In this case, some existing techniques for untimed diagnosis analysis can be brought into play in the timed context.

Generally, and as explained in Chapter 4, using on-the-fly analysis makes it possible to generate and investigate only partially the ASG state space for checking diagnosability. This is a distinct advantage compared with the traditional enumerative approaches. Furthermore, and in the same way as in the untimed context, for a diagnosable LTPN the ASG is then used for deriving a diagnoser called LTD that we develop for performing online diagnosis. As a classical diagnoser, an LTD assigns a state (or a set of states) with a tag indicating fault occurrence: Normal, F_i -certain or F_i -uncertain, while it reacts to a sequence of observable dated events.

We also discuss the relation between our developed ASG and the event-recording automata (ERA), which is an existing determinizable subclass of Timed Automata (TA). It is shown that a determinizable subclass of LTPNs also exists, whose ASGs shows some equivalence with ERA. The bisimulation between them makes it possible to bring existing TA-based techniques into PN-based timed fault diagnosis.

5.1 Splitting Time Intervals Assigned to LTPN Transitions

Splitting time intervals assigned to LTPN transitions is a key element in our technique for timed fault diagnosis analysis of DES. We assume the LTPN to be live and bounded. This section will give the motivation and introduce some preliminary concepts that will be used in this technique.

5.1.1 Motivation

First, let us look at the motivation of splitting time intervals with the following example.

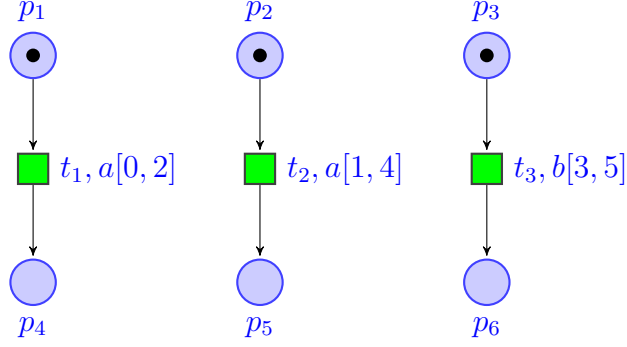
Example 27 Consider the observability problem of the LTPN N_{LT} in Figure 5.1(a), where $t_1, t_2, t_3 \in \Sigma_o$ are observable transitions.

Using the classical analytical approach [Berthomieu and Menasche, 1983], we can compute all the state classes reachable from the initial class C_0 . A part of the state class graph is shown in Figure 5.1(b). From state class C_0 , we have two reachable state classes C_1 and C_2 by the firing of transitions t_1 at a date in $[0, 2]$ and t_2 at a date in $[1, 2]$ respectively. This is convenient if we do not consider event labels assigned to the transitions.

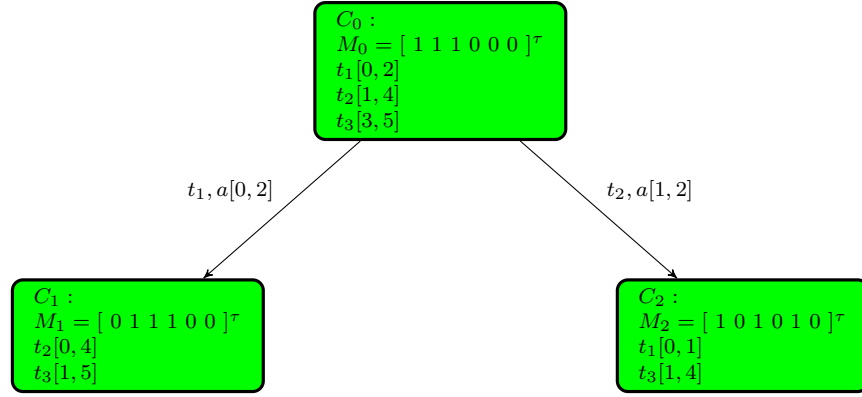
However, when dealing with LTPNs, i.e., event based observation, it is possible that we cannot distinguish between two transitions with the same label, even if additional time information is available. For example, if event a is observed at date 0.5 t.u., C_1 will be reached with certainty. However, if a is observed at 1.5 t.u., it may correspond either to the firing of transition t_1 or the firing of t_2 and it is uncertain whether C_1 or C_2 is reached.

In order to eliminate this nondeterminism, we want to modify the state class graph of N_{LT} while considering transition labeling, so that by each observation of event with its exact occurrence time, we can determine which state (or minimal set of states) the system can be in. A part of the modified state class graph of N_{LT} is shown in Figure 5.1(c). We split the time interval associated with transition t_1 , i.e., $[0, 2]$, into two disjoint intervals $[0, 1[$ and $[1, 2]$. We recompute the state classes reached from C_0 , i.e., C_{1a} and C_{1b} , by the firing of t_1 at a date belonging

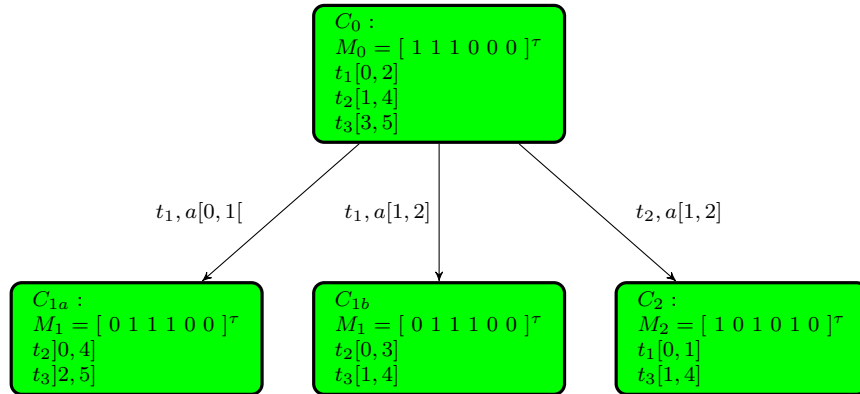
5. TIME PN-BASED DIAGNOSIS OF DES



(a) The LTPN N_{LT} in Example 27



(b) A part of the state class graph of N_{LT}



(c) A part of the modified state class graph of N_{LT}

Figure 5.1: The state class graph and the modified state class graph for LTPN N_{LT}

to these two intervals respectively, using the classical approach for TPNs. In this case we can state that, if event a is observed at a date belonging to $[0, 1[$, C_{1a} is reached with certainty; whereas if a is observed in $[1, 2]$, one of the state classes belonging to the set $\{C_{1b}, C_2\}$ is reached. Assuming that we rebuild the state class graph from the initial state class C_0 with the time intervals splitting technique, and regroup all the state classes that are reached from the same state class (or set of state classes) by a transition firing assigned with the same event and firing interval, we may solve, even partially, the problem of nondeterminism in the observation of LTPNs.

Comparing the two previous cases, we generate more state classes in Figure 5.1(c) than in Figure 5.1(b). However, the state classes obtained in Figure 5.1(c) are more precise in describing the N_{LT} behavior w.r.t event occurrence. For example, if a is observed at 0.5, we are certain that C_{1a} is reached and one of the following firable transitions t_2 and t_3 can be fired at a relative date belonging to $[0.5, 3.5]$ and $[2.4, 4.5]$ respectively from C_{1a} , rather than $[1, 5]$ as in class C_1 Figure 5.1(b). This means that using the classical state class graph for observing an LTPN leaves an overestimation on the firing domains. Moreover, this overestimation may be propagated and aggravated as the system state moves forward along the reachable state classes, resulting in a non-optimal evaluation of the system state. In summary, our objective here is to develop an efficient deterministic structure for the observation and diagnosis analysis of LTPNs.

5.1.2 Semi-Interval

In this section we formally discuss how to split time intervals assigned to LTPN transitions. The goal behind splitting a given finite set of time intervals is to generate a new set of split intervals that we call *basic interval set*. For this aim, let us first introduce some basic notations on time intervals that will be used afterward.

A *time interval* [Allen, 1983] is indeed a set of non-negative rational numbers but may also include infinity:

$$\{[:]\} \times \mathbb{Q}_{\geq 0} \times (\mathbb{Q}_{\geq 0} \times \{[:]\} \cup \{+\infty[})$$

5. TIME PN-BASED DIAGNOSIS OF DES

It can be a bounded, unbounded or half-bounded interval, e.g.,

$$[1, 5] = \{x \in \mathbb{Q}_{\geq 0} \mid 1 \leq x \leq 5\}$$

$$]9, 13[= \{x \in \mathbb{Q}_{\geq 0} \mid 9 < x < 13\}$$

$$[0, +\infty[= \{x \in \mathbb{Q}_{\geq 0} \mid x \geq 0\}$$

Definition 23 (*semi-interval*) A left semi-interval is a left-open interval, defined by

- $a[= \{x \in \mathbb{Q}_{\geq 0} \mid x < a\} = [0, a[$ with $a \in \mathbb{Q}_{\geq 0}$, or
- $+\infty[= \{x \in \mathbb{Q}_{\geq 0}\} = [0, +\infty[$, or
- $a] = \{x \in \mathbb{Q}_{\geq 0} \mid x \leq a\} = [0, a]$;

a right semi-interval is a right-open interval, defined by

- $]a = \{x \in \mathbb{Q}_{\geq 0} \mid x > a\} =]a, +\infty[$ with $a \in \mathbb{Q}_{\geq 0}$, or
- $[a = \{x \in \mathbb{Q}_{\geq 0} \mid x \geq a\} = [a, +\infty[$.

Actually, to determine whether it is a left or a right interval, it suffices to note the position of rational limit (a) relatively to the square bracket: on the left for left semi-interval and on the right for the right semi-interval.

Obviously, any time interval can be written as an intersection of a left semi-interval and a right semi-interval. For the previous three time intervals, we have

$$[1, 5] = \{x \mid x \geq 1\} \cap \{x \mid x \leq 5\}$$

$$]9, 13[= \{x \mid x > 9\} \cap \{x \mid x < 13\}$$

$$[0, +\infty[= \{x \mid x \geq 0\} \cap \{x \mid x < +\infty\}$$

Formally, we treat a time interval as an intersection of two half-open intervals that we call *semi-intervals*.

Given an interval i , the corresponding left (resp. right) semi-interval is denoted by $left(i)$ (resp. $right(i)$), and the complementary set of semi-interval

α is denoted by $\bar{\alpha}$. For $\beta = a]$ (resp. $a[;]a; [a)$, we denote $\text{bound}(\beta) = a$ and $\text{border}(\beta) =]$ (resp. $[;]$).

Example 28 Time interval $i = [2, 4[$, can be written as:

$$i = \text{left}(i) \cap \text{right}(i)$$

where

$$\text{right}(i) = [2 = \{x \in \mathbb{Q}_{\geq 0} \mid x \geq 2\}$$

and

$$\text{left}(i) = 4[= \{x \in \mathbb{Q}_{\geq 0} \mid x < 4\}$$

We can also denote by

$$\overline{\text{right}(i)} = 2[$$

$$\text{bound}(\text{right}(i)) = 2$$

$$\text{border}(\text{right}(i)) = [$$

For two left (or two right) semi-intervals α and β , we say $\alpha = \beta$, if

- $\text{bound}(\alpha) = \text{bound}(\beta)$ and
- $\text{border}(\alpha) = \text{border}(\beta)$.

We define an order relation “ \prec ” between semi-intervals as follows:

- $\alpha \prec \beta$, if $\text{bound}(\alpha) < \text{bound}(\beta)$;
- $c[\prec [c \prec c] \prec]c$, if $\text{bound}(\alpha) = \text{bound}(\beta) = c$;
- $\alpha \prec +\infty[$, if $\alpha \neq +\infty[$.

Example 29 For semi-intervals $1[; [1; 1];]1; 9[; [9; 9];]9$ and $+\infty[$, the order relation between them is as follows:

$$1[\prec [1 \prec 1] \prec]1 \prec 9[\prec [9 \prec 9] \prec]9 \prec +\infty[$$

The objective of defining this order relation between semi-intervals is to reorganize a set of semi-intervals for further computing basic interval sets, as will be introduced in Algorithm 8 in the following section (cf. Lines 3 and 8).

5. TIME PN-BASED DIAGNOSIS OF DES

5.1.3 Basic Interval Set (BIS)

In order to eliminate nondeterminism in LTPNs, time interval splitting techniques [Liu et al., 2014a] will be developed to reassign each observable event with an interval in the basic interval set, such that each firing of an observable event with its relative time brings the system to a unique minimal macro state (i.e., the ASC that will be introduced in Section 5.2.1).

Definition 24 *Given a finite time interval set A , the basic interval set (BIS) of A , denoted by $BIS(A)$, is a set of disjoint nonempty time intervals β_j subject to:*

1. $\forall k \neq j, \beta_k \cap \beta_j = \emptyset$;
2. $\forall \alpha \in A, \exists \beta_1, \beta_2, \dots, \beta_m \in BIS(A)$, such that $\alpha = \bigcup_{j=1}^m \beta_j$;
3. $\forall \beta_1, \beta_2 \in BIS(A), \beta_1 \neq \beta_2, \exists \alpha \in A$, such that $\beta_1 \cap \alpha = \emptyset, \beta_2 \cap \alpha \neq \emptyset$.

In particular, for a time interval set A such that its cardinality (the number of elements in set A) $|A| = 1$, $BIS(A) = A$. Here we emphasize that $BIS(A)$ is a finite and unique set for any finite interval set A , as will be illustrated in details in Propositions 6 and 8.

About the above definition, we make the following remarks:

- Condition 1 shows that the elements in a BIS must be disjoint from each other, e.g., $\{[1, 3]; [2, 4]\}$ cannot be the BIS of any interval set since $[1, 3] \cap [2, 4] \neq \emptyset$.
- Condition 2 indicates that any interval in set A is the union of a finite number of intervals in $BIS(A)$, e.g., for $A = \{[1, 4]; [2, 5]\}$, $BIS(A) = \{[1, 2]; [2, 4], [4, 5]\}$, we have $[1, 4] = [1, 2] \cup [2, 4]$ and $[2, 5] = [2, 4] \cup [4, 5]$.
- Condition 3 ensures that $BIS(A)$ is unique and this will be proved in Proposition 8.

Here, the term *basic* has two meanings. On the one hand, any time interval assigned to an LTPN transition can be represented by a union of finite disjoint basic intervals. Some of these basic intervals may also be parts of other time intervals of A , i.e., basic intervals can be treated as basic components for the

time intervals of A . On the other hand, Condition 3 of Definition 24 ensures that $BIS(A)$ is unique and contains as few time intervals as possible to represent all the elements in A as a union. More illustration will be given in Example 30 at the end of this section.

Proposition 6 *The BIS of a finite interval set is also finite.*

Proof. Assume that given a finite interval set A , $BIS(A)$ can be infinite and

$$BIS(A) = \bigcup_{j=1}^{+\infty} \{\beta_j\}$$

Consider that index j in $\bigcup_{j=1}^{+\infty} \{\beta_j\}$ gives the ordering of the intervals' values, i.e., β_1 holds the smaller values, and so on, as shown in Figure 5.2.

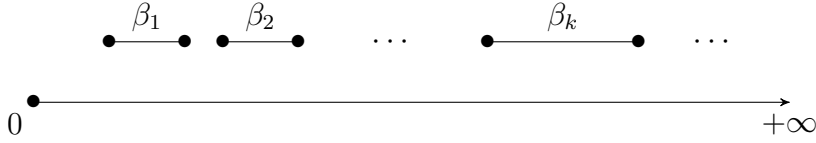


Figure 5.2: Time intervals in $BIS(A)$

Let us take $\beta_i, \beta_{i+1} \in BIS(A)$, according to Condition 1 in Definition 24,

$$\beta_i \cap \beta_{i+1} = \emptyset$$

here we have

$$\forall x \in \beta_i, y \in \beta_{i+1}, x < y$$

According to Condition 3 in Definition 24,

$$\exists \alpha_1 \in A \text{ such that } \alpha_1 \cap \beta_i \neq \emptyset \text{ and } \alpha_1 \cap \beta_{i+1} = \emptyset$$

Moreover, as intervals β_j are ordered from left to right and, given that α_1 is an interval, i.e., contains continuous values,

$$\alpha_1 \cap \beta_k = \emptyset \text{ for } k \geq i + 1 \quad (1)$$

5. TIME PN-BASED DIAGNOSIS OF DES

In the same way,

$$\exists \alpha_2 \in A \text{ such that } \alpha_2 \cap \beta_{i+1} \neq \emptyset \text{ and } \alpha_{i+1} \cap \beta_{i+2} = \emptyset$$

Likewise for β_{i+2}, β_{i+3} ,

$$\exists \alpha_3 \in A \text{ such that } \alpha_3 \cap \beta_{i+2} \neq \emptyset \text{ and } \alpha_{i+2} \cap \beta_{i+3} = \emptyset$$

Hereby, $\alpha_2 \neq \alpha_3$ and $\alpha_1 \neq \alpha_3$, since from (1), $\alpha_1 \cap \beta_k = \emptyset$ for $k \geq i+1$. This way, for two successive intervals β_i, β_{i+1} in $BIS(A)$, a new interval α_k is found. Then as i goes from 1 to $+\infty$, an infinite number of intervals α_k is generated, which means A has an infinite number of intervals and this violates the assumption. \square

The BIS of a finite interval set can be computed by Algorithm 8.

Algorithm 8 Computation of BIS

```

1: Input:  $A$ ;  $\triangleright A$  is a finite interval set.
2: Output:  $B$ ;  $\triangleright B = BIS(A)$ 
3:  $C \leftarrow \emptyset$ ;  $\triangleright C$  is an array of semi-intervals ordered according to  $\prec$ .
4: for all  $\alpha \in A$  do
5:    $C \leftarrow C \cup \{left(\alpha)\} \cup \{right(\alpha)\}$ ;
6: reorder  $C$  according to  $\prec$ ;
7:  $c_0 \leftarrow \overline{c_1}$ ;  $\triangleright c_j$  ( $j = 1, 2, \dots$ ) denotes, in the order of  $\prec$ , the  $j^{th}$  element of  $C$ .
8:  $C \leftarrow C \cup \{c_0\}$ ;  $\triangleright$  Insert  $c_0$  into  $C$ , then  $c_0$  will be the first element of  $C$  instead of the original  $c_1$ .
9: for  $j$  from 1 to  $(|C| - 1)$  do  $\triangleright$  For all the elements of  $C$  except the first and the last one.
10:   if  $c_{j-1}$  is a right semi-interval then
11:      $\alpha \leftarrow c_{j-1}$ ;  $\triangleright \alpha$  is a right semi-interval.
12:   else
13:      $\alpha \leftarrow \overline{c_{j-1}}$ ;  $\triangleright \alpha$  is a left semi-interval.
14:   if  $c_j$  is a left semi-interval then
15:      $\beta \leftarrow c_j$ ;  $\triangleright \beta$  is a left semi-interval.
16:   else
17:      $\beta \leftarrow \overline{c_j}$ ;  $\triangleright \beta$  is a right semi-interval.
18:    $B \leftarrow B \cup \{(\alpha \cap \beta)\}$ ;
19: return  $B$ ;

```

In order to prove the unicity of the BIS for a given finite interval set, we first give the following proposition, which will be used in the demonstration of Proposition 8.

Proposition 7 *Given a finite interval set A , for any $\alpha \in A$ and $\beta \in BIS(A)$*

$$\beta \cap \alpha \neq \emptyset \Rightarrow \beta \subseteq \alpha$$

Proof. According to Condition 2,

$$(\exists \beta_1, \beta_2, \dots, \beta_m \in BIS(A))(\alpha = \bigcup_{j=1}^m \beta_j)$$

and obviously $\forall j = 1, \dots, m, \beta_j \subseteq \alpha$.

