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

Baisi Liu

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	DR Joaquin RODRIGUEZ	IFSTTAR - Villeneuve d'Ascq
Rapporteur	Prof. Jean-Marc FAURE	SUPMECA - LURPA
Rapporteur	Prof. Michel COMBACAU	Université de Toulouse 3 - UPS - LAAS
Examinateur	DR Joaquin RODRIGUEZ	IFSTTAR - Villeneuve d'Ascq
Examinateur	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

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

Laboratoire d'Automatique, Génie Informatique et Signal, École Centrale de Lille

To my parents

ACKNOWLEDGEMENTS

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

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. Héléne Catsiapis for helping me to improve my French.

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

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

Many thanks to my parents and my girl friend for their consistent support.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

INTRODUCTION

This thesis deals with Petri-net-based techniques for fault diagnosis of discrete event systems (DESs). The work was 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 presents an overview of this thesis. Section 1.1 introduces the background of the fault diagnosis problem in the context of DESs. 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 hypotheses will be given in Section 2.3, after the introduction of some basic notations.

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, 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 reconfiguration can be made to prevent the system from dangerous consequences [Lin94]. 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 [Mor+07]. 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 systems.* 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 at 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 thorough examination. This means the diagnosis procedure needs to be performed online and has to be efficient.

Now let us consider 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*", "*indistinguishable*" or "*silent* " event (or state), which is difficult to be distinguished from normal behavior 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'A+99; DT89; Nel90], as shown in Figure 1.1:

1. A *transient* fault, which is often of finite duration shorter than the stable time interval of the affected signal, occurs only once and then disappears. It may be the result of external causes like interference. Generally, transient faults are the most common, and they are also hard to identify, since they may disappear after having produced errors.

2. An *intermittent* fault occurs and disappears repeatedly, making the system vibrate between normal and fault states. It can be caused by an unstable device operation.

3. A *permanent* fault occurs but does not disappear (such that the system remains in a fault state) until repairing measures are undertaken. Typically, a permanent fault is due to subsystem failures, physical damage or design error.

Diagnosis of transient [Sch+01], intermittent [Con+04; Jia+03a; Sol+07] and permanent faults have been studied. In this thesis, we only consider the diagnosis problems of permanent faults and a series of works on diagnosis of permanent faults will be reviewed in Chapter 3.

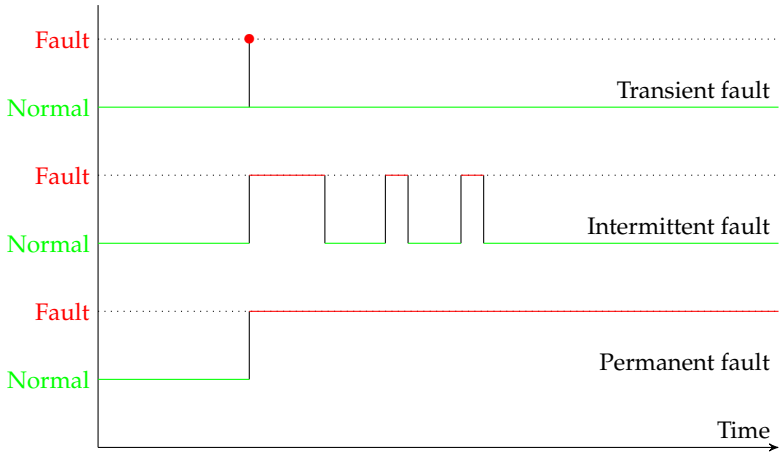


Figure 1.1: Transient fault, intermittent fault and permanent fault

1.1.2 Discrete Event System (DES)

Diagnosis techniques that we deal with in the framework of this thesis are model-based, which means that one has a behavioral model that depicts the system dynamics. Moreover, we assume that diagnosis analysis can be undertaken at an abstraction level such that the system behavior can be abstracted as a discrete-event model [Lin94].

A *DES* is informally a *discrete-state* and *event-driven* system [CL07]. 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 discrete-event model to a certain level of abstraction [Lin94; Sam+95], which can be untimed, timed or stochastic. Note that many systems around us are *Computer-Controlled Systems*, whose state space and event-driven mechanism are built on binary logic. They can be treated as DESs to a certain degree. This is why control and diagnosis of DESs have been extensively studied in recent years.

In order to characterize DESs, different modeling notations have been developed [CL07]: language and automata, PN theory, $(max, +)$ algebra, Markov chains and queuing theory, discrete-event simulation, perturbation analysis, concurrent estimation techniques, etc. 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 the untimed and the timed context, using the LPN and LTPN notation.

1.1.3 Problems

Fault diagnosis has been extensively studied. Most existing methods can be classified as follows: fault-tree-based methods, quantitative, analytic 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 computers control more and more systems, and most of them can be abstracted as discrete-event models to a certain abstraction degree.

DES-based diagnosis is first studied in the framework of automata [Sam+95], which brings the problem of "state explosion". Afterward, some research turns to the PN-based technique [Ush+98], 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* [Jia+01; Lin94; Sam+95; YL02a]. 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 occurrence. This topic is studied as K-diagnosability in the untimed context [Bas+10; Cab+12b; YG04] and as Δ-diagnosability in the timed context [Tri02], 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 that 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 interference 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 whose objective is to observe system behavior and give a verdict of potential faults, active diagnosis is an integrated approach to control and diagnosis [Sam+98]:

a) In the early period, the design of systems and the development of diagnosis 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 diagnosis tool may be difficult, due to some practical limitations. The active diagnosis in the sense of [Sam+98] aims at developing a diagnosable system by early integrating diagnosability analysis since the design phase. In distributed systems, [Rib+07] determines the characteristics and the modifications that could be useful for designers to improve and to guarantee some diagnosability objectives.

b) For existing systems, one can turn certain sensors ON or OFF as necessary, such that the system can be monitored by a dynamic observer whose set of observable events varies according to external commands. Diagnosis based on dynamic observers permits achieving cost savings [CT08], since certain sensors only operate as necessary.

5. *Enhancement of diagnosability* [Wen+06]. Although the active diagnosis is a trend in 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/or placement of sensors. This has been further studied as the problem of sensor optimization [Cab+13b; Deb+02; Jia+03b; RH10].

Besides, there are also other directions on diagnosis study, such as meta-diagnosis [Bel+11], robust diagnosis [Car+12], 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 will be interested in active diagnosis and sensor optimization issues.

1.2 Objectives

As will be discussed in Chapter 3, some conventional approaches [Cab+12b; Jia+01; Sam+95; YL02b] 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 formally proved. Our goal here is to develop a new technique, namely the on-the-fly and incremental technique, which is able to tackle the state explosion problem in the untimed context on the basis of existing proven 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 Chapter 4 and Chapter 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 conventional algebraic representation with the help of *markings*, *incidence matrix* and *state equation*. This formulation, however, is not sufficient for featuring LPNs, as there is no characterization of relations between transitions and events. To cope with the above shortcoming, we propose novel algebraic representation for LPNs, based on some new notions that will be introduced, namely *extended incidence matrix*, *event marking*, *fault 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 (RG) [Ush+98].

   c) On-the-fly analysis of conventional diagnosability based on K-diagnosability:
   By extension, we solve the conventional diagnosability by dealing with a series of K-diagnosability problems, where K increases progressively. Compared with the existing approaches based on an RG or 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 straightforwardly obtained from the FM-set tree.

   e) Development of a software tool for K-diagnosability and diagnosability analysis:
   In order to show the effectiveness of our method, we develop a diagnosis tool called On-the-Fly PEtri-Net-based Diagnosability Analyzer (OF-PENDA), and compare our approach with other existing ones with the help of the Workshop on Discrete Event Systems (WODES) diagnosis benchmark [Giu07] and our

developed railway level crossing (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 timed discrete event systems (TDESs):

   a) We deal with the diagnosability of TDESs. The model we use is the LTPN - an extension of TPN, wherein each transition is associated with an event that 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:

- In Chapter 2, we review some formalisms of DES and TDES, i.e. automata, PNs, timed automata (TAs) and TPNs. In particular, we present two extensions of PNs 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 DESs. First, we introduce our algebraic representation of LPNs. Then we provide algorithms to check K-diagnosability and conventional diagnosability. We demonstrate how to perform online diagnosis of LPN models. Finally, we developed a diagnosis tool called OF-PENDA and perform a comparative analysis 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 TDESs. We introduce the time interval splitting (TIS) technique to reformulate the diagnosis problem of TDESs in such a way to make it possible to apply the conventional 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 TDESs, and provide algorithms to check Δ-diagnosability and diagnosability for TDESs.

- 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: TAs, TPNs and LTPNs. We will explain why the LPN and the LTPN are chosen as the model for diagnosis analysis in this study. 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 [CL07; Lin94], using techniques such as language and automata, PN 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 PN models of the analyzed system.

2.1.1 Automata

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

2.1.1.1 Finite State Automaton (FSA)

Definition 1 *An* FSA *is a 5-tuple* $G = (X, \Sigma, \delta, x_0, F)$*, where:*

- X *is a finite set of states;*

- Σ *is a finite set of events;*
- $\delta : X \times \Sigma \to 2^X$ *is the* partial transition mapping*;*
- x_0 *is the initial state of the system;*
- $F \subseteq X$ *is the set of accepted states.*

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 target 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* $G = (X, \Sigma, \delta, x_0), F$*, as shown in Figure 2.1,*

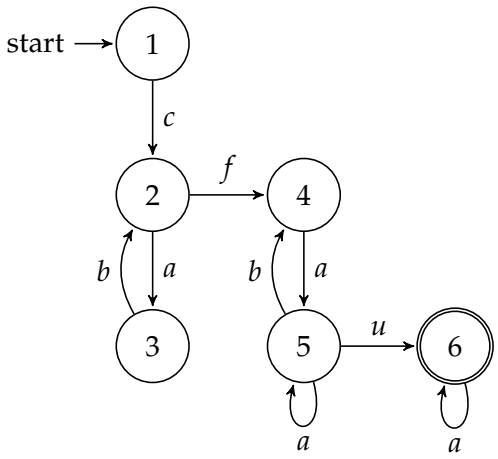


Figure 2.1: An example of automaton

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

2.1.1.2 Language Represented by Automata

The behavior of the system is described by the *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 into 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 that 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 $P_o : \Sigma^* \to \Sigma_o^*$ as

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

The inverse projection operator P_o^{-1} is defined as $P_o^{-1}(r) = \{s \in \mathcal{L} |\ P_o(s) = r\}$ for $r \in \Sigma_o^*$.

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 *post-language* of $\mathcal{L}$ after s, denoted by $\mathcal{L}/s$, is defined by:

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

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

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 non-deterministic 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 non-deterministic automaton. The algorithm of building an observer from a non-deterministic automaton is introduced in [CL07].

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 for 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* $\Sigma_u = \{f, u\}$, *the observer is built in Figure 2.2.*

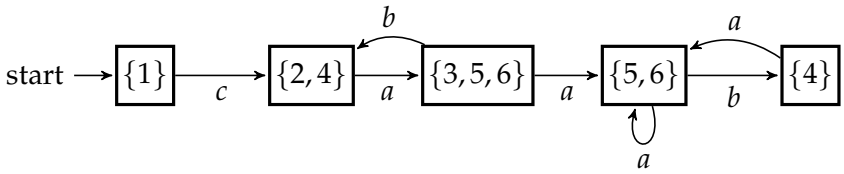


Figure 2.2: The observer of the automaton in Figure 2.1

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 be 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.*

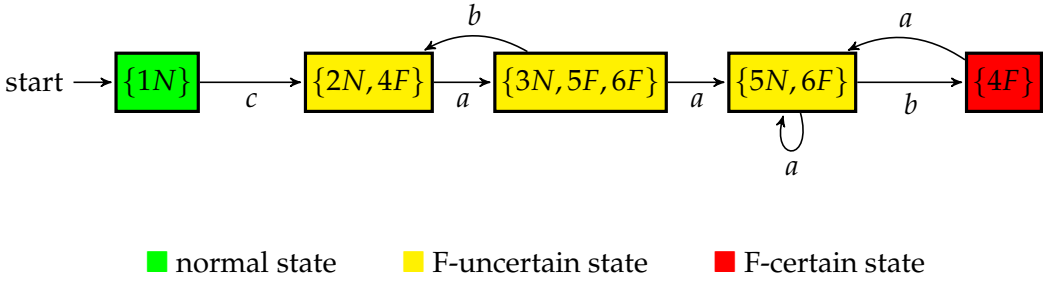


Figure 2.3: The diagnoser of the automaton in Figure 2.1

2.1.1.5 Diagnosability of Automata Models

The problem of diagnosability is to determine whether the system is diagnosable or not, i.e., once a fault has occurred, can it be detected and identified in a finite number of steps?

Definition 2 *[Sam+95] 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:*

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

where:

- $\Psi(\Sigma_{F_i})$ *denotes the set of all traces of $\mathcal{L}$ that end in a faulty event belonging to fault 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 within a finite delay.

Let us now introduce the definition of indeterminate cycle that is fundamental to test the property of diagnosability in the diagnoser automaton [Sam+95].

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 is:

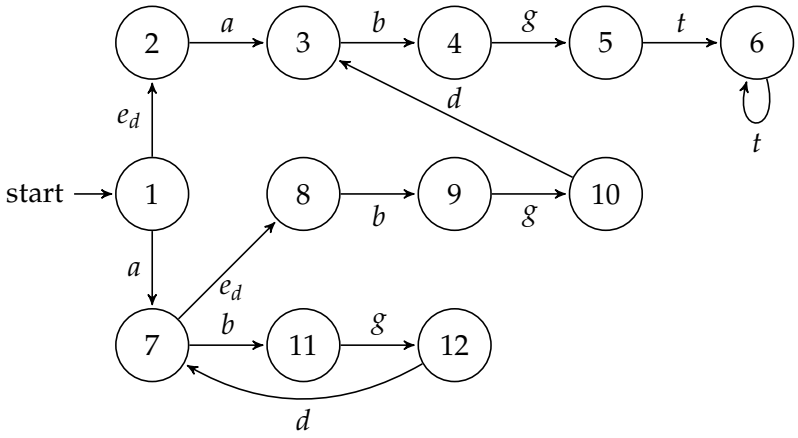
- a corresponding cycle in G involving only states that carry Y in their labels in the cycle in $Diag(G)$ and
- a corresponding cycle in G involving only states that carry N in their labels in the cycle in $Diag(G)$.

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

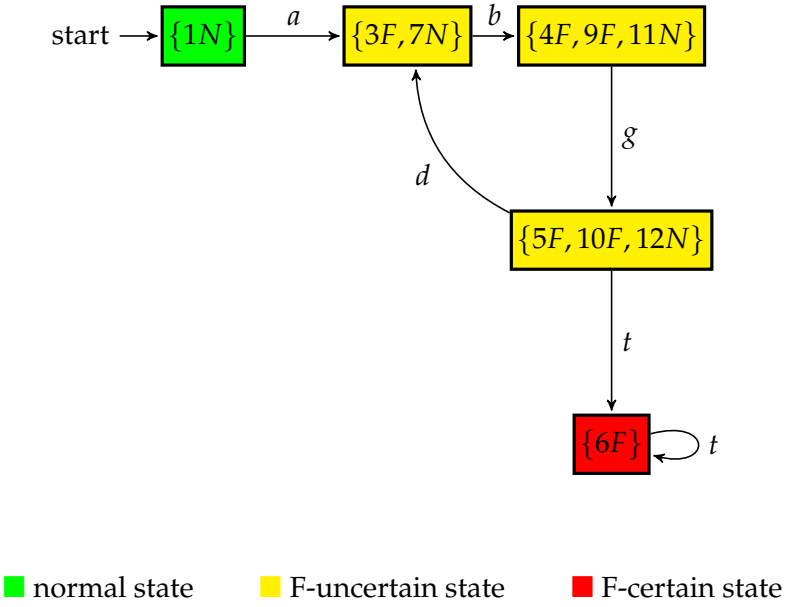
Proposition 1 *[Sam+95] A language $\mathcal{L}$ without multiple failures of the same type is diagnosable if and only if its diagnoser $Diag(G)$ has no indeterminate cycles w.r.t each failure type Σ_{F_i}.*

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), where $\Sigma_o = \{a, b, d, g, t\}$ is the set of observable events and e_d is the unique unobservable fault event. There is 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 is only a corresponding cycle of normal states (7,11 and 12) but no corresponding cycle of faulty states from $\{3, 4, 5, 9, 10\}$.

For details about diagnosability and related notations, the reader can refer to [CL07].



(a) An example of automaton [CL07]



(b) The diagnoser of the automaton in Figure 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 PNs (and their extensions) rather than automata to model DESs, since PNs have the following advantages [Giu+07]:

- 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 calculation.
- PNs can well present concurrent processes, which is also one of the original objectives to develop such a notation.
- PN 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 a finite number of places and transitions) can represent an infinite state space.

Here, we review some basics of PNs. For details, the reader can refer to [Mur89].

2.1.2.1 Definition of PN

Petri net, 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 *[Pet62] 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);*
- *$Pre\colon P \times T \to \mathbb{N}$ is the* pre-incidence mapping *that gives the arcs linking places to transitions in the net, as well as their corresponding weight;*
- *$Post\colon P \times T \to \mathbb{N}$ 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 the initial marking M_0. For simplicity, we will use the term "PN" to refer to "marked PN" afterward.

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 $M \geq Pre(\cdot, t_i)$, denoted by $M\ [\ t_i >$.

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

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

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

$$M' = M + C \cdot \vec{t_i} \tag{2.3}$$

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

$$C = Post - Pre \tag{2.4}$$

is called the *incidence matrix*.

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

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

$$M' = M + C \cdot \pi(\sigma) \tag{2.5}$$

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

$$\pi(\sigma) = \sum_{i=1}^{k} \vec{t_i} \tag{2.6}$$

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

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

Example 5 *Consider the example of N in Figure 2.5.*

- *The set of places is* $P = \{p_1, p_2\}$;

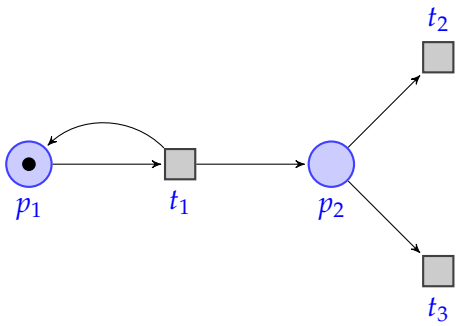


Figure 2.5: An example of PN

- *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:*

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

- *The post-incidence mapping is:*

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

- *The incidence matrix is:*

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

- $En(M_0) = \{t_1\}$ *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 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).

2.1.4 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 the initial marking M_0;
- Σ is a finite set of events for transition labeling;
- $\varphi\colon T \to \Sigma$ is the *transition labeling function*, φ is also extended to sequences of transitions, $\varphi\colon T^* \to \Sigma^*$.

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

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

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.*

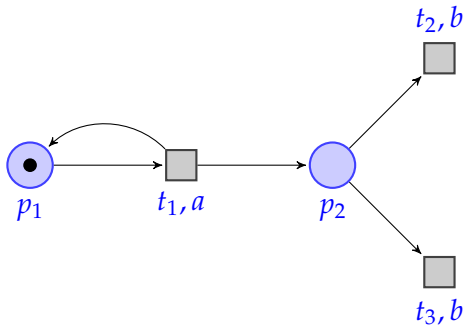


Figure 2.6: An example of LPN

- $\Sigma = \{a, b\}$ *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 an 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+13] Given an 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 classic 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 (TAs) [AD94], timed Petri nets [Ram74], time Petri nets (TPNs) [Mer74] and labeled time Petri nets (LTPNs) [Ber+05], etc.

In this section, we will review the background of TTS, TA, TPNs and LTPNs, which will be used afterward. Some notations are inspired from [Ber+05; Dia01; Gha+09; Tri02]. For more details, the reader can refer to the literature.

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 *[Hen+92] A* transition system *S is a 4-tuple $(Q, q_0, \Sigma, \rightarrow)$ where:*

- *Q is the set of 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 $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots$. A state $q' \in Q$ is reachable from a state q if a finite sequence $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} q_n$, exists such that $q_0 = q$ and $q_n = q'$.

Definition 6 *[Hen+92] 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;*
- *$l : \Sigma \rightarrow \mathbb{Q}_{\geq 0}$ is a minimum delay for each transition $e \in \Sigma$;*
- *$u : \Sigma \rightarrow \mathbb{Q}_{\geq 0} \cup \{+\infty\}$ is a maximum delay for each transition $e \in \Sigma$.*

2.2.2 Timed Automata (TA)

TAs 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 TAs, the reader can refer to the seminal paper [AD94].

2.2.2.1 Definition of TA

Definition 7 *[Alu+99] 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*

$$\varphi := x \leq c \mid c \leq x \mid x < c \mid c < x \mid \varphi_1 \wedge \varphi_2, c \in \mathbb{Q}_{\geq 0};$$

- *$E \subseteq L \times \Sigma \times 2^X \times \Phi(X) \times L$ 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.*

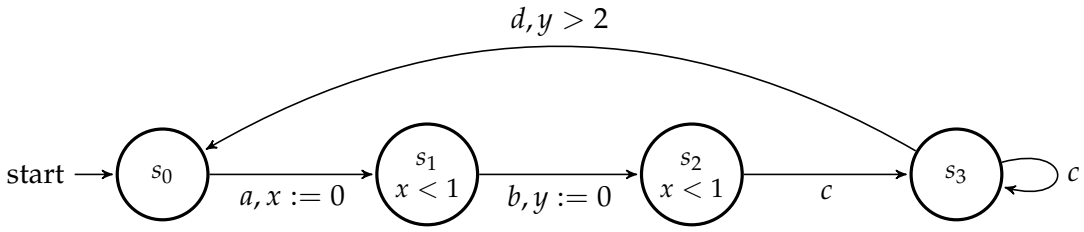


Figure 2.7: An example of timed automaton

- $L = \{s_0, s_1, s_2, s_3\}$;
- $L^0 = \{s_0\}$;
- $\Sigma = \{a, b, c, d\}$;
- $X = \{x, y\}$;
- $I(s_0) = I(s_3) = \emptyset$, $I(s_1) = I(s_2) = \{x < 1\}$;

2.2.2.2 TA Semantics

The semantics of a timed automaton A is defined as a TTS 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 non-Zeno.

Example 8 *For the TA 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 *[Mer74] A* TPN *is a 6-tuple* $(P, T, Pre, Post, M_0, SIM)$, *where:*

- $(P, T, Pre, Post, M_0)$ *is a marked PN;*

- $SIM\colon T \to \mathbb{Q}_{\geq 0} \times (\mathbb{Q}_{\geq 0} \cup \{+\infty\})$ *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,* $SIM(t_1) = [1,1]$, $SIM(t_2) = [0,2]$ *and* $SIM(t_3) = [0,3]$.

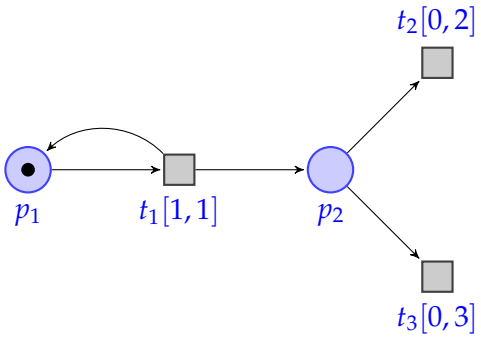


Figure 2.8: An example of TPN

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 the other transitions. Formally, I_0 is defined by:

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

2.2.3.2 Dynamics of TPN

Let us look at the state transition of the TPN from state $E = (M, I)$ towards state $E' = (M', I')$ following the firing of transition t with $I(t) = [\alpha_t, \beta_t]$ [1], at a relative date 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: $\forall k \in T$,

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

- 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:

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

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

If transition t remains enabled during its own firing (t is *multi-enabled* [Dia01]), 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. The discrete edges (relative to transitions) are defined by, for all $t_i \in T$:

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

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 $\forall k \in \{1, \cdots, n\}, M \geq Pre(t_k) \Rightarrow v'_k \leq \beta(t_k)$.

