ONLINE FAULT DIAGNOSIS OF DISCRETE EVENT SYSTEMS MODELED BY LABELED PETRI NETS USING LABELED PRIORITY PETRI NETS

Braian Igreja de Freitas

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: João Carlos dos Santos Basilio

Rio de Janeiro
Abril de 2021

ONLINE FAULT DIAGNOSIS OF DISCRETE EVENT SYSTEMS MODELED BY LABELED PETRI NETS USING LABELED PRIORITY PETRI NETS

Braian Igreja de Freitas

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Orientador: João Carlos dos Santos Basilio

Aprovada por: Prof. João Carlos dos Santos Basilio
Prof. Abdoul Karim Armand Toguyeni
Prof. José Reinaldo Silva
Prof. Lilian Kawakami Carvalho

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2021

Freitas, Braian Igreja de

Online Fault Diagnosis of Discrete Event Systems Modeled by Labeled Petri Nets Using Labeled Priority Petri Nets/Braian Igreja de Freitas. – Rio de Janeiro: UFRJ/COPPE, 2021.

XIV, 108 p.: il.; $29,7$cm.

Orientador: João Carlos dos Santos Basilio

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2021.

Referências Bibliográficas: p. 106 – 108.

1. Discrete event system. 2. Petri net. 3. Online diagnosis. I. Basilio, João Carlos dos Santos. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Acknowledgments

I thank my parents, Paulo and Denise, for their constant love and support, which motivates me to always do my best.

I thank my brother Breno for being a supportive family member and friend.

I thank all my grandparents, Adenair, Irma, Januário and Walcy for their love.

I thank all my long time friends, specially Doda, Eliésio, Isabel, Guilherme, Mateus, Marcelo and Thais, who constantly supported me throughout the tough times caused by the Covid-19 pandemic.

I thank Amanda, Amy, Daiha, Eric, Hugo, Iasmin, Mariana, Patusco, Victor and Vinicius for their great time during my undergraduate studies in the Federal University of Rio de Janeiro.

I thank Prof. João Basilio for advising this work and for always teaching me and motivating me to follow the academic career.

I thank my fellow students of the Laboratory of Control and Automation, specially, Antônio Gonzalez, Lucas Antunes, Públio Lima, Thiago Cabral and Raphael Barcelos for their short yet enjoyable time.

I thank all the professors of the Federal University of Rio de Janeiro for sharing their knowledge with me, specially, Prof. Afonso Gomes, Prof. Fernando Lizarralde, Prof. Lilian Carvalho and Prof. Marcos Moreira.

I thank the National Council for Scientific and Technological Development (CNPq) for their financial support during the first year of my master's studies and the Carlos Chagas Filho Foundation for Supporting Research in the State of Rio de Janeiro (FAPERJ) for their financial support during the second year of my master's studies.

I thank professors João Basilio, Abdoul Toguyeni, José Silva and Lilian Carvalho for their feedback after the defense of this dissertation.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

DIAGNÓSTICO DE FALHAS EM TEMPO REAL DE SISTEMAS A EVENTOS DISCRETOS MODELADOS POR REDES DE PETRI ROTULADAS UTILIZANDO REDES DE PETRI ROTULADAS COM PRIORIDADES

Braian Igreja de Freitas

Abril/2021

Orientador: João Carlos dos Santos Basilio

Programa: Engenharia Elétrica

O problema da diagnose de falha em sistemas a eventos discretos consiste na capacidade de se detectar e isolar a ocorrência de eventos de falhas. Neste trabalho, são propostos múltiplos algorítimos para, a partir da rede de Petri do sistema a ser diagnosticado, criar uma rede de Petri diagnosticadora rotulada com prioridades cujas transições são rotuladas apenas por eventos observáveis e cujos estados alcançáveis possuem informações necessárias para que o diagnosticador possa ter certeza da ocorrência de falha. Assim como em trabalhos em anteriores, supõe-se que a rede de Petri do sistema a ser diagnosticado não possui ciclos envolvendo lugares e transições não observáveis. Além disso, sob uma hipótese mais restritiva envolvendo os estados alcançáveis da rede de Petri diagnosticadora, o diagnosticador proposto terá uma estrutura que não terá crescimento indefinido em decorrência das observações de eventos, fazendo com que o diagnosticador aqui proposto seja capaz de executar o diagnóstico online de classes de redes de Petri que trabalhos anteriores somente são capazes de diagnosticar com estruturas passíveis de crescimento indefinido para determinadas sequências de observações de eventos.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ONLINE FAULT DIAGNOSIS OF DISCRETE EVENT SYSTEMS MODELED BY LABELED PETRI NETS USING LABELED PRIORITY PETRI NETS

Braian Igreja de Freitas

April/2021

Advisor: João Carlos dos Santos Basilio

Department: Electrical Engineering

The problem of fault diagnosis of discrete event systems concerns the capacity to detect and isolate the occurrence of fault events. In this work, we propose multiple algorithms to create a labeled priority diagnoser Petri net from the Petri net of the system to be diagnosed. The diagnoser Petri net transitions are labeled only by observable events and its reachable states have enough information to allow the diagnoser to be sure about the fault occurrence. As in previous works, we assume that the Petri net of the system to be diagnosed does not possess cycles involving places and unobservable transitions. In addition, under a more restrictive assumption regarding the reachable states of the diagnoser Petri net, the diagnoser will have a structure that does not grow indefinitely due to event observations, making the diagnoser proposed here able to execute the online diagnosis of a class of Petri nets that previous works were only able to diagnose with structures that are likely to grow indefinitely for specific sequences of event observations.

Contents

List of Figures

List of Tables

List of Symbols

$\mathbb{N}$	Set of natural numbers, where the first element is 1, Page 29
$\mathbb{Z}$	Set of integer numbers, Page 24
$\mathbb{Z}_+$	Set of non-negative integer numbers, Page 24
$\mathbb{Z}_+^n$	Set of n dimensional vectors where each element is a non-negative integer number, Page 13
$\lvert s \rvert$	Length of sequence s, Page 9
ϵ	Empty sequence of events, Page 9
$s \lhd t$	Indicates that sequence s can be obtained by removing some of the elements of sequence t, Page 9
Σ^*	Kleene-closure of the set of events Σ, Page 9
$\overline{L}$	Prefix-closure of language L, Page 10
L/s	Post-language of language L with respect to sequence s, Page 10
$P : \Sigma_l^* \to \Sigma_s *$	Projection of Σ_l over Σ_s, Page 10
n_P	Number of places in the set of places P, Page 12
n_T	Number of transitions in the set of transitions T, Page 12
$I(t)$	Input places of transition t, Page 13
$O(t)$	Output places of transition t, Page 13
$I(p)$	Input transitions of place p, Page 13
$O(t)$	Output transitions of place p, Page 13
$\vec{a} \geq \vec{b}$	Each i-th element of $\vec{a}$ is greater than or equal to the i-th element of $\vec{b}$, Page 14
$\vec{a} > \vec{b}$	Each i-th element of $\vec{a}$ is greater than the i-th element of $\vec{b}$, Page 14
$\vec{m}[r\rangle$	The sequence of transitions r is enabled by the marking vector $\vec{m}$, Page 16
$\vec{m}_1[r\rangle\vec{m}_2$	Firing the transition sequence r from the marking vector $\vec{m}_1$ results in the marking vector $\vec{m}_2$, Page 16
T^*	Set of all possible transition sequences formed by the transitions of the set T, Page 16