Now suppose that

$$(\forall k = 1, \dots, m)(\beta \neq \beta_k)$$

From Condition 1 in Definition 24,

$$(\forall k = 1, \dots, m)(\beta \cap \beta_k = \emptyset)$$

$$\Rightarrow \beta \cap (\bigcup_{k=1}^m \{\beta_j\}) = \emptyset$$

$$\Rightarrow \beta \cap \alpha = \emptyset$$

This violates $\beta \cap \alpha \neq \emptyset$ in the proposition, thus

$$\exists k \in \{1, 2, \dots, m\} \text{ such that } \beta = \beta_k$$

$$\Rightarrow \beta \subseteq \alpha$$

□

Proposition 8 *Given a finite set of time intervals A , $BIS(A)$ is unique.*

Proof. Assume that at least two different BIS exist for a finite set of time intervals A , i.e.,

$$(\exists B_1 = BIS(A), B_2 = BIS(A))(B_1 \neq B_2)$$

5. TIME PN-BASED DIAGNOSIS OF DES

$$\Rightarrow (\exists \beta \in B_1)(\forall \gamma \in B_2)(\beta \neq \gamma) \quad (1)$$

or conversely

$$(\exists \gamma \in B_1)(\forall \beta \in B_2)(\gamma \neq \beta) \quad (2)$$

According to Condition 3, $(\exists \alpha \in A)(\beta \cap \alpha \neq \emptyset)$

According to Proposition 7, $\beta \subseteq \alpha$

Since $B_2 = BIS(A)$, according to Condition 2,

$$(\exists \gamma' \in B_2)(\gamma' \subseteq \alpha, \gamma' \cap \beta \neq \emptyset)$$

$$(1) \Rightarrow \gamma' \neq \beta$$

We have already obtained that,

$$\beta \subseteq \alpha, \gamma' \subseteq \alpha, \beta \neq \gamma', \beta \cap \gamma' \neq \emptyset$$

According to the three possible relations between β and γ' as shown in Figure 5.3, we have either

$$(\exists \gamma'' \in B_2, \gamma'' \subseteq \alpha)(\gamma'' \cap (\overline{\gamma'} \cap \beta) \neq \emptyset) \quad (3) \text{ (cf. Figure 5.3(a) and 5.3(b))}$$

or

$$(\exists \beta' \in B_1, \beta' \subseteq \alpha)(\beta' \cap (\overline{\beta} \cap \gamma') \neq \emptyset) \quad (4) \text{ (cf. Figure 5.3(b) and 5.3(c))}$$

According to Condition 3 in Definition 24,

$$(3) \Rightarrow (\gamma'' \cap \beta \neq \emptyset) \wedge (\exists \alpha' \in A)(\gamma' \cap \alpha' \neq \emptyset, \gamma'' \cap \alpha' = \emptyset)$$

$$\beta \cap \gamma' \neq \emptyset, \gamma' \subseteq \alpha' \Rightarrow \beta \cap \alpha' \neq \emptyset$$

$$\Rightarrow \beta \subseteq \alpha'$$

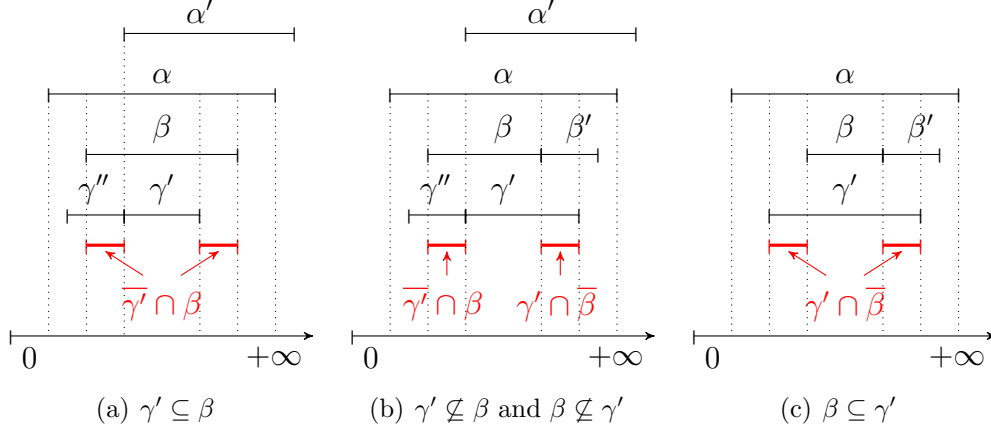


Figure 5.3: The relation between β and γ'

This violates $(\gamma'' \cap \beta \neq \emptyset) \wedge (\gamma'' \cap \alpha' = \emptyset)$. The same violations can be obtained if we consider (3). Therefore, the assumption does not hold.

We can make the same reasoning with (2) to come to the same result. Thus $B_1 = B_2$. \square

Example 30 Given a set of time intervals $A = \{[1, 3]; [2, 4]; [3, 7]; [5, +\infty[\}$, the solution of $BIS(A)$ according to Algorithm 8 can be obtained as follows:

1. Split all the intervals in A into semi-intervals which are gathered in a new set C :

$$C = \{[1; 3]; [2; 4]; [3; 7]; [5; +\infty[\}$$

2. Reorder the elements of C in the order \prec , i.e.,

$$[1 \prec [2 \prec [3 \prec 3] \prec 4] \prec [5 \prec 7] \prec +\infty[$$

3. For each pair of neighboring semi-intervals α and β in the ordering of the previous step such that $\alpha \prec \beta$, let $\alpha' = \bar{\alpha}$ (resp. $\beta' = \bar{\beta}$) if α is a left (resp. β is a right) semi-interval, otherwise $\alpha' = \alpha$ (resp. $\beta' = \beta$). Finally, $\alpha' \cap \beta'$ will be an element of $BIS(A)$, as shown in Table 5.1.
4. $BIS(A) = \{[1, 2[; [2, 3[; [3, 3]; [3, 4];]4, 5[; [5, 7];]7, +\infty[\}$, according to the fourth column in Table 5.1.

5. TIME PN-BASED DIAGNOSIS OF DES

Table 5.1: Solution of $BIS(A)$ in Example 30

$\alpha \prec \beta$	α'	β'	$\alpha' \cap \beta'$
$[1 \prec [2$	$[1$	$\overline{[2} = 2[$	$[1, 2[$
$[2 \prec [3$	$[2$	$\overline{[3} = 3[$	$[2, 3[$
$[3 \prec 3]$	$[3$	$3]$	$[3, 3]$
$3] \prec 4]$	$\overline{3]} =]3$	$4]$	$]3, 4]$
$4] \prec [5$	$\overline{4]} =]4$	$\overline{[5} = 5[$	$]4, 5[$
$[5 \prec 7]$	$[5$	$7]$	$[5, 7]$
$7] \prec +\infty[$	$\overline{7]} =]7$	$+\infty[$	$]7, +\infty[$

A graphical illustration for this solution is given in Figure 5.4.

5.2 Reachability Analysis of LTPNs with Fault Information

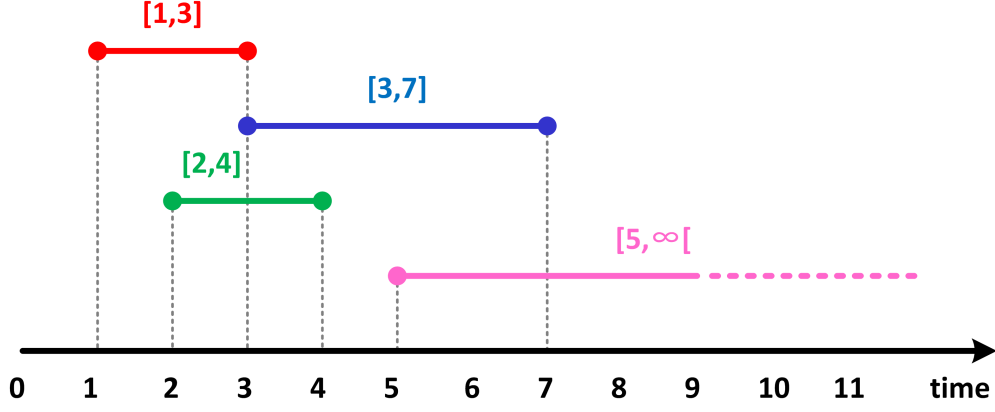
5.2.1 Augmented State Class (ASC)

In order to deal with the fault diagnosis problem using LTPNs, we need to associate each state class with a fault tag, from which we can determine whether a fault has occurred or not from the initial state class to the current one. Without loss of generality, we will deal with a unique class of fault.

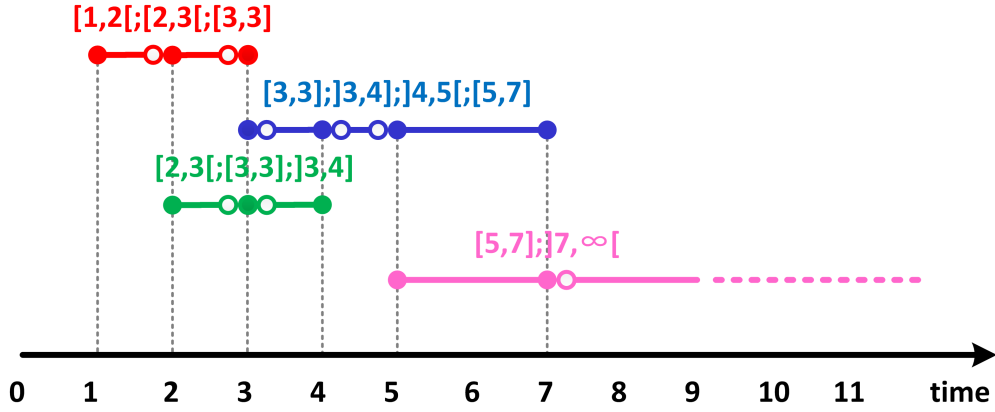
Definition 25 An ASC is a pair $x = (C, y)$, which is associated to an achievable firing sequence $\sigma \in T^*$ such that $C_0 \xrightarrow{\sigma} C$, and y is computed by:

$$y = \begin{cases} F & \text{if } \exists j, \sigma^j \in T_f \\ N & \text{otherwise} \end{cases}$$

The initial ASC is defined by $x_0 = (C_0, N)$, since we consider there is no fault in the system initially. Two ASCs $x = (C, y)$ and $x' = (C', y')$ are equivalent, iff $C = C'$, i.e., C and C' have the same marking and the same firing domain [Diaz, 2001], and $y = y'$.



(a) Intervals in set A



(b) Splitting intervals of A to compute $BIS(A)$

Figure 5.4: The computation of BIS in Example 30

Let \mathcal{N}_{ASC} be the set of ASCs relative to a given LTPN, mapping $\mathcal{T}_{ASC} : \mathcal{N}_{ASC} \times T^* \rightarrow \mathcal{N}_{ASC}$ defines transition between ASCs. We say an ASC $x' = (C', y')$ is reachable from $x = (C, y)$ by $\sigma \in T^*$, denoted by $x \xrightarrow{\sigma} x'$, iff:

- $C \xrightarrow{\sigma} C'$;

•

$$y' = \begin{cases} F & \text{if } (y = F) \vee (\exists j, \sigma^j \in T_f) \\ N & \text{otherwise} \end{cases} \quad (5.1)$$

5. TIME PN-BASED DIAGNOSIS OF DES

Consequently, the number of ASCs is at most twice the number of state classes. Recall that we consider that faults are permanent. Thus, fault propagation follows the same rules as in the untimed context, as shown in Figure 4.8 of the previous chapter.

Proposition 9 *A bounded LTPN has a finite number of ASCs.*

Proof. According to [Diaz, 2001], the number of state classes of a TPN is finite iff this net is bounded. A state class can be associated with at most two values: N or F. Therefore, given a bounded LTPN with m state classes, the number of ASCs will not exceed the number of combinations of state classes and tags, i.e., $2m$. \square

5.2.2 Sequence Duration

In the classical state class graph for a given TPN, a time interval assigned to a transition connecting two state classes, specifies the possible relative firable time of the transition from the source state class, i.e., the possible delay between the source and destination state classes. However, when considering the duration between any two state classes with some intermediate state classes, and due to diagonal constraints, one cannot simply sum up all the time intervals assigned to the transitions between them. In this section, we discuss how to compute this time duration, i.e., how to compute a firing sequence duration in a general way.

Definition 26 *Given an ASC x , we shall call a candidate sequence of x any sequence of transitions $\sigma t \in T^*$ which is achievable starting from x , where $\sigma \in T_u^*$ and $t \in T_o$.*

Note that σ may be empty.

We denote by $Can(x)$ the set of candidate sequences from ASC x . The ASCs which are reachable upon the firing of such sequences are called *candidate ASCs* of x . Formally speaking,

$$Can(x) = \bigcup_{\sigma \in T_u^* T_o} \{x' \text{ s.t. } x \xrightarrow{\sigma} x'\}$$

To each candidate sequence, one assigns a relative framing to its duration [Ghazel et al., 2009]. This obtained time interval contains all the possible firing dates of observable transition t relatively to x . Indeed, this duration is evaluated by solving the set of inequalities composed of the constraints for the intervals assigned to each transition which brings a state class to another. We use the notation $SD(\sigma)$ to denote the duration of a transition sequence σ .

Example 31 Consider the LTPN in Figure 5.5.

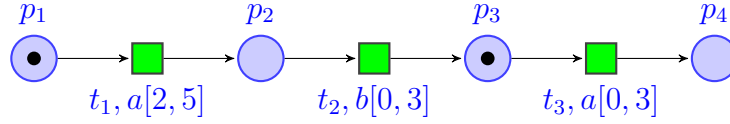


Figure 5.5: The LTPN graph for Example 30

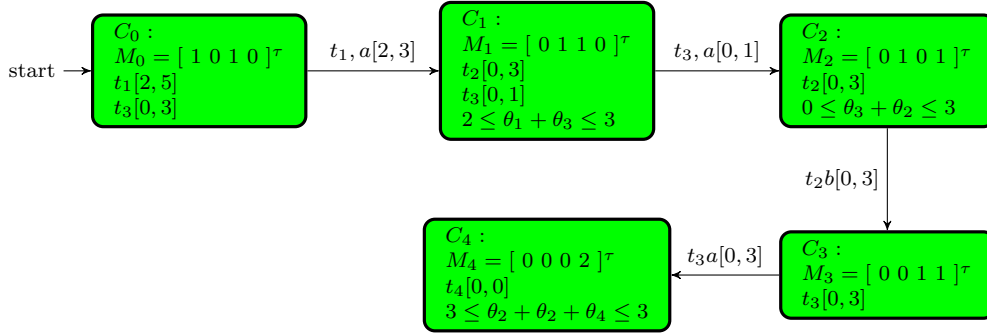


Figure 5.6: The state classes following σ for Example 30

One of the achievable firing sequences is $\sigma = t_1 t_3 t_2 t_3$, with the state classes generated along σ , i.e., from C_0 to C_4 , as shown in Figure 5.6. Assume the firing dates of the transitions in σ are respectively $\theta_1, \theta_3, \theta_2, \theta'_3$ (θ'_3 is the relative date of the second firing of t_3). Then $SD(\sigma)$ is computed by:

$$SD(\sigma) = \theta_1 + \theta_3 + \theta_2 + \theta'_3, \text{ where:}$$

5. TIME PN-BASED DIAGNOSIS OF DES

$$\left\{ \begin{array}{ll} 2 \leq \theta_1 \leq 3 & \text{constraint of } t_1 \text{ between } C_0 \text{ and } C_1 \\ 0 \leq \theta_3 \leq 1 & \text{constraint of } t_3 \text{ between } C_1 \text{ and } C_2 \\ 0 \leq \theta_2 \leq 3 & \text{constraint of } t_2 \text{ between } C_2 \text{ and } C_3 \\ 0 \leq \theta'_3 \leq 3 & \text{constraint of } t_3 \text{ between } C_3 \text{ and } C_4 \\ 2 \leq \theta_1 + \theta_3 \leq 3 & \text{constraint of } C_1 \\ 0 \leq \theta_3 + \theta_2 \leq 3 & \text{constraint of } C_2 \end{array} \right.$$

Here we obtain:

$$SD(\sigma) = [2, 9]$$

which is not simply the sum of intervals associating with each transition in the sequence (otherwise, we would have obtained $[2, 10]$).

Example 30 shows that the direct addition of time intervals assigned to transitions brings overestimation to the actual sequence duration.

5.2.3 ASC-Graph

In order to present both the reachability of ASCs upon observable events and fault propagation information of an LTPN, we develop a structure called ASC-graph. As mentioned earlier, without loss of generality, we consider a unique fault class.

Definition 27 An ASC-graph is a digraph $(\mathcal{Q}_{ASC}, \mathcal{A}_{ASC}, \mathcal{T}_{ASC}, q_0)$, where:

- $\mathcal{Q}_{ASC} \subseteq 2^{\mathcal{N}_{ASC}}$ is the set of ASC-graph nodes;
- $q_0 = (C_0, N)$ is the initial ASC-graph node;
- $\mathcal{T}_{ASC} : \mathcal{Q}_{ASC} \times \Sigma_o \rightarrow \mathcal{Q}_{ASC}$ is the transition mapping between ASC-graph nodes. Given $X \in \mathcal{Q}_{ASC}, e \in \Sigma_o$, $\mathcal{T}_{ASC}(X, e) = \{q' \mid \exists q \in X, \sigma \in Can(q), \varphi(\sigma) = e, q \xrightarrow{\sigma} q'\}$;
- \mathcal{I} is the set of time intervals (values in $\mathbb{Q}_{\geq 0}$);
- $\mathcal{A}_{ASC} \subseteq \mathcal{Q}_{ASC} \times (\Sigma_o \times 2^{\mathcal{I}}) \times \mathcal{Q}_{ASC}$ is the set of directed arcs of the ASC-graph: $\mathcal{A} = \{(q, e, i, q') \mid \exists \sigma \in Can(q), q' \in \mathcal{T}_{ASC}(q, e), \text{ s.t. } SD(\sigma) = i, \varphi(\sigma) = e\}$, where i denotes the set of possible time intervals.

The building procedure of the ASC-graph is in fact an augmented (with fault tag) ϵ -reduction problem on the state graph in which we are interested in the reachability of fault markings upon observable events. In this ϵ -reduction, only the fault markings which can be reached right after an observable event are kept; sequence duration between nodes in the ASC-graph is recomputed if there are some intermediary erased ASCs between them. In summary, an ASC-graph can be treated as an ϵ -reduced state class graph enriched by the fault propagation relation.

The ASC-graph is built *a priori* based on an existing state class graph, unlike the construction of the FM-graph in the untimed context which is performed on the fly, since the building of state class graphs is more complex than the reachability graph construction in the untimed context. Thanks to the analytical approach of building the state class graph [Berthomieu and Menasche, 1983] and the developed tools such as [Berthomieu et al., 2004; Gardey et al., 2005], the ASC-graph can be derived from the state class graph in a straightforward way. However, we will consider the on-the-fly building of ASC-graphs or other advanced models in the future, such that diagnosability analysis can be performed even more efficiently.

Example 32 Consider the LTPN in Figure 5.7, where $T_u = T_f = \{t_1\}$ and $T_o = \{t_2, t_3, t_4\}$. As a reference, the state class graph is given in Figure 5.8, where the grey boxes indicate the state classes reached right after an observable transition.

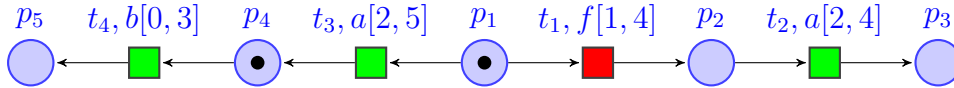


Figure 5.7: The LTPN graph for Example 32

We can compute, step by step (from Figure 5.9(a) to 5.9(g)), to obtain its ASC-graph as shown in Figure 5.9(g):

- Step 1 (cf. Figure 5.9(a)): from the initial ASC (C_0, N) , determine all the reachable ASCs, i.e., (C_1, F) , (C_2, N) and (C_3, N) .