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 [BM83] and the zone graph [Gar+04] are examples of such approaches that 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 [BD91].

2.2.3.4 State Class

One can note that using notation (M, I) will lead to an 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 with an achievable firing sequence of transitions from the initial state:

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

D is called the *firing domain* and is the set of vector solutions of the τ_j-linear inequalities, 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:

- $M \geq Pre(\cdot, t_j)$;
- inequalities 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, denoted by $C \xrightarrow{t_j} C'$. Here C' is defined by:

1. $M' = M - Pre(\cdot, t_j) + Post(\cdot, t_j)$;
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 first fired among $En(M)$.
   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 a subset of C exists 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 *[Dia01] 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 each transition of the TPN with an event, this TPN is said labeled. We then speak of labeled time Petri net (LTPN) [Ber+05], such that each firing of transition simultaneously produces the corresponding event.

2.2.4.1 Definition of LTPN

Definition 9 *[Bou+06] A LTPN is a 8-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;*
- $\varphi : T \to \Sigma$ *is the* transition labeling function *as defined for LPNs.*

Informally, we can treat an LTPN as an 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.

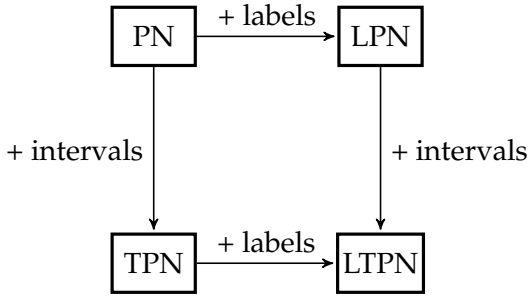


Figure 2.9: Relation between LTPN and PN, TPN, LPN

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

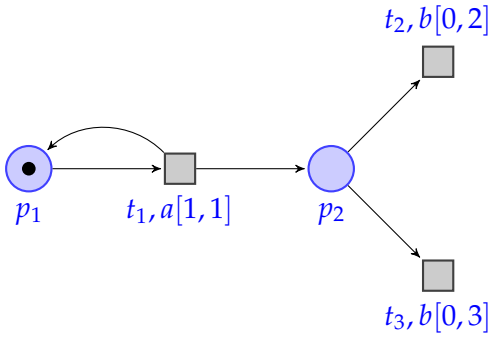


Figure 2.10: An example of LTPN

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)* [Dia01] 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 $\exists\ E_0, E_1, \cdots, E_n$ 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:*

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

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 an 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 [Tri02]. Now we will discuss these issues in the framework of LTPNs. Without loss of generality, we consider only one class of faults.

Definition 12 *[Liu+14a] Given an 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:*

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

We also say here that G is Δ-diagnosable.

In simple terms, any fault in a diagnosable LTPN can be diagnosed with a finite delay after its occurrence. Obviously, Δ_{min} exists 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 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 Hypotheses

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 hypotheses for the classic diagnosis analysis. For clarity, we also make the following remarks:

1. The system under consideration can be abstracted as an untimed or timed DES, which can be modeled by an LPN or an LTPN.

2. 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.

3. No achievable cycle of unobservable transitions exists in the LPN or LTPN.

4. During the analysis, the system structure does not change, and the system behavior does not respond to outside inferences (control commands).

5. We consider only permanent faults, i.e., the system remains in a faulty state after the occurrence of the fault.

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

2.4 Conclusion

In the section, we have reviewed the modeling formalism 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.

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 diagnosis 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 has been studied from different viewpoints with the application of various techniques, as shown in Figure 3.1:

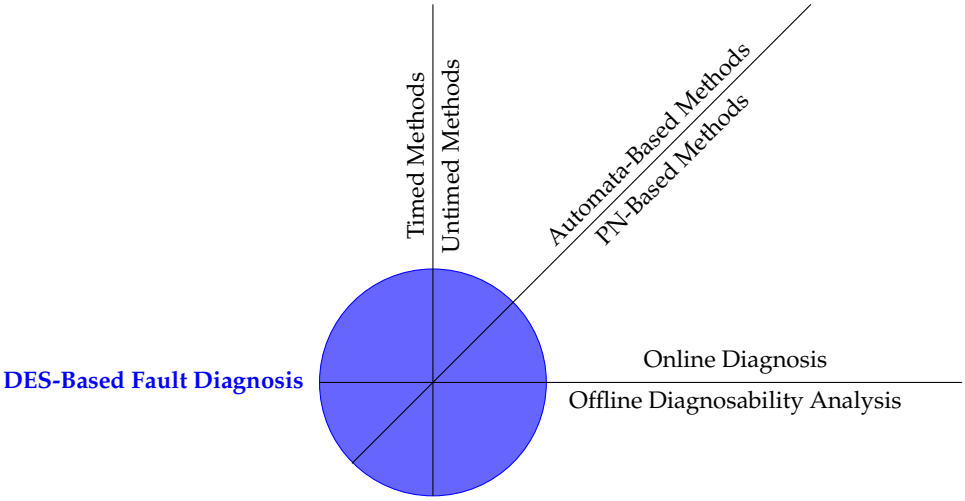


Figure 3.1: DES-based fault diagnosis

- The two most discussed topics are *online diagnosis* and *diagnosability analysis* [Lin94]. 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 formalism are *automata* and *PNs*. DES-based diagnosis is first studied in the framework of regular languages and automata [Lin94; Sam+95]. These automata-based approaches are based on state enumeration, which can induce state explosion problems. In order to overcome this problem, some subsequent automata-based approaches [Jia+01; YL02b] have been proposed for reducing the computational complexity, without the construction of a diagnoser automaton. Besides, a series of works [AF98; Bas+10; Bou+05; BW94; Cab+10; GL07; Gha+09; Haa09; RH09; Ush+98; Wen+05] 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 is investigated in both *untimed* and *timed* contexts [Cas10; Gha+09; HC94; Jia+06; Liu+13; Tri02; Wan+11; Wan+13; ZF06]. 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 different (time) dimension, so that the system behavior contains richer information. However, the complexity of dealing with such systems is significantly 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 A.1 in the appendix (cf. Appendix A).

3.2 Literature on Diagnosability of DESs

3.2.1 Classic Diagnosability

DES-based diagnosis has had a growing interest from both academia and industrial communities during the two past decades. [Sam+95] 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 [Ush+98], the language-based diagnosability of [Sam+95] is extended to unbounded PNs, where the net marking is observable and all transitions are unobservable, and the faults are associated with transitions. A simple ω diagnoser and sufficient conditions for diagnosability of unbounded PNs are proposed.

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

In [Jia+01], 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 system states and linear in the number of the failure types. In [YL02b], a comparable 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 [XZ04] and [Cab+12b], a composition net called verifier net (VN) 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 VN approach is more efficient compared with traditional diagnoser approach, however, it requires an exhaustive enumeration of the reachability set of the VN, which may be larger than the reachability set of the original PN. Note that the authors of [Cab+12b] also deal with the K-diagnosability problem and use the integer linear programming (ILP) technique.

In [Wen+05], the authors propose a sufficient condition for testing diagnosability by checking the structure of sub-net called T-components related to the T-invariants of the LPN without building a diagnoser. The indeterminate cycle in the sense of [Sam+95] 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 and b are observable and f is the faulty transition.

Note that the above mentioned literature is on the diagnosability of permanent faults.

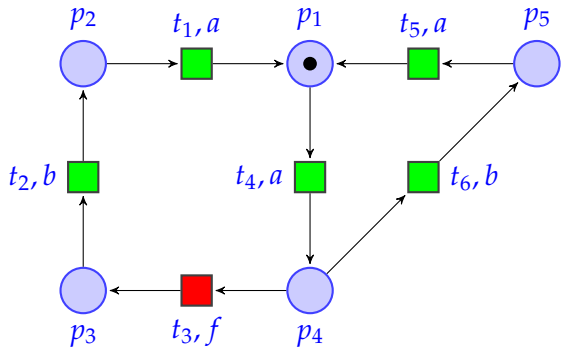


Figure 3.2: An undiagnosable LPN with 2 minimal T-invariants w.r.t the same observable projection

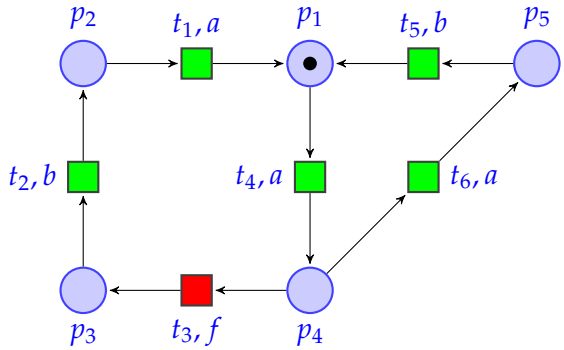


Figure 3.3: A diagnosable LPN with 2 minimal T-invariants w.r.t the same observable projection

There are also some works on the diagnosability of intermittent faults as in [Con+04; Jia+03a; Jia+06; YG04].

3.2.2 *K*-Diagnosability of Untimed DESs

The classic 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 [Bas+10; Bas+12; Cab+12b] which ensure that any fault (or class of faults) can be determined within a finite delay 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 of *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 [Bas+10], 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 [Bas+12], 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 [Cab+12b], the authors provide necessary and sufficient conditions for classic 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 VN. 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 [Bas+10; Bas+12].

Besides, there are many other studies on the other extensive versions of diagnosability for untimed DESs, such as $[1, K]$-diagnosability in [Jia+06; YG04], codiagnosability in [Wan+04], modular diagnosability in [Con+04], 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 TDESs

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 the timed 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.

[Tri02] 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 TAs. The developed algorithms are based on standard reachability analysis of some accepting states or on searching non-zeno runs.

[Xu+10] deals with diagnosis of TDESs modeled as TAs. 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 (TA) in the timed context may be rather resource-consuming.

Diagnosis of TDESs are also considered in the framework of TPN. In [Pen09], 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 [Jir+06], 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 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

Classic diagnoser-based approach [Sam+95; Sam+96] 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 an 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 [Jia+01; YL02b], 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 [Cab+12b; XZ04], a VN 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 VN.

3.3.3 Decentralized/Distributed/Modular Approaches

To overcome the problem of state explosion, approaches of decentralized, distributed and modular diagnosis [Ben+03; Cab+13a; Cas12; Con+06; Deb+00; GL07; JB05; Laf+05; Pen09; Pro02; QK06] have also been discussed. Although we put these three literally similar methods together, there are some differences between the terminology [ZL13]. 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 [GL07], 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 ILP, to solve DES diagnosis problems.

An early study on diagnosability of PNs models can be found in [Wen+05]. 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 to minimum T-invariants. The proposed algorithm shows the efficiency compared with state enumeration approaches.

In [Bas+10; Bas+12], 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 firable sequence in terms of firing count vectors may require, in the worst case, a number of firing count vectors equal to the sequence length.

[Cab+12b] deals with diagnosability of LPN models using a twin plant called 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 [Ben+03; Gra+10; Haa09; Mad+10]. Unfolding is a well-established technique for verifying properties of PNs; its use for this purpose was initially proposed by McMillan [MP95]. The unfolding of a PN 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.

[Gra+10] discusses the on-line diagnosis of distributed systems using TPN models. They propose to base the method on unfoldings. Given a partial observation, as a possibly structured set of actions, their method determines the causal relation between events in the model that explains the observation. It can also synthesize parametric constraints associated with these explanations. The method is implemented in the tool Romeo [Gar+05].

[Mad+10] 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 [Cla+94] 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 [BK08]:

- 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 to focus on the essential properties first. No complete requirement specification is needed.
- It provides diagnosis information and counterexamples in case a property is invalidated, which is useful for fault diagnosis.

In [Cim+03], 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 [Gra09], the author presents a symbolic-based approach to test diagnosability. The search can be performed in a classic forward manner, or in a backward manner which potentially avoids exploring all the search space of 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 [Hua+04], the authors study the 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 first order temporal logic model checking. An on-line algorithm for diagnoser synthesis is obtained by using predicates and predicate transformers.

[PG13] 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 [Gha+12].

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 diagnosis software tools for DESs, and the comparisons between them.

UMDES [Laf00] is a library of C routines for the study of DESs modeled by FSA. The tool provides manipulation of FSA, operations of supervisory control theory and failure diagnosis. The tool DESUMA [Ric+06] is an integration of the UMDES library with the graphical environment GIDDES for visualizing DESs. DESUMA allows the user to perform a variety of manipulations on DES modeled by FSAs, such as model editing, diagnosability analysis, verification, control under full and partial observation, and decentralized control.

In [Sta+06], the authors develop a tool for Discrete-Event Control And Diagnosis Analysis, called DECADA. DECADA checks diagnosability of an automaton model under partial observation.

[Cab+11] develops a software platform for the integration of DESs tools. The objective of this software platform is to integrate several tools dealing with PNs 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 [Cab+12a], 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.

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 Section 4.2 – Section 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+12]. Moreover, by extension, we transform the classic diagnosability problem into a series of K-diagnosability problems [Liu+14b], 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 [Sam+95],

verifier automata [YL02b], verifier (Petri) nets [Cab+12b], linear inequalities [Bas+12], 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 [SE05], since such techniques have the following advantages:

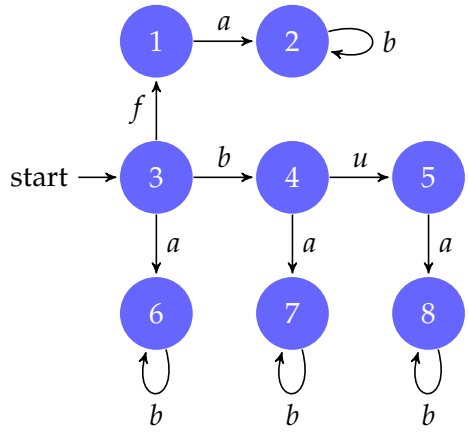
1. On-the-fly techniques can save memory resources. Generally, on-the-fly techniques permit us to generate and investigate only a part of the state space to find solutions, unlike the classic enumerative approaches which have to build the whole state space *a priori*. On-the-fly exploration techniques 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 Figure 4.2(b)), which can save time from both analysis stages.

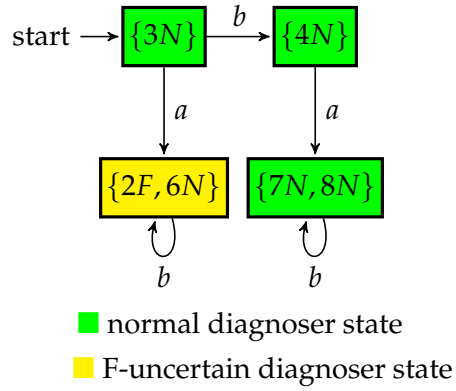
3. On-the-fly techniques can deal with some unbounded systems, since they return a verdict as soon as some specific features are found, instead of investigating the whole state space. Take the unbounded PN in Figure 4.3 (where t_1, t_2, t_4 and t_5 are observable transitions and t_3 is the only unobservable 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.

4.1.2 Incremental Analysis Technique

An *Incremental* method [Koe+04] is a search technique that reuses the information from previous searches when some parameters change. Generally, it is faster than performing



(a)



(b) *Diag*(*G*)

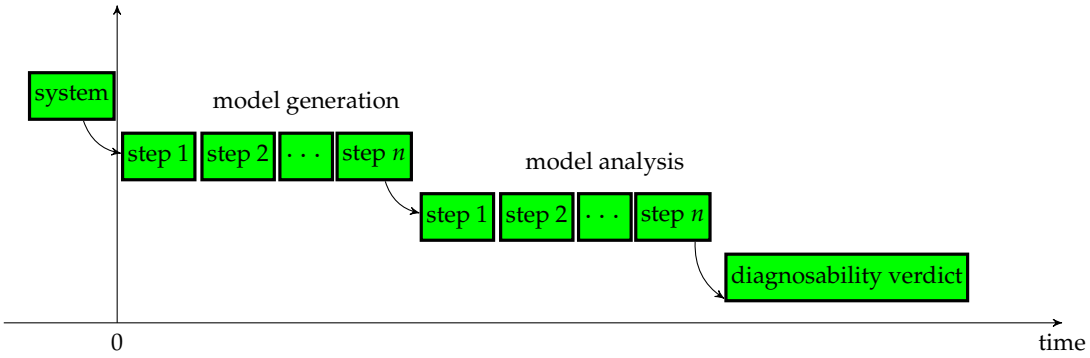
Figure 4.1: An automaton G and its diagnoser $Diag(G)$

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.

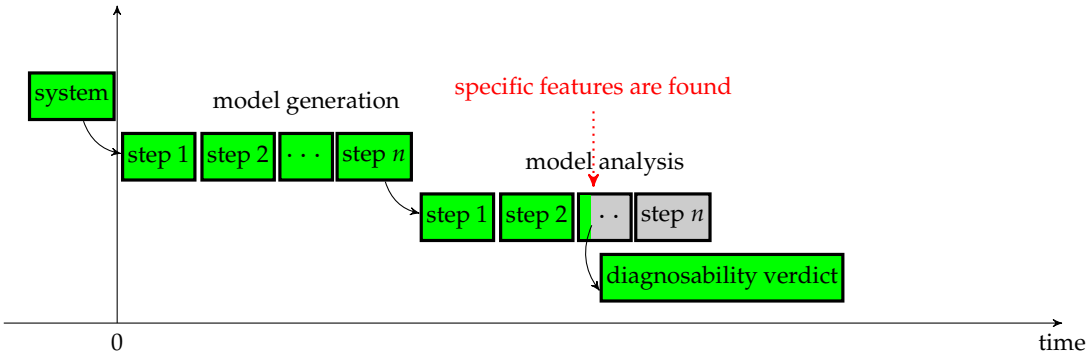
We will apply this technique in our diagnosability analysis, since it can bring the following advantages:

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 [Bas+12; Wen+05] have to rebuild and solve the equation system (or inequalities) when seeking out K_{min}, without using the previous search results.

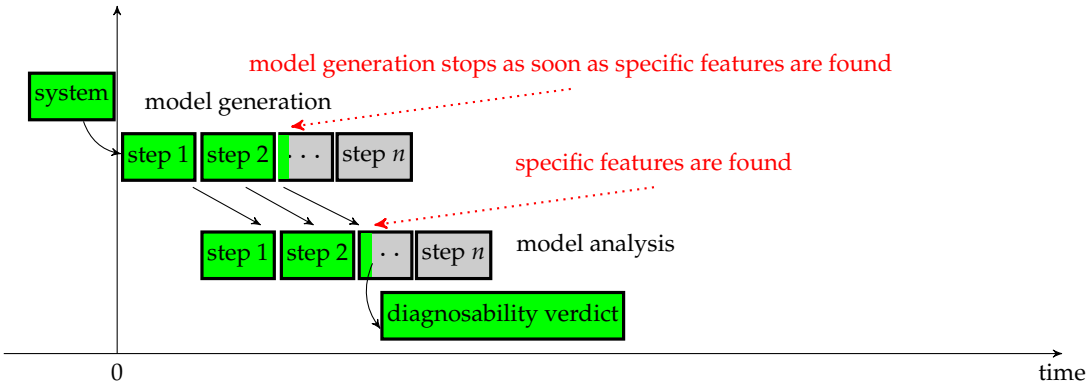
2. It is a skillful technique to speed up the search procedure. It should be used for bounded systems so that the search can terminate well. In particular, it can be used



(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

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.

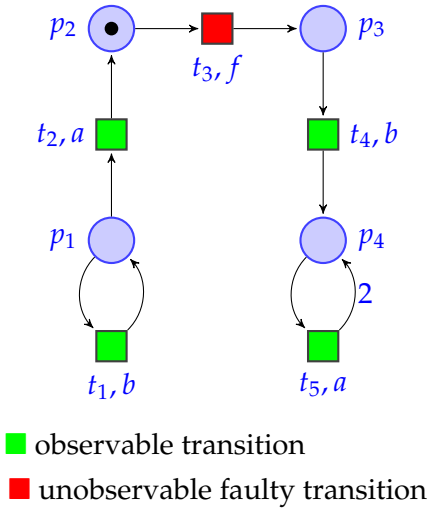


Figure 4.3: An unbounded PN

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 realty, 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 *markings*, *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* $\Sigma' = \{e_1, e_2, \ldots, e_{|\Sigma'|}\}$. *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 4.3 is an LPN, where* $\Sigma = \{a, b, u, f_1, f_2, f_3\}$, $\varphi(t_1) = f_1$, $\varphi(t_2) = f_2$, $\varphi(t_3) = f_3$, $\varphi(t_4) = u$, $\varphi(t_5) = \varphi(t_7) = \varphi(t_{10}) = a$, $\varphi(t_6) = \varphi(t_8) = \varphi(t_9) = b$.

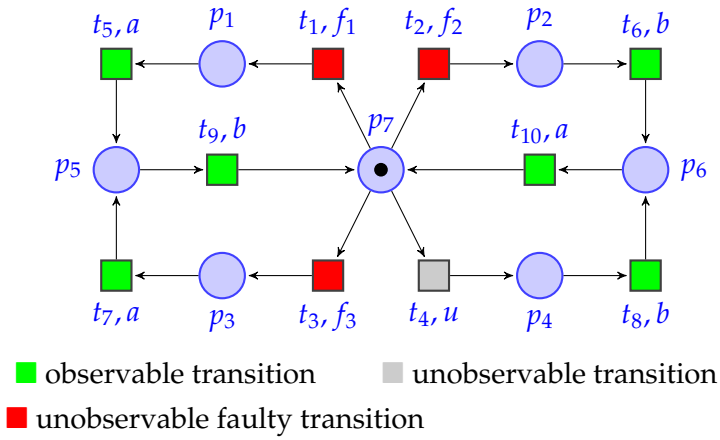


Figure 4.4: An LPN

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.*

The event-mapping matrix of this LPN is:

$$C_e = \begin{array}{c} \\ a \\ b \\ f_1 \\ f_2 \\ f_3 \end{array} \begin{array}{c} \begin{array}{cccccccccc} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 & t_9 & t_{10} \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \tag{4.1}$$

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:*

$$C_x = \begin{bmatrix} C \\ C_e \end{bmatrix} \tag{4.2}$$

Example 13 *The extended incidence matrix of the LPN in Figure 4.3 is:*

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

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:*

$$EM = \begin{bmatrix} M \\ E \end{bmatrix} \tag{4.4}$$

where:

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

The initial event marking is defined by:

$$EM_0 = \begin{bmatrix} M_0 \\ E_0 \end{bmatrix} = \begin{bmatrix} M_0 \\ \vec{0} \end{bmatrix} \tag{4.5}$$

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 4.3 is:* [1]

$$EM_0 = [\, 0\,0\,0\,0\,0\,0\,1 \mid 0\,0\,0\,0 \,]^{\tau} \tag{4.6}$$

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:

$$EM' = EM + C_x \cdot \pi(\sigma) \tag{4.7}$$

and we say that EM' is *reachable* from EM upon σ, written as $EM\,[\,\sigma\,>\,EM'$. 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, 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,

$$C_e = \begin{bmatrix} C_o \\ C_f \end{bmatrix} \tag{4.8}$$

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 $C_f(i,j) = 0$.

[1] M^{τ} is the transpose of matrix M.

Example 15 *The extended incidence matrix of the LPN in Figure 4.3, while considering two classes of faults* $\Sigma_{F_1} = \{f_1, f_3\}$ *and* $\Sigma_{F_2} = \{f_2\}$*, is then:*

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

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,

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

where:

– *mark* is a marking projection, $mark\colon \mathcal{E} \to \mathbb{N}^{|P|}$,

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

– *event* is a projection relative to the considered events, $event\colon \mathcal{E} \to \mathbb{N}^{|\Sigma_o|+m}$,

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

– *obs* is a projection relative to the observable events, $obs\colon \mathcal{E} \to \mathbb{N}^{|\Sigma_o|}$,

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

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

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

Example 16 *Considering the LPN in Figure 4.3, for sequence* $\sigma = t_1 t_5 t_9 t_3 t_7 t_9$*, we have* $EM_0\,[\,t_1\,\rangle\, EM_1\,[\,t_5\,\rangle\, EM_2\,[\,t_9\,\rangle\, EM_3\,[\,t_3\,\rangle\, EM_4\,[\,t_7\,\rangle\, EM_5\,[\,t_9\,\rangle\, EM_6$ *and* $EM_6 = EM_0 + C_x \cdot \pi(\sigma)$*, where the event markings generated successively are given in Table 4.1.*

Table 4.1: Event markings in Example 16

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

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 [RH09].