$LT(\mathcal{N})$	Set of transition sequences that may fire in the Petri net $\mathcal{N}$, Page 16
λ	Empty transition sequence, Page 16
$R(\mathcal{N})$	Set of reachable states of the Petri net $\mathcal{N}$, Page 16
$\langle \sigma, T_e, \vec{m} \rangle$	Event conflict involving the transitions of T_e labeled by event σ and enabled by the marking vector $\vec{m}$, Page 19
$L(\mathcal{N})$	Language generated by the Petri net $\mathcal{N}$, Page 20
Σ_o	Set of observable events, Page 20
Σ_{uo}	Set of unobservable events, Page 20
$P_o : \Sigma^* \rightarrow \Sigma_o*$	Projection of Σ over the observable events Σ_o, Page 20
T_o	Set of observable transitions, Page 20
T_{uo}	Set of unobservable transitions, Page 20
$PT : T_l^* \rightarrow T_s*$	Projection of T_l over T_s, Page 20
$PT_o : \Sigma^* \rightarrow \Sigma_o*$	Projection of T over the observable transitions T_o, Page 20
ω	Symbol representing that a place marking can be as large as required, Page 24
Σ_f	Set of fault events, Page 28
$\Psi(\Sigma_f)$	Set of event sequences that end with fault events, Page 28
$MT : T_D^* \rightarrow T_o^*$	Mapping function of the set transitions T_D of the diagnoser Petri net over the observable transitions T_o of the original Petri net Page 56
p_f	Place of a diagnoser Petri net that indicates the fault occurrence, Page 51
σ_{fv}	Event that indicates the possibility of the fault occurrence, Page 51
P_p	Set of places created by function NOC, Page 70
$Poss_{cache}$	List of possibilities associated with the places of P_p, Page 70
C_F	Case of the diagnoser in which we are sure that the fault event has occurred, Page 85
C_D	Case of the diagnoser in which we are sure that the fault event could have occurred, Page 85
C_N	Case of the diagnoser in which we are sure that the fault event did not occur, Page 85
M_{s_o}	Matrix where each column is a possible marking vector that the Petri net may reach with the observation of the event sequence s_o, Page 91
M_{red,s_o}	Matrix initially equal to M_{s_o}, but where each row is reduced by their minimum value and all repeated columns of M_{red,s_o} are removed, Page 91

CT(.)	Coverability tree algorithm, Page 25
$FAM(.)$	Function that obtains all the minimal markings and their associated transition sequences with respect to a transition, Page 51
$NOC(.)$	Function that solves multiple event conflicts involving a set of transitions labeled by a given event, Page 70
$AOT(.)$	Function that defines all output transitions of a place created by function NOC, Page 73
ECT(.)	Extended coverability tree algorithm, Page 94

Chapter 1

Introduction

Discrete event system (DES) is a special class of systems whose state space is discrete and the dynamic evolution is ruled by the asynchronous occurrence of events [1]. This type of system can be used to model various applications, such as manufacturing systems, operational systems, robotics and logistics.

Among the formalisms used to model DESs, automata and Petri nets are the most commonly used [1, 2]. The states and events of DESs are directly represented, respectively, by the nodes and arcs of automata, whereas Petri nets have two types of nodes, places and transitions. The places are associated with numbers of tokens whose possible combinations model the states of the DES, whereas the transitions represent the events. The structure of automata facilitates the analysis of the behavior of the DES it models, but it cannot be used to model DESs with infinite numbers of states. In contrast, the analysis of a DES modeled by a Petri net is harder, but since its structure uses places and tokens to represent the states of DESs, it can be used to represent a DES with an infinite number of states.

A few decades ago, Lin [3] and Sampath et al. [4] introduced the notion of fault diagnosis of DESs, where it is shown how to infer the occurrence of some unobservable events, the so-called fault events, through the observation of other events that are observable. This notion has shown to have great potential due to its capacity to allow the detection of the occurrence of unobservable events without the need to directly detect them, thus, reducing the cost of implementing sensors for the fault events. As DESs become more complex, the detection of their fault events becomes less straightforward, increasing the difficulty of doing the fault diagnosis of particular systems. In order to solve this issue, recent works address the general problem of fault diagnosis of DESs, where each work solves the problem for a class of DESs.

The problem of fault diagnosis can be divided into two main problems: diagnosablity and diagnosis. The diagnosability problem with respect to a given DES is associated with the capacity to assert whether it is possible of infer the occurrence of a fault event of the DES, and the diagnosis is related to the construction of an online diagnoser that follows a procedure in order to determine the occurrence of fault events during the DES operation.

Since the structure of automata are simpler, several works studied the diagnosability and the online diagnosis of DESs modeled by automata have been proposed, such as [4–13]. Although a large number of aspects regarding fault diagnosis are approached by those works, their contribution are limited to DESs modeled by automata, which means that their findings are not applicable to DESs that can only be modeled by more complex models such as Petri nets.

The complex structure of Petri nets, on the other hand, allows the study of DESs that contain infinite numbers of states. In this regard, some works that approach the problem of diagnosability and online diagnosis of DESs modeled by Petri nets appear in the literature [14–23]. Although some of these works assume that the Petri net model only have a finite number of states, they are still relevant due to the known space explosion when using automaton models.

In [14], necessary and sufficient conditions for the diagnosability and K-diagnosability of bounded and unbounded Petri net systems based on event observations are presented. It is also proposed an algorithm that is able to verify language diagnosability using linear programming and a verifier Petri net, which is constructed from the Petri net system whose diagnosability is being tested. Although [14] contributes to the studies of diagnosability of DESs modeled by Petri nets, its scope regarding online diagnosis is rather limited.

Liu et al. (2017) [15] claims that all of the problems of diagnosability, K-diagnosability and online diagnosis for bounded Petri nets can be solved by the on-the-fly construction of two graphs, named fault marking graph and fault marking set graph. The nodes of the latter enumerate the set of states that are consistent with the events that are observed on-the-fly, being each node associated with a number that indicates the fault occurrence. Additionally, to the whole graph there corresponds a value that represents the maximum length of the sequences of nodes that are associated with the occurrence of the fault event. Those components allow the solution of the aforementioned problems without the need to completely construct the fault marking set graph.

In [16], it is proposed an online diagnoser for ordinary Petri nets that is based on the observation of the number of tokens of some of their places. In addition, an algorithm capable of defining the minimal set of places that need to be observed in a Petri net for the diagnosis to be possible is also proposed in [16]. The approach of [17] for the online diagnosis is similar to [16] in the sense that it also uses the observation of the number of tokens of some places of the Petri net. Nevertheless, [17] also takes into account partial observation of event occurrences, and combines both types of observations in order to diagnose the occurrence of fault events in interpreted Petri nets using the solution of linear programming problems. It is worth mentioning that among the works cited here, [16] and [17] are the only approaches that consider the observation of tokens of some places. All of the others, including the one considered in this dissertation, only rely on the observation of event occurrences.