5. TIME PN-BASED DIAGNOSIS OF DES

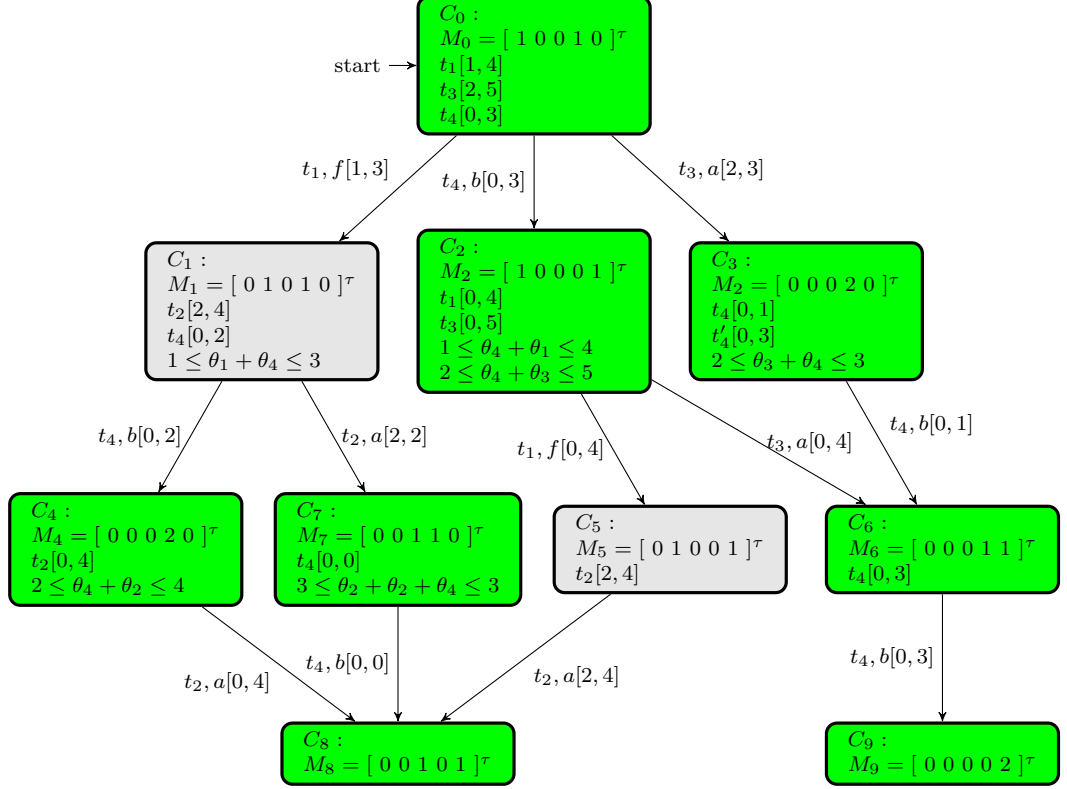


Figure 5.8: The state graph of the LTPN in Figure 5.7

- *Step 2 (cf. Figure 5.9(b))*: for each ASC obtained in Step 1 and reached from an unobservable transition (denoted by gray boxes in the figures) (here (C_1, N)), compute its reachable ASCs (here (C_4, F) and (C_7, F)). For general cases, this step will be repeated until each branch arrives at an ASC reached right after an observable transition.
- *Step 3 (cf. Figure 5.9(c))*: for any pair of ASCs reached after an observable transition (green boxes in the figures), connect them directly and label this new path with the observable event in this transition sequence with its corresponding sequence duration intervals. Here, (C_1, F) is erased with the addition of $b[1, 3]$ linking (C_0, N) and (C_4, F) and $a[3, 3]$ linking (C_0, N) and (C_7, F) ; $t_4, b[0, 3]$ and $t_3, a[2, 3]$ are replaced by $b[0, 3]$ and $a[2, 3]$ respectively.
- *Step 4 (cf. Figure 5.9(d))*: for each new ASC, do as Step 1 and 2.

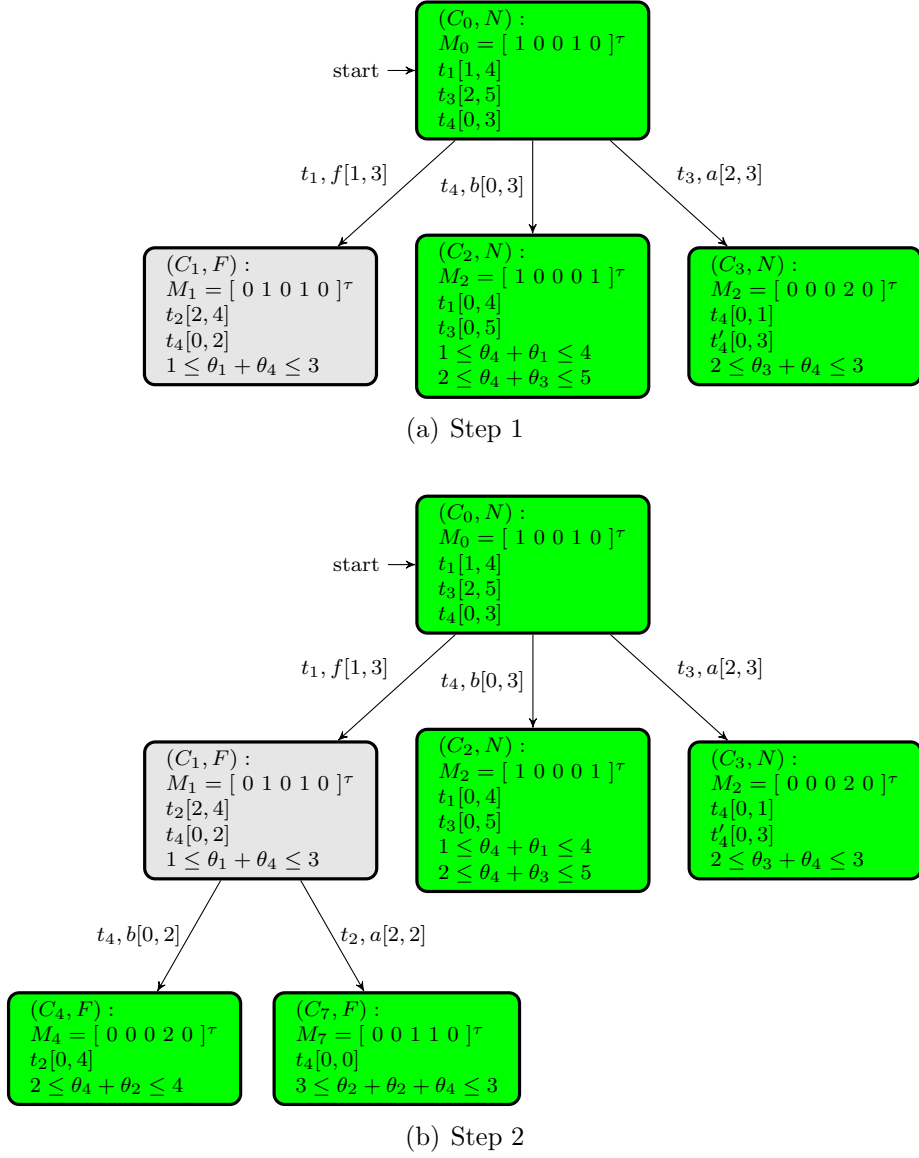


Figure 5.9: The construction of ASC-graph

- Step 5 (cf. Figure 5.9(e)): for each new pair of ASC reached after an observable transition, do as Step 3.
- Step 6 (cf. Figure 5.9(f)): repeat Step 1 to obtain ASC (C_9, N) .
- Step 7 (cf. Figure 5.9(g)): repeat Step 3 to replace $t_4, b[0, 3]$ with $b[0, 3]$. Until now no new ASC is reachable. Thus, the construction of this ASC-

5. TIME PN-BASED DIAGNOSIS OF DES

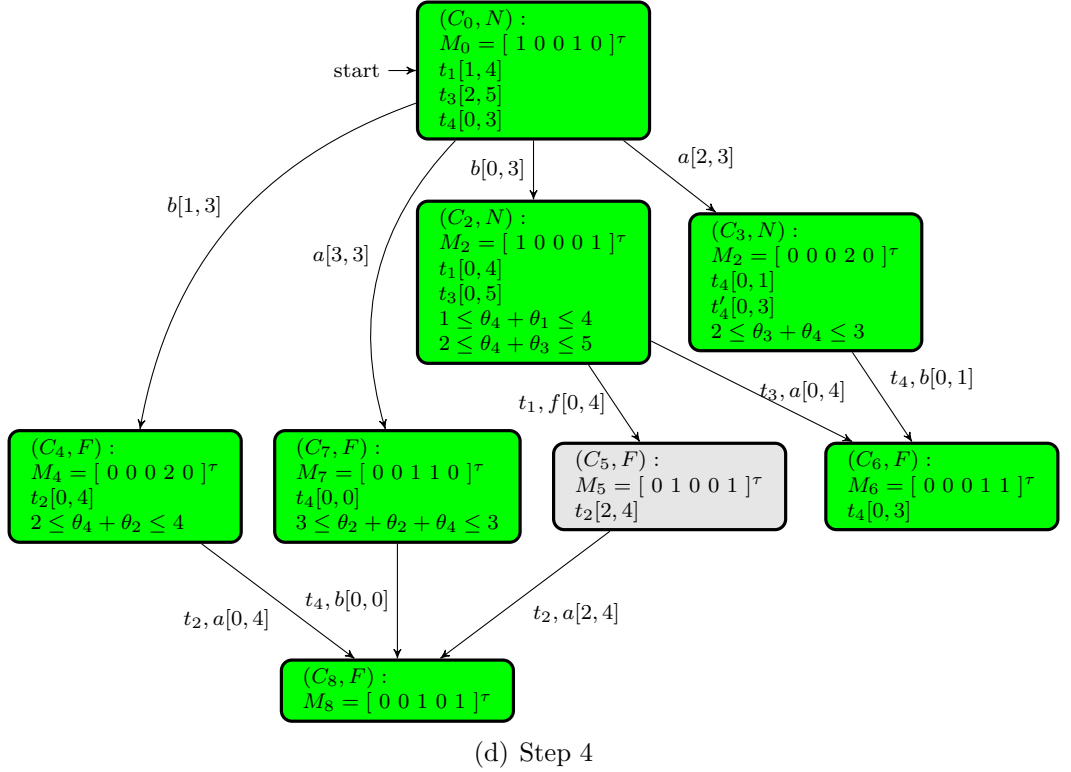
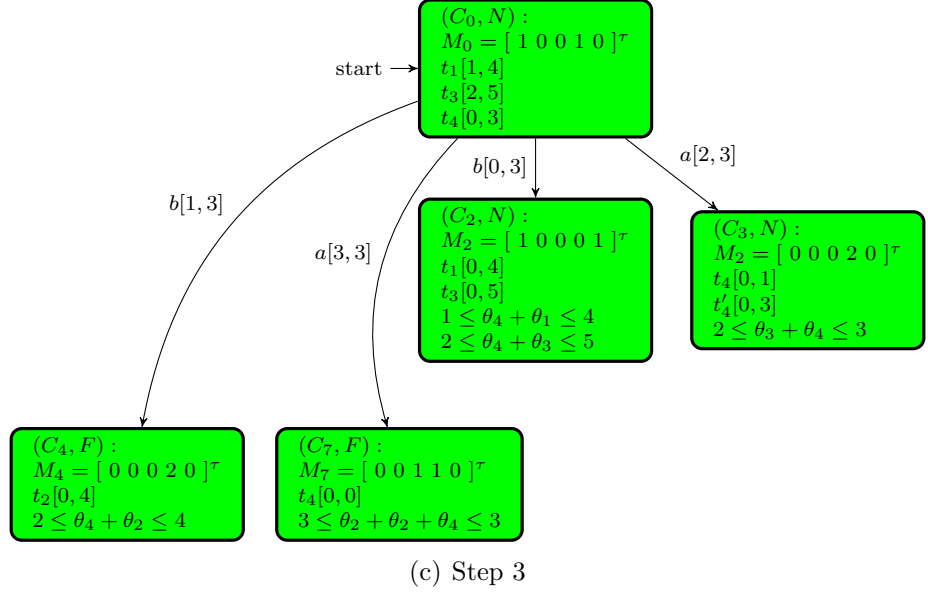
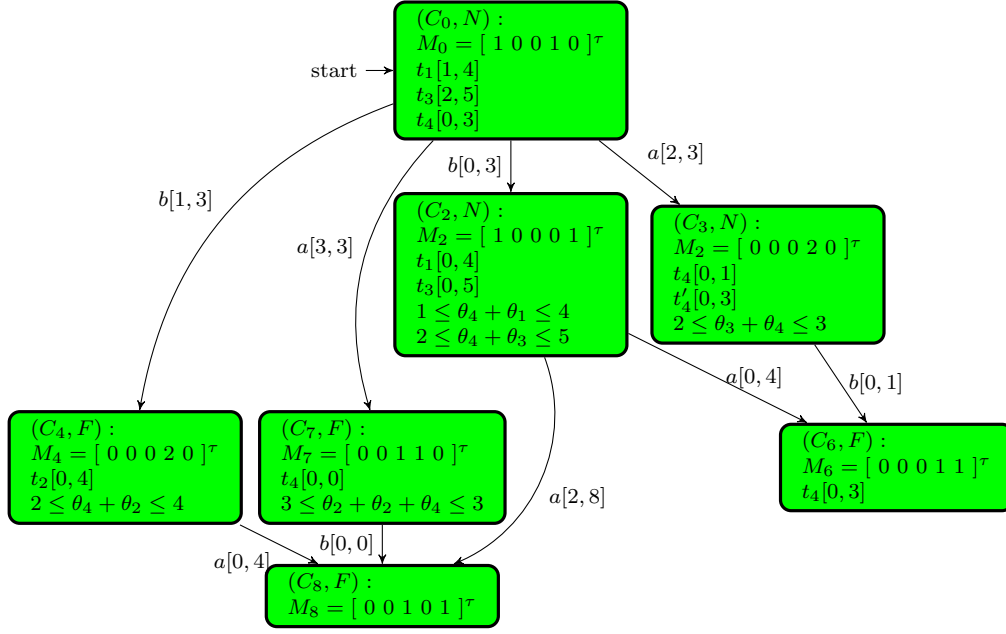
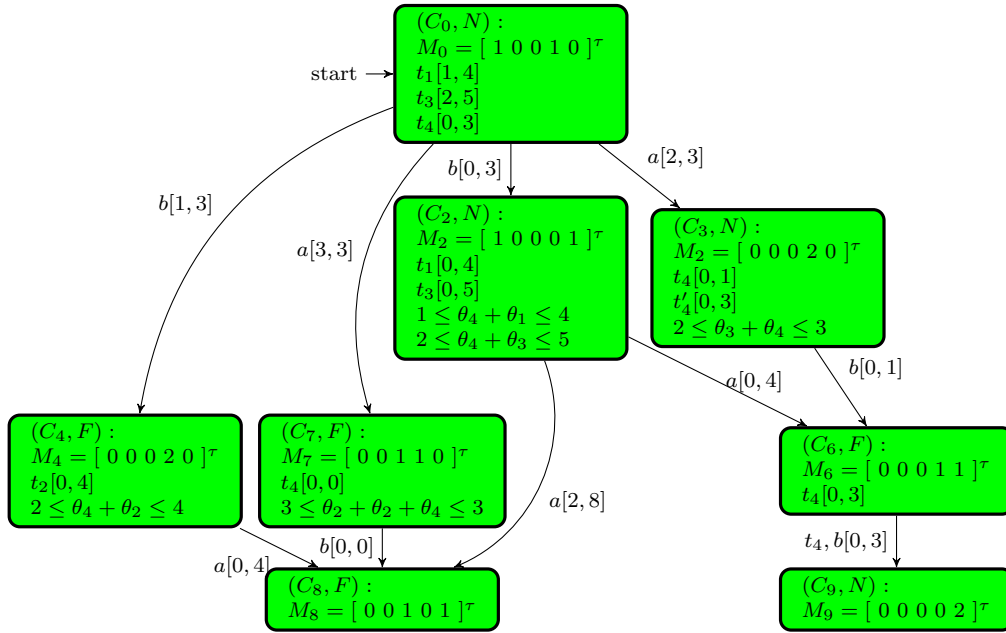


Figure 5.9: The construction of ASC-graph (Continued)



(e) Step 5



(f) Step 6

Figure 5.9: The construction of ASC-graph (Continued)

5. TIME PN-BASED DIAGNOSIS OF DES

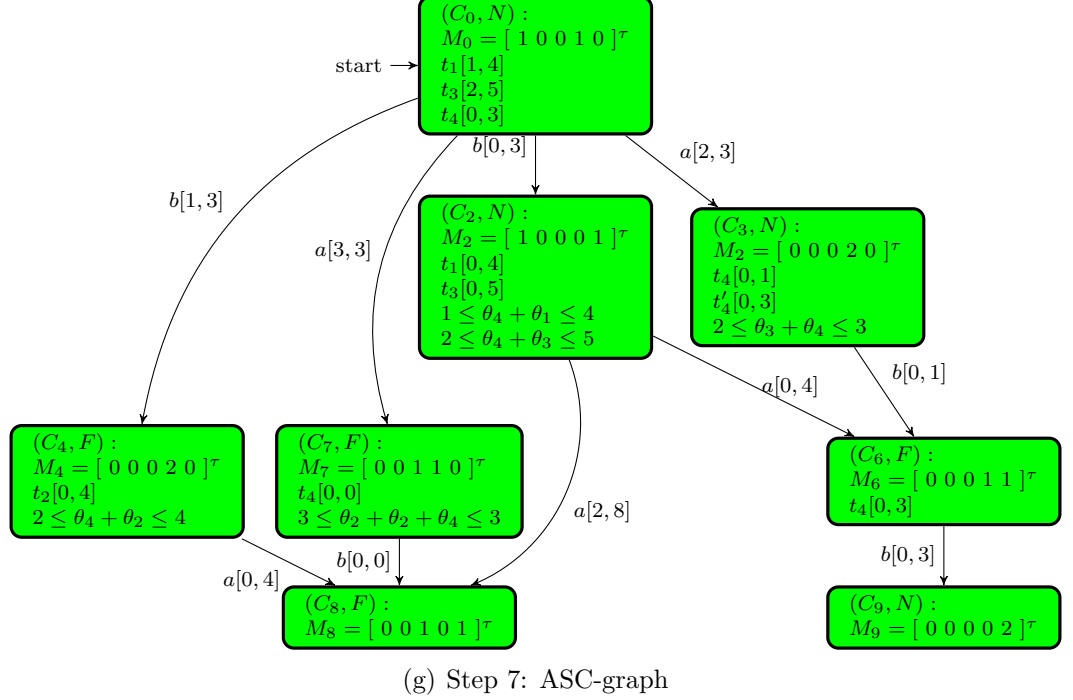


Figure 5.9: The construction of ASC-graph (Continued)

graph is terminated.

5.2.4 ASC-Set

In order to determinize an LTPN for state estimation and diagnosability analysis, we will gather the states reached by the same timed observation (observable event and its corresponding occurrence date) in some sets called ASC-sets.

An ASC-set is then an element of $2^{\mathcal{N}_{ASC}}$. The initial ASC-set is defined by $\{x_0\}$.

Given an ASC-set g , we say $e \in \Sigma_o$ is a *candidate event* of g , if $\exists x \in g, \mathcal{T}_{ASCg}(x, e) \neq \emptyset$. We denote by $CES(g)$ the candidate event set of g . The *candidate interval set (CIS)* of g relative to e is defined by $CIS(g, e) = BIS(Y)$, where $Y = \{SD(\sigma) \mid \exists x \in g, \sigma \in Can(x), \text{s.t. } \varphi(\sigma) = e\}$.

In other words, $CIS(g, e)$ is the basic interval set relative to the intervals corresponding to the possible delays for e to occur from an element in g .

Example 33 Consider the CIS of the initial ASC-set $g_0 = \{(C_0, N)\}$ of the LTPN in Figure 5.7. As shown in Figure 5.9(g), from the only element (C_0, N) in g_0 , 4 firable transitions $a[3, 3]$, $b[1, 3]$, $b[0, 3]$ and $a[2, 3]$ exist. Therefore, all the possible firing time of observable event a relative to g_0 can be gathered in the set $CIS(g_0, a) = \{[2, 3[; [3, 3]\}$. In other words, one can obtain a new ASC from g_0 upon observable event a at a date in $[2, 3[; [3, 3]$ relative to g_0 . Likewise, $CIS(g_0, b) = \{[0, 1[; [1, 3]\}$.

Let \mathcal{G} be the set of reachable ASC-sets. Given $g \in \mathcal{G}$, $e \in CES(g)$ and $i \in CIS(g, e)$, the transition mapping between ASC-sets $\xi : \mathcal{G} \times \Sigma_o \times \mathcal{I} \rightarrow \mathcal{G}$ is defined by:

$\xi(g, e, i) = \{x' \mid \exists x \in g, \sigma \in Can(x), \varphi(\sigma) = e, x \xrightarrow{\sigma} x' \text{ s.t. } i \subseteq SD(\sigma)\}$, where \mathcal{I} is the set of time intervals.

The ASC-set g is said to be

- normal, if $\forall (C, y) \in g, y = N$;
- F-certain, if $\forall (C, y) \in g, y = F$;
- F-uncertain, otherwise.

We denote $tag(g) = N$ (resp. F, U), if g is normal (resp. F-certain, F-uncertain).

Example 34 This example shows how to compute the reachable ASC-set from the initial ASC-set $g_0 = \{(C_0, N)\}$ of the LTPN in Figure 5.1(c):

- Step 1 (cf. Figure 5.10(a)): for each ASC in g_0 , here (C_0, N) , determine all the reachable ASCs: (C_4, F) , (C_7, F) , (C_2, N) and (C_3, N) .
- Step 2: for each candidate event of g_0 , here a and b , compute its CIS:

$$CIS(g_0, a) = BIS([3, 3]; [2, 3]) = \{[2, 3[; [3, 3]\}$$

$$CIS(g_0, b) = BIS([1, 3]; [0, 3]) = \{[0, 1[; [1, 3]\}$$

- Step 3 (cf. Figure 5.10(b)): for each ASC in g_0 , recompute its reachable ASCs upon each observable event $e \in \Sigma_o$ with the relative firing interval in $CIS(g_0, e)$.