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*.

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* $Post^{\Sigma'} : T \times P' \to \{1\}$ *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 4.3. Initially, each event counter contains no tokens, since there is no occurrence of any event at the initial marking.*

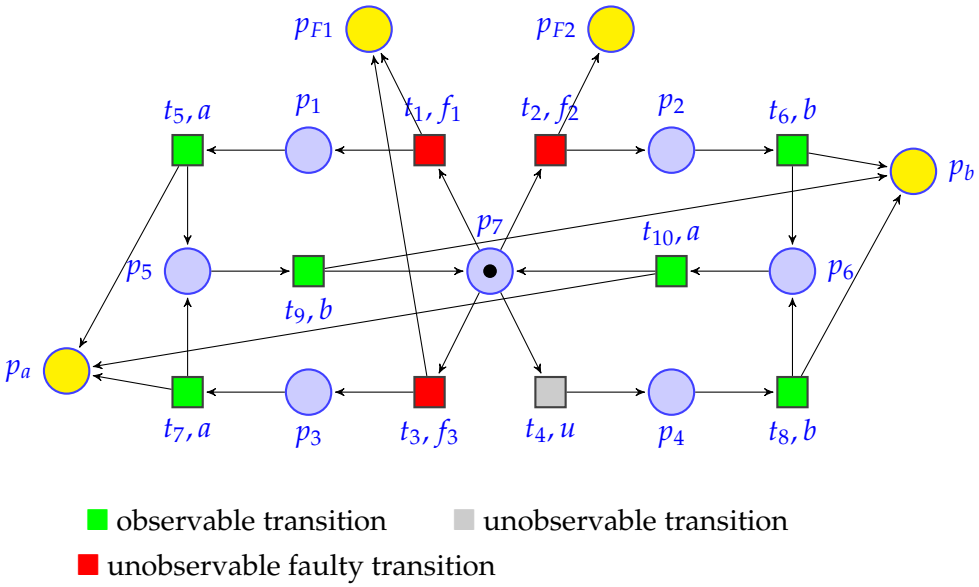


Figure 4.5: A LPN-EC

The LPN-EC can reach a new marking upon the firing of a sequence of transitions $\sigma = t_1t_5t_9t_3t_7t_9$, *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.

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.

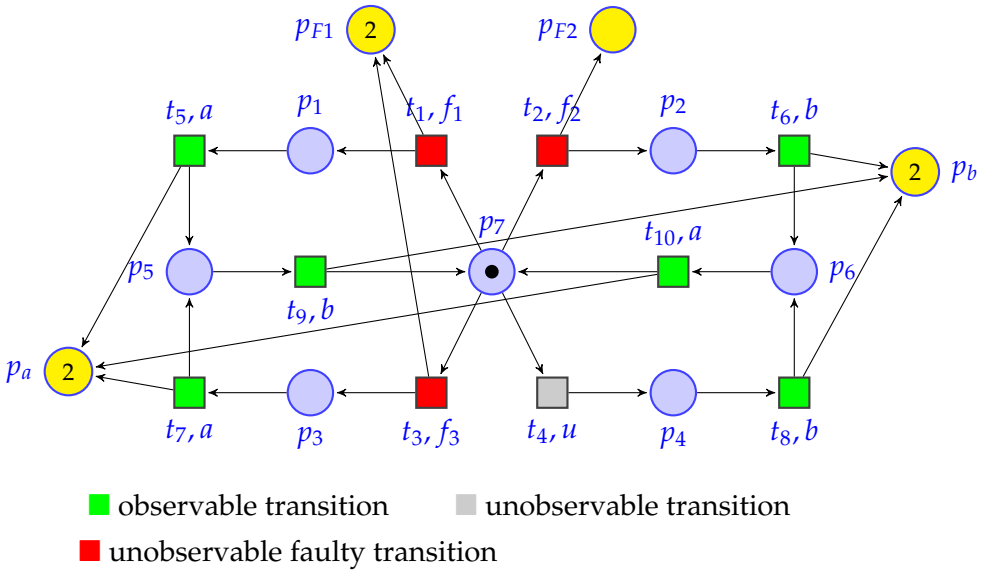


Figure 4.6: The reached marking of the LPN-EC in Figure 4.5 by the firing of $t_1 t_5 t_9 t_3 t_7 t_9$

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 [Cab+12b]. 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.

4.3 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 $\vec{x} = [x_1, \ldots, x_{|\vec{x}|}]^\tau$, $[\vec{x}]^k$ 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:

$$FM_0^i = \begin{bmatrix} M_0 \\ 0 \end{bmatrix} \tag{4.11}$$

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.*

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

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

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 $EM_1\,[\,\sigma\,>\,EM_2, \sigma \in T^*$, we write $FM_1^i\,[\,\sigma\,>\,FM_2^i$, 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.*

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. □

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 $N, \Sigma_o, \delta, FM_0$), where:*

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

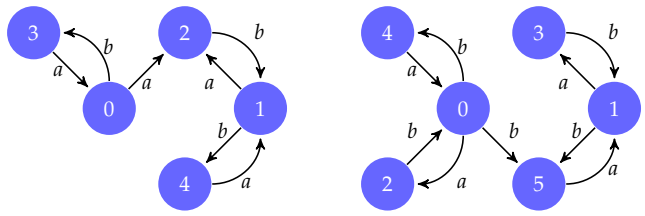
An FM^i-graph is a directed non-deterministic graph. Each node indicates a given fault marking and each arc indicates an observable event. Note that an arc from one node to itself is permitted. For a given observable event e and two nodes FM_1^i and FM_2^i, at most one arc labeled with e may link FM_1^i 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 [Sam+95] 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.

Example 20 *Consider the LPN in Figure 4.3, 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.*

Algorithm 1 Algorithm for δ function

Input: a fault marking FM and an observable event e;
Output: $\mathcal{F} = \delta(FM, e)$;
function $\delta(FM, e)$
    $\mathcal{F}_{con} \leftarrow \{FM\}$; ▷ $\mathcal{F}_{con}$ is the set of fault markings under consideration.
    $\mathcal{F} \leftarrow \emptyset$; ▷ $\mathcal{F}$ is the set of fault markings reached from FM immediately after the occurrence of e.
    for all $y \in \mathcal{F}_{con}$ **do**
        if $mark(y)\,[\,t >$ **then** ▷ t is enabled at marking $mark(y)$.
            if $t \in T_o$ **then**
                if $\varphi(t) = e$ **then**
                    $z \leftarrow y\,[\,t >$; ▷ z is reached from y by the firing of t.
                    $\mathcal{F} \leftarrow \mathcal{F} \cup \{z\}$;
            else ▷ If t is an unobservable transition.
                $z \leftarrow y\,[\,t >$;
                $\mathcal{F}_{con} \leftarrow \mathcal{F}_{con} \cup \{z\}$;
    return $\mathcal{F}$;



(a) FM^1-graph (b) FM^2-graph

Figure 4.7: The FM-graphs of the LPN in Figure 4.3

Table 4.3: Fault markings in Example 20

j	FM_j^1	FM_j^2
0	$[\,0\,0\,0\,0\,0\,0\,1 \mid 0\,]^\tau$	$[\,0\,0\,0\,0\,0\,0\,1 \mid 0\,]^\tau$
1	$[\,0\,0\,0\,0\,0\,0\,1 \mid 1\,]^\tau$	$[\,0\,0\,0\,0\,0\,0\,1 \mid 1\,]^\tau$
2	$[\,0\,0\,0\,0\,1\,0\,0 \mid 1\,]^\tau$	$[\,0\,0\,0\,0\,1\,0\,0 \mid 0\,]^\tau$
3	$[\,0\,0\,0\,0\,0\,1\,0 \mid 0\,]^\tau$	$[\,0\,0\,0\,0\,1\,0\,0 \mid 1\,]^\tau$
4	$[\,0\,0\,0\,0\,0\,1\,0 \mid 1\,]^\tau$	$[\,0\,0\,0\,0\,0\,1\,0 \mid 0\,]^\tau$
5	–	$[\,0\,0\,0\,0\,0\,1\,0 \mid 1\,]^\tau$

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 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 $\mathcal{X} = 2^{\mathcal{Q}}$ and the initial FM-set $x_0 = \{FM_0\}$.

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

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

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 $\Lambda : \mathcal{X} \to 2^{\mathcal{X}}$,

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

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 not exceed the number of all fault marking combinations:

$$C_{2p}^1 + C_{2p}^2 + \cdots + C_{2p}^{2p}$$

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

$$= 2^{2p} - 1$$

□

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

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

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 [Sam+95].

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 [Sam+95].

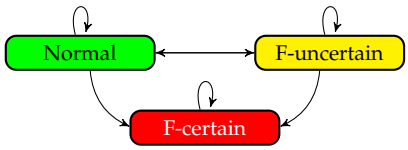


Figure 4.8: Fault propagation between FM-sets

Example 21 *Considering the LPN in Figure 4.3, 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 [Sam+95]. 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.

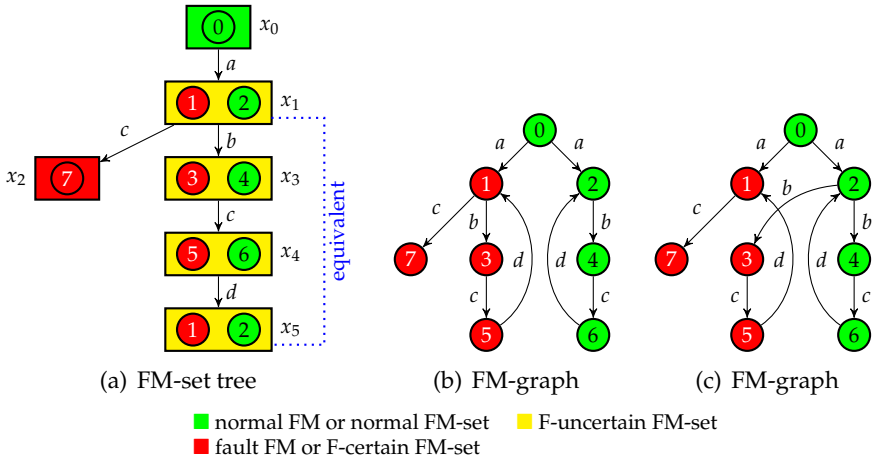


Figure 4.9: Checking an indeterminate cycle

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* $delay\colon \mathcal{X} \to \mathbb{N}$ *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

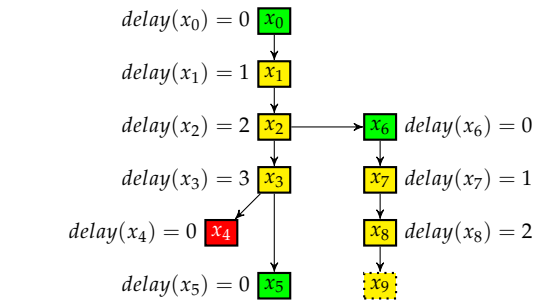
$tag(x)$	$tag(x')$	$delay(x')$
N	N or F	0
N	U	1
U	N or F	0
U	U	$delay(x)+1$

- *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*).*

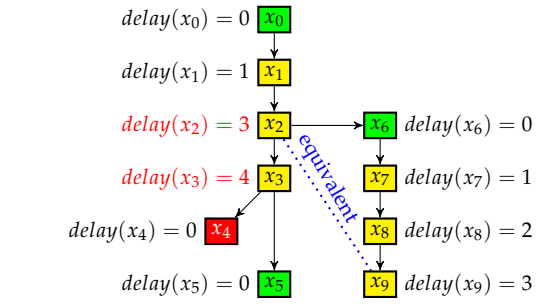
We do not consider the case where $tag(x) = F$, since, as will be shown later when building the FM-set tree, the processing of a branch is stopped as soon as a faulty node (an FM-set tagged F) is obtained since, obviously, the subsequent states will all be faulty (permanent faults) and, online, a diagnosis 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 where two equivalent uncertain nodes are met in the same branch). Indeed, when meeting the relation $x = x'$, we have to compare $delay(x)$ and $delay(x')$, 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 $delay(x_9) > delay(x_2)$, we update the delay value of x_2 and x_3 as shown in Figure 4.10(b).*



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



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

normal FM-set F-uncertain FM-set F-certain FM-set

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

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.

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

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 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: $(\mathcal{T}', y, n) \leftarrow$ 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.

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).

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;

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: $x' \leftarrow$ NEXTFMSET(x, e) ; ▷ x' is the child node of x.
6: update $\mathcal{T}$ to $\mathcal{T}'$ by adding x' from x upon e;
7: **if** $(x' \neq \emptyset) \wedge [tag(x') = N]$ **then**
8: **if** $(\forall\, x'' \in \mathcal{X}_v)(x' \neq x'')$ **then** ▷ No equivalent node already exists.
9: $\mathcal{X}_v \leftarrow \mathcal{X}_v \cup \{x'\}$;
10: $(\mathcal{T}'', y, n) \leftarrow$ KDIAG($\mathcal{T}', x', K$);
11: **if** $n \neq 1$ **then**
12: **return** $(\mathcal{T}'', y, n)$;
13: **if** $(x' \neq \emptyset) \wedge [tag(x') = U]$ **then**
14: **if** $delay(x') = K$ **then**
15: **return** $(\mathcal{T}', x', 0)$; ▷ 0 denotes that N_L is not K-diagnosable.
16: ▷ 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** ▷ Use of function
20: ▷ *path_exists* from the library *digraph* [Rus12].
21: **return** $(\mathcal{T}', x', -1)$; ▷ -1 denotes N_L is not
22: ▷ (K-)diagnosable due to the indeterminate cycle.
23: **else if** $delay(x') > delay(x'')$ **then**
24: $d \leftarrow delay(x') - delay(x'')$;
25: **if** UPDDELAY(x'', d, K) = FALSE **then**
26: **return** $(\mathcal{T}', x', 0)$; ▷ cf. Algorithm 5.
27: **else**
28: $\mathcal{X}_v \leftarrow \mathcal{X}_v \cup \{x'\}$;
29: $(\mathcal{T}'', y, n) \leftarrow$ KDIAG($\mathcal{T}', x', K-1$);
30: **if** $n \neq 1$ **then return** $(\mathcal{T}'', y, n)$;
31: **return** $(\mathcal{T}', x', 1)$; ▷ 1 denotes that N_L is K-diagnosable.

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 $delay(x') = K$, 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 $delay(x') < K$,

i. If there is an existing node x'' (in $\mathcal{X}_v$) such that $x' = x''$, (Lines 18 – 26 of Algorithm 3)

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' ← ∅;
5:     for all y ∈ x do
6:         x' ← x' ∪ δ(y, e);                ▷ Function δ is given in Algorithm 1.
7:         𝒩 ← 𝒩 ∪ x';                       ▷ 𝒩 is the set of the FM-graph nodes.
8:         for all z ∈ δ(y, e) do
9:             𝒜 ← 𝒜 ∪ {(y, e, z)};          ▷ 𝒜 is the set of the FM-graph arcs.
10:    return x';
```

Algorithm 5 UPDDELAY(): a sub-function 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 ⇝ z do
5:         if tag(z) = U then
6:             delay(z) ← delay(z) + d;
7:             if delay(z) ≥ K then
8:                 return FALSE;             ▷ N_L is not K-diagnosable.
9:             else
10:                return UPDDELAY(z, d, K);
11:    return TRUE;
```

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 $delay(x''') \geq K$ 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 4.3. 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}.*

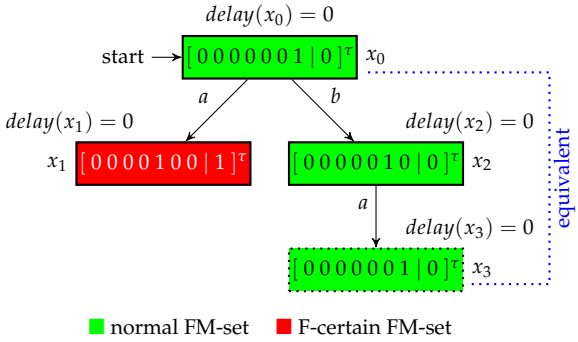


Figure 4.11: Solution process of K-diagnosability for Σ_{F_1}

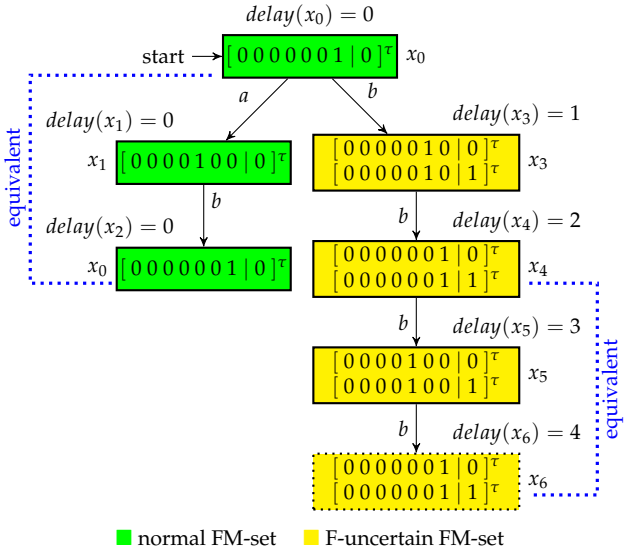


Figure 4.12: Solution process of K-diagnosability for Σ_{F_2}

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

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

Example 24 *Using Algorithm 6, we have analyzed the diagnosability of LPN N_L in Figure 4.3 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

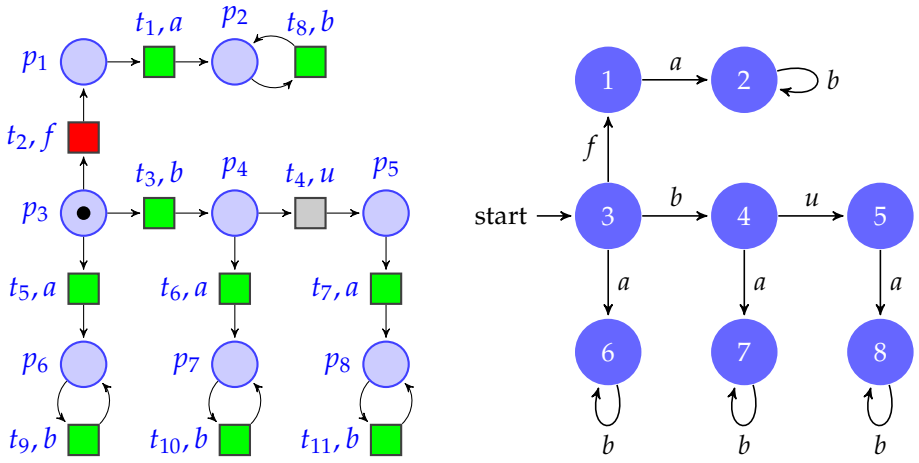
Let us analyze the proposed algorithm in terms of memory complexity. According to Proposition 1 and 2, for a given class of faults, if the number of markings of the considered LPN is p (which is also the number of automaton states if we consider the reachability graph as an automaton), the number of 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 [Sam+95]. 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.*

Table 4.5: The markings in Figure 4.13(b)

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

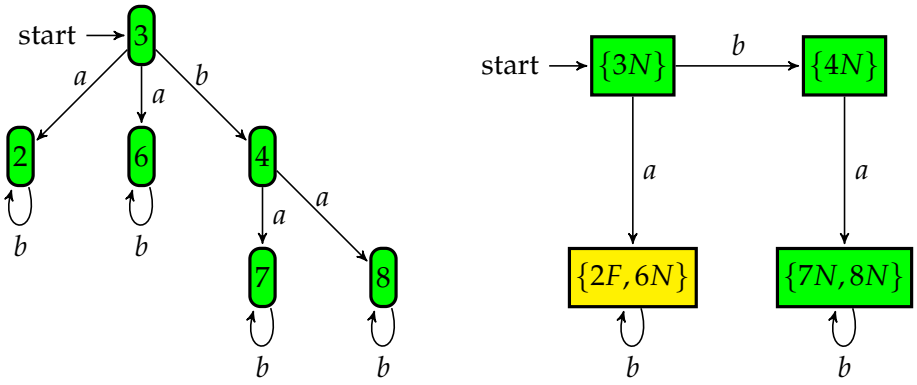


(a) N_L

(b) The language equivalent automaton G of the reachability graph of N_L

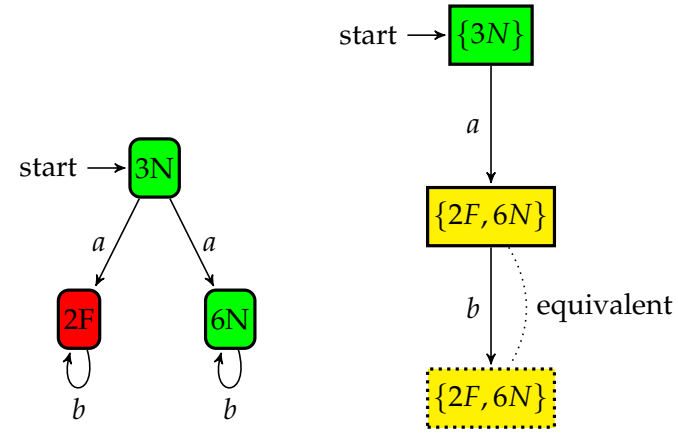
Figure 4.13: LPN N_L and its reachability graph G

Using the traditional diagnoser approach, diagnoser Diag(G) (cf. Figure 4.14(b)) can be built based on observer 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)).



(a) *Obs*(*G*): the observer of *G* (b) *Diag*(*G*): the diagnoser of *G*

Figure 4.14: The observer and diagnoser of *G*



(a) The FM-graph of N_L (b) The FM-set tree of N_L

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

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

Table 4.6: The markings in Figure 4.15

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

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, 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 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.

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

Input: A sequence composed of observable event e_j;
Output: The fault occurrence information of the current system state;
$q_0 \leftarrow N$; ▷ The system is normal (N) after the initialization.
$j \leftarrow 1$;
while the system is in operation **do**
    Wait for the input observable event e_j;
    $q_j \leftarrow$ the state from q_{j-1} upon e_j;
    switch q_j **do**
        case N
            `assert`(No fault belonging to Σ_{F_i} has happened;)
        case U
            `assert`(A fault belonging to Σ_{F_i} has probably happened;)
        case F
            `assert`(A fault belonging to Σ_{F_i} has happened;)
        case $\emptyset$ ▷ The system arrives at a blocked faulty state "F".
            `assert`(A fault belonging to Σ_{F_i} has happened;)
    $j \leftarrow j + 1$;

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.