In [18], it is proposed an online diagnoser that stores all of the states that a Petri net may reach that are consistent with the observation of a sequence of events, where each state is associated with labels that indicate the occurrence of fault events. Even though [18] does not impose any restrictions regarding the Petri net that models the DES, the number of states of the diagnoser consistent with an event sequence observation may be either infinite or grow with respect to the length of the observed sequence; the former would cause the online diagnoser to attempt to store an infinite number of states, whereas the latter would cause the online diagnoser to compute an undetermined number of states as the observed event sequence grows, which would slow down the online diagnosis process after each event observation.

Inequalities that are obtained from the Petri net together with the Fourier-Motzkin elimination method are deployed in [19] and [20] to execute the online diagnosis of Petri net systems. If, on one hand, the use of inequalities increases the speed of the fault events diagnosis, on the other, the online diagnosers proposed in [19] and [20] are only suitable to acyclic and reversible Petri nets, respectively.

The approaches presented in [21] and [22] use online diagnosers that detect the occurrence of fault events by solving linear problems, whose variables and constraints are defined by the Petri net systems. They both require that the Petri net to be diagnosed does not contain any cycles of unobservable transitions. Furthermore, the online diagnosers presented in [21] and [22] require that each observable event must not be associated with more than two transitions of the Petri net.

Finally, [23] uses the notion of basis markings and justifications to implement the online diagnoser associated with the Petri net systems. The work proposed an online diagnoser for Petri nets that do not contain cycles of unobservable transitions and a more optimized online diagnoser for bounded Petri nets. Although [23] presents the computation of a general online diagnoser, the proposed diagnoser must store both the basis markings and justifications that are consistent with the event observations; therefore, similar to what may occur with the online diagnoser proposed in [18], the number of basis markings and justifications may grow indefinitely as the observed event sequence increases, which may slow down the online diagnosis computation process required after each observation.

1.1 Objective of this work

The aforementioned works sometimes fail to diagnose the fault occurrences of diagnosable DESs modeled by labeled Petri net using limited structures. In order to increase the class of labeled Petri nets systems whose fault events can be diagnosed with limited structures, we propose, in this work, a new approach for the online diagnosis of Petri net system that consists of using the Petri net to be diagnosed to create a diagnoser Petri net whose structure is a λ-free (no unobservable transitions) labeled priority Petri net that is able to replicate the behavior of the original Petri net. Furthermore, the states of the diagnoser Petri net that may be reached with the observed transition sequences of the original Petri nets contain enough information to allow the diagnoser to be sure about the occurrence of the fault events of the system. It is worth remarking that, in order for the diagnoser Petri net to be built, it is required that the Petri net system does not have any cycles of unobservable transitions, which is a common assumption among works on online diagnosers of DESs [15, 21–23]. One of the advantages of the proposed diagnoser Petri net is that, for some classes of Petri net systems, its structure does not grow indefinitely with the growth of observed event sequences, as it is the case of all existing diagnosers of previous works that can diagnose these classes.

Notice that it is possible for observable events of the diagnoser Petri net to be associated with multiple transitions, which may cause one event observation to be able to generate multiple possible states, which requires the analysis of multiple states in order to determine the fault occurrence. In order to circumvent this issue, we propose an algorithm that modifies a previously obtained diagnoser Petri net by replacing those transitions that can model the same event occurrence with new ones in order to make the diagnoser Petri net always generate only one possible state after each event observation. Finally, it will be shown that, under an additional assumption, it will be possible to move to the offline computation all modifications on the diagnoser Petri net to solve the aforementioned problem, therefore ensuring that the diagnoser Petri net will be able to diagnose the fault occurrence of a Petri net system without the need of a structure that grows indefinitely with the event observations. It is worth remarking that our approach will also be able to diagnose the fault occurrence of Petri nets where the aforementioned assumption does not hold by doing the modifications on the diagnoser Petri net during the online diagnosis.

1.2 Dissertation structure

This work is organized as follows. Chapter 2 presents a review on DESs, the structures of the Petri nets that will be considered in this work and an algorithm that finds the coverability trees of Petri nets. In Chapter 3, we explain with more detail the definition of fault diagnosability and diagnosis of DESs and Petri net systems, and we briefly explain the online diagnoser proposed in [23]. After that, in Chapter 4 our approach for online diagnoser of Petri net systems using a λ-free labeled priority diagnoser Petri net is presented. Chapter 5 summarizes all of the contributions of this work to the problem of online diagnosis of Petri net systems and indicates possible future directions that may be taken from this work.

Chapter 2

Theoretical background

In this chapter, we present the basic concepts of DESs and the structures of the Petri nets that are used in this work. Additionally, we present the algorithm that obtains the coverability trees of Petri nets. Section 2.1 introduces the main concepts of discrete event systems and how it operates. In Section 2.2, we present languages and their operations and properties, which are commonly used to describe the event occurrences of discrete event systems. After that, in Section 2.3, we present all the Petri net structures and their properties that are used in this work, where the structures that we use are the following: Petri nets, marked Petri nets, labeled Petri nets and labeled priority Petri nets. Finally, in Section 2.4, we present an algorithm for the construction of the coverability tree of Petri nets.

2.1 Discrete event systems

A discrete event system is a system whose states are described by a discrete set and the dynamic evolution is ruled by event occurrences, *i.e.*, a transition from one state of the system to another occurs given the occurrence of an event [1]. Due to the nature of DESs, two main concepts exist in this type of system: states and events.

The states of a DES, which are contained into a discrete set, are able to represent the system current situation through symbols. For example, a switch that can be turned on or off may have two states, one that models that the system is *on*, and another one that models that it is *off*. Those states can be grouped into the set of states $X = \{On, Off\}$. Although in this example the state set has two states only, the set of states of a DES may contain infinite states, such as an infinite queue, whose states may be modeled by the number of occupants, which may vary from 0 to infinity.

An event, on the other hand, is associated with some occurrence that may cause a state of the DES to change. Regarding the example of the switch, the action of someone pressing the switch may be considered an event labeled by σ. If someone were to press the switch while its state is *On*, its state would change to *Off*. Notice that the change of state from *Off* to *On* may also be caused by the occurrence of event σ. Thus, when it occurs, the same event may be able to cause different state changes on the DES. In the DESs considered in this work, all events will be presumed to be instantaneous and their occurrence will not depend on time; thus, the notion of the system dynamics being ruled by time, as usually used in time-driven systems, will not be used in this work, since an event-driven approach, where the dynamic of the system is described by the occurrence of events, will be the approach adopted here.

Although DESs are based on a discrete approach, as illustrated by the example of the switch, most systems, including time-driven systems, can be modeled by a DES given the correct abstraction. Therefore, analyzing tools developed for DESs can be beneficial to all kinds of systems, which motivates the ongoing study on DES. Those studies are mainly developed based on the two kinds of event driven formalisms: Automaton and Petri net. Although the approach of this work will be based on Petri nets, it will also use the concept of language, which is commonly regarded in DESs studies with the aim of listing all event sequence occurrences during the system operation.

2.2 Language

A language is defined over a set of events Σ and is a set of finite event sequences constructed with the events of Σ [1]. For example, if $\Sigma = \{a, b, c\}$ is a set of three events, $L_1 = \{abc, bc, ca, cac\}$ is a possible language of Σ that contains four sequences composed by events of Σ. Although every sequence of a language must be finite, a language may contain an infinite number of sequences. For example, language $L_2 = \{\textit{Every possible sequence of events of } \Sigma \textit{ that contains } cb\}$, which is also defined over Σ, has an infinite number of finite length sequences.