5. TIME PN-BASED DIAGNOSIS OF DES

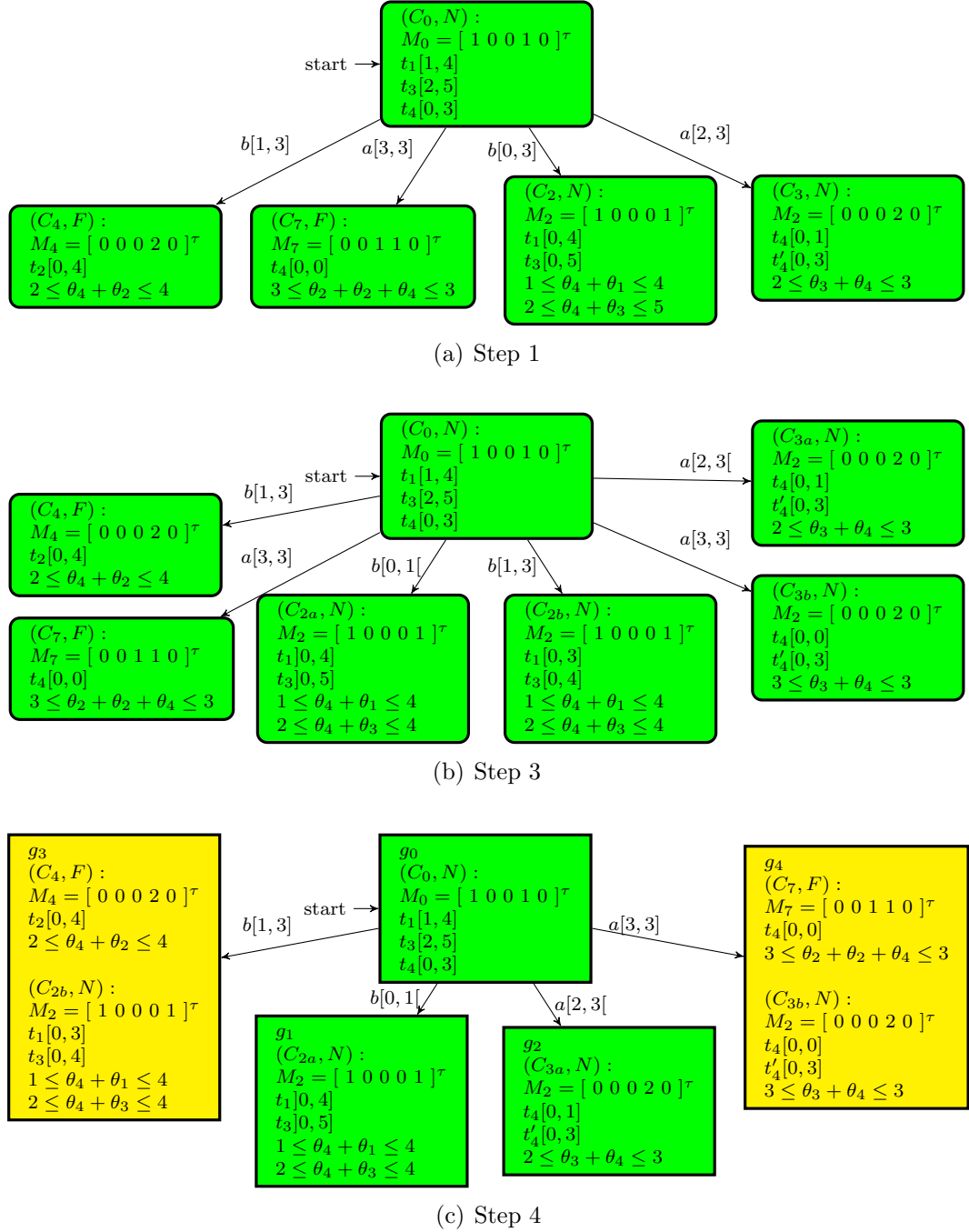


Figure 5.10: Computation of the reachable ASC-set of the initial ASC-set of LTPN in Figure 5.7

-
- *Step 4 (cf. Figure 5.10(c))*: collect the ASCs reached after the same label (observable event and the corresponding firing interval) as a new ASC-set from g_0 .

5.2.5 ASC-Set Graph (ASG)

The augmented state class set graph (ASG) is introduced as a deterministic digraph which will serve as a basis to check diagnosability. Here the term “deterministic” means that, given an ASC-set and a candidate sequence, we can deduce with certainty which candidate ASCs the system will be possibly in. In other words, the ASG can be treated as a timed diagnoser such that, given any timed trace, it estimates the possible system states, as well as the corresponding fault information.

The ASG is a digraph $(\mathcal{G}, \mathcal{R}, \xi, g_0)$, where:

- $\mathcal{G} \subseteq 2^{\mathcal{N}_{ASC}}$ is the set of ASG nodes;
- $g_0 = \{x_0\} = \{(C_0, N)\}$ is the initial node;
- ξ is the transition mapping between ASC-sets;
- $\mathcal{R} \subseteq \mathcal{G} \times \Sigma_o \times \mathcal{I} \times \mathcal{G}$ is the set of ASG arcs: $\mathcal{R} = \{(g, e, i, g') \mid g' = \xi(g, e, i)\}$.

The ASG can be computed by Algorithm 9.

Example 35 *The ASG of the LTPN in Figure 5.7 can be computed by Algorithm 9. The main procedure is performed as follows:*

- *Step 1: select an existing ASC-set. Initially, there is only ASC-set g_0 . Compute all the reachable ASC-sets from the selected ASC-set g_0 , as shown in Figure 5.10(c).*
- *Step 2: for each newly obtained ASC-set, repeat Step 1 until no new ASC-sets can be obtained, which means that the ASG is completed, as shown in Figure 5.11.*

5. TIME PN-BASED DIAGNOSIS OF DES

Algorithm 9 Construction of the ASG

```

1: Input: the ASC-graph  $(\mathcal{N}_{ASC}, \mathcal{A}, \gamma, x_0)$ ;
2: Output: the ASG;
3:  $g_0 \leftarrow \{x_0\}$ ;  $\triangleright$  initialization
4:  $\mathcal{G}_{con} \leftarrow \{g_0\}$ ;  $\triangleright \mathcal{G}_{con}$  is the set of ASC-sets to be considered.
5:  $\mathcal{G}_{vst} \leftarrow \emptyset$ ;  $\triangleright \mathcal{G}_{vst}$  is the set of ASC-sets that have been considered.
6: while  $\mathcal{G}_{con} \neq \emptyset$  do
7:   pick a node  $g \in \mathcal{G}_{con}$  s.t.  $g \notin \mathcal{G}_{vst}$ ;
8:   for all  $e \in CES(g)$  do
9:      $Y \leftarrow CIS(g, e)$ ;
10:    for all  $i \in Y$  do
11:       $\mathcal{G}_{con} \leftarrow \mathcal{G}_{con} \cup \{\xi(g, e, i)\}$ ;
12:     $\mathcal{G}_{con} \leftarrow \mathcal{G}_{con} \setminus \{g\}$ ;
13:     $\mathcal{G}_{vst} \leftarrow \mathcal{G}_{vst} \cup \{g\}$ ;

```

5.3 Checking Diagnosability

The definition of diagnosability is introduced in Section 2.2.4.3. Without loss of generality, we first discuss the diagnosability for one class of faults Σ_{F_i} . The generalization of our approach can be obtained just by repeating the same process for each class Σ_{F_i} . For the sake of checking diagnosability based on observable events with their occurrence dates, we will propose a deterministic structure called *ASG* on the basis of *ASC-sets*.

Before discussing the timed diagnosability of DES, we make the following assumptions, in the same way as in the untimed context:

- The LTPN is bounded;
- No achievable cycle of unobservable transitions exists;
- Faults are permanent, i.e., when a fault occurs the system remains indefinitely faulty.

Note that the liveness condition is relaxed.

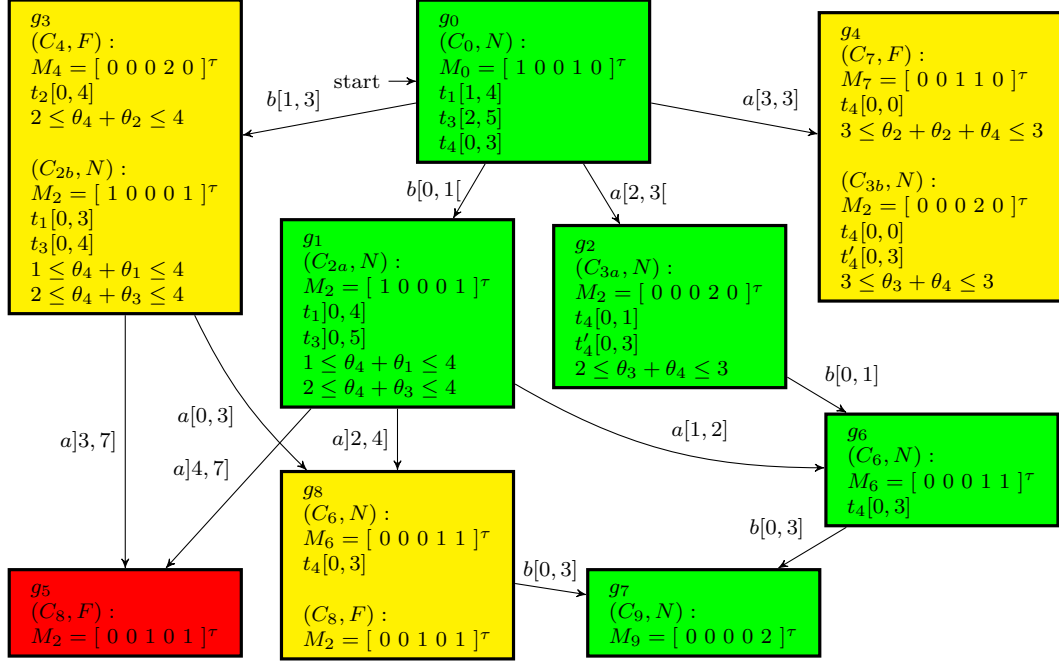


Figure 5.11: The ASG for Example 35

5.3.1 Conditions for Undiagnosability

As we have explained earlier, the ASG offers a state representation that distinguishes between reachable states, based on an explicit discrimination taking into account both observable events, and their possible occurrence dates. Defining such a structure makes it possible to use similar analysis as in the untimed context. However, some other considerations related to time still need to be added, as will be presented in the following.

Condition 1: indeterminate cycle

Recall that the condition for undiagnosability of an automaton is the existence of an indeterminate cycle as proved in [Sampath et al., 1995]. We can extend this condition for the analysis of diagnosability of LTPN on the basis of the ASG, since our technique, which consists in splitting time intervals, makes it possible to derive an untimed-diagnoser-like structure, by making the distinction between sequences on the basis of temporal criteria explicit in the ASG model structure.

By analogy with the untimed context, we define an indeterminate cycle in an ASG as a cycle composed of finite nodes in the graph, such that for any node g in

5. TIME PN-BASED DIAGNOSIS OF DES

this cycle, there are two ASCs $x_1, x_2 \in g$, x_1 is a faulty ASC in a cycle composed of faulty nodes in the ASC-graph, while x_2 is a normal ASC in a cycle composed of normal nodes in the ASC-graph.

Proposition 10 *The LTPN is undiagnosable if an indeterminate cycle in the ASG exists.*

This is obvious according to the explanation of indeterminate cycle. Note that a cycle of F-uncertain ASC-sets in ASG (Figure 5.12(a), where the black boxes are faulty ASCs and the white ones are normal) is not necessarily an indeterminate cycle. If this cycle corresponds to two ASC cycles in an ASC-graph such that one is a normal cycle (x_1, x_3) and the other is a faulty one (x_2, x_4) as in Figure 5.12(b), then g_1 and g_2 form an indeterminate cycle. Otherwise, they do not (Figure 5.12(c)).

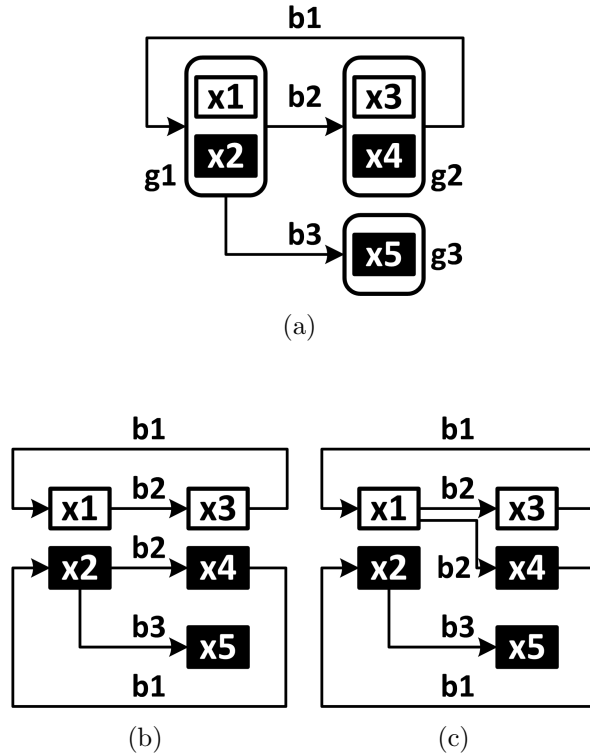


Figure 5.12: Illustration of indeterminate cycle

Condition 2: infinite sequence duration in certain cases

Given an ASC-set g (which is not the initial ASC-set) and its predecessor g' , we define the *maximum delay mapping*:

$$mdelay : \mathcal{N}_{ASC} \times \mathcal{N}_{ASC} \rightarrow \mathbb{Q}_{\geq 0} \cup \{+\infty\}$$

by $mdelay(g', g) =$

$$\max\{SD(\sigma) \mid \exists x' \in g', x \in g, \sigma' \in (T_u \setminus T_f)^*, t_f \in T_f, \sigma \in T_u^* T_o \text{ s.t. } x' \xrightarrow{\sigma' t_f \sigma} x\}$$

In other words, $mdelay(g', g)$ is the maximum delay between the first possible occurrence of a fault and g , relatively to a predecessor ASC-set g' .

We also define

$$fdelay : \mathcal{N}_{ASC} \rightarrow \mathbb{Q}_{\geq 0} \cup \{+\infty\}$$

by $fdelay(g) =$

$$\max\{SD(\sigma) \mid \exists x \in g, \sigma' \in (T \setminus T_f)^*, t_f \in T_f, \sigma \in T^* T_o \text{ s.t. } x_0 \xrightarrow{\sigma' t_f \sigma} x\}.$$

Here $fdelay(g)$ is the maximum delay between the first occurrence of a fault and g , relatively to the initial ASC-set $\{x_0\}$.

We define

$$SD_{max}(g', g) = \max\{SD(\sigma) \mid \exists x' \in g', x \in g, \sigma \in T_u^* T_o \text{ s.t. } x' \xrightarrow{\sigma} x\}$$

Proposition 11 *An LTPN is undiagnosable if g' is the predecessor of ASC-set g , and*

1. $mdelay(g', g) = +\infty$ or
2. $SD_{max}(g', g) = +\infty$ if $tag(g') = U$ and $tag(g) \in \{U, F\}$.

Proof. We prove the above proposition according to the definition of diagnosability. For an ASC-set g , it may be normal, F-uncertain or F-certain, i.e., $tag(g) \in \{N, U, F\}$.

- If $tag(g) = N$, no diagnosability verdict can be concluded and further investigation is needed.
- If $tag(g) = U$, for its predecessor g' , $tag(g') \in \{N, U\}$.

5. TIME PN-BASED DIAGNOSIS OF DES

- If $tag(g) = N$, there must be a fault in one of the paths (sequences of transitions) between an ASC in g' and an ASC in g . If $mdelay(g', g) = +\infty$, the system is undiagnosable. Otherwise, no diagnosability verdict can be concluded.
- If $tag(g) = U$ and $SD_{max}(g', g) = +\infty$, the system is undiagnosable. Otherwise, no diagnosability verdict can be concluded.
- If $tag(g) = F$, for its predecessor g' , $tag(g') \in \{N, U, F\}$.
 - If $tag(g) = N$, there must be a fault in one of the paths between an ASC in g' and an ASC in g . If $mdelay(g', g) = +\infty$, the system is undiagnosable. Otherwise, no diagnosability verdict can be concluded.
 - If $tag(g) = U$ and $SD_{max}(g', g) = +\infty$, the system is undiagnosable. Otherwise, no diagnosability verdict can be concluded. \square

Condition 3: dead subset in certain cases

Note that here we also deal with non-live systems. For this, let us introduce the following definitions. An ASC-set g is said to be:

- *undead or nonblocking*, if $\forall x \in g, \exists t \in T$ s.t. $x \xrightarrow{t}$;
- *dead*, if $\forall x \in g, \nexists t \in T$ s.t. $x \xrightarrow{t}$;
- *quasi-dead*, otherwise.

Given a quasi-dead ASC-set g , we define the *dead subset* of g as the set of all dead ASCs in g , which can be formalized as: $DS(g) = \{x \in g \mid \nexists t \in T \text{ s.t. } x \xrightarrow{t}\}$.

Example 36 *Let us consider the example in Figure 5.11. There are two dead ASC-sets g_6, g_7 , and a quasi-dead ASC-set g_5 with dead subset $DS(g_5) = \{(C_8, F)\}$.*

We will now discuss some conditions for undiagnosability w.r.t the liveness of ASC-sets.

Proposition 12 *An LTPN is undiagnosable if a quasi-dead ASC-set g exists, such that*

3. $DS(g)$ is *F-uncertain*, or

4. $DS(g)$ is *F-certain* and a normal successor ASC-set g' exists such that g' may be reached upon an infinite delay $(+\infty)$.

Proof. For condition (3), an F-uncertain dead subset means that, some ASCs in this set may be reachable by firing a sequence containing a fault, while others can be reached without any fault having been occurred. Furthermore, it is not possible to distinguish them by further observation, since they are all dead and the system will remain in F-uncertain state forever.

For condition (4), if g' is reachable upon an infinite delay, one cannot determine whether the system is blocked in the (faulty) dead subset of g ($DS(g)$), or it is still in the way to g' , which means that it is possible that no fault has occurred in the state g' or $DS(g)$ in a finite delay after the fault, i.e., we do not know if a fault has occurred. \square

5.3.2 On-the-Fly Checking of Diagnosability

Proposition 13 *A bounded LTPN is diagnosable iff none of the conditions in Propositions 10, 11 and 12 holds.*

Proof. (\Rightarrow) : This condition is proposed from three perspectives that we consider:

1. With the help of splitting intervals, the behavior of LTPN is characterized as in the untimed context, where non-existence of indeterminate cycle has been proved to be necessary and sufficient condition for diagnosability [Sampath et al., 1995].
2. This is the restriction from the definition of diagnosability of LTPN.
3. This is the restriction from the perspective of considering non-live TDES.

(\Leftarrow) : The negation of these three conditions has been proved to be necessary by Propositions 10, 11 and 12, since each of the conditions in Propositions 10, 11 and 12 is sufficient for undiagnosability. \square

5. TIME PN-BASED DIAGNOSIS OF DES

We have shown that diagnosability can be checked while building ASG. Actually, building the whole ASG would be similar to the approach based on state enumeration, often consuming much memory while dealing with large systems, even if this burdensome work could be performed off line. Yet, there is still a difference w.r.t this approach, since ASG branch building is stopped as soon as an F-certain ASC-set is found or if one of the conditions for undiagnosability (cf. Propositions 10, 11 and 12). In order to tackle this problem, we will propose a new approach to check diagnosability on the basis of on-the-fly building of the ASG, as shown in Algorithm 10. Moreover, we determine the minimum value Δ_{min} for which the system is diagnosable. Hence, when the system is diagnosable and with Δ_{min} being determined, the system is Δ -diagnosable for any $\Delta \geq \Delta_{min}$ and is not Δ -diagnosable for any $\Delta < \Delta_{min}$.

5.4 Discussion on Bisimulation between ERA and ASG

The ASG we have developed is a special structure that is similar to a type of TA called event-recording automata (ERA). In this section, we discuss the similarities between ASG and ERA and discuss the bisimulation between them under certain conditions.

Definition 28 [*Cassez, 2012*] *The timed automaton A is said to be deterministic if*

1. *there is no ϵ labeled transition in A , and*
2. *whenever (l, g, a, r, l') and (l, g', a, r', l'') are transitions of A , $g \wedge g' \equiv \text{FALSE}$.*

Condition 1, like that in untimed automata, does not allow the existence of empty event ϵ . Condition 2 is quite different from that for untimed automata. It implies that the non-deterministic case for untimed automata, such as a state having two output transitions with the same event, could be deterministic in TA whenever the two transitions cannot occur at the same time.