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 [Giu07] and the LC benchmark are used in the comparative simulation using

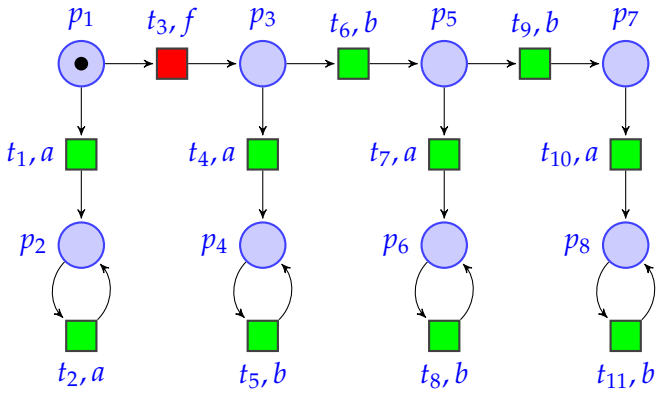


Figure 4.16: LPN N_L

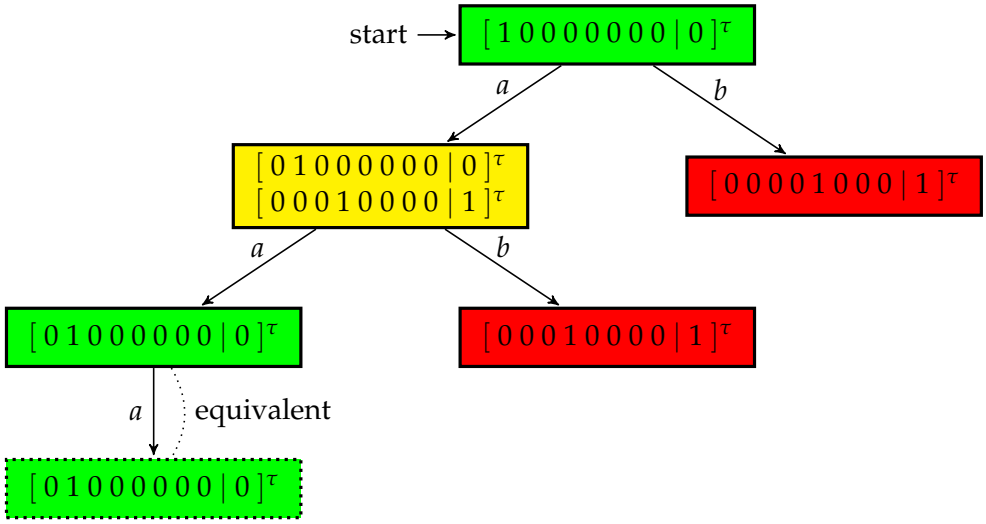


Figure 4.17: The FM-set tree of N_L

OF-PENDA and the UMDES library [Laf00], with the help of TINA tool [Ber+04]. 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;

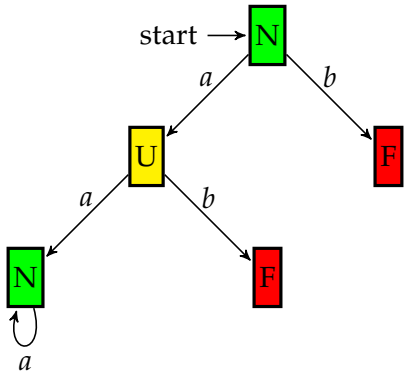


Figure 4.18: The diagnoser of N_L

– 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 [Giu07].

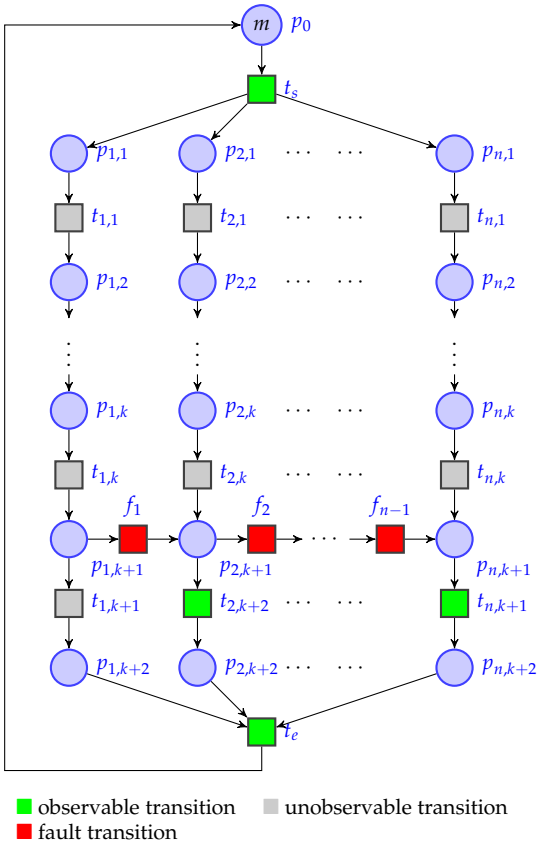


Figure 4.19: The WODES diagnosis benchmark

4.6.1.2 Comparative Simulation

We now analyze the diagnosability upon the fault class $\Sigma_{F_1} = \{f_1, \ldots, f_{n-1}\}$. 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 4^{th} 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}".
- The 5^{th} 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 6^{th} column titled "$|\mathcal{N}|$" is the number of the FM-graph nodes.
- The 7^{th} column titled "$|\mathcal{X}_v|$" is the number of the FM-set tree nodes.
- The 8^{th} column titled "$|Diag|$" is the number of the diagnoser automaton states generated by UMDES.
- The 9^{th} 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 [Laf00] does not.

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

Table 4.7: Diagnosability analysis results based on WODES benchmark

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.

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 [Bas+12], K is considered to be the number of both observable and unobservable transitions after a faulty transition. While here, as well as in [Cab+12b] 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.

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 [Bas+12], 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 TAs in [Tri02].

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 [Cab+10; Sam+95] 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 [Cab+12b], 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 [Bas+12], 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. 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 emmitted. 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 [Sam+98]. Besides, the WODES diagnosis benchmark we dealt with is unlive when $n \geq 2$, which

is against the assumption of liveness in [Sam+95]. 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 (LC) 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 LC Benchmark

In this section, we develop an n-track LC benchmark based on the single-track LC model in [LS85]. Different from the original model of [LS85], our benchmark is live and integrated with the operation principles for multi-track LC, which can be sufficiently complex for testing diagnosability analysis approaches.

Figure B.7 describes a global LPN model for a signle-track LC with a unidirectional track. Based on this model, a more general model is given in Figure B.8 – involving n railway tracks, which can be obtained from the single-track 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-track LC benchmark, the reader can refer to Appendix B.

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});

- $|P|$ 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, the following remarks can be made:

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-track 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-track benchmark even if it can be calculated eventually. For the case of $n = 9$, the calculation stops with an "out of memory" error. Thus, the analysis of 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. 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.

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 $\pm$1s. For the OF-PENDA, an automatic timer has been integrated with an error margin of less than 1ms.

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 in terms of memory consumption. Besides, it is worthwhile to mention that the FM-sets can be far fewer than the diagnoser states when an indeterminate cycle exists (i.e., the model is undiagnosable) and is detected early.

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

n	Σ_{Fi}	$\|P\|$	$\|T\|$	$\|A\|$	$\|R\|$	$\|\mathcal{N}\|$	$\|Diag\|$	$\|\mathcal{X}_v\|$	$\mathcal{D}_D$	$\mathcal{D}_V$	$\mathcal{D}_O$	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 *verifier_dia.exe* of UMDES

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

results obtained by OF-PENDA

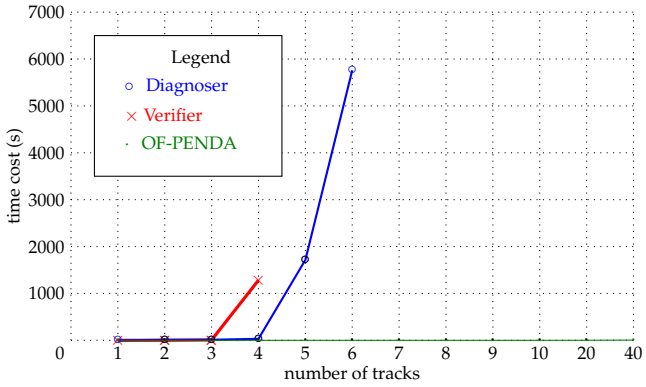


Figure 4.20: Time cost for the diagnosability analysis on Σ_{F1}

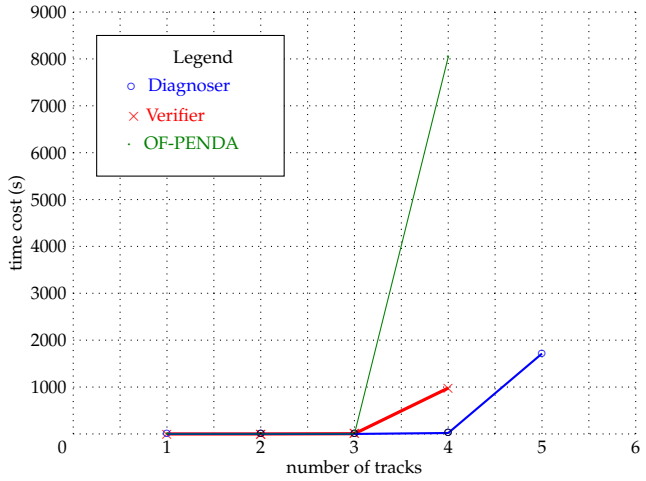


Figure 4.21: Time cost for the diagnosability analysis on Σ_{F2}

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, whereas, on the contrary, the reachable markings are computed on the fly while investigating diagnosability by OF-PENDA.

8. 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-track 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-track 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 $t_1t_4t_{2,1}t_{2,2}t_{2,3}t_{2,4}t_2t_5$ exists, which corresponds to an indeterminate cycle. Therefore, the system is undiagnosable.

- For Σ_{F2} in the cases of n-track LC ($n \geq 1$), t_6 can fire or not right after the firing of sequence $t_{i,1}t_1t_4$. Then the system can remain F_2-uncertain for as long as 6 steps during the firing of sequence $t_{i,2}t_{i,3}t_{i,4}t_{i,1}t_2t_1$. The 7^{th} 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.

More importantly, 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. For example, the diagnosability analysis for the LC model is performed in less than 6 seconds, for even when $n = 40$, whereas UMDES (diagnoser and verifier techniques) do not issue a verdict within 6 hours for $n > 4$. However, for the diagnosable cases (Σ_{F2}), we spend less memory but more time than UMDES, although the obtained results remain comparable: OF-PENDA and UMDES-verifier block from $n = 4$, whereas UMDES-diagnoser blocks at $n = 5$. This gap in terms of efficiency depending on whether the model is diagnosable or not can be explained as follows: For the undiagnosable models, the analysis by OF-PENDA is completed as soon as an indeterminate cycle is found which can occur quickly, hence avoiding the generation and analysis of an important part of the state space. However, for diagnosable models, it is likely that a bigger part of the state space is generated/analyzed since, in this case, the only stopping condition which allows us to avoid building/investigating the whole state space is when a faulty node is generated. Indeed, since we deal with permanent faults, it is useless to continue investigating the following nodes since they are faulty as well.

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 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 [Tri02], fault diagnosis of timed DESs has been presented in the framework of TAs. Necessary and sufficient conditions for diagnosability of TAs have been given. For the analysis based on TPNs, it is possible to first transform TPN models into TAs using existing techniques, and then to reuse the results of [Tri02]. 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 TAs and more suitable for modeling a real system. In the contrary, generating the whole state space (TAs) 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.

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 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

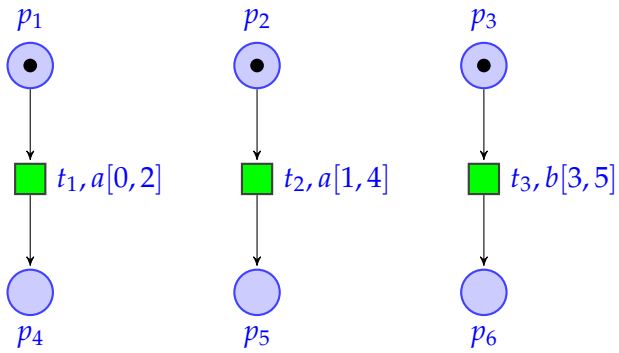
First, let us look at the motivation of splitting time intervals with the following example.

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.*

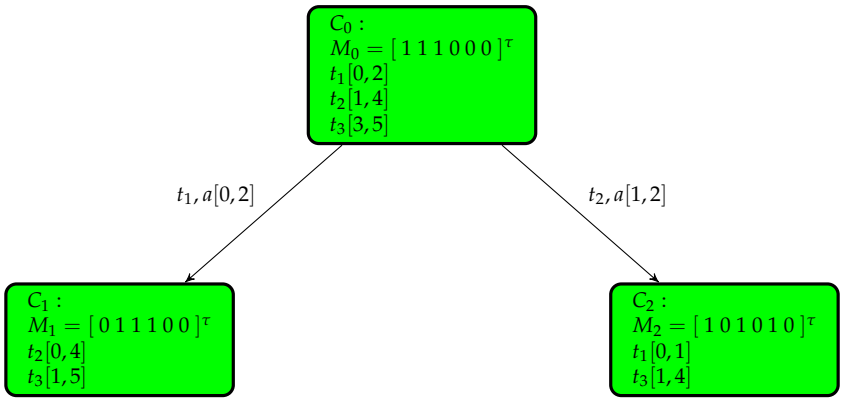
Using the classical analytical approach [BM83], we can compute all the state classes reachable from the initial class C_0. A part of the state class graph is shown in Figure 5.1(b). From state class C_0, we have two reachable state classes C_1 and C_2 by the firing of transitions t_1 at a date in $[0, 2]$ 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.

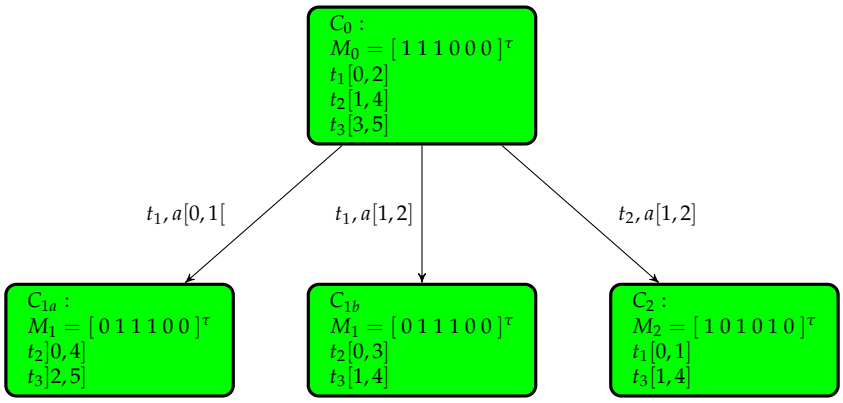
In order to eliminate this nondeterminism, we want to modify the state class graph of N_{LT} while considering transition labeling, so that by each observation of event with its exact occurrence



(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}

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 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 with regard to (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* [All83] is indeed a set of non-negative rational numbers but may also include infinity:

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

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 $\overline{\alpha}$. For $\beta = a]$ (resp. $a[;]a; [a$), we denote $bound(\beta) = a$ and $border(\beta) =]$ (resp. $[;]; [$).

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

$$i = left(i) \cap right(i)$$

where

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

and

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

We can also denote by

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

$$bound(right(i)) = 2$$

$$border(right(i)) = [$$

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

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

We define an order relation "$\prec$" between semi-intervals as follows:

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

5.1.3 Basic Interval Set (BIS)

In order to eliminate nondeterminism in LTPNs, time interval splitting techniques [Liu+14a] 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 augmented state class (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. $\forall\, \alpha \in A, \exists\, \beta_1, \beta_2, \cdots, \beta_m \in BIS(A)$, *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 $\cup_{j=1}^{+\infty}\{\beta_j\}$ gives the ordering of the intervals' values, i.e., β_1 holds the smaller values, and so on, as shown in Figure 5.2.

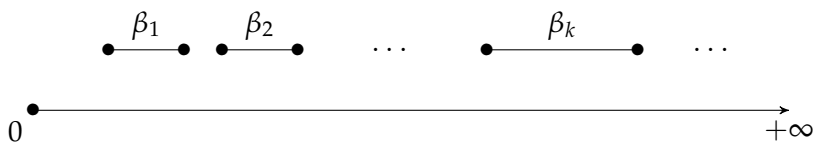


Figure 5.2: Time intervals in $BIS(A)$

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,

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

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. □

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

Algorithm 8 Computation of BIS

Input: A; ▷ A is a finite interval set.
Output: B; ▷ $B = BIS(A)$
$C \leftarrow \emptyset$; ▷ C is an array of semi-intervals ordered according to $\prec$.
for all $\alpha \in A$ **do**
    $C \leftarrow C \cup \{left(\alpha)\} \cup \{right(\alpha)\}$;
reorder C according to $\prec$;
$c_0 \leftarrow \overline{c_1}$; ▷ c_j $(j = 1, 2, \ldots)$ denotes, in the order of $\prec$, the j^{th} element of C.
$C \leftarrow C \cup \{c_0\}$; ▷ Insert c_0 into C, then c_0 will be the first element of C instead of the original c_1.
for j from 1 to $(|C| - 1)$ **do** ▷ For all the elements of C except the first and the last one.
    if c_{j-1} is a right semi-interval **then**
        $\alpha \leftarrow c_{j-1}$; ▷ α is a right semi-interval.
    else
        $\alpha \leftarrow \overline{c_{j-1}}$; ▷ α is a left semi-interval.
    if c_j is a left semi-interval **then**
        $\beta \leftarrow c_j$; ▷ β is a left semi-interval.
    else
        $\beta \leftarrow \overline{c_j}$; ▷ β is a right semi-interval.
    $B \leftarrow B \cup \{(\alpha \cap \beta)\}$;
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,

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

and obviously $\forall j = 1, \cdots, m, \beta_j \subseteq \alpha$.
Now suppose that

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

From Condition 1 in Definition 24,

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

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

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

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

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

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

□

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.,

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

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

or conversely

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

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

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

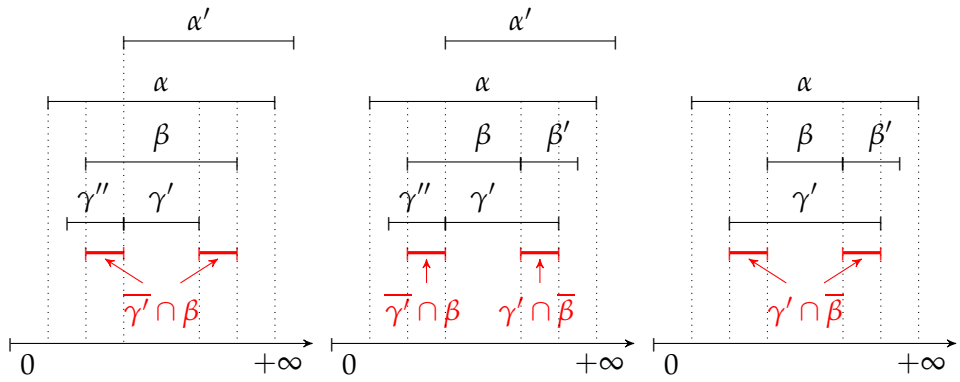
or

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

According to Condition 3 in Definition 24,

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

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



(a) $\gamma' \subseteq \beta$ (b) $\gamma' \not\subseteq \beta$ and $\beta \not\subseteq \gamma'$ (c) $\beta \subseteq \gamma'$

Figure 5.3: The relation between β and γ'

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

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 $B_1 = B_2$. □

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.,*

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

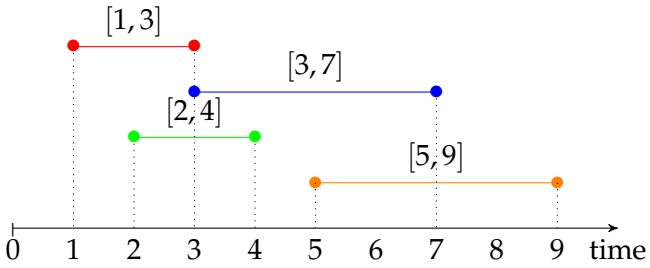
3. *For each pair of neighboring semi-intervals* α *and* β *in the ordering of the previous step such that* $\alpha \prec \beta$*, let* $\alpha' = \overline{\alpha}$ *(resp.* $\beta' = \overline{\beta}$*) if* α *is a left (resp.* β *is a right) semi-interval, otherwise* $\alpha' = \alpha$ *(resp.* $\beta' = \beta$*). 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.*

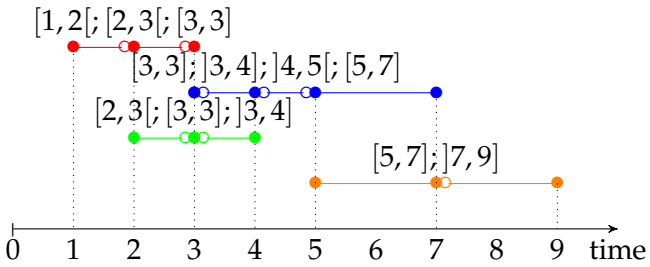
A graphical illustration for this solution is given in Figure 5.4.

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) Intervals in set A



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

Figure 5.4: The computation of BIS in Example 30

5.2 Reachability Analysis of LTPN 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 [Dia01], and $y = y'$.

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:

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

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 [Dia01], 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., $2m$. □

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 [Gha+09]. 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.*

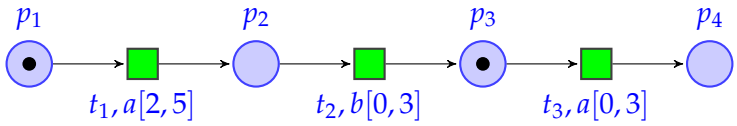


Figure 5.5: The LTPN graph for Example 30

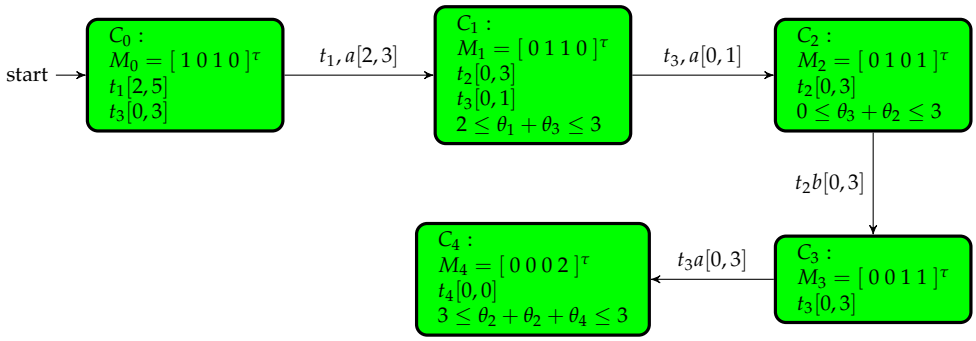


Figure 5.6: The state classes following σ for Example 30

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:

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

$$\begin{cases} 2 \leq \theta_1 \leq 3 & \textit{constraint of } t_1 \textit{ between } C_0 \textit{ and } C_1 \\ 0 \leq \theta_3 \leq 1 & \textit{constraint of } t_3 \textit{ between } C_1 \textit{ and } C_2 \\ 0 \leq \theta_2 \leq 3 & \textit{constraint of } t_2 \textit{ between } C_2 \textit{ and } C_3 \\ 0 \leq \theta'_3 \leq 3 & \textit{constraint of } t_3 \textit{ between } C_3 \textit{ and } C_4 \\ 2 \leq \theta_1 + \theta_3 \leq 3 & \textit{constraint of } C_1 \\ 0 \leq \theta_3 + \theta_2 \leq 3 & \textit{constraint of } C_2 \end{cases}$$

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* $(\mathcal{Q}_{ASCG}, \mathcal{A}_{ASCG}, \mathcal{T}_{ASCG}, q_0)$*, where:*

- $\mathcal{Q}_{ASCG} \subseteq 2^{\mathcal{N}_{ASC}}$ *is the set of ASC-graph nodes;*
- $q_0 = (C_0, N)$ *is the initial ASC-graph node;*
- $\mathcal{T}_{ASCG} : \mathcal{Q}_{ASCG} \times \Sigma_o \rightarrow \mathcal{Q}_{ASCG}$ *is the transition mapping between ASC-graph nodes. Given* $X \in \mathcal{Q}_{ASCG}, e \in \Sigma_o$, $\mathcal{T}_{ASCG}(X, e) = \{q' \mid \exists\, q \in X, \sigma \in Can(q), \varphi(\sigma) = e, q \xrightarrow{\sigma} q'\}$*;*
- $\mathcal{I}$ *is the set of time intervals (values in* $\mathbb{Q}_{\geq 0}$*);*
- $\mathcal{A}_{ASCG} \subseteq \mathcal{Q}_{ASCG} \times (\Sigma_o \times 2^{\mathcal{I}}) \times \mathcal{Q}_{ASCG}$ *is the set of directed arcs of the ASC-graph:* $\mathcal{A} = \{(q, e, i, q') \mid \exists\, \sigma \in Can(q), q' \in \mathcal{T}_{ASCG}(q, e),\ s.t.\ SD(\sigma) = i, \varphi(\sigma) = e\}$*, 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 [BM83] and the developed tools such as [Ber+04; Gar+05], 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.*