The length of a sequence s is denoted by $|s|$ and a sequence that does not contain events is referred to as an empty sequence, being denoted by ϵ; thus, $|\epsilon| = 0$. Sequences may also be concatenated into larger sequences; for example, sequence $s = abcd$ may be considered the result of a concatenation between sequences $t = ab$, $u = c$ and $v = d$ in such way that $s = tuv$. The empty sequence when concatenated with another sequence does not add any events to the sequence; for example, for a sequence s, $\epsilon s = s\epsilon = s$. Finally, we will use $s \lhd t$ to denote that sequence s can be obtained from sequence t by removing some events from t; for example, considering sequences $s_1 = abcb$, $s_2 = ab$, $s_3 = bcb$ and $s_4 = bb$, it can be inferred that $s_2 \lhd s_1$, $s_3 \lhd s_1$ and $s_4 \lhd s_1$.

The language that contains all finite sequences that can be generated by Σ, including the empty sequence, is denoted by Σ^* and is named Kleene-closure of Σ. If $\Sigma = \{a, b\}$, for example, then $\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, baa, abb, bab, bba, bbb, \ldots\}$. Notice that every language L created from the events of Σ must satisfy $L \subseteq \Sigma^*$.

2.2.1 Language operations and properties

Since languages are sets of sequences, all operations defined over sets, such as union, intersection and complement with respect to Σ^* are applicable to languages. Besides those operations, the following operations and properties of languages are also used in this work:

- ***Concatenation:*** Given two languages $L_1, L_2 \subseteq \Sigma^*$, the concatenation of L_1 with L_2 is $L_1L_2 = \{s \in \Sigma^* : (\exists s_1 \in L_1) \wedge (\exists s_2 \in L_2)[s = s_1s_2]\}$.

Example 2.1. *Let* $\Sigma = \{a, b, c\}$ *be the set of events of languages* $L_1 = \{a, c, ac, bca\}$ *and* $L_2 = \{\epsilon, c\}$. *The concatenation of both languages is* $L_1L_2 = \{a, c, ac, bca, cc, acc, bcac\}$.

- ***Prefix-closure:*** Given a language $L \subseteq \Sigma^*$, the prefix-closure of L is $\overline{L}$, where

$$\overline{L} = \{s \in \Sigma^* : (\exists t \in \Sigma^*)[st \in L]\}.$$

Additionally, a language $L \subseteq \Sigma^*$ is said to be prefix-closed if $L = \overline{L}$.

Example 2.2. *Let* $\Sigma = \{a, b, c\}$ *be the set of events of languages* $L_1 = \{a, c, ac, bca\}$ *and* $L_2 = \{\epsilon, a, b, ab\}$. *Notice that* L_1 *is not prefix-closed, due to* L_1 *being different than* $\overline{L_1} = \{\epsilon, a, b, c, ac, bc, bca\}$. *On the other hand, every prefix of each element of* L_2 *is contained in itself, which makes* $L_2 = \overline{L_2}$. *Therefore,* L_2 *is prefix-closed.*

- ***Post-language:*** Given a language $L \subseteq \Sigma^*$ and a sequence $s \in \Sigma^*$, the post-language L after s is

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

Example 2.3. *Let* $\Sigma = \{a, b, c\}$ *be the set of events of language* $L = \{a, b, c, bc, ac, abc, bac\}$. *The post-language of* L *after sequence* b *is* $L/b = \{\epsilon, c, ac\}$.

- ***Natural projection:*** The natural projection, also known simply as projection, is the mapping of a sequence from a larger set of events Σ_l to a smaller set of events Σ_s. This mapping is represented by the function $P : \Sigma_l^* \to \Sigma_s^*$ and can be defined by the following recursion:

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

$$P(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_s \\ \epsilon, & \text{if } \sigma \in \Sigma_l \setminus \Sigma_s \end{cases}$$

$$P(s\sigma) = P(s)P(\sigma), \forall s \in \Sigma_l^*, \forall \sigma \in \Sigma_l.$$

Notice that the projection operation removes the events of $s \in \Sigma_l^*$ that do not belong to Σ_s. The projection function can also be inverted, resulting in the inverse mapping. The inverse function, named inverse projection $P^{-1} : \Sigma_s \to 2^{\Sigma_l}$, is defined, for a sequence $s \in \Sigma_s^*$, as $P^{-1}(s) = \{t \in \Sigma_l^* : P(t) = s\}$. Notice that the result of the inverse projection is a set of sequences generated by all possible additions in s of events of Σ_l that are and not Σ_s.

The projection operation can also be extended to languages $L \subseteq \Sigma_l^*$ by applying the projection operation on all sequences of L. Therefore, $P(L) = \{s \in \Sigma_s^* : (\exists t \in L)[P(t) = s]\}$. Likewise, the inverse projection of a language $L \subseteq \Sigma_s^*$ is defined as $P^{-1}(L) = \{s \in \Sigma_l^* : (\exists t \in L)[P(s) = t]\}$.

Example 2.4. *Let $\Sigma_l = \{a, b, c\}$ be the set of events of language $L_1 = \{a, b, c, ac, acbc\}$, $\Sigma_s = \{a, b\}$ be the set of events of language $L_2 = \{ab, b\}$ and $P : \Sigma_l^* \to \Sigma_s^*$ be a projection from Σ_l to Σ_s. By applying the projection P to L_1, we obtain $P(L_1) = \{a, b, \epsilon, ab\}$, whereas by applying the inverse projection P^{-1} on L_2, we obtain $P^{-1}(L_2) = \{\{c\}^* a \{c\}^* b \{c\}^*\} \cup \{\{c\}^* b \{c\}^*\}$.*

- ***Liveness:*** A language $L \in \Sigma^*$ is said to be a live language if $(\forall s \in L)(\exists \sigma \in \Sigma) : (s\sigma \in L)$. In words, for every sequence $s \in L$, there is another sequence in L that is s concatenated with an event $\sigma \in \Sigma$.

Example 2.5. *Let $\Sigma = \{a, b, c\}$ be the set of events of language $L_1 = \{a\}\{c\}^* = \{a, ac, acc, accc, \dots\}$ and language $L_2 = \{a, ab, abb\}$. Notice that L_1 is live, since for every sequence $s \in L_1$, $sc \in L_1$, whereas L_2 is not live, since there does not exist an event $\sigma \in \Sigma$ such that $abb\sigma \in L_2$.*

2.3 Petri net

One of the formalisms capable of representing DES is the Petri net. Its structure allows a graphic representation of the relation between the states and events of a great variety of DES, including DESs with an infinite number of states [1].

2.3.1 Petri net structure

A Petri net is a bipartite graph with two types of vertices that are connected by weighted arcs: places and transitions. Places are represented by circles and they are related to the Petri net states, whereas transitions are represented by line segments and, in most cases, are associated with events. Arcs are represented by arrows and they represent the relation between places and transitions; therefore, they never connect vertices of the same type, *i.e.* the arcs of a Petri net can only connect places to transitions and transitions to places. Notice that each arc has an associated weight that represents the amount of resources that will either be removed from the places where the arcs start or be added to the places whose arcs start from some transition. Weights appears alongside each arc, if their values are greater than 1.

Formally, Petri nets are defined as follows [2].