Algorithm 10 On-the-fly building of ASG, checking diagnosability and computing Δ_{min}

```

1: Input: the ASC-graph;
2: Output: diagnosability of  $G$  and (if  $G$  is diagnosable)  $\Delta_{min}$ ;
3:  $g_0 = \{x_0\}$ ;
4:  $\Delta_{min} = 0$ ;
5:  $\mathcal{G}_{vst} \leftarrow \emptyset$ ;
6:  $\mathcal{G}_{con} \leftarrow \{g_0\}$ ;
7: while  $\mathcal{G}_{con} \neq \emptyset$  do pick a node  $g \in \mathcal{G}_{con}$  with  $g \notin \mathcal{G}_{vst}$ ;
8:   for all  $e \in \Sigma_o$  do
9:      $\mathcal{I} \leftarrow CIS(g, e)$ ;
10:    for all  $i \in \mathcal{I}$  do
11:       $g' \leftarrow \xi(g, e, i)$ ;
12:      if  $tag(g') = U$  then
13:        if  $\exists g'' \in \mathcal{G}_{vst}$  s.t.  $g'' = g'$  then
14:          if  $g'$  is in an indeterminate cycle then
15:            return  $G$  is undiagnosable;
16:          if  $(tag(g) = N) \wedge (mdelay(g, g') = +\infty)$  then
17:            return  $G$  is undiagnosable;
18:          if  $(tag(g) = U) \wedge (SD_{max}(g, g') = +\infty)$  then
19:            return  $G$  is undiagnosable;
20:          if  $tag(DS(g')) = U$  then
21:            return  $G$  is undiagnosable;
22:           $\Delta_{min} \leftarrow \max(\Delta_{min}, fdelay(g'))$ ;
23:      if  $tag(g') = F$  then
24:        if  $(tag(g) = N) \wedge (mdelay(g, g') = +\infty)$  then
25:          return  $G$  is undiagnosable;
26:        if  $(tag(g) = U) \wedge (SD_{max}(g, g') = +\infty)$  then
27:          return  $G$  is undiagnosable;
28:         $\Delta_{min} \leftarrow \max(\Delta_{min}, fdelay(g'))$ ;
29:     $\mathcal{G}_{con} \leftarrow \mathcal{G}_{con} \setminus \{g\}$ ;
30:     $\mathcal{G}_{vst} \leftarrow \mathcal{G}_{vst} \cup \{g\}$ ;
31: return  $G$  is  $\Delta_{min}$ -diagnosable;

```

Unlike an untimed (non-deterministic) automaton, from which a deterministic automaton observer can always be built, a timed automaton is not always determinizable [Alur et al., 1994]. Moreover, the determinization of a timed automaton is proved to be undecidable [Tripakis, 2006].

5. TIME PN-BASED DIAGNOSIS OF DES

However, a subclass of determinizable TA exists, which is called Event-Clock Automata (ECA) [Alur et al., 1994, 1999]. According to the type of clocks, ECA is further divided into two subclasses: Event-Recording Automata (ERA) and Event-Predicting Automata (EPA). It is worth discussing the ERA here, since it has some similarities in the expressiveness as an ASG of LTPN.

Let Σ be a finite set of events. For every event $a \in \Sigma$, we write x_a to denote the *event-recording clock* of a . Given a timed word $\omega = (a_0, t_0)(a_1, t_1) \dots (a_n, t_n)$, the value of the clock x_a at the j^{th} position of ω is $t_j - t_i$, where i is the largest position preceding j such that a_i equals a . If no occurrence of a precedes the j^{th} position of ω , then the value of the clock x_a is “undefined”, denoted by \perp . We write $\mathbb{R}_\perp = \mathbb{R} \cup \{\perp\}$ for the set of nonnegative real numbers together with the special value \perp .

Definition 29 (*event-recording clock*) [Alur et al., 1994] For all $0 \leq j \leq n$,

$$\lambda(\omega, j)(x_a) = \begin{cases} t_j - t_i & \text{if } i \text{ exists such that } 0 \leq i < j \text{ and } a_i = a \\ & \text{and for all } k \text{ with } i < k < j, a_k \neq a, \\ \perp & \text{if } a_k \neq a \text{ for } k \text{ with } 0 \leq k < j. \end{cases}$$

An event-recording automaton (ERA) is a timed automaton with event-recording clocks. It contains, for every event a , a clock that records the time of the last occurrence of a . The class of event-recording automata is, on the one hand, expressive enough to model (finite) timed transition systems and, on the other hand, determinizable and closed under all boolean operations. The translation from timed transition systems to event-recording automata, which leads to an algorithm for checking if two timed transition systems have the same set of timed behavior, has been presented in [Alur et al., 1994].

It should be noted that our developed ASG for LTPN has the following features:

- An ASG is a specific structure for analyzing diagnosability of LTPN. Given an LTPN, different ASGs may be, in general, built on the fly according to different search strategies. However, different ASGs result in the same

diagnosability verdict. In particular, the on-the-fly-built ASG will be the complete ASG, if all the behavior of LTPN has been investigated.

- An ASG is an observer-like, but not exactly an observer for LTPN, since it is not necessarily determinizable. The ASG is just the observer only if the LTPN is determinizable and all the behavior are well considered while building the ASG.
- There is no strong connection between diagnosability and “determinizability” for LTPNs. A diagnosable LTPN may be either determinizable or not; and a determinizable LTPN may be either diagnosable or not. Given a diagnosable LTPN, (at least) the on-the-fly-built ASG is a time deterministic structure such that any timed behavior corresponds to a state of ASG.
- On the fly technique allows analyzing diagnosability for certain non-determinizable LTPNs, since the generation of state space of ASG always stops when an F-certain state is obtained, no matter whether future behavior is determinizable or not.

The general LTPNs can be non-deterministic and not necessarily determinizable. However, a subclass of LTPNs exists which are determinizable and their ASG is deterministic to be transformed into ERA. For example, the ASG in Figure 5.11 can be transformed into an ERA as shown in Figure 5.13.

We guess that our developed ASG can always be transformed into a language-equivalent ERA. However, we do not prove it here and further study on the bisimulation between the two models will be performed in the future.

5.5 Online Diagnosis

In this section, we discuss how online diagnosis for a diagnosable LTPN model is performed, using a deterministic structure called *LTD* that will be developed. By observing events with their corresponding occurrence dates online, one can deduce with certainty which state (Normal, F-uncertain or F-certain) the system can be in and give the verdict pertinent to fault occurrences.

5. TIME PN-BASED DIAGNOSIS OF DES

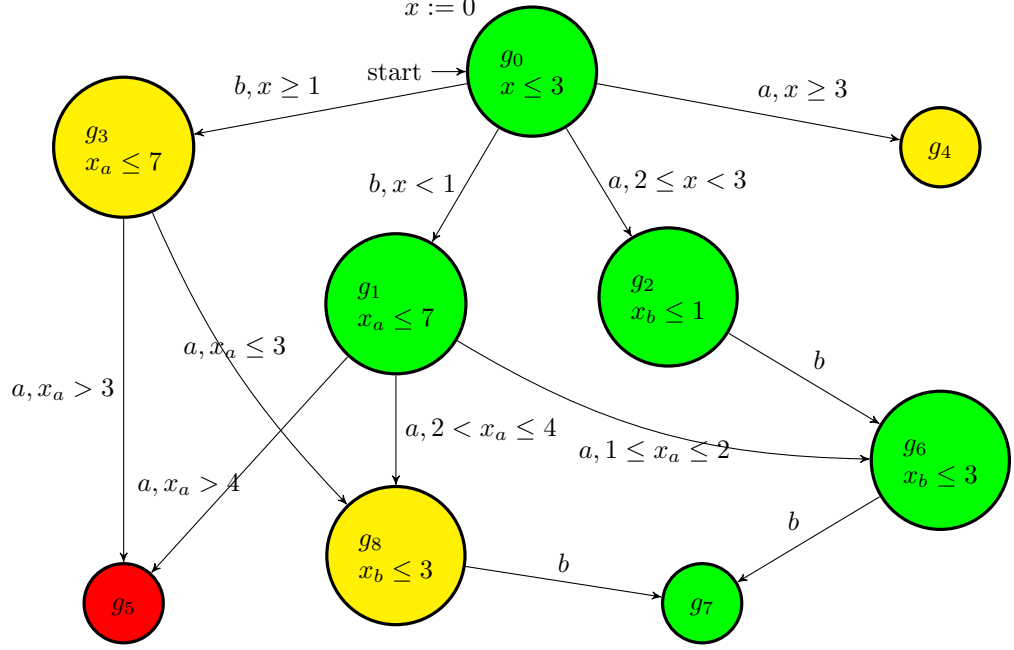


Figure 5.13: A language equivalent ERA for the ASG in Figure 5.11

The LTD is obtained from the ASG by erasing all the information except fault tags for each node in the ASG and observable events labeling the arcs with their corresponding intervals. This procedure deletes all the information unnecessary for diagnosis.

For each F-uncertain quasi-dead node g , a virtual node g' labeled with “F” is created as a successor to g , and the arc from g to g' is labeled with (ϵ, i) , where ϵ is an empty event indicating that no event is observed, i is the interval from the maximum firing date of the other firable transitions to $+\infty$. Note that, this virtual component does not belong to the LTD, while it helps to diagnose a fault when dealing with non-live systems, as will be illustrated through Example 37.

The tags associated with each node in an LTD provide the same information as that in the ASG:

- “N” means that no fault has occurred;
- “U” denotes that a fault has possibly occurred, and further observation is needed before being able to give a precise diagnosis verdict;

- “F” denotes that a fault has occurred with certainty.

Given a system behavior presented by a sequence of observable events with their corresponding event occurrence dates, one can find whether a fault has occurred or not, with the help of the LTD. The algorithm for online diagnosis of LTPNs is given in Algorithm 11.

Algorithm 11 Fault detection using the LTD

```

1: Input: An LDFS composed of  $(e_j, d_j)$ ;
2: Output: The current system state;
3:  $q_0 \leftarrow N$ ; ▷ The system is normal ( $N$ ) after the initialization.
4:  $j \leftarrow 1$ ;
5: while the system is in operation do
6:   Wait for the input observable event  $e_j$  with its occurrence date  $d_j$ ;
7:    $q_j \leftarrow$  the state from  $q_{j-1}$  upon  $e_j$  at date  $d_j$ ;
8:   switch  $q_j$  do
9:     case  $N$ 
10:      assert(No fault belonging to  $\Sigma_{F_i}$  has happened;);
11:     case  $U$ 
12:      assert(A fault belonging to  $\Sigma_{F_i}$  has probably happened;);
13:     case  $F$ 
14:      assert(A fault belonging to  $\Sigma_{F_i}$  has happened;);
15:     case  $\emptyset$  ▷ The system arrives at a blocked faulty state “F”.
16:      assert(A fault belonging to  $\Sigma_{F_i}$  has happened;);
17:    $j \leftarrow j + 1$ ;

```

Example 37 The LTD for the diagnosable LTPN in Figure 5.7 using its corresponding ASG (Figure 5.11) is given in Figure 5.14. The part in dashed line corresponds to the virtual nodes associated to the quasi-dead node (note that the considered system is non-live). Given two event sequences: $s_1 = abb$ with the event firing relative dates 2.5, 0.5, 2 and $s_2 = ba$ with the event firing relative dates 2, 3, one can conclude that the system is in:

- a normal (blocking) state upon the firing of s_1 ;
- an uncertain state right upon the firing of s_2 ;

5. TIME PN-BASED DIAGNOSIS OF DES

- suppose that no event is observed within 3 t.u upon s_2 , Then one concludes that a faulty state has been reached (indicated by the dashed part added to the diagnoser).

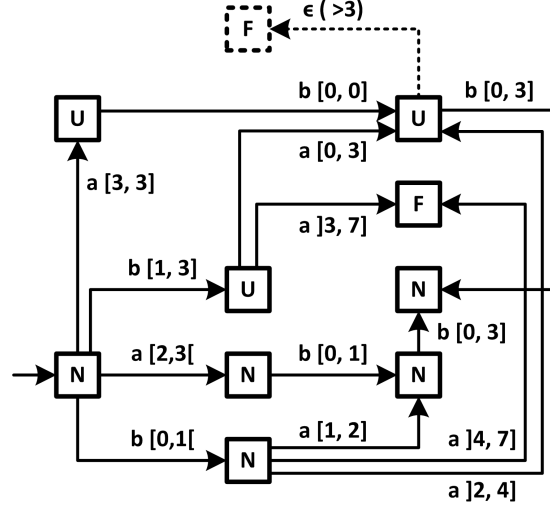


Figure 5.14: Diagnoser of the LTPN in Figure 5.7

5.6 Conclusion

5.6.1 Summary

The first contribution of this chapter is the development of the time interval splitting technique and the ASG structure which makes it possible to take advantage of the results obtained on the diagnosis of DES under an untimed context. The considered timed model which depicts the DES behavior is LTPN. The ASG structure carries necessary information to check diagnosability, and allows for computing the parameter Δ_{min} that characterizes the minimum delay to ensure diagnosability.

Our second contribution is the reduction of the overestimation of the firing domain of an ASC-set graph. Thus, we propose the necessary and sufficient conditions for diagnosability of LTPN. Note, moreover, that the ASG is built on the fly, and that the online diagnoser is derived in a straightforward way.

5.6.2 Perspectives

Example 37 has presented a general diagnoser for LTPN. However, in practice, this structure can be further simplified to a *modified labeled timed diagnoser (MLTD)*, as shown in Figure 5.15. For a given node, if an output arc (e, i) is the only one with an event component e , then we erase the interval component, e.g., the arc labeled with $b[0, 1]$ in Figure 5.14 can be replaced with b in Figure 5.15.

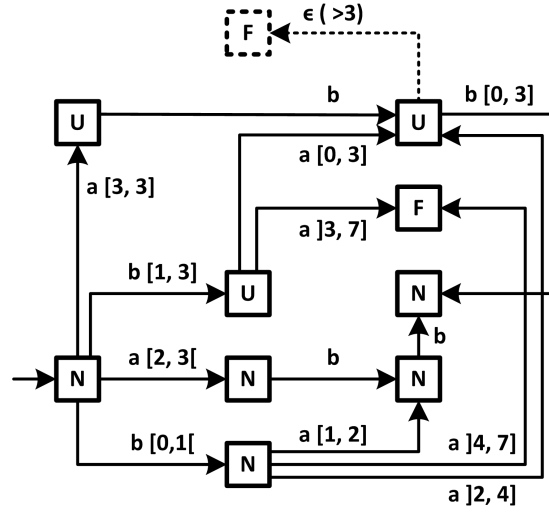


Figure 5.15: The MLTD of the LTPN in Figure 5.7

From the implementation viewpoint, the modified LTD takes as few inputs as possible to deliver diagnosis verdict. As shown in Example 37, one can determine that the system is normal after the observation of sequence $s_1 = abb$, just by taking into account the occurrence date of the first event a without that of the following two b s, since after the occurrence of a at the date 2.5, the following diagnoser state is unique and must be followed by event b regardless of its occurrence date.

The developed ASCs, which are derived from state classes, have been used for describing the state space of LTPNs. However, we note that another notion called *zone graph* [Gardey et al., 2004] has been reported to be more efficient than state classes in presenting TPN state space. Thus, we are interested in using zone

5. TIME PN-BASED DIAGNOSIS OF DES

graph to improve our work.

In addition, for the online diagnosis we intend to take into account the actual occurrence dates of the observable events to make the diagnosis procedure more efficient. Note however that this would need to perform the same computations online in order to propagate this actual occurrence dates as additional constraints.

Chapter 6

Conclusions and Perspectives

6.1 Conclusions

In this thesis, we are interested in fault diagnosis issues of discrete event systems modeled by labeled Petri nets. On-the-fly and incremental techniques are used to cope with state explosion problem. The main contributions of this work can be summarized as follows:

- In the untimed context, we have developed algebraic representations, namely the event-mapping matrix and extended state equation for LPNs, to characterize their static and dynamic features w.r.t failure occurrence. The FM-graph and FM-set tree are built simultaneously and on the fly to analyze diagnosability. Classical diagnosability is analyzed by a series of K -diagnosability problems, in which K is increased progressively. The incremental search technique is used to make full use of the previous search information pertaining to $(K-1)$ -diagnosability, while dealing with K -diagnosability, which allows us to avoid recomputing the state space from scratch.
- In the timed context, we deal with diagnosis issues using LTPN models. First, (temporal) determinization for LTPNs is considered on the basis of a technique that we have developed called Time Interval Splitting (TIS). Necessary and sufficient conditions for diagnosability are then given based on on-the-fly building and analysis of the ASG. Finally, online diagnosis can be performed on the basis of some structures called LTD or MLTD, derived

6. CONCLUSIONS AND PERSPECTIVES

from the ASG. Our study shows that for diagnosable LTPNs, the corresponding ASGs have some semantic equivalence with ERA (event-recording automata)

- The study we have carried out shows the advantages of using an on-the-fly and incremental technique in tackling diagnosis problems: 1) on-the-fly techniques have been shown to be an efficient means to tackle the state explosion phenomenon in some cases. In particular, for diagnosability analysis we have proved that generating the whole state space is not always necessary. 2) incremental technique avoids recomputing state space from scratch when dealing with K (resp. Δ)-diagnosability with progressively increasing K (resp. Δ). These computing techniques do not reduce complexity in terms of time and memory in the worst case. However, in general, only a part of the state space is investigated. Even when the system is diagnosable, the investigation of the whole state space is not necessarily required. Moreover, the theoretically worst case seems to be rather rare in practice and the simulation results obtained on some benchmarks show the efficiency of such techniques.

6.2 Perspectives

Our work on diagnosis of DESs gives rise to some interesting perspectives in the short and medium terms:

- While analyzing the diagnosability of DESs based on incremental search of K (resp. Δ)-diagnosability for LPNs (resp. LTPNs), the value K (resp. Δ) continues to increase until a diagnosability verdict can be emitted or when the necessary (but not necessarily the whole) state space is investigated. For unbounded models, the approach may work as well, depending on the structure of the model and the search strategy. However, it generally risks increasing K (resp. Δ) to infinity in such a way that no verdict would be eventually given. Therefore, it is worth looking for an optimal threshold value K_{opt} (resp. Δ_{opt}) such that the diagnosability verdict can be given out immediately upon K_{opt} (resp. Δ_{opt})-diagnosability is investigated. Besides,

this would be important also when dealing with bounded models. This would require investigating some structural features on the PN model. We intend to investigate this issue in the near future.

- The LPN-EC introduced in Section 4.2.4 is useful for recording event occurrences, whereas it is not necessarily bounded even if the original net is bounded. We intend to modify the LPN-EC structure in such a way as to ensure boundedness.
- Our approaches use depth-first search to investigate the state space (nodes in the developed tree-like structures) branch by branch. Moreover, no rules are defined to select the branch to be built/investigated first, i.e., the order of branch exploring is arbitrary. The strategy could be improved to direct the search in such a way as to increase the chances of quickly obtaining a diagnosability verdict. A similar idea can be found in [Huang, 2013].
- The algebraic representation of LPNs can simplify the description of system behavior, particularly in the event- and/or state(marking)-based analysis for LPNs, e.g., [Basile et al., 2012; Ru and Hadjicostis, 2009]. The drawback of this technique is that event-markings record only occurrence number but no ordering of events. This could be improved by integrating the ordering information of events such that the diagnosability of LPNs could be directly investigated using mathematical representations and their corresponding tools.
- We are interested in introducing zone graph representation [Gardey et al., 2004] into our analysis for LTPNs, since test results show that the zone graph approach is more efficient than the traditional state class graph [Berthomieu and Menasche, 1983] when dealing with bounded TPNs. This could improve our analysis process while investigating diagnosability in the timed context.
- For online diagnosis, we intend to take the actual occurrence dates of the observable events into account to make the online diagnosis procedure more efficient. Note, however, that this would require performing the same com-

6. CONCLUSIONS AND PERSPECTIVES

putations online in order to propagate the actual occurrence dates as additional constraints.

- As the on-the-fly approach shows its efficiency in terms of time and memory, we intend to apply this technique to the verifier approach, which has been proved to be of low complexity.