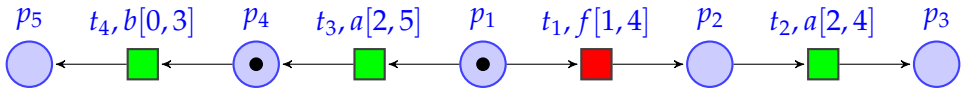


Figure 5.7: The LTPN graph for Example 32

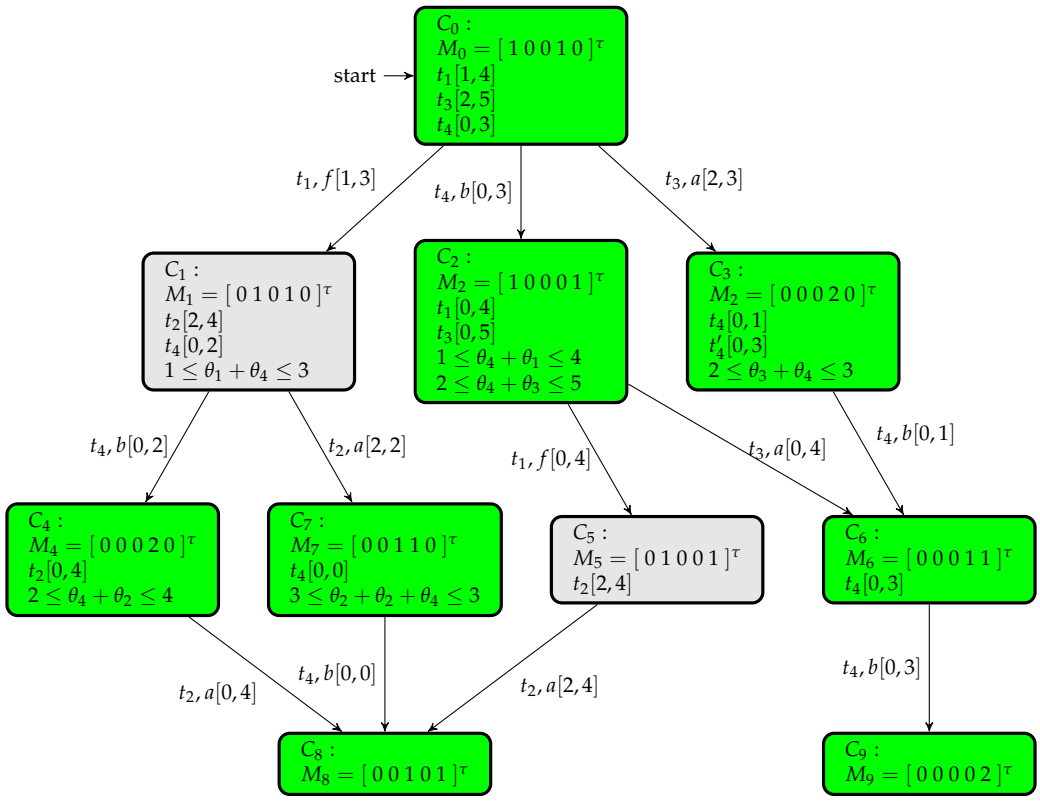
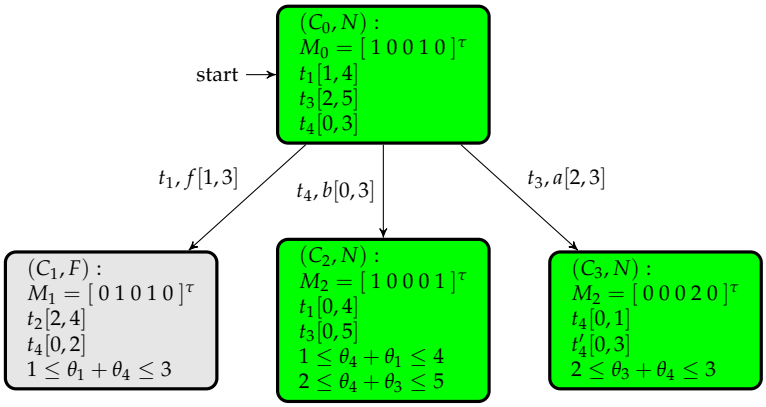


Figure 5.8: The state graph of the LTPN in Figure 5.7

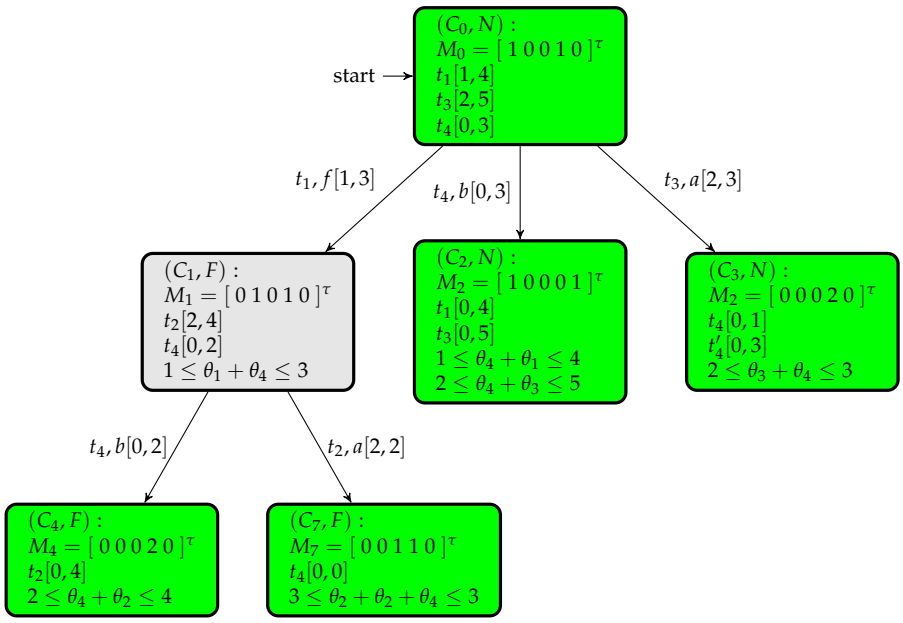
We can compute, step by step (from Figure 5.9(a) to 5.9(a)), to obtain its ASC-graph as shown in Figure 5.9(a):

- *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).*

- *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.*



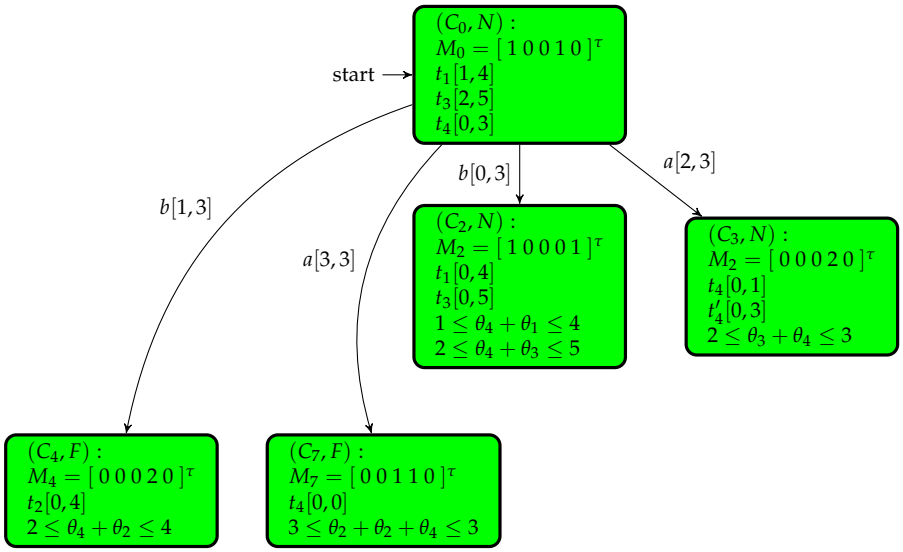
(a) Step 1



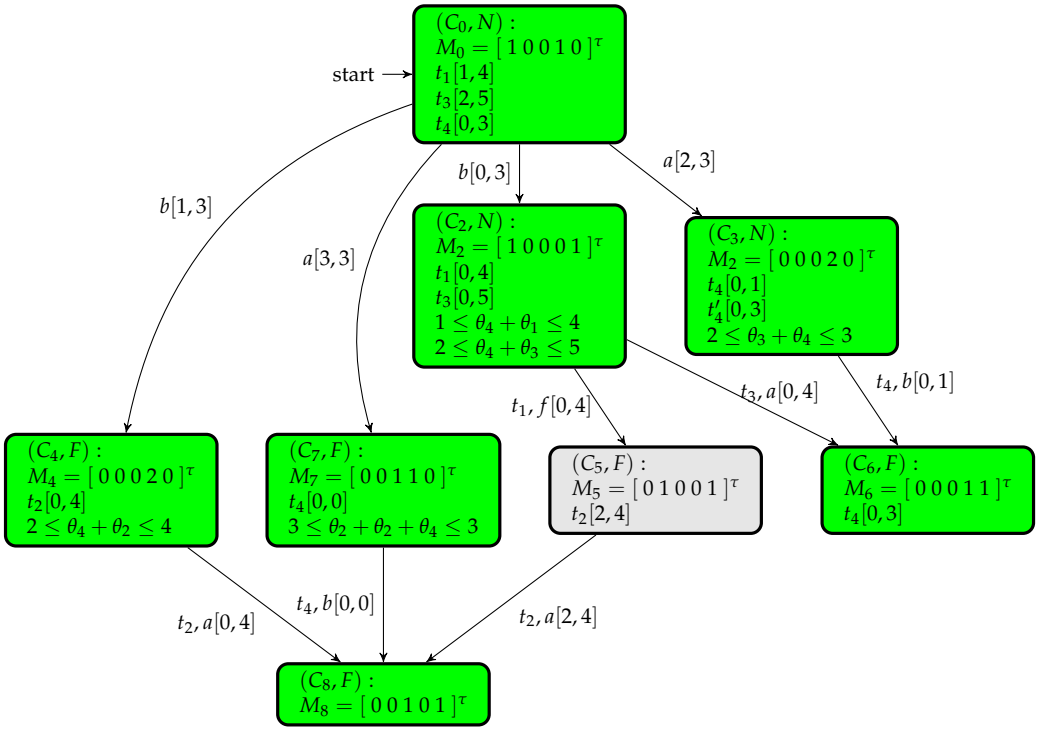
(b) Step 2

Figure 5.9: The construction of ASC-graph

- *Step 3 (cf. Figure 5.9(a)): 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,*



(a) Step 3

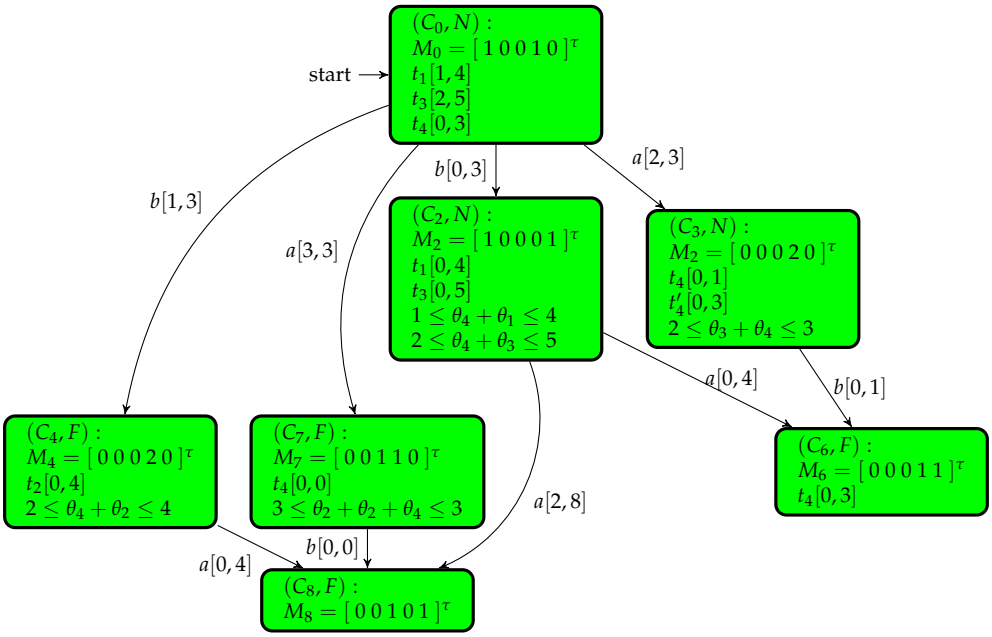


(b) Step 4

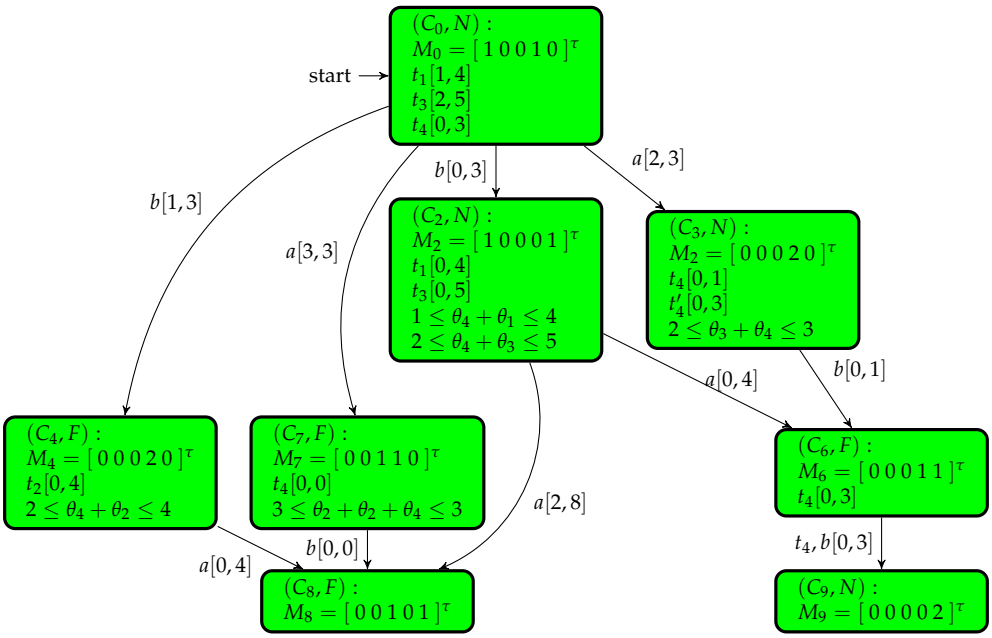
Figure 5.9: The construction of ASC-graph (Continued)

(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(b)): for each new ASC, do as Step 1 and 2.*
- *Step 5 (cf. Figure 5.9(a)): for each new pair of ASC reached after an observable transition,*



(a) Step 5

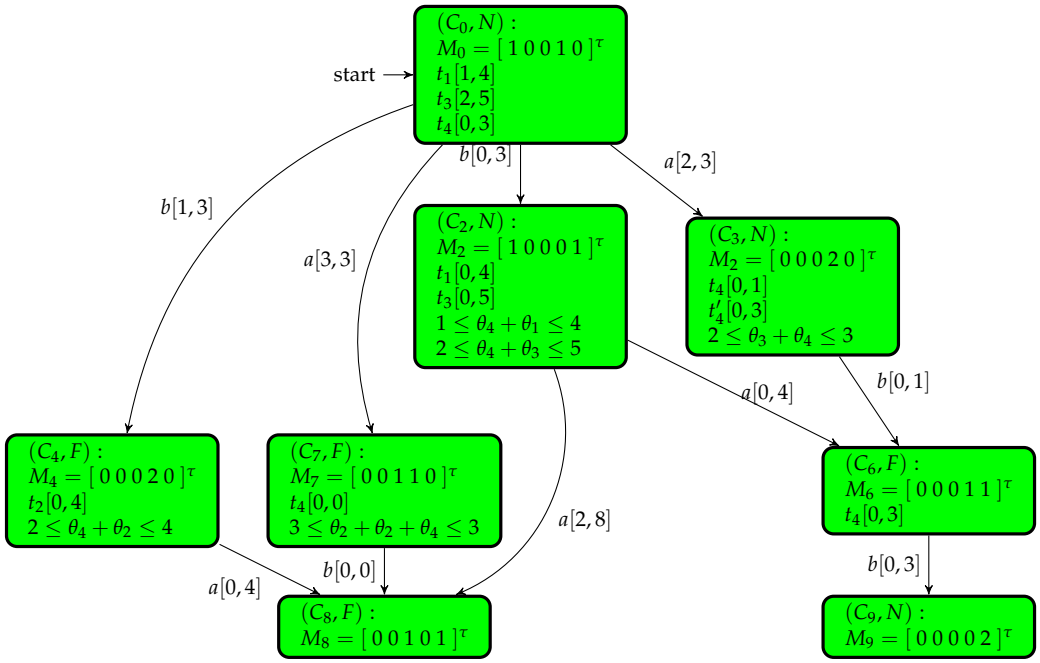


(b) Step 6

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

do as Step 3.

- *Step 6 (cf. Figure 5.9(b)): repeat Step 1 to obtain ASC* (C_9, N).
- *Step 7 (cf. Figure 5.9(a)): 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-graph is terminated.*



(a) Step 7: ASC-graph

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

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 inFigure 5.9(a), 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 $\xi : \mathcal{G} \times \Sigma_o \times \mathcal{I} \to \mathcal{G}$ 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:*

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

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

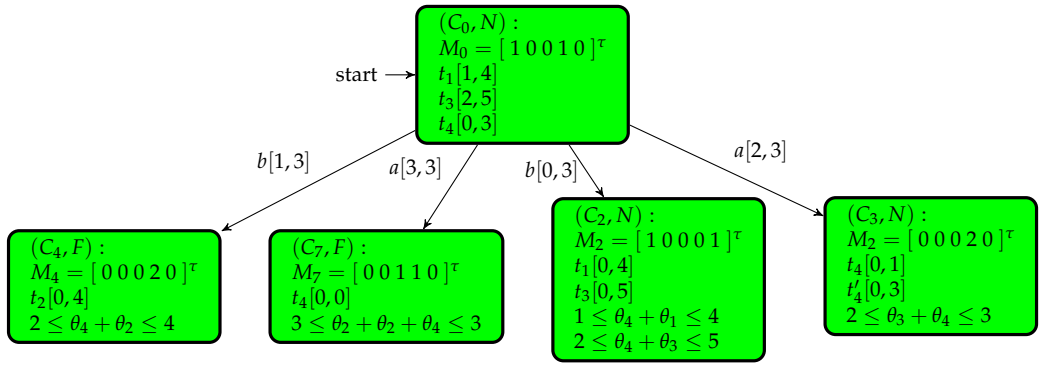
- *Step 3 (cf. Figure 5.10(b)): for each ASC in g_0, recompute its reachable ASCs upon each observable event $e \in \Sigma_o$ with the relative firing interval in $CIS(g_0, e)$.*
- *Step 4 (cf. Figure 5.10(c)): collect the ASCs reached after the same label (observable event and the corresponding firing interval) as a new ASC-set from g_0.*

5.2.5 ASC-Set Graph (ASG)

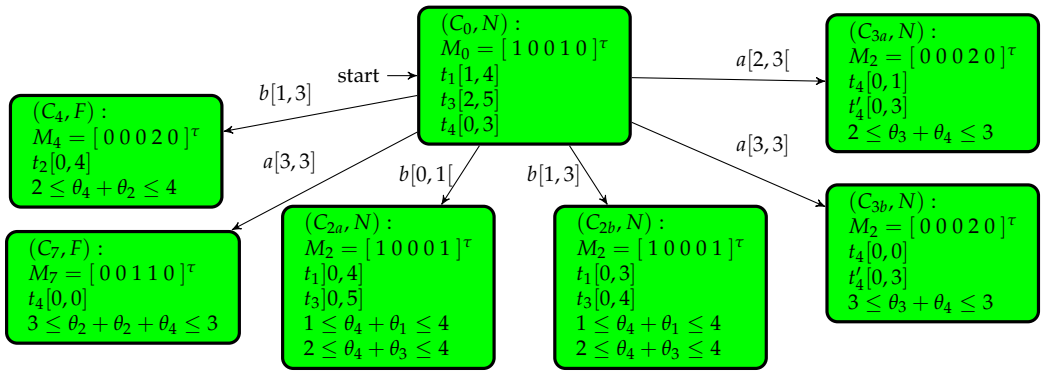
The augmented state class set graph (ASG) is introduced as a deterministic digraph which will serve as a basis to check diagnosability. Here the term "deterministic" means that, given an ASC-set and a candidate sequence, we can deduce with certainty which candidate ASCs the system will be possibly in. 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:

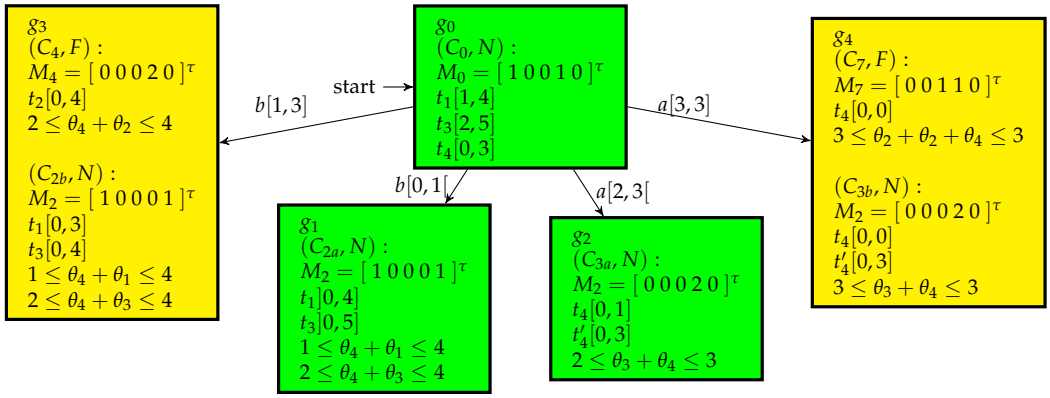
- $\mathcal{G} \subseteq 2^{\mathcal{N}_{ASC}}$ is the set of ASG nodes;
- $g_0 = \{x_0\} = \{(C_0, N)\}$ is the initial node;
- ξ is the transition mapping between ASC-sets;
- $\mathcal{R} \subseteq \mathcal{G} \times \Sigma_o \times \mathcal{I} \times \mathcal{G}$ is the set of ASG arcs: $\mathcal{R}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



(a) Step 1



(b) Step 3



(c) Step 4

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

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).*

Algorithm 9 Construction of the ASG

Input: the ASC-graph $(\mathcal{N}_{ASC}, \mathcal{A}, \gamma, x_0)$;
Output: the ASG;
$g_0 \leftarrow \{x_0\}$; ▷ initialization
$\mathcal{G}_{con} \leftarrow \{g_0\}$; ▷ $\mathcal{G}_{con}$ is the set of ASC-sets to be considered.
$\mathcal{G}_{vst} \leftarrow \emptyset$; ▷ $\mathcal{G}_{vst}$ is the set of ASC-sets that have been considered.
while $\mathcal{G}_{con} \neq \emptyset$ **do**
    pick a node $g \in \mathcal{G}_{con}$ s.t. $g \notin \mathcal{G}_{vst}$;
    for all $e \in CES(g)$ **do**
        $Y \leftarrow CIS(g, e)$;
        for all $i \in Y$ **do**
            $\mathcal{G}_{con} \leftarrow \mathcal{G}_{con} \cup \{\xi(g, e, i)\}$;
    $\mathcal{G}_{con} \leftarrow \mathcal{G}_{con} \backslash \{g\}$;
    $\mathcal{G}_{vst} \leftarrow \mathcal{G}_{vst} \cup \{g\}$;

- *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.*

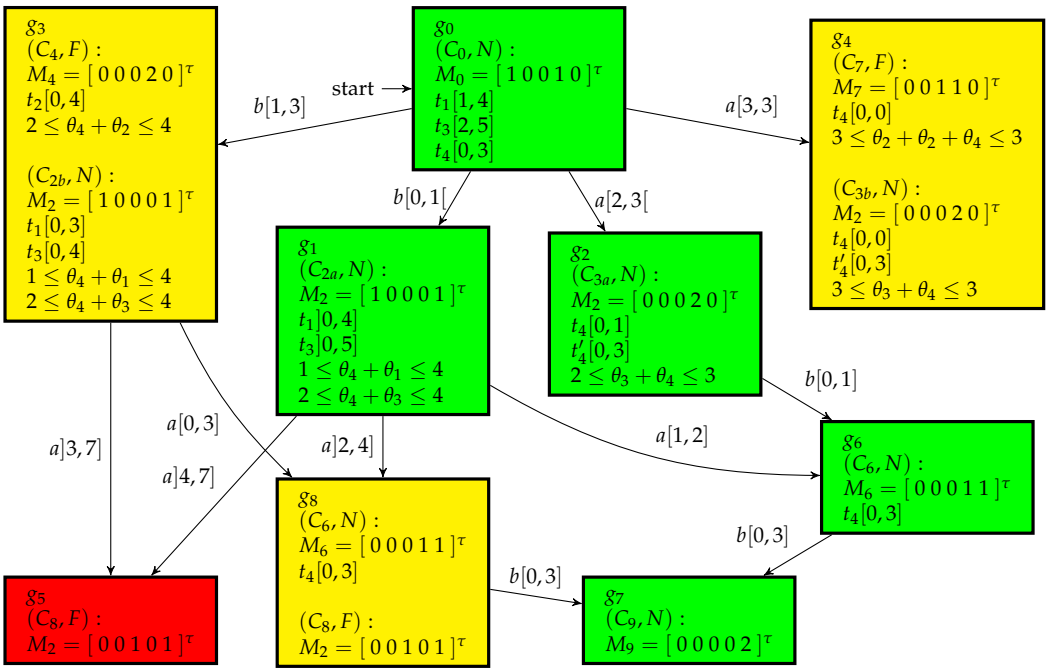


Figure 5.11: The ASG for Example 35

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.