Definition 2.1 (Petri net). *A Petri net is a quadruple*

$$\mathcal{N} = (P, T, Pre, Post),$$

where:

- $P = (p_1, p_2, ..., p_{n_P})$ *is the finite set of places;*
- $T = (t_1, t_2, ..., t_{n_T})$ *is the finite set of transitions;*
- Pre *is a* $(n_P \times n_T)$ *matrix whose element of the i-th row and j-th column is a positive integer equal to the weight of the arc connecting place* p_i *to* t_j*, if such arc exists, or zero, otherwise;*
- $Post$ *is a* $(n_P \times n_T)$ *matrix whose element of the i-th row and j-th column is a positive integer equal to the weight of the arc connecting transition* t_j *to* p_i*, if such arc exists, or zero, otherwise.*

When an arc originates from a place $p_i \in P$ (resp. transition $t_i \in T$) and is connected to a transition $t_j \in T$ (resp. place $p_j \in P$), p_i (resp. t_i) is an input

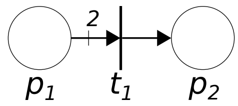


Figure 2.1: Petri net example.

place of t_j (resp. input transition of p_j). The set of input places of t_j (resp. input transitions of p_j) are denoted by $I(t_j)$ (resp. $I(p_j)$). Likewise, t_j (resp. p_j) is an output transition of p_i (resp. output place of t_i), while the set of output transitions of p_i (resp. output places of t_i) are denoted by $O(p_i)$ (resp. $O(t_i)$).

Example 2.6. *A simple Petri net graph is shown in Figure 2.1. Its structure is defined by* $P = \{p_1, p_2\}$, $T = \{t_1\}$, $Pre(p_1, t_1) = 2$ *and* $Post(p_2, t_1) = 1$. *Notice that* $O(p_1) = I(p_2) = \{t_1\}$, $I(t_1) = \{p_1\}$ *and* $O(t_1) = \{p_2\}$

2.3.2 Petri net markings

In order to represent the state that a DES modeled by a Petri net is currently in, each place of the net is associated with a number of tokens, which are represented by small dots inside their associated places. Each possible combination of the number of tokens in each place is considered a single state of the DES, and whenever an event occurs, a transition associated with it changes the number of tokens of places according to the weights of the arcs that connect those places with the associated transition.

The current number of tokens of a place $p \in P$ is called place marking and is represented by the function $m : P \rightarrow \mathbb{Z}_+^{n_P}$, while the Petri net current state, also known as current marking vector, is represented by the column vector $\vec{m} = [m(p_1), m(p_2), \ldots, m(p_{n_P})]^T$. Notice that the current marking vector represents the Petri net state, and thus, whenever the Petri net state changes, the marking vector also changes. Therefore, the importance of a marking vector for the description of the DES dynamic justifies the definition of marked Petri net, as shown below.

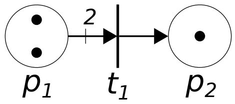


Figure 2.2: Marking Petri net example.

Definition 2.2 (Marked Petri net)**.** *A marked Petri net is a quintuple*

$$\mathcal{N} = (P, T, Pre, Post, \vec{m}_0),$$

where:

- $(P, T, Pre, Post)$ *is a Petri net and*
- $\vec{m}_0 \in \mathbb{Z}_+^{n_P}$ *is the initial marking vector.*

Example 2.7. *By adding tokens to the places of the Petri net of Example 2.6, we obtain the marked Petri net depicted in Figure 2.2, in which* $\vec{m}_0 = [2, 1]^T$

2.3.3 Petri net dynamics

In order to model the change of state caused by the occurrence of events of a DES, the Petri net dynamic makes an enabled transition fire, changing the net current state based on a set of rules that adds and removes tokens from places. A transition $t \in T$ is enabled whenever the current numbers of tokens of each place $p \in I(t)$ is greater than or equal to their respective arc weight connecting p to t. Thus, the formal definition of enabled transition is as follows.

Definition 2.3 (Enabled transition of a Petri net)**.** *A transition* $t \in T$ *of a Petri net* $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0)$ *with the current marking vector* $\vec{m}$ *is enabled if the following is true*

$$(\forall p \in I(t)), (m(p) \geq Pre(p, t))$$

Given two vectors of the same length $\vec{a}$ and $\vec{b}$, let $\vec{a} \geq \vec{b}$ (resp. $\vec{a} > \vec{b}$) denote that the i-th element of $\vec{a}$ is greater than or equal to (resp. greater than) the i-th element of $\vec{b}$. Additionally, since *Pre* and *Post* are matrices, we may denote the j-th column of matrix *Pre* as $Pre(:, t_j)$. Thus, another possible way to verify if transition t_j is enabled is by checking if each element of $\vec{m}$ is greater than or equal to the corresponding element of $Pre(:, t_j)$, which is equivalent to checking whether $\vec{m} \geq Pre(:, t_j)$.

Whenever a transition $t \in T$ is enabled, it may fire, removing tokens from its input places and adding tokens to its output places according to the weight of the arcs connecting each place and t. Therefore, when t fires, the new number of tokens of each input place $p_i \in I(t)$ is:

$$m'(p_i) = m(p_i) - Pre(p_i, t),$$

whereas the new number of tokens of each output place $p_o \in O(t)$ is:

$$m'(p_o) = m(p_o) + Post(p_o, t).$$

Thus, the new marking of each place $p \in P$ of the Petri net, given that t has fired, can be described as:

$$m'(p) = m(p) + Post(p, t) - Pre(p, t).$$

The new marking vector $\vec{m}'$ after the firing of t can also be described either by matrices *Pre* and *Post*, or by the incidence matrix $A = Post - Pre$, as follows:

$$\vec{m}' = \vec{m} + Post(:, t) - Pre(:, t) = \vec{m} + A(:, t).$$

Finally, the fundamental equation that describes the Petri net dynamic evolution after through multiple transition firings is given by:

$$\vec{m}' = \vec{m} + A\vec{r}_k,$$

where $\vec{r}_k$ is a column vector whose j-th component corresponds to the number of times transition t_j has fired. Notice that $\vec{r}_k$ can either represent transitions that fire simultaneously or transitions that fire sequentially, the former requires that all transitions in $\vec{r}_k$ must be able to fire simultaneously, *i.e.* there must be enough tokens in $\vec{m}$ in order to fire all transitions, whereas the latter requires a marking $\vec{m}$ that enables the first transition of the sequence, and after that enables the second transitions with the marking $\vec{m}'$, which is generated after the first transition fires, and so forth. In this work, we will assume that only one transition can fire at a time.

Let $r = t_1 t_2 t_3 \dots t_n$ be a sequence of n transitions enabled by $\vec{m}$, *i.e.* t_1 is enabled by $\vec{m}$, t_2 is enabled by the marking generated after the firing of t_1, and so on. The sequential firing of the transitions of r starting from the marking $\vec{m}$ that results in $\vec{m}'$ is denoted by $\vec{m}[r\rangle\vec{m}'$; $\vec{m}[r\rangle$ is used to denote that the sequence r is enabled by $\vec{m}$, meaning that all transitions from r can fire sequentially starting from $\vec{m}$.

Let T^* be all possible transition sequences that can be generated by the transitions of a set T. The set of all finite sequences that are enabled in a Petri net $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0)$ is denoted by:

$$LT(\mathcal{N}) = \{r \in T^* : \vec{m}_0[r\rangle\}.$$

Notice that $LT(\mathcal{N})$ also contains the empty sequence of transitions, denoted by λ and we can use $r_1 \lhd r_2$ to denote that the the transition sequence r_1 can be obtained by removing transitions from the transition sequence r_2.