Appendix A: Literature on DES-Based Diagnosis

Table 1: Literature on DES-based diagnosis

Reference	Automata TA	general PN LPN	TPN	Diagnosability K/ Δ -diagnosability	Online diagnosis	Unfolding ILP	Decentralized/Distributed Other techniques
[Basile et al., 2012]		•	•		•		•
[Benveniste et al., 2003]		•				•	
[Boel and Jiroveanu, 2013]					•		
[Cabasino et al., 2010]		•			•		
[Cabasino et al., 2012a]			•	•	•	•	
[Cabasino et al., 2013a]			•	•			•
[Contant et al., 2004]	•			•			
[Contant et al., 2006]	•			•			•
[Debouk et al., 2000]	•			•			•
[Dotoli et al., 2009]		•			•	•	
[Genc and Lafortune, 2007]			•		•		•
[Ghazel et al., 2009]			•		•		
[Grabiec et al., 2010]			•		•	•	

. APPENDIX A: LITERATURE ON DES-BASED DIAGNOSIS

Table 1 continued											
Reference	Automata TA general PN LPN TPN Diagnosability K/ Δ -diagnosability Online diagnosis Unfolding ILP Decentralized/Distributed Other techniques										
[Haar, 2009]				•		•			•		
[Hashtrudi Zad, 2003]	•					•					
[Hashtrudi Zad, 2005]	•							•			
[Holloway and Chand, 1994]		•						•			
[Jiang et al., 2001]	•					•					
[Jiang et al., 2003a]							•				
[Jiang and Kumar, 2004]											
[Jiang et al., 2006]		•				•					
[Jiroveanu and Boel, 2005]			•					•		•	
[Jiroveanu and Boel, 2010]			•			•					
[Lafortune et al., 2005]	•					•				•	
[Lai et al., 2008]			•					•			
[Lefebvre and Delherm, 2007]			•					•			
[Lin, 1994]								•			
[Madalinski et al., 2010]				•		•			•		
[Pandalai and Holloway, 2000]								•		•	
[Paoli and Lafortune, 2005]	•					•					
[Pencolé, 2009]					•	•		•			
[Provan, 2002]						•				•	
[Qiu et al., 2006]	•					•				•	
[Qiu et al., 2009]	•					•					
[Ramirez-Trevino, 2007]								•		•	
[Ramirez-Trevino et al., 2012]						•				•	
[Ru and Hadjicostis, 2009]				•				•			

Table 1 continued	
Reference	Automata TA general PN LPN TPN Diagnosability K/ Δ -diagnosability Online diagnosis Unfolding ILP Decentralized/Distributed Other techniques
[Sampath et al., 1995]	•
[Sampath et al., 1996]	•
[Sampath et al., 1998]	•
[Simeu-Abazi et al., 2010]	•
[Sztipanovits and Misra, 1996]	
[Tripakis, 2002]	•
[Ushio et al., 1998]	•
[Wang and Girard, 2011]	•
[Wen et al., 2005]	•
[Xu et al., 2010]	•
[Xue and Zheng, 2004]	•
[Yoo and Lafortune, 2002a]	•
[Yoo and Garcia, 2004]	•

. APPENDIX A: LITERATURE ON DES-BASED DIAGNOSIS

Appendix 2: Development of the Level Crossing Benchmark

An LC, is an intersection where a railway line (or multiple railway lines) crosses a road or path at the same level, as opposed to the railway line crossing over or under using a bridge or a tunnel.

An Overview on the Case Study

In France, there are more than 18,000 LCs. Every day they are traversed by an average of 16 million vehicles and nearly 450,000 closing cycles take place for the passage of trains. LCs are identified as critical safety points in both road and railway infrastructures [Ghazel, 2009]. On average, 400 people are killed every year in the European Union [Ćirović and Pamučar, 2013]. In France, 100 collisions happened and 33 people were killed in 2012 [RFF, 2013]. Therefore, safety of LCs always attracts great attention in railway operation and also in research area [Ghazel and El Koursi, 2007; Khoudour and Ghazel, 2009; Mekki et al., 2012].

In this section, we apply our diagnosis techniques to an LC system. We consider an LC with bidirectional multi-line tracks (or unidirectional single-line track for the simple case) and a bidirectional road. Generally, an LC plant is composed of train sensors set on the railway infrastructure, local control system, sound alarm, road lights and barriers, as shown in Figure 1 [Ghazel and El-Koursi, 2014]. The LC global dynamics can be depicted while considering three subsystems, namely the railway traffic, the LC controller and the barriers, which will be introduced in Section 6.2.

. APPENDIX 2: DEVELOPMENT OF THE LEVEL CROSSING BENCHMARK

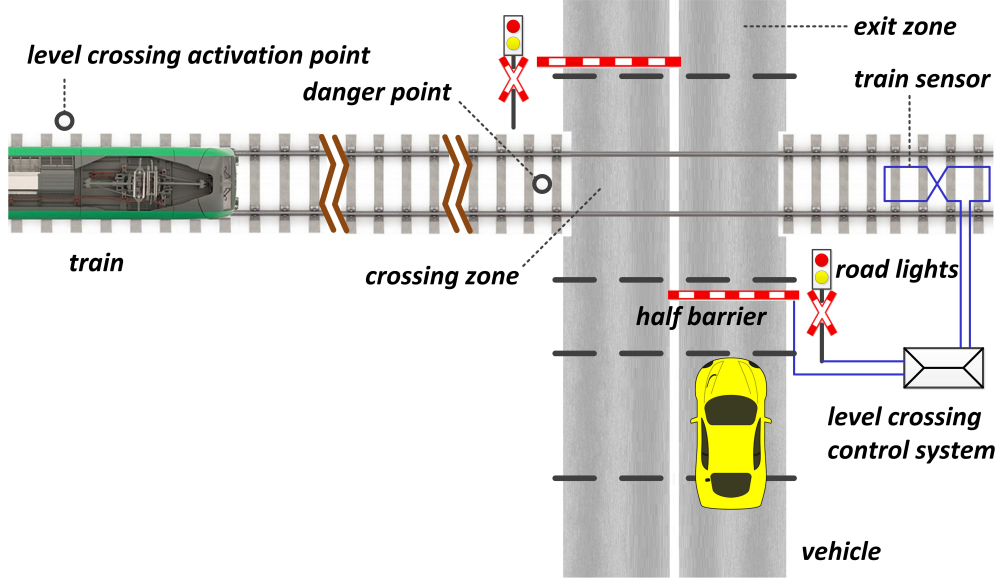


Figure 1: The construction of a single-line track LC system

The logic of a single-line track LC is as follows: when a train approaching the LC is detected by the sensors, the barriers are lowered and the road lights show red. The LC is reopened to road traffic as soon as the train is detected (also by train sensors) out of the crossing zone. As for a multi-track LC, the control on barriers depends on the railway traffic on each line:

- The LC is closed when a train approaching the LC from any line is detected by the train sensors;
- The LC is reopened to road traffic only if no train is still in the crossing zone.

The LC dynamics will be depicted by means of PN models in the next section.

Modeling of the LC Subsystems

This section introduces the modeling of the LC subsystems, namely the railway traffic, the LC controller and the barriers. The n -line LC benchmark will be built based on the single-line LC model of [Leveson and Stolzy, 1985] with some modifications. We will give their corresponding LPN models and operating principles.

Note that, as a first step, only the normal behavior will be modeled; some failures will be introduced afterward.

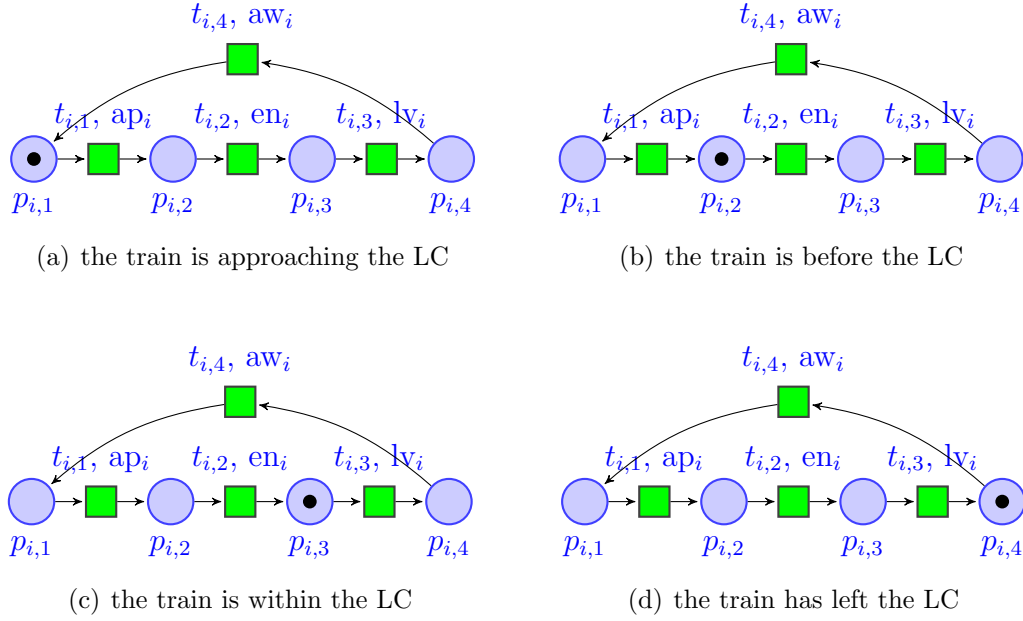


Figure 2: The LPN model for a train passing an LC

Railway Traffic

Railway traffic can be modeled as an LPN composed of 4 places and 4 transitions as shown in Figure 2, where:

- Marked place $p_{i,1}$ (here the subscript i denotes the track index) denotes that a train is approaching the LC, as shown in Figure 2(a);
- Marked place $p_{i,2}$ denotes that the train has come into the section before the LC, which can be detected by sensor $(t_{i,1}, ap_i)$ (here “ap” denotes “approaching”), as shown in Figure 2(b);
- Marked place $p_{i,3}$ denotes that the train has entered the LC, which can be detected by sensor $(t_{i,2}, en_i)$ (here “en” denotes “entering”), as shown in Figure 2(c);

. APPENDIX 2: DEVELOPMENT OF THE LEVEL CROSSING BENCHMARK

- Marked place $p_{i,4}$ denotes that the train has left the LC, which can be detected by sensor $(t_{i,3}, lv_i)$ (here “lv” denotes “leaving”), as shown in Figure 2(d). The zone delimited by transition $t_{i,1}$ and $t_{i,3}$ will be called the crossing zone;
- Finally, place $p_{i,4}$ is linked with $p_{i,1}$ through transition $(t_{i,4}, aw_i)$ (here “aw” denotes that the train is “away” from the LC), which implies that the next train can approach the LC (when $p_{i,1}$ is marked) only if the previous train has left the LC crossing zone (when $p_{i,4}$ is marked). In other words, there is no overlapping between successive train passages.

LC Controller

The LC controller is equipment to collect trains position information from the sensors along the track (in the railway traffic subsystem) and send controlling commands to the barriers and the road lights. The road lights will be omitted in the model as their status can be directly deduced from that of the barriers. It is a processing subsystem between railway traffic and the protection subsystem. The LPN model pertaining to the LC controller is shown in Figure 3 and the operating principles are explained below:

- When a train enters the LC crossing zone, an alert signal is sent from sensor $t_{i,1}$ to the LC controller (place p_1 will be marked). Then (t_1, cr) (here “cr” denotes “closing request”) is fired and a token is added into place p_5 , which means that the condition for closing barriers is satisfied. A token is also added to place p_3 upon t_1 firing, to store the information about the train arrival.
- When a train has left the LC crossing zone, its position is detected by sensor $t_{i,3}$ and this information is sent to the LC controller (place p_4 will be marked). Then (t_2, or) (here “or” denotes “open request”) can be fired and a token is added into place p_6 , which means that the condition for reopening barriers is satisfied.

The LC controller holds, among others, a component called *interlock* [Leveson and Stolzy, 1985]. An interlock can be a hardware or a software mechanism for

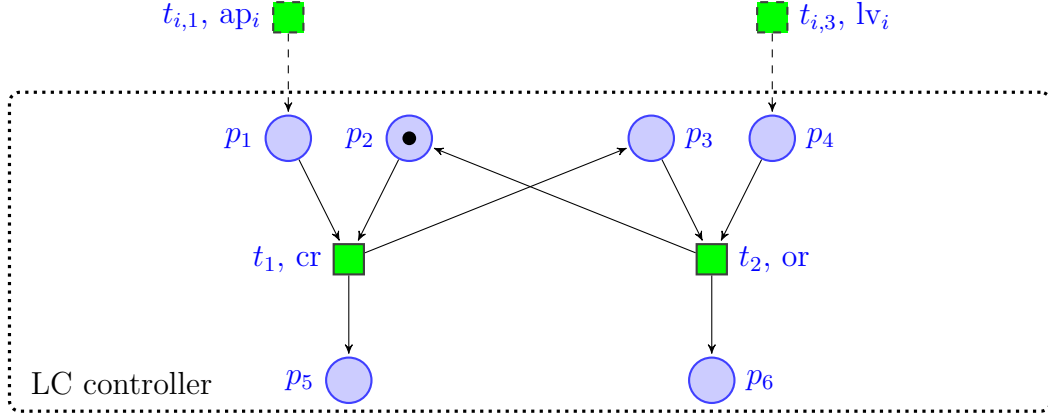


Figure 3: The LPN model for LC controller

ensuring correct sequences of events.

The LPN model for an interlock is shown in Figure 4. In order to make sure that t_1 has to fire before t_2 , a new place p_5 is added as an output place of t_1 and as an input place of t_2 , as shown in Figure 4(b). In other words, the introduction of the interlock (place p_5 and its input/output arcs) ensures that the firing of t_2 is conditioned by the firing of t_1 .

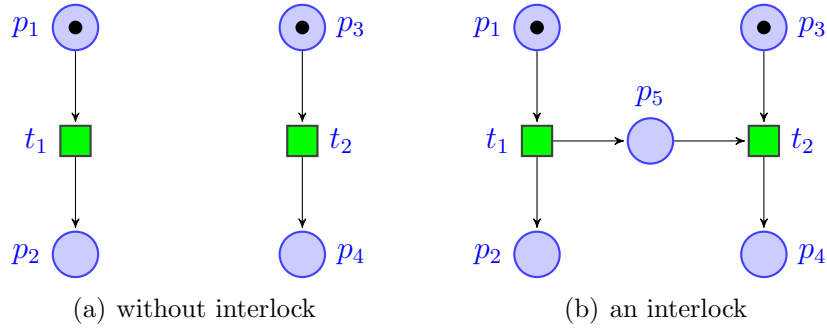


Figure 4: The PN model for an interlock

In a given system, multiple interlocks may exist for ensuring the order of event in some sequences. For example, two interlocks in the LC controller module exist, as shown in Figure 3: the one is formed by $t_1 \rightarrow p_3 \rightarrow t_2$ ensuring the firing priority of t_1 over t_2 ; and the other by $t_2 \rightarrow p_2 \rightarrow t_1$ ensuring the firing priority of

. APPENDIX 2: DEVELOPMENT OF THE LEVEL CROSSING BENCHMARK

t_2 over t_1 after t_1 has been first fired. Such a double-interlock can make sure that t_1 and t_2 fire alternatively. In practice, this means that the LC may be closed only if it was open and reopened only if it was closed.

Barriers Subsystem

The barriers are a subsystem passively responding to the commands from the LC controller. The barrier state switches between “up” (place p_7 is marked) and “down” (place p_8 is marked), i.e., the intermediary positions are ignored. The barriers can be set to “down” (resp. “up”) to prevent (resp. permit) the vehicles from crossing only if the closing (resp. reopening) condition is satisfied. Here p_7 and p_8 are mutually exclusive, i.e., they cannot be marked at the same time, since a barrier can be only up or down. The LPN model for the barrier system is given in Figure 5, where the labels of t_7 and t_8 transitions denote “lower” and “raise” respectively.

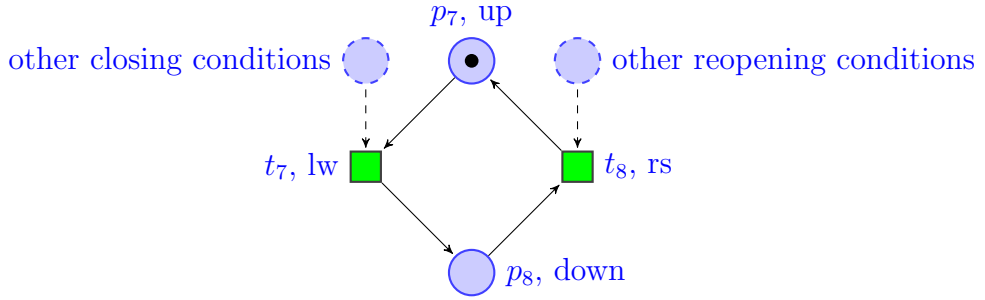


Figure 5: The LPN model for a barrier system

Single-Line LC Model

After having set up the models for the three LC subsystems, let us now establish the global single-line LC model depicted in Figure 6.

The railway traffic subsystem “communicates” with the LC controller through the train sensors which send train position information. The LC controller sends close or open command to switch the “up” and “down” states of the barriers. Place p_9 , together with transitions t_4 and $t_{i,2}$, forms an interlock ensuring that normally

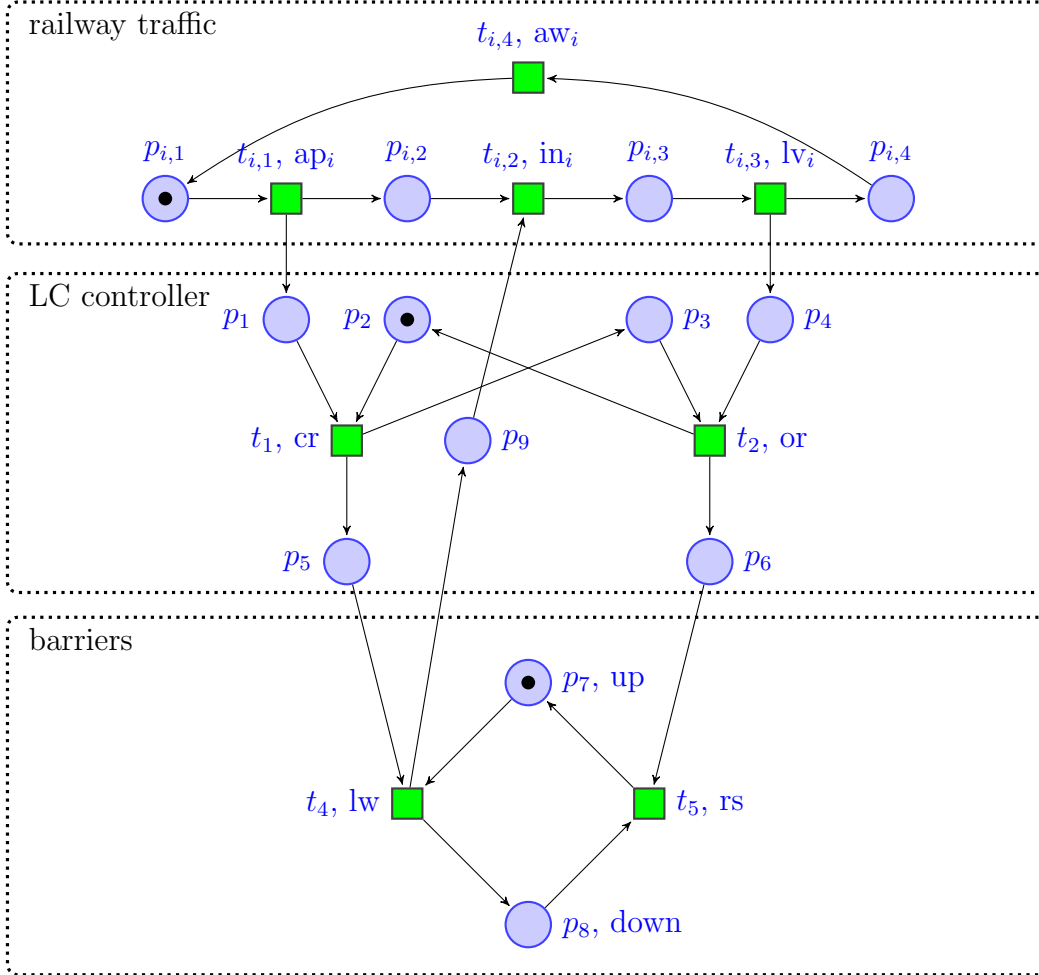


Figure 6: A single-line LC

the barriers must be well lowered (transition t_4 has been fired) before the train enters the LC (transition $t_{i,2}$ is fired).

In the LC, there are two classes of faults which are denoted by red colored transitions in Figure 7: the first one is modeled by transition $(t_{i,5}, ig)$ (here “ig” denotes “ignore”) indicating that the train may enter the LC crossing zone before the barriers are ensured to be lowered; the other modeled by transition (t_6, bf) (here “bf” denotes “barrier fault”) indicating a barrier failure that results in a premature barrier raising. Each of these two faults can induce train-car collision.