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 [Sam+95]. 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 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)).

Condition 2: infinite sequence duration in certain cases

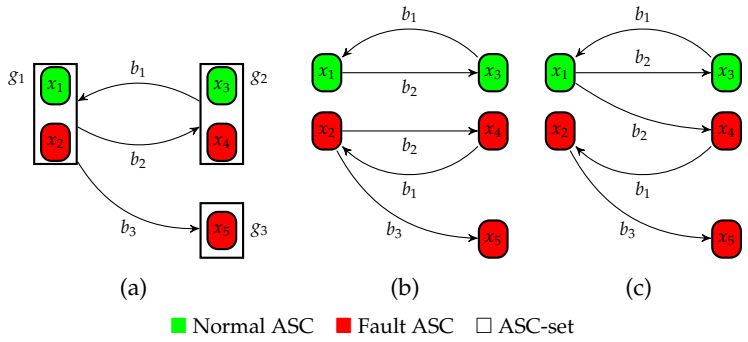


Figure 5.12: Illustration of indeterminate cycle

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

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

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

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

by $fdelay(g) =$
$max\{SD(\sigma) \mid \exists\, x \in g, \sigma' \in (T \backslash T_f)^*, t_f \in T_f, \sigma \in T^* T_o \text{ s.t. } x_0 \xrightarrow{\sigma' t_f \sigma} x\}$. 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', $tag(g') \in \{N, U\}$.
  - 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. □

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. □

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 [Sam+95].

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. □

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 Event-Recording Automata (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 *[Cas12] The TA A is said to be* deterministic *if*

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

Input: the ASC-graph;
Output: diagnosability of G and (if G is diagnosable) Δ_{min};
$g_0 = \{x_0\}$;
$\Delta_{min} = 0$;
$\mathcal{G}_{vst} \leftarrow \emptyset$;
$\mathcal{G}_{con} \leftarrow \{g_0\}$;
while $\mathcal{G}_{con} \neq \emptyset$ **do** pick a node $g \in \mathcal{G}_{con}$ with $g \notin \mathcal{G}_{vst}$;
    for all $e \in \Sigma_o$ **do**
        $\mathcal{I} \leftarrow CIS(g, e)$;
        for all $i \in \mathcal{I}$ **do**
            $g' \leftarrow \xi(g, e, i)$;
            if $tag(g') = U$ **then**
                if $\exists\, g'' \in \mathcal{G}_{vst}$ s.t. $g'' = g'$ **then**
                    if g' is in an indeterminate cycle **then**
                        return G is undiagnosable;
                if $(tag(g) = N) \wedge (mdelay(g, g') = +\infty)$ **then**
                    return G is undiagnosable;
                if $(tag(g) = U) \wedge (SD_{max}(g, g') = +\infty)$ **then**
                    return G is undiagnosable;
                if $tag(DS(g')) = U$ **then**
                    return G is undiagnosable;
                $\Delta_{min} \leftarrow max(\Delta_{min}, fdelay(g'))$;
            if $tag(g') = F$ **then**
                if $(tag(g) = N) \wedge (mdelay(g, g') = +\infty)$ **then**
                    return G is undiagnosable;
                if $(tag(g) = U) \wedge (SD_{max}(g, g') = +\infty)$ **then**
                    return G is undiagnosable;
                $\Delta_{min} \leftarrow max(\Delta_{min}, fdelay(g'))$;
    $\mathcal{G}_{con} \leftarrow \mathcal{G}_{con} \backslash \{g\}$;
    $\mathcal{G}_{vst} \leftarrow \mathcal{G}_{vst} \cup \{g\}$;
return G is Δ_{min}-diagnosable;

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, $g \wedge g' \equiv$ FALSE.*

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

Unlike an untimed (non-deterministic) automaton, from which a deterministic automaton observer can always be built, a TA is not always determinizable [Alu+94]. Moreover, the determinization of a TA is proved to be undecidable [Tri06].

However, a subclass of determinizable TA exists, which is called Event-Clock Automata (ECA) [Alu+94; Alu+99]. 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) [Alu+94] 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 TA 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 [Alu+94].

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.

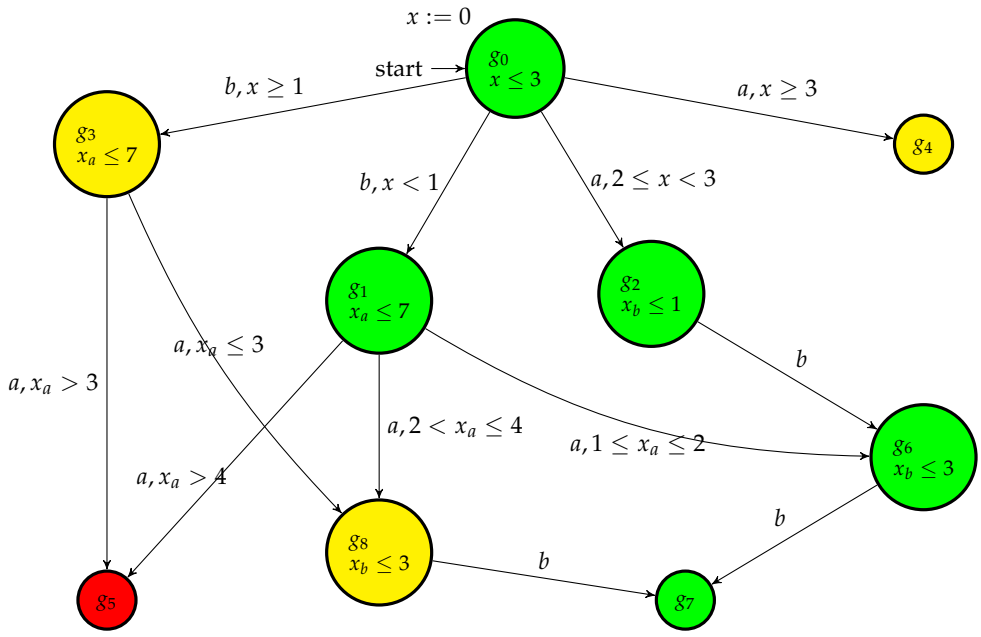


Figure 5.13: A language equivalent ERA for the ASG in Figure 5.11

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-centain) the system can be in and give the verdict pertinent to fault occurrences.

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

Input: An LDFS composed of (e_j, d_j);
Output: The current system state;
$q_0 \leftarrow N$; ▷ The system is normal (N) after the initialization.
$j \leftarrow 1$;
while the system is in operation **do**
    Wait for the input observable event e_j with its occurrence date d_j;
    $q_j \leftarrow$ the state from q_{j-1} upon e_j at date d_j;
    switch q_j **do**
        case N
            `assert`(No fault belonging to Σ_{F_i} has happened;)
        case U
            `assert`(A fault belonging to Σ_{F_i} has probably happened;)
        case F
            `assert`(A fault belonging to Σ_{F_i} has happened;)
        case $\emptyset$ ▷ The system arrives at a blocked faulty state "F".
            `assert`(A fault belonging to Σ_{F_i} has happened;)
    $j \leftarrow j + 1$;

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;*
- *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).*

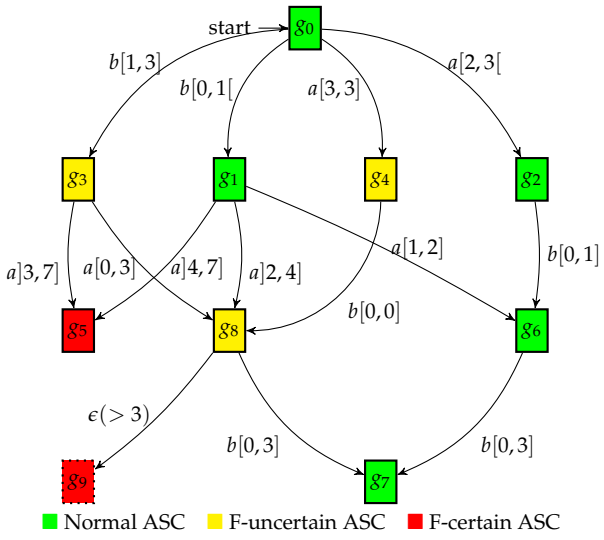


Figure 5.14: Diagnoser of the LTPN in Figure 5.7

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.

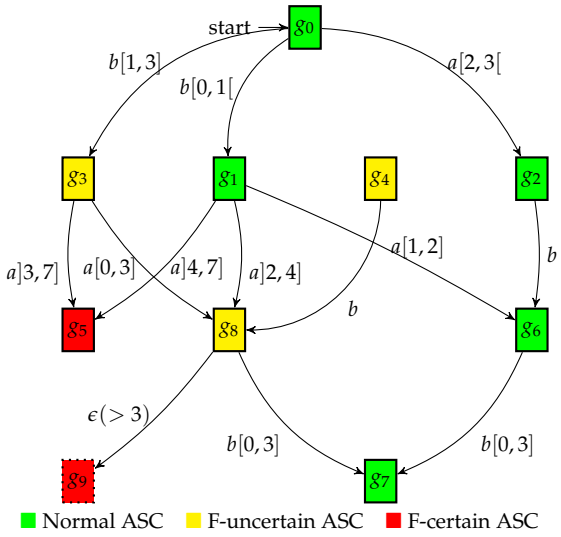


Figure 5.15: The MLTD of the LTPN in Figure 5.7

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 bs, 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* [Gar+04] has been reported to be more efficient than state classes in presenting TPN state space. Thus, we are interested in using zone 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 DESs modeled by LPNs. The on-the-fly and incremental technique is 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 representation, namely the *event-mapping matrix* and the *extended state equation* for LPNs, to characterize their static and dynamic features w.r.t. failure occurrence. The FM-graph and the FM-set tree are built simultaneously and on the fly to analyze diagnosability. Conventional 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 TIS technique. 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 from the ASG. Our study shows that for diagnosable LTPNs, the corresponding ASGs have some semantic equivalence with ERA.

- The study we have carried out shows the advantages of using an on-the-fly and incremental technique in tackling diagnosis problems: 1) the on-the-fly technique 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. 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 [Hua13].

- The algebraic representation of LPNs can simplify the description of system behavior, particularly in the event- and/or state-based analysis for LPNs, e.g., [Bas+12; RH09]. 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 algebraic representation and their corresponding tools.

- We are interested in introducing zone graph representation [Gar+04] into our analysis for LTPNs, since test results show that the zone graph approach is more efficient than the traditional state class graph [BM83] 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 computations 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.

BIBLIOGRAPHY

[AF98] A. Aghasaryan and E. Fabre. "Fault detection and diagnosis in distributed systems: an approach by partially stochastic Petri nets". In: *Discrete Event Dynamic Systems* 8.2 (1998), pp. 203–231.

[All83] J. F. Allen. "Maintaining knowledge about temporal intervals". In: *Communications of the ACM* 26.11 (Nov. 1983), pp. 832–843.

[AD94] R. Alur and D. L. Dill. "A theory of timed automata". In: *Theoretical Computer Science* 126.2 (1994), pp. 183–235.

[Alu+94] R. Alur, L. Fix, and T. Henzinger. "A determinizable class of timed automata". In: 126 (1994), pp. 183–235.

[Alu+99] R. Alur, L. Fix, and T. A. Henzinger. "Event-clock automata: a determinizable class of timed automata". In: *Theoretical Computer Science* 11.97 (1999), pp. 253–273.

[BK08] C. Baier and J. Katoen. *Principles of model checking*. MIT Press, 2008.

[Bas+10] F. Basile, P. Chiacchio, and G. De Tommasi. "Diagnosability of labeled Petri nets via integer linear programming". In: *Discrete Event Systems* 10.1 (2010), pp. 71–77.

[Bas+12] F. Basile, P. Chiacchio, and G. De Tommasi. "On K-diagnosability of Petri nets via integer linear programming". In: *Automatica* 48.9 (Sept. 2012), pp. 2047–2058.

[Bel+11] N. Belard, Y Pencolé, and M. Combacau. "A theory of meta-diagnosis: reasoning about diagnostic systems". In: *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two* (2011), pp. 731–737.

[Ben+03] A. Benveniste, E. Fabre, S. Haar, and C. Jard. "Diagnosis of asynchronous discrete-event systems: a net unfolding approach". In: *IEEE Transactions on Automatic Control* 48.5 (May 2003), pp. 714–727.

[Ber+05] B. Berard, F. Cassez, and S. Haddad. "Comparison of the expressiveness of timed automata and time Petri nets". In: *Formal Modeling and Analysis of Timed Systems*. Springer, 2005, pp. 211–225.

[BD91] B. Berthomieu and M. Diaz. "Modeling and verification of time dependent systems using time Petri nets". In: *IEEE Transactions on Software Engineering* 17.3 (1991), pp. 259–273.

[BM83] B. Berthomieu and M. Menasche. "An enumerative approach for analyzing time Petri nets". In: *Proceedings IFIP* (1983), pp. 41–46.

[Ber+04] B. Berthomieu, P.-O. Ribet, and F. Vernadat. "The tool TINA - Construction of abstract state spaces for Petri nets and time Petri nets". In: *International Journal of Production Research* 42.14 (July 2004), pp. 2741–2756.

[BJ13] R. K. Boel and G. Jiroveanu. "The on-line diagnosis of time Petri nets". In: *Control of Discrete-Event Systems*. Springer, 2013. Chap. 17, pp. 343–364.

[Bou+05] P. Bouyer, F. Chevalier, and D. D'Souza. "Fault diagnosis using timed automata". In: *Foundations of Software Science and Computational Structures*. Springer, 2005, pp. 219–233.

[Bou+06] P. Bouyer, S. Haddad, and P.-A. Reynier. "Extended timed automata and time Petri nets". In: *International Conference on Application of Concurrency to System Design*. 2006, pp. 91–100.

[BW94] B. Brandin and W. Wonham. "Supervisory control of timed discrete-event systems". In: *IEEE Transactions on Automatic Control* 39.2 (1994), pp. 329–342.

[Cab+10] M. P. Cabasino, A. Giua, and C. Seatzu. "Fault detection for discrete event systems using Petri nets with unobservable transitions". In: *Automatica* 46.9 (Sept. 2010), pp. 1531–1539.

[Cab+11] M. P. Cabasino, L. Contini, A. Giua, C. Seatzu, and A. Solinas. "A software platform for the integration of discrete event systems tools". In: *IEEE International Conference on Automation Science and Engineering*. IEEE, Aug. 2011, pp. 45–51.

[Cab+12a] M. P. Cabasino, A. Giua, L. Marcias, and C. Seatzu. "A comparison among tools for the diagnosability of discrete event systems". In: *IEEE International Conference on Automation Science and Engineering*. IEEE, Aug. 2012, pp. 218–223.

[Cab+12b] M. P. Cabasino, A. Giua, S. Lafortune, and C. Seatzu. "A new approach for diagnosability analysis of Petri nets using verifier nets". In: *IEEE Transactions on Automatic Control* 57.12 (Dec. 2012), pp. 3104–3117.

[Cab+13a] M. P. Cabasino, A. Giua, A. Paoli, and C. Seatzu. "Decentralized diagnosis of discrete-event systems using labeled Petri nets". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2013), pp. 1–9.

[Cab+13b] M. P. Cabasino, S. Lafortune, and C. Seatzu. "Optimal sensor selection for ensuring diagnosability in labeled Petri nets". In: *Automatica* 49.8 (Aug. 2013), pp. 2373–2383.

[Car+12] L. Carvalho, M. Moreira, J. C. Basilio, and S. Lafortune. "Robust diagnosis of discrete-event systems against permanent loss of observations". In: *Automatica* 49.1 (Jan. 2012), pp. 223–231.

[CL07] C. G. Cassandras and S. Lafortune. *Introduction to discrete event systems*. 2nd ed. Springer, 2007.

[Cas10] F. Cassez. "Dynamic observers for fault diagnosis of timed systems". In: *49th IEEE Conference on Decision and Control* (2010), pp. 4359–4364.

[Cas12] F. Cassez. "The complexity of codiagnosability for discrete event and timed systems". In: *IEEE Transactions on Automatic Control* 57.7 (2012), pp. 1752 – 1764.

[CT08] F. Cassez and S. Tripakis. "Fault diagnosis with static and dynamic observers". In: *Fundamenta Informaticae* 88.4 (2008), pp. 497–540.

[Cim+03] A. Cimatti, C. Pecheur, and R. Cavada. "Formal verification of diagnosability via symbolic model checking". In: *International Joint Conference on Artificial Intelligence*. 2003, pp. 363–369.

[CP13] G. Ćirović and D. Pamučar. "Decision support model for prioritizing railway level crossings for safety improvements: Application of the adaptive neuro-fuzzy system". In: *Expert Systems with Applications* 40.6 (2013), pp. 2208–2223.

[Cla+94] E. Clarke, O Grumberg, and D. Long. "Model checking and abstraction". In: *ACM Transactions on Programming Languages and Systems* (1994), pp. 343–354.

[Con+04] O. Contant, S. Lafortune, and D. Teneketzis. "Diagnosis of intermittent faults". In: *Discrete Event Dynamic Systems* 14.2 (Apr. 2004), pp. 171–202.

[Con+06] O. Contant, S. Lafortune, and D. Teneketzis. "Diagnosability of discrete event systems with modular structure". In: *Discrete Event Dynamic Systems* 16.1 (Feb. 2006), pp. 9–37.

[D'A+99] S D'Angelo, C. Metra, and G. Sechi. "Transient and permanent fault diagnosis for FPGA-based TMR systems". In: *International Symposium on Defect and Fault Tolerance in VLSI Systems* (1999), pp. 330 –338.

[Deb+00] R. Debouk, S. Lafortune, and D. Teneketzis. "Coordinated decentralized protocols for failure diagnosis of discrete event systems". In: *Discrete Event Dynamic Systems* 10.1-2 (2000), pp. 33–86.

[Deb+02] R. Debouk, S. Lafortune, and D. Teneketzis. "On an optimization problem in sensor selection". In: *Discrete Event Dynamic Systems* 12.4 (2002), pp. 417–445.

[Dia01] M. Diaz. *Les réseaux de Petri - modèles fondamentaux*. Hermés, Paris, 2001.

[Dot+09] M. Dotoli, M. Fanti, A. Mangini, and W. Ukovich. "On-line fault detection in discrete event systems by Petri nets and integer linear programming". In: *Automatica* 45.11 (2009), pp. 1–23.

[DT89] J. Dugan and K. Trivedi. "Coverage modeling for dependability analysis of fault-tolerant systems". In: *IEEE Transactions on Computers* 38.6 (1989), pp. 775–787.

[Gar+04] G. Gardey, O. F. Roux, and O. H. Roux. "Using zone graph method for computing the state space of a time Petri net". In: *Formal modeling and analysis of timed systems*. Springer, 2004, pp. 246–259.

[Gar+05] G. Gardey, D. Lime, M. Magnin, and O. H. Roux. "Romeo: A Tool for analyzing time Petri nets". In: *Computer Aided Verification*. Springer, 2005, pp. 418–423.

[GL07] S. Genc and S. Lafortune. "Distributed diagnosis of place-bordered Petri nets". In: *IEEE Transactions on Automation Science and Engineering* 4.2 (Apr. 2007), pp. 206–219.

[Gha09] M. Ghazel. "Using stochastic Petri nets for level-crossing collision risk assessment". In: *IEEE Transactions on Intelligent Transportation Systems* 10.4 (2009), pp. 668–677.

[GE07] M. Ghazel and E.-M. El Koursi. "Automatic level crossings: From informal functional requirements' specifications to the control model design". In: *IEEE International Conference on System of Systems Engineering 2007*. IEEE. 2007, pp. 1–6.

[GEK14] M. Ghazel and E.-M. El-Koursi. "Two-half-barrier level crossings versus four-half-barrier level crossings: A comparative risk analysis study". In: *IEEE Transactions on Intelligent Transportation Systems* (Jan. 2014).

[Gha+09] M. Ghazel, A. Toguyéni, and P. Yim. "State observer for DES under partial observation with time Petri nets". In: *Discrete Event Dynamic Systems* 19.2 (2009), pp. 137–165.

[Gha+12] M. Ghazel, F. Peres, A. Belhaj Alaya, and A. Jemai. "A DBMS framework for diagnosability analysis of discrete event systems". In: *42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. Jan. 2012.

[Giu07] A. Giua. *A benchmark for diagnosis*. 2007.

[Giu+07] A. Giua, C. Seatzu, and D. Corona. "Marking estimation of Petri nets with silent transitions". In: *IEEE Transactions on Automatic Control* 52.9 (Sept. 2007), pp. 1695–1699.

[Gra+10] B. Grabiec, L. Traonouez, and C. Jard. "Diagnosis using unfoldings of parametric time Petri nets". In: *Formal Modeling and Analysis of Timed Systems*. Springer Berlin Heidelberg, 2010, pp. 137–151.

[Gra09] A. Grastien. "Symbolic testing of diagnosability". In: *International Workshop on Principles of Diagnosis*. 2009, pp. 131–138.

[Haa09] S. Haar. "Qualitative diagnosability of labeled Petri nets revisited". In: *Proceedings of the 48h IEEE Conference on Decision and Control held jointly with the 28th Chinese Control Conference*. IEEE, Dec. 2009, pp. 1248–1253.

[Has03] S. Hashtrudi Zad. "Fault diagnosis in discrete-event systems: Framework and model reduction". In: *IEEE Transactions on Automatic Control* 48.7 (2003), pp. 1199–1212.

[Has05] S. Hashtrudi Zad. "Fault diagnosis in discrete-event systems: incorporating timing information". In: *IEEE Transactions on Automatic Control* 50.7 (2005), pp. 1010–1015.