A marking $\vec{m}$ is reachable in a Petri net $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0)$ if there exists a transition sequence r such that $\vec{m}_0[r\rangle\vec{m}$. The set of all makings that are reachable from $\vec{m}_0$ in $\mathcal{N}$, called reachability set, is denoted by:

$$R(\mathcal{N}) = \{\vec{m} \in \mathbb{Z}_+^{n_P} : (\exists r \in T^*)[\vec{m}_0[r\rangle\vec{m}]\}.$$

It is straightforward from the definition of $R(\mathcal{N})$ that $\vec{m}_0 \in R(\mathcal{N})$. In addition, notice that the reachability set can be infinite if there exists a transition sequence that, after firing, does not decrease the number of tokens of any place of the Petri net current marking and adds at least one token to any place, thus adding tokens indefinitely to the net.

Example 2.8. *Consider the Petri net $\mathcal{N}$ of Figure 2.2. Notice that transition t_1 is enabled, since $m(p_1) = 2 \geq Pre(p_1, t_1)$. If it fires, the new marking will be*

$$\vec{m} = \vec{m}_0 + Post(:, t_1) - Pre(:, t_1) = [2,1]^T + [0,1]^T - [2,0]^T = [0,2]^T.$$

Notice that $LT(\mathcal{N}) = \{\lambda, t_1\}$ and $R(\mathcal{N}_1) = \{[2,1]^T, [0,2]^T\}$.

2.3.4 Petri net operations and properties

The following Petri net operation and properties will be used in this work:

•***T'-induced subnet of $\mathcal{N}$:*** given a Petri net $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0)$ and a subset of transitions T', the T'-induced subnet of $\mathcal{N}$ is $\mathcal{N}' = (P, T', Pre', Post', \vec{m}_0)$, where $Pre' = Pre(:, T')$ and $Post' = Post(:, T')$, with $Pre(:, T')$ and $Post(:, T')$ denoting the matrices composed by the columns of *Pre* and *Post* that are associated with the transitions in T'. The T'-induced subnet $\mathcal{N}'$ is also denoted as $\mathcal{N}' \prec_{T'} \mathcal{N}$.

•***Bounded Petri net:*** a Petri net $\mathcal{N}$ is bounded if $R(\mathcal{N})$ is finite, meaning that all states of the Petri net can be enumerated. If $R(\mathcal{N})$ is infinite, then the Petri net is unbounded instead, rendering it impossible to enumerate all possible states.

•***Deadlock-free Petri net:*** a Petri net $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0)$ is deadlock-free when the following is true:

$$(\forall \vec{m} \in R(\mathcal{N}))(\exists t \in T)[\vec{m}[t\rangle].$$

In words, every state has at least one enabled transition, guaranteeing that the Petri net will always have an enabled transition during its operation.

•***Acyclic Petri net*** a Petri net $\mathcal{N}$ is acyclic if it does not include a directed circuit formed by places and transitions. Among the properties acyclic Petri nets have, we will use the following two: *(i)* the positive elements of A are equal to the

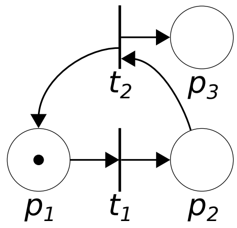


Figure 2.3: Cyclic, unbounded and deadlock-free Petri net

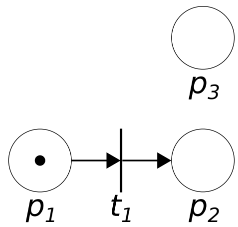


Figure 2.4: Acyclic, bounded Petri net that posseses a deadlock and is the $\{t_1\}$-induced Petri net of the one depicted in Figure 2.3

elements of *Post*, whereas the negative elements of A are equal to the elements of $-Pre$, meaning that an element of *Pre* that is different from zero is zero in *Post* and vice-versa; *(ii)* given a Petri net current marking $\vec{m}$ and vector $\vec{r}_k$, which may have components greater than 1 and satisfies the condition $\vec{m} + A\vec{r}_k \geq \vec{0}$, then there exists a transition sequence $s \in T^*$ composed by the repetitions of the transitions of $\vec{r}_k$, if they exist, such that $\vec{m}[s\rangle$.

Example 2.9. *Consider the Petri net $\mathcal{N}$ of Figure 2.3. As t_1 and t_2 fire sequentially, tokens are added indefinitely to p_3, which means that $R(\mathcal{N}_1)$ is infinite and that the Petri net is unbounded. The Petri net is also cyclic, since it is possible to describe the cycle $p_1t_1p_2t_2p_1$ from the net. Finally, since every marking of the Petri net always has a token in either p_1 or p_2, there won't be a marking in which t_1 and t_2 are both not enabled; therefore, the Petri net is deadlock-free.*

Let $T' = \{t_1\}$. The T'-induced Petri net $\mathcal{N}'$ of the net depicted in Figure 2.3 is shown in Figure 2.4. Notice that t_1 only fires once in $\mathcal{N}'$, moving the token from p_1 to p_2. After the firing of t_1, no transitions are enabled by the current marking, meaning that the Petri net has a limited number of reachable states and possesses a deadlock; thus, the Petri net is bounded and is not deadlock-free. Finally, notice that $\mathcal{N}'$ is acyclic, since there are no transitions connecting p_2 to p_1.

2.3.5 Labeled Petri net

A labeled Petri net is a Petri net whose transitions are associated with the events of a set of events Σ. This addition allows us to define the language $L \subseteq \Sigma^*$ generated by the Petri net, in which every sequence $s \in L$ is associated with a transition sequence $t \in LT(\mathcal{N})$ that may be fired from the initial state of the Petri net. The definition of labeled Petri nets is as follows [1].

Definition 2.4 (Labeled Petri net)**.** *A labeled Petri net is a septuple*

$$\mathcal{LN} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell),$$

where:

- $(P, T, Pre, Post, \vec{m}_0)$ *is a marking Petri net;*
- Σ *is the set of events and*
- $\ell : T \to \Sigma$ *is the transition labeling function.*

Since the firing of a transition in a labeled Petri net is associated with the occurrence of an event, whenever event $\sigma \in \Sigma$ occurs, a transition $t \in T$, such that $\ell(t) = \sigma$, fires as well. In this regard, we make the following assumption.

A1. If multiple transitions that are associated with a same event σ are enabled and σ occurs, only one of the enabled transitions associated with σ fires.

It is worth remarking that Assumption **A1** is usually made in works that use labeled Petri net with multiple transitions labeled by a same event, as seen in [14, 17, 18, 23, 24]. A possible consequence of this assumption is that it becomes necessary to determine which transition fired given that an event has occurred. Such a situation will be referred to as an event conflict, whose definition is inspired by the definition of conflicts for simple Petri nets [2], as follows.