Note that compared with Figure 3, there are two more arcs into and out of

. APPENDIX 2: DEVELOPMENT OF THE LEVEL CROSSING BENCHMARK

place p_9 : the arc from $t_{i,2}$ to p_9 ensures that p_9 is remarked after the firing of $t_{i,2}$; the other arc from p_9 to $t_{i,3}$ takes p_9 as one of the conditions for firing $t_{i,3}$. Both of the two arcs ensure the boundedness of the LPN model. More precisely, the LPN here is 1-bounded (or n -bounded for the n -line LC model afterward).

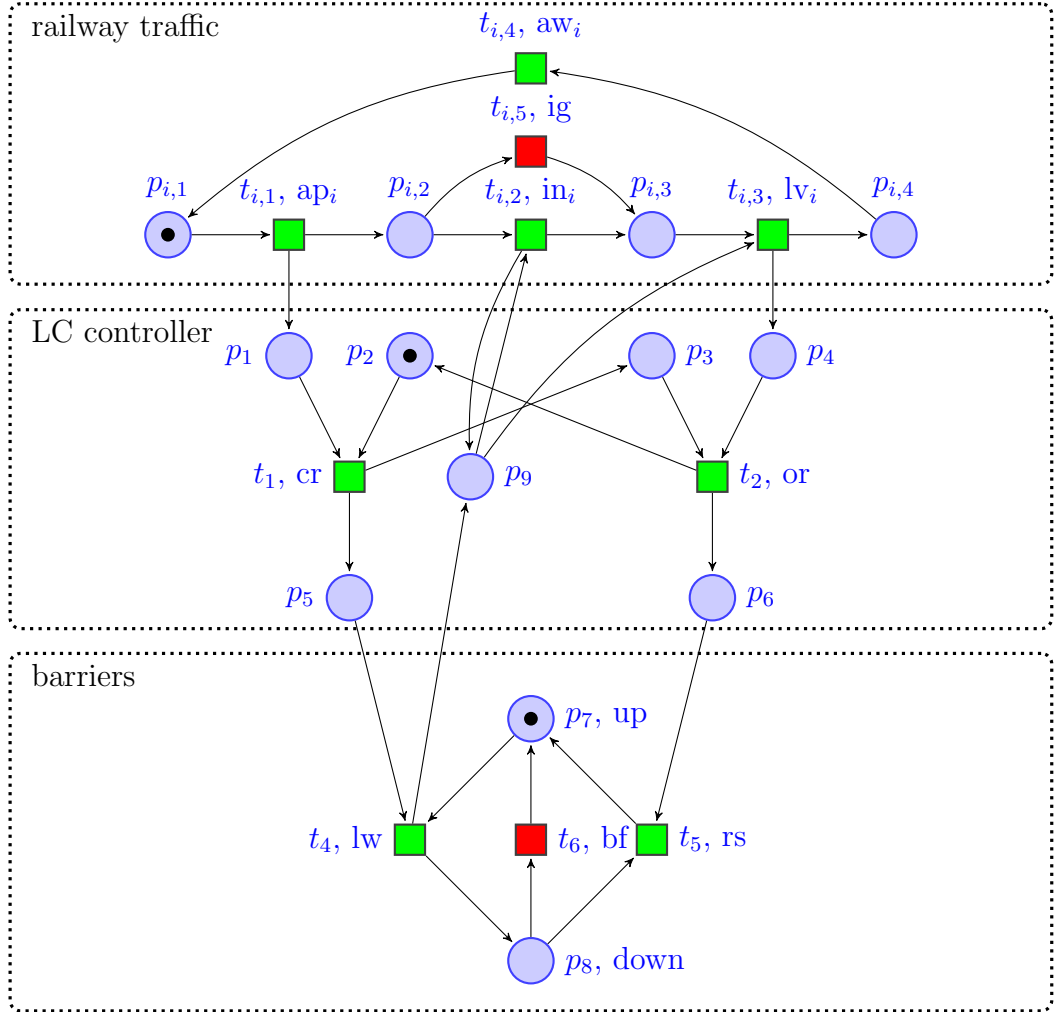


Figure 7: A single-line LC with two classes of faults

In the following section, we will introduce a more general LPN model for the LC system, while taking into account n railway lines.

n -line LC Model

Figure 7 describes a global LPN model for a single-line LC with a unidirectional track. Based on this model, a more general model is given in Figure 8 – involving n railway tracks, which can be obtained from the single-line LC model while fulfilling the following controlling rules under a nominal situation:

- The LC must be closed if any approaching train is detected in any line;
- The LC can be reopened if there is no train in the “within” or “before” sections in any line.

In other terms, the above rules eliminate all the possibilities that the collision between railway and road traffic may take place.

Compared with the single-line LC (cf. Figure 7), there are several changes when generating the n -line LC model:

- Transition t_3 is newly added. In the n -line LC model, t_3 can be fired if both places p_5 and p_8 are marked. This means that if there is an LC closing request from one of the n lines (place p_5 is marked), whereas the barriers are already in the low position (place p_8 is marked) due to a previous closing command from any other line. Then the barriers shall remain down (transition t_3 fires for clearing the request from (marked) place p_5 while keeping the token in p_8).
- Place p_2 is marked with n tokens to ensure that at most n closing requests can be proceeded by the LC controller (place p_1 is n -bounded).
- Place p_6 is also marked with n tokens and denotes the reopening condition. Each firing of t_1 removes a token from p_6 and puts a token into p_3 , meaning that the LC cannot be reopened when at least one closing request is proceeded by the LC controller. The LC can be reopened only if p_6 is n -marked, i.e., all the trains have passed the LC and their closing LC requests have already been treated by the LC controller.
- The two arcs linking t_5 and p_6 have a weight of n . t_5 can be fired only if p_6 holds the reopening condition (n tokens), i.e., no train is still in the crossing

. APPENDIX 2: DEVELOPMENT OF THE LEVEL CROSSING BENCHMARK

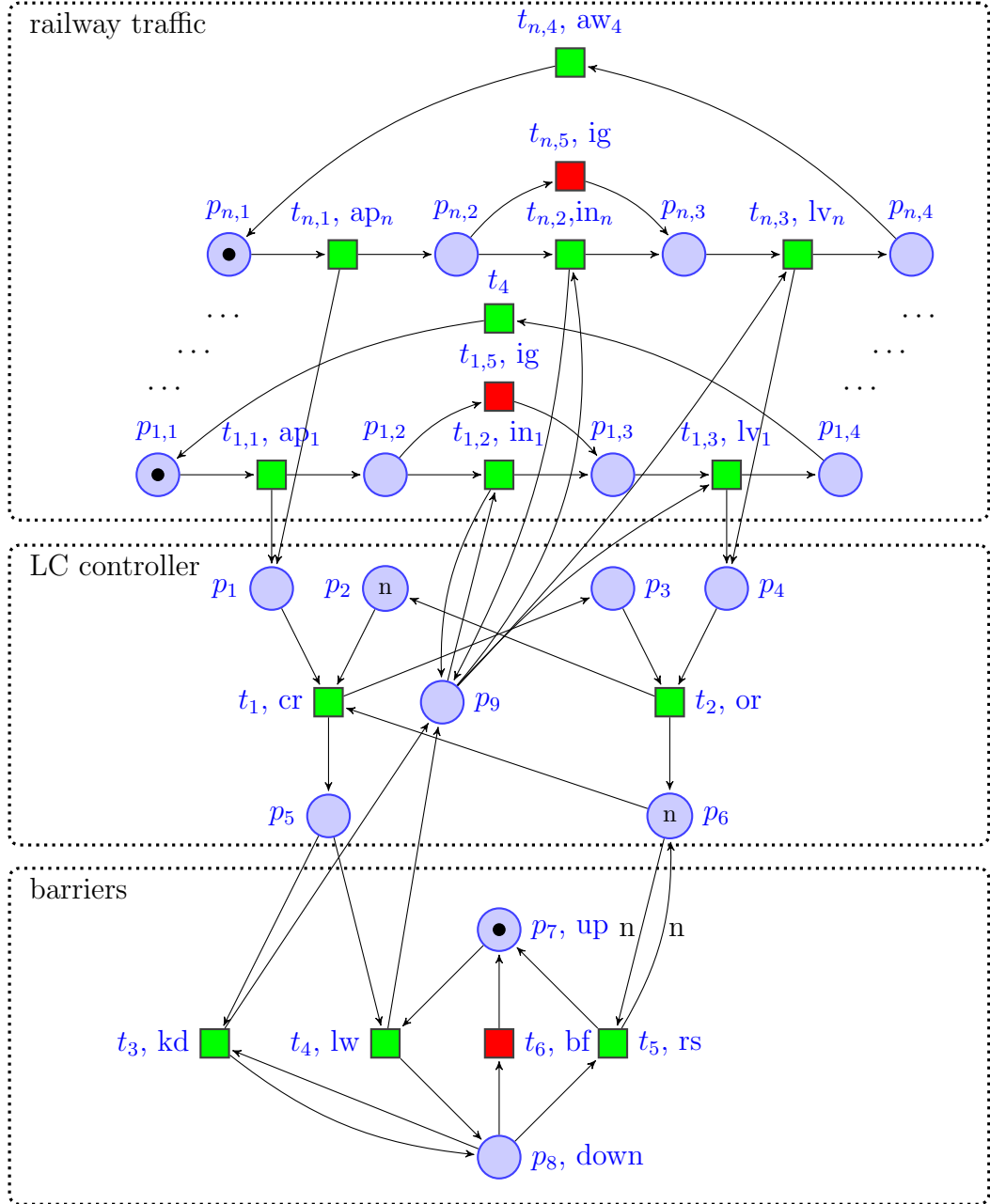


Figure 8: n -line LC benchmark

zone. The firing of t_5 also returns n tokens to p_6 to indicate that at most n closing requests can be treated (as no train is still crossing on any of n tracks).

-
- The arcs linking $t_{i,2}$ to p_9 and p_9 to $t_{i,3}$ ensure that, whether the train passes the LC normally ($t_{i,2}$ is fired) or upon a fault “ig” ($t_{i,5}$ is fired), the token in p_9 will be removed when the train leaves the LC. This ensures the boundedness of the LPN model, since, without these two arcs, the LPN will be unbounded due to the unbounded place p_9 .

In order to obtain sufficiently large LC models for analysis, one can add as many “railway traffic” blocks as necessary and connect them with the “LC controller” and “barriers” blocks in the same way.

In this global model, all the transitions are observable, but the faulty transitions, i.e., $T_o = T \setminus T_u$ and $T_u = T_f = \{t_6\} \cup (\cup_i \{t_{i,5}\})$

The n -line LPN model can be rather big when n takes great values. The state space of the corresponding LPN models for the various values of n can be calculated by the TINA tool [Berthomieu et al., 2004], as shown in Table 2, where:

- n denotes the number of tracks in the n -line LC;
- $|P|$ denotes the number of places in the LPN;
- $|T|$ denotes the number of transitions in the LPN;
- $|A|$ denotes the number of arcs in the reachability graph, i.e., the number of automaton arcs in the diagnoser approach [Sampath et al., 1995].
- $|R|$ denotes the number of nodes in the reachability graph, i.e., the number of automaton states when analyzing diagnosability with the diagnoser approach;
- \mathcal{T}_T denotes the time used for computing the reachability graph (here the value of $|R|$ and $|A|$) of the PN by means of TINA on an Inter Mac (CPU: 2.8 GHz, RAM: 16 GB).

Recall here that not the whole state space will be generated while using our on-the-fly technique. However, the reachability graphs are generated in order to transform them into the input file (language equivalent automata) for UMDDES.

As shown in Table 2, the size of the reachability graph grows very quickly as n increases, since places p_2 and p_6 can hold as many as n tokens, due to which so many markings exist.

. APPENDIX 2: DEVELOPMENT OF THE LEVEL CROSSING BENCHMARK

Table 2: Some figures about the state space of the various LC models

n	$ P $	$ T $	$ A $	$ R $	\mathcal{T}_T
1	13	11	28	24	<1s
2	17	16	540	216	<1s
3	21	21	6,256	1,632	<1s
4	25	26	56,704	11,008	<1s
5	29	31	442,880	68,608	2s
6	33	36	3,126,272	403,456	11s
7	37	41	20,500,480	2,269,184	140s
8	41	46	127,074,304	12,320,768	29m
9	45	51	o.m.	o.m.	o.m.

Note: o.m. = out of memory

References

- Aghasaryan, A. and Fabre, E. (1998). Fault detection and diagnosis in distributed systems: an approach by partially stochastic Petri nets. *Discrete Event Dynamic Systems*, 8(2):203–231. [38](#)
- Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843. [109](#)
- Alur, R. and Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235. [23](#), [24](#)
- Alur, R., Fix, L., and Henzinger, T. (1994). A determinizable class of timed automata. *Computer Aided Verification*, 126:183–235. [139](#), [140](#)
- Alur, R., Fix, L., and Henzinger, T. A. (1999). Event-clock automata: a determinizable class of timed automata. *Theoretical Computer Science*, 11(97):253–273. [25](#), [140](#)
- Baier, C. and Katoen, J. (2008). *Principles of model checking*. MIT Press. [47](#)
- Basile, F., Chiacchio, P., and De Tommasi, G. (2010). Diagnosability of labeled Petri nets via integer linear programming. *Discrete Event Systems*, 10(1):71–77. [6](#), [38](#), [41](#), [42](#), [46](#)
- Basile, F., Chiacchio, P., and De Tommasi, G. (2012). On K-diagnosability of Petri nets via integer linear programming. *Automatica*, 48(9):2047–2058. [41](#), [42](#), [46](#), [52](#), [56](#), [93](#), [95](#), [149](#), [151](#)

REFERENCES

- Belard, N., Pencolé, Y., and Combacau, M. (2011). A theory of meta-diagnosis: reasoning about diagnostic systems. *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two*, pages 731–737. [7](#)
- Benveniste, A., Fabre, E., Haar, S., and Jard, C. (2003). Diagnosis of asynchronous discrete-event systems: a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727. [45](#), [46](#), [151](#)
- Bérard, B., Cassez, F., and Haddad, S. (2005). Comparison of the expressiveness of timed automata and time Petri nets. In *Formal Modeling and Analysis of Timed Systems*, pages 211–225. Springer. [23](#), [31](#)
- Berthomieu, B. and Diaz, M. (1991). Modeling and verification of time dependent systems using time Petri nets. *IEEE Transactions on Software Engineering*, 17(3):259–273. [29](#)
- Berthomieu, B. and Menasche, M. (1983). An enumerative approach for analyzing time Petri nets. *Proceedings IFIP*, pages 41–46. [29](#), [107](#), [123](#), [149](#)
- Berthomieu, B., Ribet, P.-O., and Vernadat, F. (2004). The tool TINA - Construction of abstract state spaces for Petri nets and time Petri nets. *International Journal of Production Research*, 42(14):2741–2756. [90](#), [123](#), [165](#)
- Boel, R. K. and Jiroveanu, G. (2013). The on-line diagnosis of time Petri nets. In *Control of Discrete-Event Systems*, chapter 17, pages 343–364. Springer. [151](#)
- Bouyer, P., Chevalier, F., and D’Souza, D. (2005). Fault diagnosis using timed automata. In *Foundations of Software Science and Computational Structures*, pages 219–233. Springer. [38](#)
- Bouyer, P., Haddad, S., and Reynier, P.-A. (2006). Extended timed automata and time Petri nets. In *International Conference on Application of Concurrency to System Design*, pages 91–100. [31](#)
- Brandin, B. and Wonham, W. (1994). Supervisory control of timed discrete-event systems. *IEEE Transactions on Automatic Control*, 39(2):329–342. [38](#)

REFERENCES

- Cabasino, M. P., Contini, L., Giua, A., Seatzu, C., and Solinas, A. (2011). A software platform for the integration of discrete event systems tools. In *IEEE International Conference on Automation Science and Engineering*, pages 45–51. IEEE. [49](#)
- Cabasino, M. P., Giua, A., Lafortune, S., and Seatzu, C. (2012a). A new approach for diagnosability analysis of Petri nets using verifier nets. *IEEE Transactions on Automatic Control*, 57(12):3104–3117. [6](#), [7](#), [40](#), [41](#), [42](#), [45](#), [46](#), [52](#), [65](#), [93](#), [95](#), [151](#)
- Cabasino, M. P., Giua, A., Marcias, L., and Seatzu, C. (2012b). A comparison among tools for the diagnosability of discrete event systems. In *IEEE International Conference on Automation Science and Engineering*, pages 218–223. IEEE. [49](#)
- Cabasino, M. P., Giua, A., Paoli, A., and Seatzu, C. (2013a). Decentralized diagnosis of discrete-event systems using labeled Petri nets. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pages 1–9. [45](#), [151](#)
- Cabasino, M. P., Giua, A., and Seatzu, C. (2010). Fault detection for discrete event systems using Petri nets with unobservable transitions. *Automatica*, 46(9):1531–1539. [38](#), [95](#), [151](#)
- Cabasino, M. P., Lafortune, S., and Seatzu, C. (2013b). Optimal sensor selection for ensuring diagnosability in labeled Petri nets. *Automatica*, 49(8):2373–2383. [7](#)
- Carvalho, L., Moreira, M., Basilio, J. a. C., and Lafortune, S. (2012). Robust diagnosis of discrete-event systems against permanent loss of observations. *Automatica*, 49(1):223–231. [7](#)
- Cassandras, C. G. and Lafortune, S. (2007). *Introduction to discrete event systems*. Springer, 2 edition. [4](#), [11](#), [12](#), [14](#), [17](#)
- Cassez, F. (2010). Dynamic observers for fault diagnosis of timed systems. *49th IEEE Conference on Decision and Control (CDC)*, pages 4359–4364. [38](#)

REFERENCES

- Cassez, F. (2012). The complexity of codiagnosability for discrete event and timed systems. *IEEE Transactions on Automatic Control*, 57(7):1752 – 1764. [45](#), [138](#)
- Cassez, F. and Tripakis, S. (2008). Fault diagnosis with static and dynamic observers. *Fundamenta Informaticae*, 88(4):497–540. [7](#)
- Cimatti, A., Pecheur, C., and Cavada, R. (2003). Formal verification of diagnosability via symbolic model checking. In *International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 363–369. [47](#)
- Ćirović, G. and Pamučar, D. (2013). Decision support model for prioritizing railway level crossings for safety improvements: Application of the adaptive neuro-fuzzy system. *Expert Systems with Applications*, 40(6):2208–2223. [155](#)
- Clarke, E., Grumberg, O., and Long, D. (1994). Model checking and abstraction. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, pages 343–354. [47](#)
- Contant, O., Lafortune, S., and Teneketzis, D. (2004). Diagnosis of intermittent faults. *Discrete Event Dynamic Systems*, 14(2):171–202. [3](#), [40](#), [42](#), [151](#)
- Contant, O., Lafortune, S., and Teneketzis, D. (2006). Diagnosability of discrete event systems with modular structure. *Discrete Event Dynamic Systems*, 16(1):9–37. [45](#), [151](#)
- D’Angelo, S., Metra, C., and Sechi, G. (1999). Transient and permanent fault diagnosis for FPGA-based TMR systems. *International Symposium on Defect and Fault Tolerance in VLSI Systems*, pages 330 – 338. [3](#)
- Debouk, R., Lafortune, S., and Teneketzis, D. (2000). Coordinated decentralized protocols for failure diagnosis of discrete event systems. *Discrete Event Dynamic Systems*, 10(1-2):33–86. [45](#), [151](#)
- Debouk, R., Lafortune, S., and Teneketzis, D. (2002). On an optimization problem in sensor selection. *Discrete Event Dynamic Systems*. [7](#)
- Diaz, M. (2001). *Les réseaux de Petri - modèles fondamentaux*. Hermès, Paris. [24](#), [28](#), [30](#), [31](#), [32](#), [118](#), [120](#)