[Hen+92] T. Henzinger, Z. Manna, and A. Pnueli. "Timed transition systems". In: *Real-Time: Theory in Practice* 3096 (1992).

[HC94] L. E. Holloway and S. Chand. "Time templates for discrete event fault monitoring in manufacturing systems". In: *American Control Conference*. 1994, pp. 701–706.

[Hua13] Y. Huang. "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, 2013.

[Hua+04] Z. Huang, S. Bhattacharyya, V. Chandra, S. Jiang, and R. Kumar. "Diagnosis of discrete event systems in rules-based model using first-order linear temporal logic". In: *American Control Conference*. Vol. 6. 2004, pp. 5114–5119.

[JK04] S. Jiang and R. Kumar. "Failure diagnosis of discrete-event systems with linear-time temporal logic specifications". In: *IEEE Transactions on Automatic Control* 49.6 (2004), pp. 934–945.

[Jia+01] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. "A polynomial algorithm for testing diagnosability of discrete-event systems". In: *IEEE Transactions on Automatic Control* 46.8 (2001), pp. 1318–1321.

[Jia+03a] S. Jiang, R. Kumar, and H. Garcia. "Diagnosis of repeated/intermittent failures in discrete event systems". In: *IEEE Transactions on Robotics and Automation* 19.2 (2003), pp. 310–323.

[Jia+03b] S. Jiang, R. Kumar, and H. Garcia. "Optimal sensor selection for discrete-event systems with partial observation". In: *IEEE Transactions on Automatic Control* 48.3 (2003), pp. 369–381.

[Jia+06] S. Jiang, R. Kumar, and S. Member. "Diagnosis of repeated failures for discrete event systems with linear-time temporal-logic specifications". In: *IEEE Transactions on Automation Science and Engineering* 3.1 (2006), pp. 47–59.

[JB05] G. Jiroveanu and R. K. Boel. "Distributed diagnosis for Petri nets models with unobservable interactions via common places". In: *Proceedings of the 44th IEEE Conference on Decision and Control*. IEEE, 2005, pp. 6305–6310.

[JB10] G. Jiroveanu and R. K. Boel. "The diagnosability of Petri net models using minimal explanations". In: *IEEE Transactions on Automatic Control* 55.7 (2010), pp. 1663–1668.

[Jir+06] G. Jiroveanu, R. K. Boel, and B. De Schutter. "Fault diagnosis for time Petri nets". In: *8th International Workshop on Discrete Event Systems*. IEEE, 2006, pp. 313–318.

[KG09] L. Khoudour and M. Ghazel. "Towards safer level crossings: existing recommendations, new applicable technologies and a proposed simulation model". In: *European transport research review* (2009).

[Koe+04] S. Koenig, M. Likhachev, Y. Liu, and D. Furcy. "Incremental heuristic search in AI". In: *AI Magazine* 25.2 (2004), pp. 99–112.

[Laf00] S. Lafortune. *UMDES-Lib software library*. http://www.eecs.umich.edu/umdes/toolboxes.html. 2000.

[Laf+05] S. Lafortune, Y. Wang, and T.-S. Yoo. "Diagnostic décentralisé des systèmes à événements discrets". In: *Journal Européen des Systèmes Automatisés* 39.1-3 (2005), pp. 95–110.

[Lai+08] S. Lai, D. Nessi, M. P. Cabasino, A. Giua, and C. Seatzu. "A comparison between two diagnostic tools based on automata and Petri nets". In: *International Workshop on Discrete Event Systems*. IEEE, 2008, pp. 144–149.

[LD07] D. Lefebvre and C. Delherm. "Diagnosis of DES with Petri net models". In: *IEEE Transactions on Automation Science and Engineering* 4.1 (2007), pp. 114–118.

[LS85] N. G. Leveson and J. Stolzy. "Analyzing safety and fault tolerance using time Petri nets". In: *Formal Methods and Software Development* 186 (1985).

[Lin94] F. Lin. "Diagnosability of discrete event systems and its applications". In: *Discrete Event Dynamic Systems* 4.2 (1994), pp. 197–212.

[Liu+12] B. Liu, M. Ghazel, and A. Toguyéni. "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*. 2012.

[Liu+13] B. Liu, M. Ghazel, and A. Toguyéni. "Évaluation à la volée de la diagnosticabilité des systèmes à événements discrets temporisés". In: *Journal Européen des Systèmes Automatisés, Édition spéciale MSR'13, Modélisation des Systèmes Réactifs* 47.1-2-3 (2013), pp. 227–242.

[Liu+14a] B. Liu, M. Ghazel, and A. Toguyéni. "Diagnosis of labeled time Petri nets using time interval splitting". In: *The 19th World Congress of the International Federation of Automatic Control*. Accepted, 2014.

[Liu+14b] B. Liu, M. Ghazel, and A. Toguyéni. "Toward an efficient approach for diagnosability analysis of DES modeled by labeled Petri nets". In: *13th European Control Conference*. 2014.

[Mad+10] A. Madalinski, F. Nouioua, and P. Dague. "Diagnosability verification with Petri net unfoldings". In: *International Journal of Knowledge-Based and Intelligent Engineering Systems* 2 (2010), pp. 49–55.

[MP95] K. L. McMillan and D. K. Probst. "A technique of state space search based on unfolding". In: *Formal Methods in System Design* 6.1 (Jan. 1995), pp. 45–65.

[Mek+12] A. Mekki, M. Ghazel, and A. Toguyeni. "Validation of a new functional design of automatic protection systems at level crossings with model-checking techniques". In: *Intelligent Transportation Systems, IEEE Transactions on* 13.2 (2012), pp. 714–723.

[Mer74] P. Merlin. "A study of the recoverability of computing systems". PhD thesis. University of California, 1974.

[Mor+07] G. Morel, P. Valckenaers, and J. M. Faure. "Manufacturing plant control challenges and issues". In: *Control Engineering Practice* (2007).

[Mur89] T. Murata. "Petri nets: Properties, analysis and applications". In: *Proceedings of the IEEE* 77.4 (Apr. 1989), pp. 541–580.

[Nel90] V. Nelson. "Fault-tolerant computing: fundamental concepts". In: *Computer* 23.7 (July 1990), pp. 19–25.

[PH00] D. Pandalai and L. Holloway. "Template languages for fault monitoring of timed discrete event processes". In: *IEEE Transactions on Automatic Control* 45.5 (2000), pp. 868–882.

[PL05] A. Paoli and S. Lafortune. "Safe diagnosability for fault-tolerant supervision of discrete-event systems". In: *Automatica* 41.8 (Aug. 2005), pp. 1335–1347.

[Pen09] Y Pencolé. "A chronicle-based diagnosability approach for discrete timed-event systems: Application to web-services". In: *Journal of Universal Computer Science* 15.17 (2009), pp. 3246–3272.

[PG13] F. Peres and M. Ghazel. *A μ-calculus framework for the diagnosability of discrete event systems*. Tech. rep. IFSTTAR, 2013.

[Pet62] C. A. Petri. "Kommunikation mit automaten". PhD thesis. Institut fur Instrumentelle Mathematik, Schriften des IIM, 1962.

[Pro02] G. Provan. "On the diagnosability of decentralized, timed discrete event systems". In: *Proceedings of the 41st IEEE Conference on Desision and Control*. 2002, pp. 405–410.

[QK06] W. Qiu and R. Kumar. "Decentralized failure diagnosis of discrete event systems". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 36.2 (2006), pp. 384–395.

[Qiu+09] W. Qiu, Q. Wen, and R. Kumar. "Decentralized diagnosis of event-driven systems for safely reacting to failures". In: *IEEE Transactions on Automation Science and Engineering* 6.2 (2009), pp. 362–366.

[Ram74] C Ramchandani. "Analysis of asynchronous concurrent systems by timed Petri nets". PhD thesis. Massachusetts Institute of Technology, 1974.

[RT07] A. Ramirez-Trevino. "Online fault diagnosis of discrete event systems. A Petri net-based approach". In: *IEEE Transactions on Automation Science and Engineering* 4.1 (2007), pp. 31–39.

[RT+12] A. Ramirez-Trevino, E. Ruiz-Beltran, J. Arámburo-lizárraga, and E. López-mellado. "Structural diagnosability of DES and design of reduced Petri net diagnosers". In: *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 42.2 (2012), pp. 416–429.

[RFF13] RFF. *Les passages à niveau*. http://www.rff.fr/fr/le-reseau/le-reseau-en-projets/la-modernisation-du-reseau/les-passages-a-niveau. 2013.

[Rib+07] P Ribot, Y Pencolé, and M. Combacau. "Characterization of requirements and costs for the diagnosability of distributed discrete event systems". In: *5th Workshop on Advanced Control and Diagnosis* (2007).

[Ric+06] L. Ricker, S. Lafortune, and S. Genc. "DESUMA: A tool integrating GIDDES and UMDES". In: *International Workshop on Discrete Event Systems*. IEEE, 2006, pp. 392–393.

[RH09] Y. Ru and C. N. Hadjicostis. "Fault diagnosis in discrete event systems modeled by partially observed Petri nets". In: *Discrete Event Dynamic Systems* 19.4 (June 2009), pp. 551–575.

[RH10] Y. Ru and C. N. Hadjicostis. "Sensor selection for structural observability in discrete event systems modeled by Petri nets". In: *IEEE Transactions on Automatic Control* 55.7 (2010), pp. 1–14.

[Rus12] A. Rushton. *A directed graph container*. http://www.andyrushton.co.uk/programming/stlplus-library-collection. 2012.

[Sam+95] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. "Diagnosability of discrete-event systems". In: *IEEE Transactions on Automatic Control* 40.9 (1995), pp. 1555–1575.

[Sam+96] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. "Failure diagnosis using discrete-event models". In: *IEEE Transactions on Control Systems Technology* 4.2 (Mar. 1996), pp. 105–124.

[Sam+98] M. Sampath, S. Lafortune, and D. Teneketzis. "Active diagnosis of discrete-event systems". In: *IEEE Transactions on Automatic Control* 43.7 (July 1998), pp. 908–929.

[Sch+01] F. Schiller, J. Schröder, and J. Lunze. "Diagnosis of transient faults in quantised systems". In: *Engineering Applications of Artificial Intelligence* 14.4 (2001), pp. 519–536.

[SE05] S. Schwoon and J. Esparza. *A note on on-the-fly verification algorithms*. Springer, 2005, pp. 174–190.

[SA+10] Z. Simeu-Abazi, M. Di Mascolo, and M. Knotek. "Fault diagnosis for discrete event systems: Modelling and verification". In: *Reliability Engineering and System Safety* 95.4 (Apr. 2010), pp. 369–378.

[Sol+07] S. Soldani, M. Combacau, and A. Subias. "Intermittent fault diagnosis: A diagnoser derived from the normal behavior". In: *Proceedings of the 18th International Workshop on Principles of Diagnosis*. 2007, pp. 391–398.

[Sta+06] G. Stamp, Y. Ong, R. Kumar, and C. Zhou. "DECADA: Tool for discrete-event control and diagnosis analysis". In: *International Workshop on Discrete Event Systems*. IEEE, 2006, pp. 396–397.

[SM96] J. Sztipanovits and A. Misra. "Diagnosis of discrete event systems using ordered binary decision diagrams". In: *Seventh International Workshop on Principles of Diagnosis*. 1996.

[Tri02] S. Tripakis. "Fault diagnosis for timed automata". In: *Formal Techniques in Real-Time and Fault-Tolerant Systems*. Springer. 2002, pp. 205–221.

[Tri06] S. Tripakis. "Folk theorems on the determinization and minimization of timed automata". In: *Information Processing Letters* 99.6 (Sept. 2006), pp. 222–226.

[Ush+98] T. Ushio, I. Onishi, and K. Okuda. "Fault detection based on Petri net models with faulty behaviors". In: *IEEE International Conference on Systems, Man, and Cybernetics*. Vol. 1. 1998, pp. 113–118.

[WG11] W. Wang and A. Girard. "On codiagnosability and coobservability with dynamic observations". In: *IEEE Transactions on Automatic Control* 56.7 (2011), pp. 1551–1566.

[Wan+11] X. Wang, C. Mahulea, J Júlvez, and M. Silva. "On state estimation of timed choice-free Petri nets". In: *Proceedings of the 18th IFAC World Congress*. 2011, pp. 8687–8692.

[Wan+13] X. Wang, C. Mahulea, and M. Silva. "Fault diagnosis graph of time Petri nets". In: *European Control Conference*. 2013, pp. 2459–2464.

[Wan+04] Y. Wang, T.-S. Yoo, and S. Lafortune. "New results on decentralized diagnosis of discrete event systems". In: *Proceedings of 2004 Annual* (2004).

[Wen+05] Y. Wen, C. Li, and M. Jeng. "A polynomial algorithm for checking diagnosability of Petri nets". In: *IEEE International Conference on Systems, Man and Cybernetics*. Vol. 3. IEEE, 2005, pp. 2542–2547.

[Wen+06] Y. Wen, C. Li, and M. Jeng. "Diagnosability enhancement of discrete event systems". In: *IEEE International Conference on Systems, Man and Cybernetics*. Vol. 5. 2006, pp. 4096–4101.

[Xu+10] S. Xu, S. Jiang, and R. Kumar. "Diagnosis of dense-time systems under event and timing masks". In: *IEEE Transactions on Automation Science and Engineering* 7.4 (Oct. 2010), pp. 870–878.

[XZ04] F. Xue and D.-Z. Zheng. "Diagnosability for discrete event systems based on Petri net language". In: *8th Control, Automation, Robotics and Vision Conference*. Vol. 3. IEEE, 2004, pp. 2111–2116.

[YG04] T.-S. Yoo and H. Garcia. "Event diagnosis of discrete-event systems with uniformly and nonuniformly bounded diagnosis delays". In: *Proceedings of 2004 American Control Conference*. Vol. 6. 2004, pp. 5102–5107.

[YL02a] T.-S. Yoo and S. Lafortune. "NP-completeness of sensor selection problems arising in partially observed discrete-event systems". In: *IEEE Transactions on Automatic Control* 47.9 (2002), pp. 1495–1499.

[YL02b] T.-S. Yoo and S. Lafortune. "Polynomial-time verification of diagnosability of partially observed discrete-event systems". In: *IEEE Transactions on Automatic Control* 47.9 (Sept. 2002), pp. 1491–1495.

[ZL13] J. Zaytoon and S. Lafortune. "Overview of fault diagnosis methods for discrete event systems". In: *Annual Reviews in Control* 37.2 (Dec. 2013), pp. 308–320.

[ZF06] R. Zemouri and J. M. Faure. "Diagnosis of discrete event system by stochastic timed automata". In: *IEEE International Conference on Control Applications*. IEEE. 2006, pp. 1861–1866.

APPENDIX A

LITERATURE ON DES-BASED DIAGNOSIS

Reference	Automata	TA	general PN	LPN	TPN	Diagnosability	K/Δ-diagnosability	Online diagnosis	Unfolding	ILP	Decentralized/Distributed	Other techniques
[Bas+12]			•	•			•			•		
[Ben+03]			•					•	•			
[BJ13]					•			•				
[Cab+10]			•					•				
[Cab+12b]				•		•	•	•		•		
[Cab+13a]				•		•					•	
[Con+04]	•					•						
[Con+06]	•					•					•	
[Deb+00]	•					•					•	
[Dot+09]			•					•		•		
[GL07]				•				•			•	
[Gha+09]					•			•				
[Gra+10]					•			•	•			
[Haa09]				•		•			•			
[Has03]	•					•						
[Has05]	•							•				

Table A.1 continued

Reference	Automata	TA	general PN	LPN	TPN	Diagnosability	K/Δ-diagnosability	Online diagnosis	Unfolding	ILP	Decentralized/Distributed	Other techniques
[HC94]		•						•				
[Jia+01]	•					•						
[Jia+03a]							•					
[JK04]												
[Jia+06]		•				•						
[JB05]			•					•			•	
[JB10]			•			•						
[Laf+05]	•					•					•	
[Lai+08]			•					•				
[LD07]			•					•				
[Lin94]								•				
[Mad+10]				•		•			•			
[PH00]								•			•	
[PL05]	•					•						
[Pen09]					•	•		•				
[Pro02]						•					•	
[QK06]	•					•					•	
[Qiu+09]	•					•						
[RT07]								•		•		
[RT+12]						•				•		
[RH09]				•				•				
[Sam+95]	•					•						
[Sam+96]	•					•						
[Sam+98]	•											
[SA+10]		•						•				
[SM96]												
[Tri02]		•				•	•	•				
[Ush+98]			•			•		•				
[WG11]	•					•						
[Wen+05]			•			•				•		
[Xu+10]		•				•						
[XZ04]			•			•						

Table A.1 continued

Reference	Automata	TA	general PN	LPN	TPN	Diagnosability	K/Δ-diagnosability	Online diagnosis	Unfolding	ILP	Decentralized/Distributed	Other techniques
[YL02a]	•					•						
[YG04]	•						•					

APPENDIX B

DEVELOPMENT OF THE LC BENCHMARK

A level crossing (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.

B.1 An Overview on the Case Study

In France, there are more than 18,000 LCs. Every day they are traversed by an average of 16,000,000 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 [Gha09]. On average, 400 people are killed every year in the European Union (EU) [CP13]. In France, 100 collisions happened and 33 people were killed in 2012 [RFF13]. Therefore, safety of LCs always attracts great attention in railway operation and also in the research area [GE07; KG09; Mek+12].

In this section, we apply our diagnosis techniques to an LC system. We consider a bidirectional multi-track LC (or unidirectional single-track LC 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 B.1 [GEK14]. The LC global dynamics can be depicted while considering three subsystems, namely the railway traffic, the LC controller and the barriers, which will be detailed in Appendix B.2.

The logic of a single-track 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:

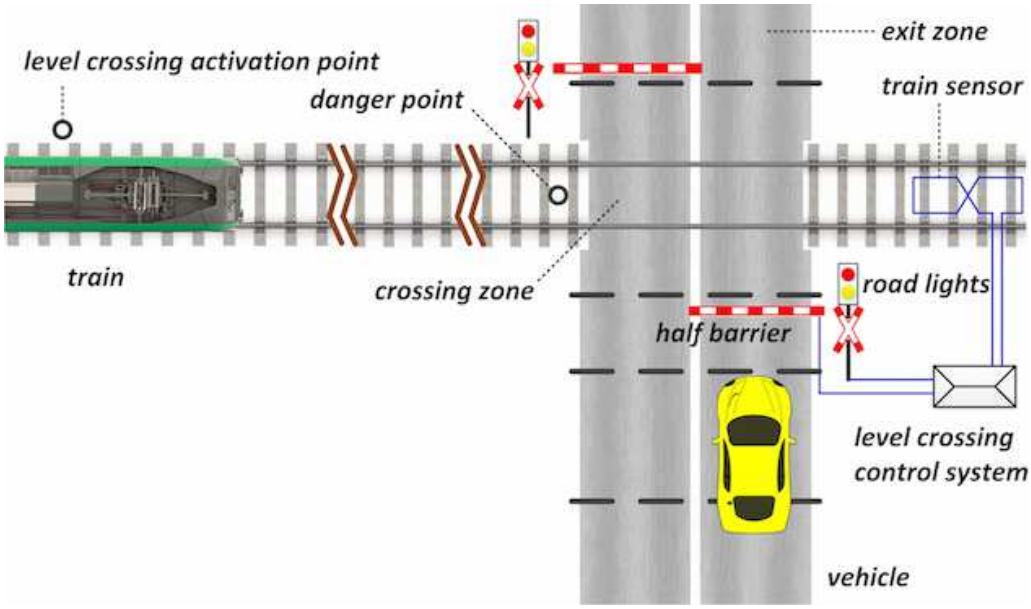


Figure B.1: The construction of a single-track track LC system

- 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.

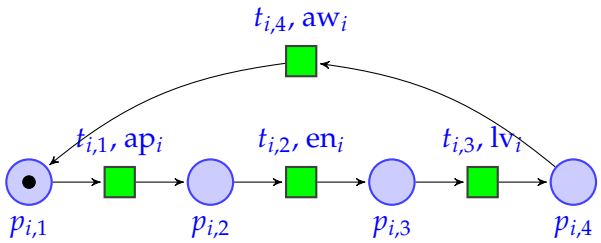
B.2 Modeling of the LC Subsystems

This section presents the modeling of the LC subsystems, namely the railway traffic, the LC controller and the barriers. The n-track LC benchmark will be built based on the single-track LC model [LS85] with some modifications. We will give their corresponding LPN models and operating principles. Note that, as the first step, only the normal behavior will be modeled; some failures will be introduced afterward.

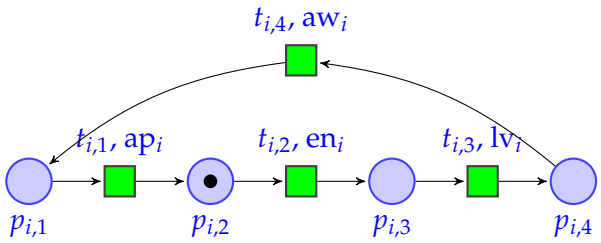
B.2.1 Railway Traffic

Railway traffic can be modeled as an LPN composed of 4 places and 4 transitions as shown in Figure B.2, where:

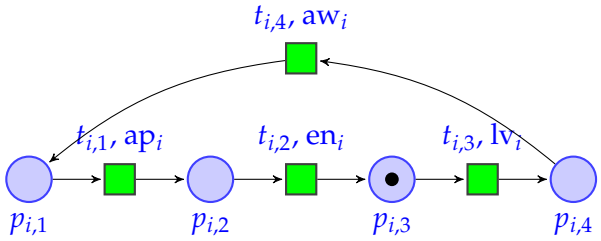
- Marked place $p_{i,1}$ (here the subscript i denotes the track index) denotes that a train is approaching the LC, as shown in Figure B.2(a);
- Marked place $p_{i,2}$ denotes that the train has come into the section before the LC, which can be detected by sensor ($t_{i,1}$, ap_i) (here "ap" denotes "approaching"), as shown in Figure B.2(b);



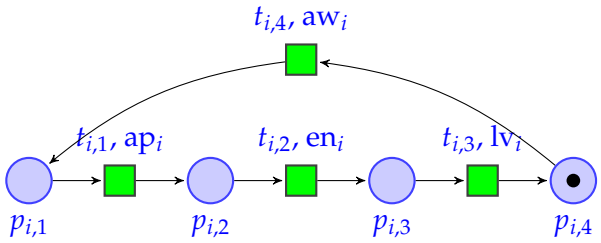
(a) the train is approaching the LC



(b) the train is before the LC



(c) the train is within the LC



(d) the train has left the LC

Figure B.2: The LPN model for a train passing an LC

- Marked place $p_{i,3}$ denotes that the train has entered the LC, which can be detected by sensor $(t_{i,2}, \mathrm{en}_i)$ (here "en" denotes "entering"), as shown in Figure B.2(c);
- Marked place $p_{i,4}$ denotes that the train has left the LC, which can be detected by sensor $(t_{i,3}, \mathrm{lv}_i)$ (here "lv" denotes "leaving"), as shown in Figure B.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 $(t_{i,4}, \mathrm{aw}_4)$ (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.

B.2.2 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 B.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.