Definition 2.5 (Event conflict)**.** *Let* $\mathcal{LN} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ *be a labeled Petri net,* $\sigma \in \Sigma$ *an event of* $\mathcal{LN}$*,* $\vec{m} \in R(\mathcal{LN})$ *the current marking vector of* $\mathcal{LN}$*. An event conflict is a triple* $\langle \sigma, T_e, \vec{m} \rangle$*, where* $T_e = \{t_1, t_2, \ldots, t_k\} \in 2^T$ *is a set of* k *transitions enabled by* $\vec{m}$ *such that* $(\forall t \in T_e)(\ell(t) = \sigma)$*.*

The sequences of events of a labeled Petri net can be associated with sequences of transitions by extending function $\ell : T \rightarrow \Sigma$ into $\ell : T^* \rightarrow \Sigma^*$. The labeling function ℓ can be further extended to $\ell : 2^{T^*} \rightarrow 2^{\Sigma^*}$ in order to associate a set of sequences of transitions with a set of sequences of events. This extension allows the definition of the prefix-closed language $L(\mathcal{LN})$ composed of all finite sequences of events that can occur on a labeled Petri net $\mathcal{LN} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$, as follows:

$$L(\mathcal{LN}) = \ell(LT(\mathcal{LN})).$$

Given that the events of a DES are associated with occurrences within the system, they can be classified as events that can be observed and events that cannot be be observed. This implies that an external observer can only acknowledge the occurrence of observable events, missing the occurrence of unobservable events. In order to divide these events, the set of events Σ is partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o is the set of observable events, and Σ_{uo} is the set of unobservable events. Based on that, we can define the projection $P_o : \Sigma^* \rightarrow \Sigma_o^*$, which projects the sequences of some language $L \subseteq \Sigma^*$ into Σ_o^* in order to obtain the observed behavior. With that in mind, the observed language of a Petri net $\mathcal{LN}$ is given by $P_o(L(\mathcal{LN}))$.

Since the transitions of a labeled Petri net are associated with events, the concept of observability can also be applied to transitions. Thus, the set of transitions T can be partitioned into $T = T_o \dot{\cup} T_{uo}$, where t_o and t_{uo} are the set of observable and unobservable transitions, respectively. In order to distinguish between the two types of transitions, observable transitions are graphically represented by solid bars, whereas unobservable transitions are represented simply by line segments. If a labeled Petri net is such that T_{uo} is empty, it can also be called a λ-free labeled Petri net, since it only contains observable transitions.

Similar to the projection $P_o : \Sigma^* \rightarrow \Sigma_o^*$, it is also possible to define a mapping of the transition sequences of a set of transitions T into a set of observable transitions T_o by using the mapping function $PT_o : T^* \rightarrow T_o^*$, which is refereed to as a transition projection. The definition of a transition projection PT is similar to the definition of natural projection of events in such a way that a projection $PT : T_l^* \to T_s^*$ of a transition sequence $r \in T_l^*$ into the set T_s^* is defined by the recursion:

$$PT(\lambda) = \lambda$$

$$PT(t) = \begin{cases} t, & \textit{if } t \in T_s \\ \lambda, & \textit{if } t \in T_l \backslash T_s \end{cases}$$

$$PT(rt) = PT(r)PT(t), \forall r \in T_l^*, \forall t \in T_l.$$

Finally, the extension of transition projection to a set of transition sequences TS is similar to the extension of natural projection on events, *i.e.*, $PT(TS) = \{r \in T_s^* : (\exists s \in TS)[PT(s) = r]\}$.

Notice that the transition projection $PT_o : T^* \to T_o^*$ can be applied to the set $LT(\mathcal{LN})$ in order to obtain the set of all observed sequences of transitions $s \in T_o^*$. It is not difficult to see that the resulting set of sequences of transitions $PT_o(LT(\mathcal{LN}))$ is such that $\ell(PT_o(LT(\mathcal{LN}))) = P_o(L(\mathcal{LN}))$.

Example 2.10. *Consider the labeled Petri net $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ depicted in Figure 2.5, where, $\Sigma = \{a, b, y, w\}$, $\ell(t_1) = \ell(t_2) = a$, $\ell(t_3) = y$, $\ell(t_4) = w$ and $\ell(t_5) = b$. Additionally, the sets of observable and unobservable events of the Petri net are $\Sigma_o = \{a, b\}$ and $\Sigma_{uo} = \{y, w\}$, respectively. At first, both observable transitions t_1 and t_2 are enabled. Since both are associated with event a, the Petri net currently possesses the event conflict $\langle a, \{t_1, t_2\}, [1, 1, 0, 0, 0, 0]^T \rangle$. In other words, if the occurrence of event a is observed, then either t_1 or t_2 fires. If t_1 fires, for example, the token of p_1 moves to p_3, enabling transition t_3 that is labeled by the unobservable event y. Since y is unobservable, the firing of t_3 cannot be realized by an observer, which means that the observer would not be able to directly infer if t_3 has fired.*

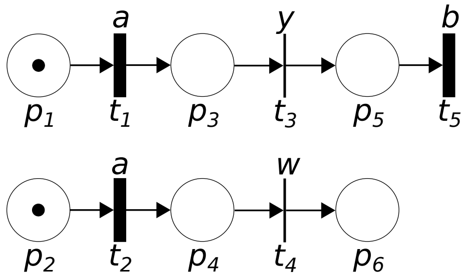


Figure 2.5: Example of a labeled Petri net.

2.3.6 Labeled priority Petri net

A possible approach that may be used to solve the problem of event conflicts presented in Section 2.3.5 is to extend labeled Petri nets to labeled priority Petri nets. This new structure allows the assignment of priorities between transitions associated with the same event so that whenever the Petri net current marking enables those transitions simultaneously, only the transition with the highest priority is allowed to fire when its corresponding event occurs. The definition of labeled priority Petri nets used in this work will be adapted from the definition of priority system presented in [25], as shown below.

Definition 2.6 (Labeled priority Petri net)**.** *A labeled priority Petri net is an octuple*

$$\mathcal{LPN} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell, \rho),$$

where:

- $(P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ *is a labeled Petri net;*
- $\rho \subseteq T \times T$ *is a set of pairs of transitions called priority relation.*

Each priority relation is represented by the pair of transitions $(t, u) \in \rho$ that indicates that transition t has lower priority than transition u; therefore, if the Petri net current marking has enough tokens to fire both t and u, only u would actually be enabled. Based on that idea, the definition enabled transition, as presented in Definition 2.3, changes as follows.

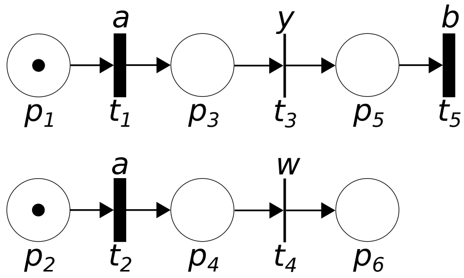


Figure 2.6: Example of a labeled priority Petri net, assuming the priority relation $\rho = \{(t_1, t_2)\}$.

Definition 2.7 (Enabled transition of a labeled priority Petri net)**.** *A transition $t \in T$ of a labeled priority Petri net $\mathcal{LPN} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell, \rho)$ with the current marking vector $\vec{m}$ is enabled if the following are true*

$$(\forall p \in P)(m(p) \geq Pre(p,t)) \text{ and}$$

$$(\forall t_u \in T \backslash \{t\})(\exists p \in P)\,[(t, t_u) \in \rho \implies m(p) < Pre(p, t_u)]$$

In words, Definition 2.7 states that for a transition t to be enabled, not only the Petri net current marking must have enough tokens to fire t, but also the current marking cannot allow any other transition t_u that has higher priority than t to fire.