REFERENCES

- Dotoli, M., Fanti, M., Mangini, A., and Ukovich, W. (2009). On-line fault detection in discrete event systems by Petri nets and integer linear programming. *Automatica*, 45(11):1–23. [151](#)
- Dugan, J. and Trivedi, K. (1989). Coverage modeling for dependability analysis of fault-tolerant systems. *IEEE Transactions on Computers*, 38(6):775 – 787. [3](#)
- Gardey, G., Lime, D., Magnin, M., and Roux, O. H. (2005). Romeo: A Tool for analyzing time Petri nets. In *Computer Aided Verification*, pages 418–423. Springer. [123](#)
- Gardey, G., Roux, O., and Roux, O. H. (2004). Using zone graph method for computing the state space of a time Petri net. In *Formal modeling and analysis of timed systems*, pages 246–259. Springer. [29](#), [145](#), [149](#)
- Genc, S. and Lafortune, S. (2007). Distributed diagnosis of place-bordered Petri nets. *IEEE Transactions on Automation Science and Engineering*, 4(2):206–219. [38](#), [45](#), [151](#)
- Ghazel, M. (2009). Using stochastic petri nets for level-crossing collision risk assessment. *IEEE Transactions on Intelligent Transportation Systems*, 10(4):668–677. [155](#)
- Ghazel, M. and El Kourssi, E.-M. (2007). Automatic level crossings: From informal functional requirements’ specifications to the control model design. In *IEEE International Conference on System of Systems Engineering 2007 (SoSE’07)*, pages 1–6. IEEE. [155](#)
- Ghazel, M. and El-Kourssi, E.-M. (2014). Two-Half-Barrier Level Crossings Versus Four-Half-Barrier Level Crossings : A Comparative Risk Analysis Study. *IEEE Transactions on Intelligent Transportation Systems*. [155](#)
- Ghazel, M., Peres, F., Belhaj Alaya, A., and Jemai, A. (2012). A DBMS framework for diagnosability analysis of discrete event systems. In *DSN 2012, 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, France. [48](#)

REFERENCES

- Ghazel, M., Toguyéni, A., and Yim, P. (2009). State observer for DES under partial observation with time Petri nets. *Discrete Event Dynamic Systems*, 19(2):137–165. [24](#), [38](#), [121](#), [151](#)
- Giua, A. (2007). A benchmark for diagnosis. http://www.diee.unica.it/giua/WODES/WODES08/media/benchmark_diagnosis.pdf. [9](#), [89](#), [91](#)
- Giua, A., Seatzu, C., and Corona, D. (2007). Marking estimation of Petri nets with silent transitions. *IEEE Transactions on Automatic Control*, 52(9):1695–1699. [17](#)
- Grabiec, B., Traonouez, L., and Jard, C. (2010). Diagnosis using unfoldings of parametric time Petri nets. In *Formal Modeling and Analysis of Timed Systems*, pages 137–151. Springer Berlin Heidelberg. [46](#), [151](#)
- Grastien, A. (2009). Symbolic testing of diagnosability. In *International Workshop on Principles of Diagnosis*, pages 131–138. [47](#)
- Haar, S. (2009). Qualitative diagnosability of labeled Petri nets revisited. In *Proceedings of the 48th IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 1248–1253. IEEE. [38](#), [46](#), [152](#)
- Hashtrudi Zad, S. (2003). Fault diagnosis in discrete-event systems: Framework and model reduction. *IEEE Transactions on Automatic Control*, 48(7):1199–1212. [152](#)
- Hashtrudi Zad, S. (2005). Fault diagnosis in discrete-event systems: incorporating timing information. *IEEE Transactions on Automatic Control*, 50(7):1010–1015. [152](#)
- Henzinger, T., Manna, Z., and Pnueli, A. (1992). Timed transition systems. *Real-Time: Theory in Practice*, 3096. [24](#)
- Holloway, L. and Chand, S. (1994). Time templates for discrete event fault monitoring in manufacturing systems. In *American Control Conference, 1994*, pages 701–706. [38](#), [152](#)

- Huang, Y. (2013). *An incremental approach for the extraction of firing sequences in timed Petri nets: Application to the reconfiguration of flexible manufacturing systems*. PhD thesis, Ecole Centrale de Lille. [149](#)
- Huang, Z., Bhattacharyya, S., Chandra, V., Jiang, S., and Kumar, R. (2004). Diagnosis of discrete event systems in rules-based model using first-order linear temporal logic. In *American Control Conference*, volume 6, pages 5114–5119. [48](#)
- Jiang, S., Huang, Z., Chandra, V., and Kumar, R. (2001). A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 46(8):1318–1321. [5](#), [7](#), [38](#), [39](#), [44](#), [152](#)
- Jiang, S. and Kumar, R. (2004). Failure diagnosis of discrete-event systems with linear-time temporal logic specifications. *IEEE Transactions on Automatic Control*, 49(6):934–945. [152](#)
- Jiang, S., Kumar, R., and Garcia, H. (2003a). Diagnosis of repeated/intermittent failures in discrete event systems. *IEEE Transactions on Robotics and Automation*, 19(2):310–323. [3](#), [40](#), [152](#)
- Jiang, S., Kumar, R., and Garcia, H. (2003b). Optimal sensor selection for discrete-event systems with partial observation. *IEEE Transactions on Automatic Control*, 48(3):369–381. [7](#)
- Jiang, S., Kumar, R., and Member, S. (2006). Diagnosis of repeated failures for discrete event systems with linear-time temporal-logic specifications. *IEEE Transactions on Automation Science and Engineering*, 3(1):47–59. [38](#), [40](#), [42](#), [152](#)
- Jiroveanu, G. and Boel, R. K. (2005). Distributed diagnosis for Petri nets models with unobservable interactions via common places. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 6305–6310. IEEE. [45](#), [152](#)
- Jiroveanu, G. and Boel, R. K. (2010). The diagnosability of Petri net models using minimal explanations. *IEEE Transactions on Automatic Control*, 55(7):1663–1668. [152](#)

REFERENCES

- Jiroveanu, G., Boel, R. K., and De Schutter, B. (2006). Fault diagnosis for time Petri nets. In *WODES06, 8th International Workshop on Discrete Event Systems*, pages 313–318, Ann Arbor, MI. IEEE. 43
- Khoudour, L. and Ghazel, M. (2009). Towards safer level crossings: existing recommendations, new applicable technologies and a proposed simulation model. *European transport research review*. 155
- Koenig, S., Likhachev, M., Liu, Y., and Furcy, D. (2004). Incremental heuristic search in AI. *AI Magazine*, 25(2):99–112. 55
- Lafortune, S. (2000). UMDES-Lib software library. <http://www.eecs.umich.edu/umdes/toolboxes.html>. 48, 90, 93
- Lafortune, S., Wang, Y., and Yoo, T.-S. (2005). Diagnostic décentralisé des systèmes à événements discrets. *Journal Européen des Systèmes Automatisés*, 39(1-3):95–110. 45, 152
- Lai, S., Nessi, D., Cabasino, M. P., Giua, A., and Seatzu, C. (2008). A comparison between two diagnostic tools based on automata and Petri nets. In *International Workshop on Discrete Event Systems*, pages 144–149. IEEE. 152
- Lefebvre, D. and Delherm, C. (2007). Diagnosis of DES with Petri net models. *IEEE Transactions on Automation Science and Engineering*, 4(1):114–118. 152
- Leveson, N. G. and Stolzy, J. (1985). Analyzing safety and fault tolerance using time petri nets. *Formal Methods and Software Development*, 186. 96, 156, 158
- Lin, F. (1994). Diagnosability of discrete event systems and its applications. *Discrete Event Dynamic Systems*, 4(2):197–212. 2, 3, 4, 5, 11, 37, 152
- Liu, B., Ghazel, M., and Toguyéni, A. (2012). K-diagnosability of labeled Petri nets. In *9ème édition de la conférence MANifestation des JEunes Chercheurs en Sciences et Technologies de l’Information et de la Communication (MajecSTIC 2012)*, volume 2012, Villeneuve d’Ascq. 51
- Liu, B., Ghazel, M., and Toguyéni, A. (2013). Évaluation à la volée de la diagnosticabilité des systèmes à événements discrets temporisés. *Journal Européen des*

REFERENCES

- Systèmes Automatisés, Édition spéciale MSR'13, Modélisation des Systèmes Réactifs*, 47(1-2-3):227–242. [23](#), [34](#), [38](#)
- Liu, B., Ghazel, M., and Toguyéni, A. (2014a). Diagnosis of labeled time Petri nets using time interval splitting. In *The 19th World Congress of the International Federation of Automatic Control (IFAC'14)*. Accepted. [34](#), [112](#)
- Liu, B., Ghazel, M., and Toguyéni, A. (2014b). Toward an efficient approach for diagnosability analysis of DES modeled by labeled Petri nets. In *The 13th European Control Conference (ECC'14)*. [51](#)
- Madalinski, A., Nouioua, F., and Dague, P. (2010). Diagnosability verification with Petri net unfoldings. *International Journal of Knowledge-Based and Intelligent Engineering Systems*, 2:49–55. [46](#), [47](#), [152](#)
- McMillan, K. L. and Probst, D. K. (1995). A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65. [46](#)
- Mekki, A., Ghazel, M., and Toguyeni, A. (2012). Validation of a New Functional Design of Automatic Protection Systems at Level Crossings with Model-Checking Techniques. *Intelligent Transportation Systems, IEEE Transactions on*, 13(2):714–723. [155](#)
- Merlin, P. (1974). *A study of the recoverability of computing systems*. PhD thesis, University of California, Irvine. [23](#), [27](#)
- Morel, G., Valckenaers, P., and Faure, J. M. (2007). Manufacturing plant control challenges and issues. *Control Engineering Practice*. [2](#)
- Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580. [18](#)
- Nelson, V. (1990). Fault-tolerant computing: fundamental concepts. *Computer*, 23(7):19–25. [3](#)
- Pandalai, D. and Holloway, L. (2000). Template languages for fault monitoring of timed discrete event processes. *IEEE Transactions on Automatic Control*, 45(5):868–882. [152](#)

REFERENCES

- Paoli, A. and Lafortune, S. (2005). Safe diagnosability for fault-tolerant supervision of discrete-event systems. *Automatica*, 41(8):1335–1347. [152](#)
- Pencolé, Y. (2009). A chronicle-based diagnosability approach for discrete timed-event systems : Application to web-services. *Journal of Universal Computer Science*, 15(17):3246–3272. [43](#), [45](#), [152](#)
- Peres, F. and Ghazel, M. (2013). A u-calculus framework for the diagnosability of discrete event systems. Technical report, IFSTTAR. [48](#)
- Petri, C. A. (1962). *Kommunikation mit automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Schriften des IIM. [18](#)
- Provan, G. (2002). On the diagnosability of decentralized, timed discrete event systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, pages 405–410. [45](#), [152](#)
- Qiu, W., Kumar, R., and Member, S. (2006). Decentralized failure diagnosis of discrete event systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 36(2):384–395. [45](#), [152](#)
- Qiu, W., Wen, Q., and Kumar, R. (2009). Decentralized diagnosis of event-driven systems for safely reacting to failures. *IEEE Transactions on Automation Science and Engineering*, 6(2):362–366. [152](#)
- Ramchandani, C. (1974). *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, Massachusetts Institute of Technology. [23](#)
- Ramirez-Trevino, A. (2007). Online fault diagnosis of discrete event systems. A Petri net-based approach. *IEEE Transactions on Automation Science and Engineering*, 4(1):31–39. [152](#)
- Ramirez-Trevino, A., Ruiz-Beltran, E., Arámburo-lizárraga, J., and López-mellado, E. (2012). Structural diagnosability of DES and design of reduced Petri net diagnosers. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 42(2):416–429. [152](#)

REFERENCES

- RFF (2013). Les passages à niveau. <http://www.rff.fr/fr/le-reseau/le-reseau-en-projets/la-modernisation-du-reseau/les-passages-a-niveau>. 155
- Ribot, P., Pencolé, Y., and Combacau, M. (2007). Characterization of requirements and costs for the diagnosability of distributed discrete event systems. *5th Workshop on Advanced Control and Diagnosis (ACD'07)*. 6
- Ricker, L., Lafortune, S., and Genc, S. (2006). DESUMA: A tool integrating GIDDES and UMDDES. In *International Workshop on Discrete Event Systems*, pages 392–393. IEEE. 48
- Ru, Y. and Hadjicostis, C. N. (2009). Fault diagnosis in discrete event systems modeled by partially observed Petri nets. *Discrete Event Dynamic Systems*, 19(4):551–575. 38, 63, 149, 152
- Ru, Y. and Hadjicostis, C. N. (2010). Sensor selection for structural observability in discrete event systems modeled by Petri nets. *IEEE Transactions on Automatic Control*, 55(7):1–14. 7
- Rushton, A. (2012). A directed graph container. <http://www.andyrushton.co.uk/programming/stlplus-library-collection>. 78
- Sampath, M., Lafortune, S., and Teneketzis, D. (1998). Active diagnosis of discrete-event systems. *IEEE Transactions on Automatic Control*, 43(7):908–929. 6, 96, 153
- Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and Teneketzis, D. (1995). Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575. 4, 5, 7, 15, 16, 37, 39, 40, 44, 52, 69, 72, 73, 84, 95, 96, 133, 137, 153, 165
- Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and Teneketzis, D. (1996). Failure diagnosis using discrete-event models. *IEEE Transactions on Control Systems Technology*, 4(2):105–124. 44, 153
- Schiller, F., Schröder, J., and Lunze, J. (2001). Diagnosis of transient faults in quantised systems. *Engineering Applications of Artificial Intelligence*, 14(4):519–536. 3

REFERENCES

- Schwoon, S. and Esparza, J. (2005). *A note on on-the-fly verification algorithms*. Springer. [52](#)
- Simeu-Abazi, Z., Di Mascolo, M., and Knotek, M. (2010). Fault diagnosis for discrete event systems: Modelling and verification. *Reliability Engineering and System Safety*, 95(4):369–378. [153](#)
- Soldani, S., Combacau, M., and Subias, A. (2007). Intermittent fault diagnosis: A diagnoser derived from the normal behavior. In *Proceedings of the 18th International Workshop on Principles of Diagnosis*, pages 391–398. [3](#)
- Stamp, G., Ong, Y., Kumar, R., and Zhou, C. (2006). DECADA: Tool for discrete-event control and diagnosis analysis. In *International Workshop on Discrete Event Systems*, pages 396–397. IEEE. [49](#)
- Sztipanovits, J. and Misra, A. (1996). Diagnosis of discrete event systems using ordered binary decision diagrams. In *Seventh International Workshop on Principles of Diagnosis*. [153](#)
- Tripakis, S. (2002). Fault diagnosis for timed automata. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 205–221. Springer. [6](#), [24](#), [33](#), [38](#), [43](#), [95](#), [105](#), [153](#)
- Tripakis, S. (2006). Folk theorems on the determinization and minimization of timed automata. *Information Processing Letters*, 99(6):222–226. [139](#)
- Ushio, T., Onishi, I., and Okuda, K. (1998). Fault detection based on Petri net models with faulty behaviors. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 1, pages 113–118. [5](#), [8](#), [38](#), [39](#), [153](#)
- Wang, W. and Girard, A. (2011). On codiagnosability and coobservability with dynamic observations. *IEEE Transactions on Automatic Control*, 56(7):1551–1566. [153](#)
- Wang, X., Mahulea, C., Júlvez, J., and Silva, M. (2011). On state estimation of timed choice-free Petri nets. In *Proceedings of the 18th IFAC World Congress*, pages 8687–8692. [38](#)

REFERENCES

- Wang, X., Mahulea, C., and Silva, M. (2013). Fault Diagnosis Graph of Time Petri Nets. In *European Control Conference (ECC)*, pages 2459–2464. [38](#)
- Wang, Y., Yoo, T.-S., and Lafortune, S. (2004). New results on decentralized diagnosis of discrete event systems. *Proceedings of 2004 Annual*. [42](#)
- Wen, Y., Li, C., and Jeng, M. (2005). A polynomial algorithm for checking diagnosability of Petri nets. In *IEEE International Conference on Systems, Man and Cybernetics (SMC'05)*, volume 3, pages 2542–2547. IEEE. [38](#), [40](#), [45](#), [56](#), [153](#)
- Wen, Y., Li, C., and Jeng, M. (2006). Diagnosability enhancement of discrete event systems. In *IEEE International Conference on Systems, Man and Cybernetics (SMC'06)*, volume 5, pages 4096–4101. [7](#)
- Xu, S., Jiang, S., and Kumar, R. (2010). Diagnosis of dense-time systems under event and timing masks. *IEEE Transactions on Automation Science and Engineering*, 7(4):870–878. [43](#), [153](#)
- Xue, F. and Zheng, D.-Z. (2004). Diagnosability for discrete event systems based on Petri net language. In *8th Control, Automation, Robotics and Vision Conference (ICARCV 2004)*, volume 3, pages 2111–2116. IEEE. [40](#), [45](#), [153](#)
- Yoo, T.-S. and Garcia, H. (2004). Event diagnosis of discrete-event systems with uniformly and nonuniformly bounded diagnosis delays. In *Proceedings of 2004 American Control Conference*, volume 6, pages 5102–5107. [6](#), [40](#), [42](#), [153](#)
- Yoo, T.-S. and Lafortune, S. (2002a). NP-completeness of sensor selection problems arising in partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, 47(9):1495–1499. [5](#), [153](#)
- Yoo, T.-S. and Lafortune, S. (2002b). Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, 47(9):1491–1495. [7](#), [38](#), [39](#), [44](#), [52](#)
- Zaytoon, J. and Lafortune, S. (2013). Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2):308–320. [45](#)

REFERENCES

Zemouri, R. and Faure, J. M. (2006). Diagnosis of discrete event system by stochastic timed automata. In *IEEE International Conference on Control Applications*, pages 1861–1866. IEEE. [38](#)

REFERENCES

Titre : Une approche efficace pour l'étude de la diagnosticabilité et le diagnostic des SED modélisés par Réseaux de Petri labellisés – contextes atemporel et temporel

Cette thèse s'intéresse à l'étude des problèmes de diagnostic des fautes sur les systèmes à événements discrets en utilisant des modèles réseau de Petri. Des techniques d'exploration incrémentale et à-la-volée sont développées pour combattre le problème de l'explosion de l'espace d'état lors de l'analyse de la diagnosticabilité. Dans le contexte atemporel, la diagnosticabilité des modèles Réseaux de Petri labellisés (RdP-L) est abordée par l'analyse d'une série de problèmes K -diagnosticabilité. Pour ce fait, deux modèles nommés respectivement FM-graph et FM-set tree sont développés à-la-volée. Un diagnostiqueur peut être dérivé à partir du FM-set tree pour le diagnostic en ligne. Dans le contexte temporel, des techniques de fractionnement des intervalles de temps sont élaborées pour développer une représentation de l'espace d'état des Réseaux de Petri labellisés et temporels (RdP-LT) pour laquelle des techniques d'analyse de la diagnosticabilité du contexte atemporel peuvent être utilisées. Sur cette base, les conditions nécessaires et suffisantes pour la diagnosticabilité des RdP-LT ont été déterminées. En pratique, l'analyse de la diagnosticabilité est effectuée sur la base de la construction à-la-volée d'une structure nommée ASG et qui contient notamment des informations relatives à l'occurrence de fautes. D'une manière générale, l'analyse effectuée sur la base des techniques à-la-volée et incrémentales permet de construire et explorer seulement une partie de l'espace d'état. Les résultats des analyses effectuées sur certains benchmarks montrent l'efficacité de ces techniques en termes de temps et de mémoire par rapport aux approches traditionnelles basées sur l'énumération des états.

Mots clés : Diagnostic des fautes, Systèmes à événements discrets, Réseaux de Petri labellisés, Réseaux de Petri labellisés et temporels, Analyse à-la-volée, Approche incrémentale, Fractionnement d'intervalles temporels.

Title: An Efficient Approach for Diagnosability and Diagnosis of DES Based on Labeled Petri Nets – Untimed and Timed Contexts

This PhD thesis deals with fault diagnosis of discrete event systems using Petri net models. Some on-the-fly and incremental techniques are developed to tackle the state explosion problem while analyzing diagnosability. In the untimed context, an algebraic representation for labeled Petri nets (LPNs) is developed for featuring the system behavior. The diagnosability of LPN models is tackled by analyzing a series of K -diagnosability problems. For this, two models called respectively FM-graph and FM-set tree are developed and built on the fly to record the necessary information for diagnosis purpose. Finally, a diagnoser is derived from the FM-set tree for online diagnosis. In the timed context, time interval splitting techniques are developed in order to generate a state representation of labeled time Petri net (LTPN) models, for which techniques from the untimed context can be used. Based on this, necessary and sufficient conditions for the diagnosability of LTPN models are determined. Moreover, we provide the solution for the minimum delay Δ that ensures diagnosability. From a practical point of view, diagnosability analysis is performed on the basis of on-the-fly building of a structure that we call ASG and which holds fault information about the LTPN states. Generally, using on-the-fly analysis and incremental technique makes it possible to build and investigate only a part of the state space. Analysis results obtained on some chosen benchmarks show the efficiency in terms of time and memory compared with the traditional approaches based on state enumeration.

Keywords: Fault diagnosis, Discrete event systems, Labeled Petri nets, Labeled time Petri nets, On-the-fly analysis, Incremental approach, Time interval splitting.