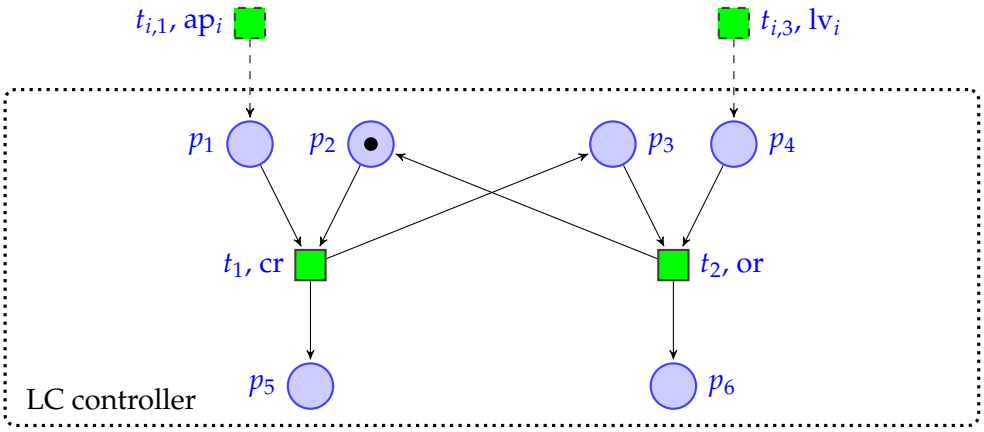
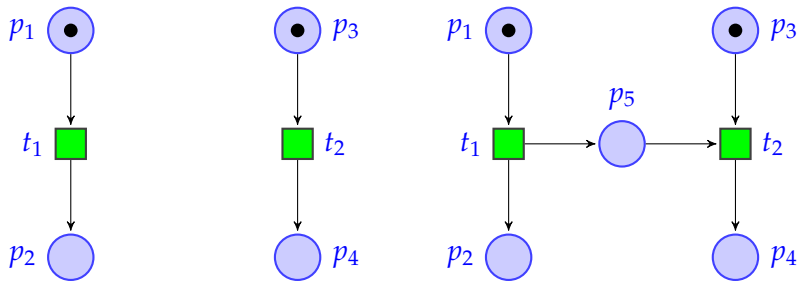


Figure B.3: The LPN model for LC controller

The LC controller holds, among others, a component called *interlock* [LS85]. An interlock can be a hardware or a software mechanism for ensuring correct sequences of events.

The LPN model for an interlock is shown in Figure B.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 B.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.



(a) without interlock (b) an interlock

Figure B.4: The PN model for an interlock

In a given system, multiple interlocks may exist for ensuring the order of events in some sequences. For example, two interlocks in the LC controller module exist, as shown in Figure B.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 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.

B.2.3 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) 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 B.5, where the labels of t_7 and t_8 transitions denote "lower" and "raise" respectively.

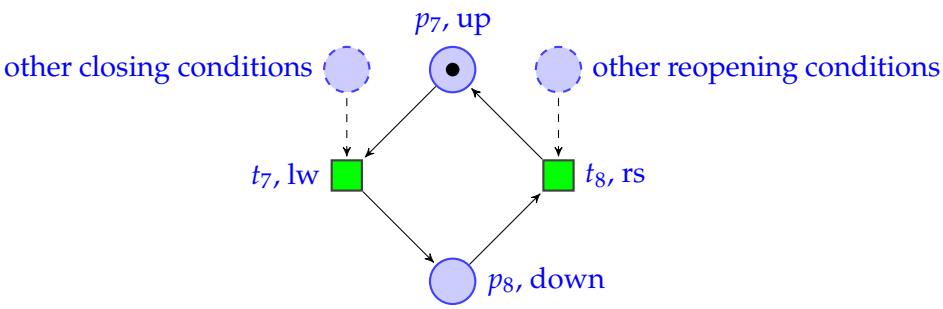


Figure B.5: The LPN model for a barrier system

B.3 Single-Track LC Model

After having set up the models for the three LC subsystems, let us now establish the global single-track LC model depicted in Figure B.6.

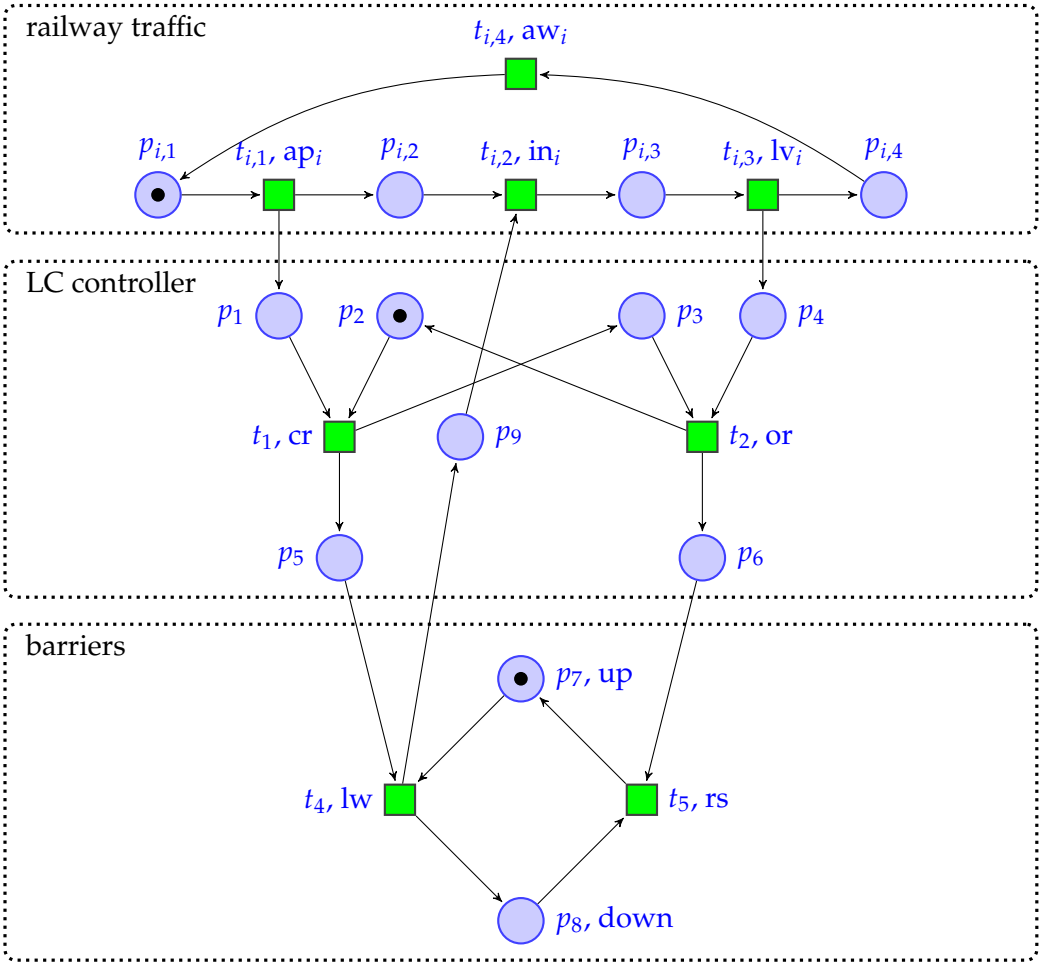


Figure B.6: A single-track LC

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 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 B.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 a train-car collision.

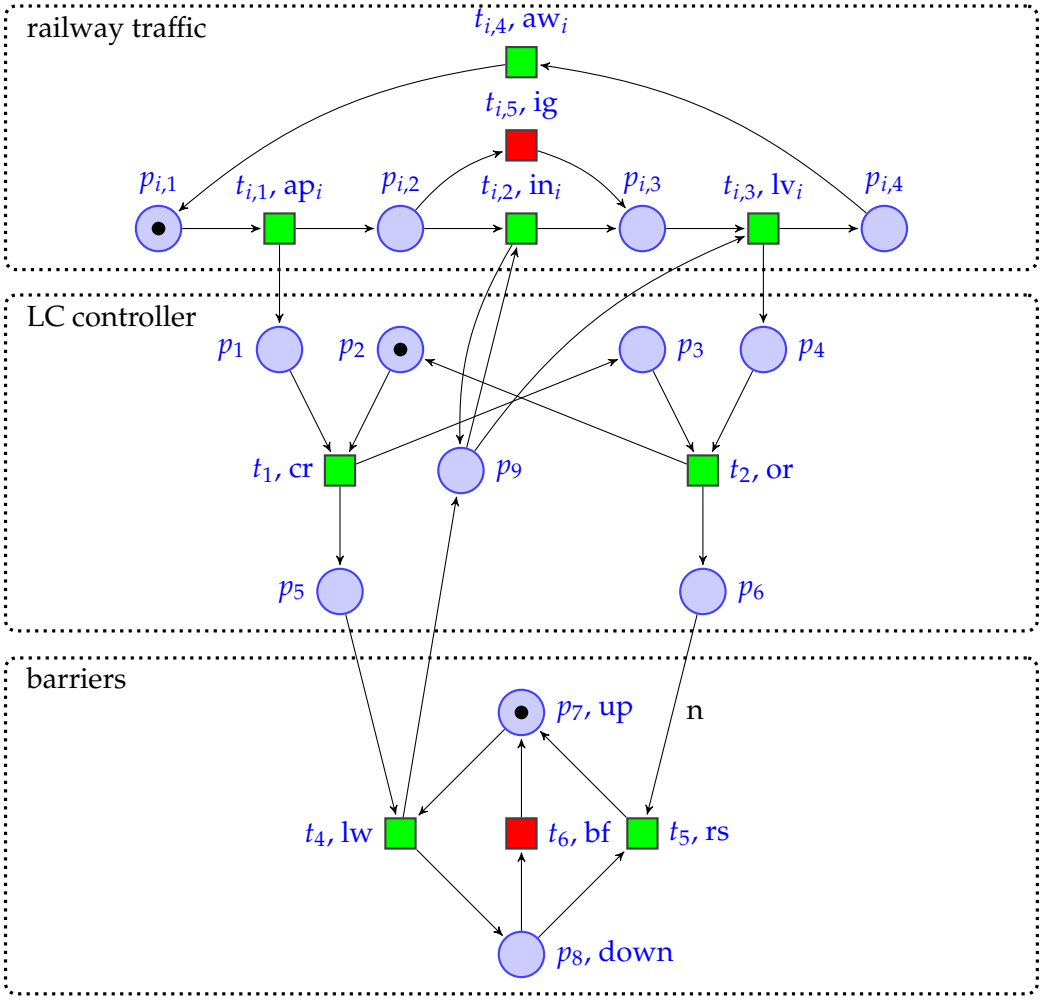


Figure B.7: A single-track LC with two classes of faults

Note that compared with the model shown in Figure B.3, there are two more arcs into and out of 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-track LC model afterward).

In the following section, we will introduce a more general LPN model for the LC system, while taking into account n railway lines.

B.4 n-Track LC Model

Figure B.7 describes a global LPN model for a unidirectional signle-track LC. Based on this model, a more general model is given in Figure B.8 – involving n railway tracks,

which can be obtained from the single-track 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.

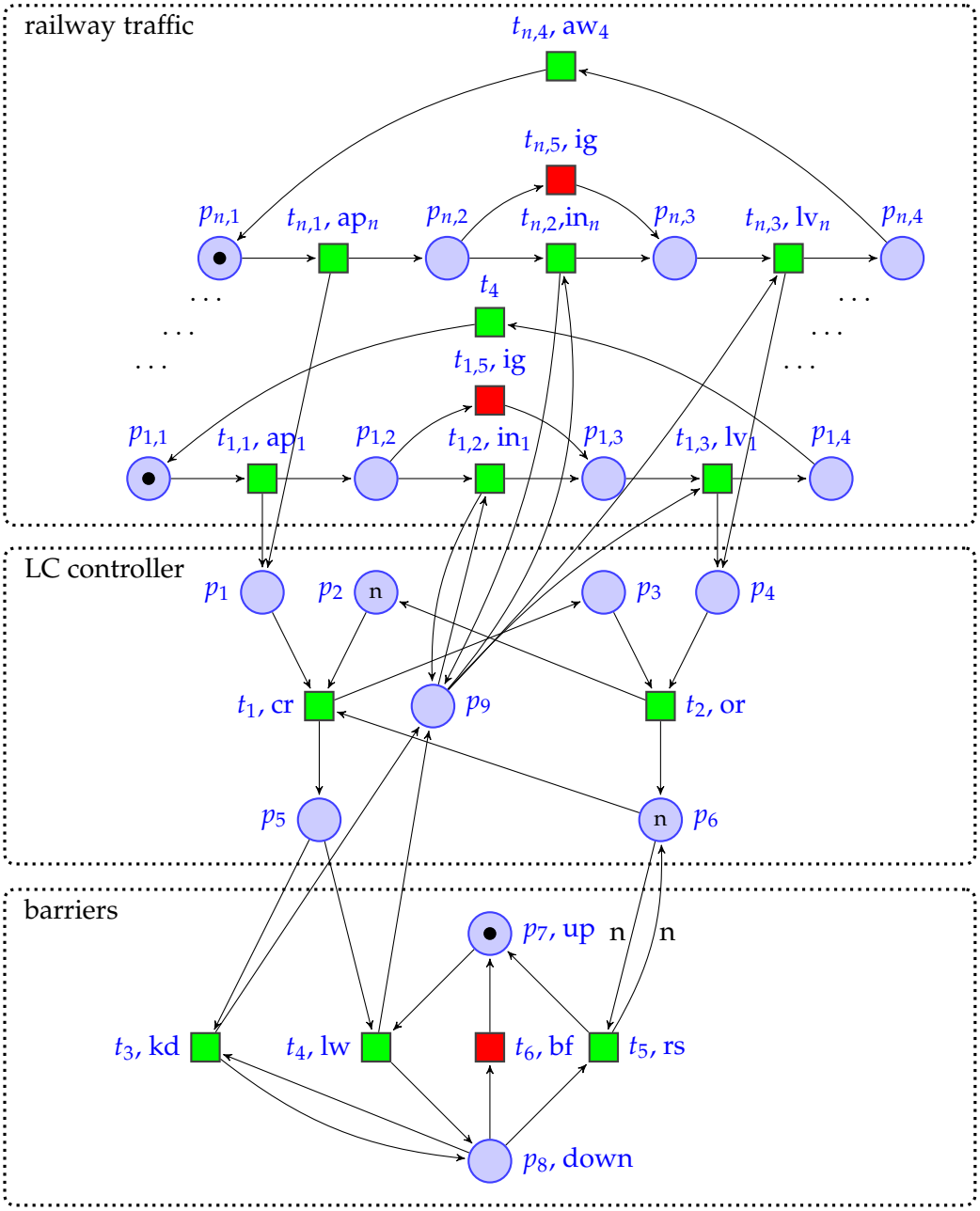


Figure B.8: n-track LC benchmark

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-track LC (cf. Figure B.7), there are several changes when generating the n-track LC model:

- Transition t_3 is newly added. In the n-track 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 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., $T_o = T \backslash T_u$ and $T_u = T_f = \{t_6\} \cup (\cup_i \{t_{i,5}\})$.

The n-track 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 [Ber+04], as shown in Table B.1, where:

- n denotes the number of tracks in the n-track 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 RG, i.e., the number of automaton arcs in the diagnoser approach [Sam+95].
- $|R|$ denotes the number of nodes in the RG, i.e., the number of automaton states when analyzing diagnosability with the diagnoser approach;
- $\mathcal{T}_T$ denotes the time used for computing the RG (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).

Table B.1: 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

Recall here that not the whole state space will be generated while using our on-the-fly technique. However, the RGs are generated in order to transform them into the input files (language-equivalent automata) for UMDES.

As shown in Table B.1, the size of the RG 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.

RÉSUMÉ ÉTENDU EN FRANÇAIS

Cette thèse s'intéresse à l'étude des problèmes de diagnostic des fautes sur les systèmes à événements discrets (SED) en utilisant des modèles réseau de Petri (RdP). Cette étude est effectuée dans l'équipe STF (Systèmes Tolérants aux Fautes) du LAGIS (Laboratoire d'Automatique, Génie Informatique et Signal, UMR CNRS 8219), à l'École Centrale de Lille, en collaboration avec le département COSYS/ESTAS (Composants et systèmes / Évaluation des systèmes de transports automatisés et de leur sécurité) à l'IFSTTAR (Institut français des sciences et technologies des transports, de l'aménagement et des réseaux), sous la direction du Prof. Armand Toguyéni et du Dr. Mohamed Ghazel, Chargé de Recherche à l'IFSTTAR - COSYS/ESTAS.

Cette mémoire de thèse se divise en six chapitres.

Chapitre 1

Le premier chapitre fournit un aperçu du manuscrit.

Ce chapitre débute par une introduction et une mise en contexte sur des problèmes de diagnostic des fautes sur les SED. Nous considérons les SED dans les contextes atemporel et temporel. Les modèles que nous utilisons sont les RdP-L et les RdP-LT. Les fautes que nous traitons sont permanentes, i.e., le système reste fautif dès qu'une occurrence de faute. Nous discutons quatre principaux problèmes :

- Diagnosticabilité classique : est-ce que le système est diagnosticable (une faute avec son type peut être déduit après un délai fini dès son occurrence) ou pas ?
- K/Δ-diagnosticabilité : est-ce que le système est K/Δ-diagnosticable (une faute avec son type peut être déduit après un délai donné et fini de K dès son occurrence dans le contexte atemporel ou Δ dans le contexte temporel) ou pas ?
- K/Δ_{min} : quel est le minimum valeur K/Δ pour assurer la diagnosticabilité (si le système est diagnosticable) ?
- Diagnostic en ligne : comment développer un outil diagnostiqueur pour effectuer le diagnostic en ligne ?

Les objectifs de cette thèse sont comme suit :

- Réduction des problèmes de l'explosion combinatoire dans les approches classiques ;
- Développement de la technique à-la-volée et incrémentale pour résoudre les principaux problèmes mentionnés en utilisant le même formalisme.

Chapitre 2

Le deuxième chapitre introduit certaines formalismes pour la modélisation des SED dans le contexte atemporel : automates, RdP et RdP labellisés (RdP-L); et dans le contexte temporel : automates temporisés, RdP temporels (RdP-T) et RdP labellisés et temporels (RdP-LT). Nous présentons aussi les définitions formelles de la diagnosticabilité des SED dans les contextes atemporel et temporels en utilisant les notations mentionnées.

Nous avons choisi les RdP comme les modèles pour l'analyse des SED, parce que les RdP ont les avantages de l'expressivité et la représentation compacte.

Chapitre 3

Le troisième chapitre examine les travaux existants sur le diagnostic des fautes des SED. Les références sont classifiées et résumées en trois parties : la diagnosticabilité et ses extensions, les approches de diagnostic, et les outils logiciels pour l'analyse de diagnostic.

Les deux problèmes les plus discutés sont le diagnostic en ligne et l'analyse de diagnosticabilité. En termes simples, le diagnostic est la détecter les occurrences des fautes et de localiser la cause des fautes. La diagnosticabilité se réfère à la capacité de détecter et localiser n'importe quelle faute dans un délai fini dès l'occurrence de cette faute. L'analyse de diagnosticabilité est effectuée hors ligne. En toute logique, une faute peut finalement être diagnostiquée si le système est diagnosticable. L'analyse de diagnosticabilité hors ligne est donc la base de diagnostic en ligne.

Les deux modèles les plus utilisés sont les automates et les RdP. Le diagnostic basé sur SED est premièrement étudié dans le cadre des langages réguliers et automates. Ces approches à base d'automates sont basées sur l'énumération des états, qui porte les problèmes de l'explosion combinatoire.

Afin de combattre ces problèmes, certaines approches basées sur l'automate ont été proposées pour réduire la complexité, sans construction d'un diagnostiqueur automate. De plus, une série de travaux concernant le diagnostic et la diagnosticabilité de SED a été discutée dans le contexte de RdP, pour profiter de ses avantages de l'expressivité et la représentation compacte.

Le diagnostic basé sur SED a été étudié dans les contextes atemporel et temporel. Les modèles à événement discret atemporel et temporel sont les deux abstractions des systèmes réels. Un modèle à événement discret atemporel caractérise le comportement logique des systèmes, i.e., seulement l'ordre des événements est considéré. Un modèle temporel peut préciser explicitement les contraintes temporelles et quantitatives sur le comportement du système. Le temps peut caractériser un SED dans une dimension temporelle, telle que le comportement du système contient des informations plus riche. Cependant, le traitement des tels systèmes est plus complexe.

Chapitre 4

Ce chapitre vise à développer la technique à-la-volée et incrémentale pour le diagnostic en ligne pour résoudre le problème de l'explosion combinatoire en traitant les systèmes complexes.

Comme illustré dans ce chapitre, l'analyse à base d'un automate connu ou la construction du graphe d'accessibilité entier n'est pas nécessaire pour l'analyse de diagnostic. Au lieu de cela, l'approche que nous proposons peut présenter plus d'efficacité comparée avec les approches existantes.

La motivation de l'utilisation de ces techniques est illustrée dans la section 4.1. Dans les sections suivantes, nous discutons le diagnostic des SED modélisés par les RdP-L, où les fautes correspondent aux transitions inobservables. Basé sur certains nouveaux concepts que nous avons introduits, une structure d'arbre qui porte le marquage et les informations d'occurrence de faute est élaborée. Cette structure que nous appelons FM-set tree est calculée à la volée pour vérifier la K-diagnosticabilité en utilisant un algorithme récursif que nous avons développé. De plus, par extension, nous transformons le problème de diagnosticabilité classique en une série des problèmes de K-diagnosticabilité, où la valeur K augmente progressivement. En outre, lorsque le système est K-diagnosticable, le diagnostic en ligne est effectué sur la base d'un diagnostiqueur qui est obtenu à partir de FM-set tree d'une manière simple.

Enfin, un teste comparatif basé sur le WODES benchmark est effectué à l'aide du logiciel OF-PENDA que nous avons développé pour prouver l'efficacité de notre approche. Comme certaines limitations du WODES benchmark apparaît dans le teste, nous développons un benchmark à partir du système ferroviaire : passage à niveau avec multiple lignes. Ce benchmark vivant et bornée porte deux types de faute : l'un diagnosticable et l'autre non-diagnosticable. De plus, nous pouvons obtenir un modèle suffisamment gros si le nombre de ligne augmente. Le résultat obtenu par le benchmark passage à niveau présente très bonne efficacité en termes de temps et mémoire comparé avec les approches diagnostiqueur et vérifieur lors que le système et non-diagnosticable à cause de l'existence d'un cycle indéterminé.

Chapitre 5

Le cinquième chapitre discute le problème de diagnostic des fautes dans le contexte temporel.

Le modèle que nous utilisons est les RdP-LT qui est une extension des RdP-T pour laquelle à chaque transition est associé un événement qui peut être observable ou non.

La première contribution est le développement du technique fractionnement des intervalles de temps, qui permet de transformer l'analyse des problèmes dans le contexte temporel en l'analyse à l'aide des techniques dans le contexte atemporel. La deuxième contribution est les conditions nécessaires et suffisantes pour la diagnosticabilité des RdP-LT.

Nous proposons une approche pour vérifier la diagnosticabilité et nous fournissons une solution pour calculer le délai minimum pour garantir cette diagnosticabilité. L'analyse de cette diagnosticabilité est effectuée par un algorithme à la volée, basé sur une structure de données que nous nommerons ASG. Cette structure contient les informations sur l'état du RdP-LT.

Les algorithmes à la volée permettent souvent d'obtenir un résultat sans avoir nécessairement à explorer tout l'espace d'états. C'est un avantage important par rapport aux approches classiques d'énumération systématique de tous les états. La discussion dans ce chapitre est la première contribution sur la diagnosticabilité dans le cadre de RdP-T d'après notre meilleure connaissance.

Chapitre 6

Le sixième chapitre termine ce manuscrit avec les contributions et respectives. L'originalité de cette thèse est le développement du technique à-la-volée et incrémentale pour l'analyse des problèmes de diagnostic, qui permet de partiellement résoudre le problème de l'explosion combinatoire.

Les principales contributions de cette thèse sont comme suit :

- Dans le contexte atemporel, nous avons développé les notations pour représenter le modèle RdP-L de manière mathématique, la technique à-la-volée et incrémentale pour l'analyse du diagnostic et un outil logiciel OF-PENDA pour implémenter notre approche ;

- Dans le contexte temporel, nous avons développé la technique « fractionnement des intervalles de temps » pour l'analyse de estimation d'états et diagnostic des fautes. De plus, nous avons aussi proposé et prouvé les conditions nécessaires et suffisantes pour la diagnosticabilité des SED-T.

Certaines directions sont indiquées pour l'étude à l'avenir : l'application de la technique a-la-volée sur l'approche vérifieur, le placement des capteurs, l'introduction de la technique « zone graph », etc.

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 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 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. Analytical 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 permet de construire et explorer seulement une partie de l'espace d'état. Les résultats des analyses effectuées sur certains bancs d'essais 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.