Example 2.11. *By adding the priority relation $\rho = \{(t_1, t_2)\}$ to the Petri net presented in Example 2.10, we obtain the labeled priority Petri net depicted in Figure 2.6. Notice that the new Petri net no longer has the event conflict $\langle a, \{t_1, t_2\}, [1, 1, 0, 0, 0, 0]^T \rangle$, due to the fact that when both transitions t_1 and t_2 are enabled, the priority relation (t_1, t_2) forces t_2 to be the only enabled transition. Thus, if event a is observed, then t_2 fires.*

2.4 Coverability tree of Petri nets

The coverability tree (CT) [2] of a Petri net is a tree whose nodes represent the possible reachable states of the Petri net, and whose edges are arcs connecting those states and represent transitions that may fire from their origin (or parent) states, changing the Petri net state to the destiny (or child) states. This tree allows us to get a good grasp of the state evolution of a Petri net caused by the firing of transitions.

It is worth remarking that even though an unbounded Petri net has an infinite number of states, it is still possible to generate an associated CT with a finite number of nodes. In this case, whenever the number of tokens of a place $p \in P$ grows indefinitely, the value of $\vec{m}(p)$ in the marking vector is replaced by the symbolic marking ω, indicating that the number of tokens of place p can be as large as required. The symbol ω has the following proprieties with respect to integers: $n < \omega$ and $\omega + n = \omega$, $\forall n \in \mathbb{Z}$. This means that a place p for which $\vec{m}(p) = \omega$ will always have enough tokens to enable any of its output transitions $t \in O(p)$ since $Pre(p,t) \in \mathbb{Z}_+$, which implies that $\omega > Pre(p,t)$. In addition, the Petri net dynamic will not affect this place marking since $\omega + Post(p,t) - Pre(p,t) = \omega$.

Algorithm 1 presents a pseudocode for the construction of a CT for labeled priority Petri nets, whose outputs are the matrices *Nodes* and *Arcs*. Each column of matrix *Nodes* is formed with the marking vector of a node of the tree. Element (i,j) of matrix *Arcs* corresponds either to the transition $t \in T$ that labels the edge that starts at the node defined by the i-th column and ends at the node defined by the j-th column of matrix *Nodes*, or zero, otherwise. The central idea of the algorithm is that is starts by assigning the initial marking vector $\vec{m}_0$ to the first node of the tree. After that, for each transition t enabled by $\vec{m}_0$, the algorithm creates a new node as a child of $\vec{m}_0$, and associates the created node with the marking $\vec{m}'$, where $\vec{m}_0[t\rangle\vec{m}'$. In order to verify if the number of tokens of a place of $\vec{m}'$ is growing indefinitely, the algorithm compares $\vec{m}'$ with each marking vectors of its predecessors to check whether the number of tokens of each place of the former is greater than or equal to the number of tokens of each corresponding place of the latter, whenever both places are different from ω. Let $\vec{m}_p$ denote the marking vector of a predecessor of $\vec{m}'$. We say that $\vec{m}' \geq \vec{m}_p$ when, $\forall p \in P$, the following is true:

$$(\vec{m}'(p) \geq \vec{m}_p(p)) \vee (\vec{m}'(p) = \omega \wedge \vec{m}_p(p) = \omega).$$

Based on the above comparison, if the CT algorithm finds a predecessor $\vec{m}_p$ of $\vec{m}'$ such that $\vec{m}' \geq \vec{m}_p$, then, for all places $p \in P$ such that $\vec{m}'(p) > \vec{m}_p(p)$, the algorithm replaces $\vec{m}'(p)$ with ω, preventing the number of tokens of that place from growing indefinitely on the generated tree. Lastly, if all of the predecessors of $\vec{m}'$ are different from $\vec{m}'$, the steps described for the creation of the children of $\vec{m}_0$ are repeated for $\vec{m}'$ in order to find the nodes that may be generated from the node of $\vec{m}'$, otherwise, $\vec{m}'$ is a terminal node (a node that has no children).

Algorithm 1 Algorithm CT to obtain the coverability tree of a labeled priority Petri net

Inputs:

- $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell, \rho)$: labeled priority Petri net model

Outputs:

- *Nodes*: $n_P \times l$ matrix whose columns are the l nodes of the tree
- *Arcs*: $l \times l$ matrix whose element (i, j) corresponds either to the transition $t \in T$ that labels the edge that starts at the node defined by the i-th column and ends at the node defined by the j-th column of matrix *Nodes*, or zero, otherwise

```
Set Nodes ← [m⃗_0]
Set l ← 1
Set Arcs ← [0]
Set nodesToCheck ← [1]
Set parents ← [0]
While nodesToCheck is not empty do
  Set currentNode ← nodesToCheck(1)
  Remove nodesToCheck(1) from nodesToCheck
  For each t such that Nodes(:, currentNode)[t⟩ do
    Set newNode ← Nodes(:, currentNode) + Post(:, t) − Pre(:, t)
    Set currentParent ← currentNode
    While (currentParent ≠ 0) do
      If newNode ≥ Nodes(:, currentParent)
        For each p ∈ P such that newNode(p) > Nodes(p, currentParent)
          Set newNode(p) ← ω
      Set currentParent ← parents(currentParent)
    Set Nodes ← [Nodes, newNode]
    Set parents ← [parents, currentNode]
    Set Arcs ← [[Arcs, 0⃗_{l×1}]^T, 0⃗_{l+1×1}]^T
    Set l ← l + 1
    Set Arcs(currentNode,l) ← t
    Set flag ← True
    While (currentParent ≠ 0) and flag is True do
      If (Nodes(:, currentParent) = newNode)
        Set flag ← False
      Set currentParent ← parents(currentParent)
    If flag is True
      Set nodesToCheck ← [nodesToCheck, l]
```

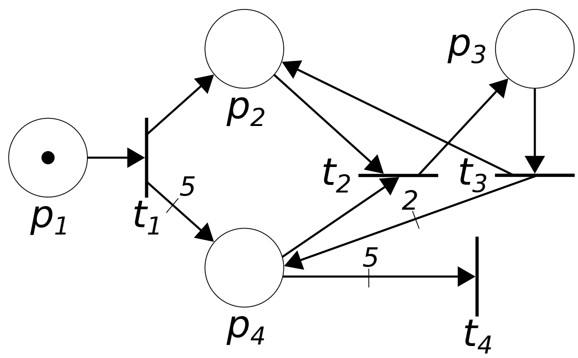


Figure 2.7: Petri net considered in Example 2.12.

Example 2.12. *Consider the Petri net $\mathcal{N}$ of Figure 2.7. If we execute Algorithm 1 with $\mathcal{N}$ as an input, we obtain the CT of $\mathcal{N}$ described by the output matrices Nodes and Arcs, given by*

$$Nodes = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 5 & 4 & 0 & \omega & \omega & \omega & \omega & \omega \end{bmatrix} \quad and$$

$$Arcs = \begin{bmatrix} 0 & t_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & t_2 & t_4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & t_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & t_2 & t_4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & t_3 & t_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Furthermore, we are able to graphically represent the CT, as shown in Figure 2.8

Notice from Figure 2.7 that after transition t_1 fires, the transition sequence t_2t_3 can fire indefinitely, adding a token to p_4 after the firing of the sequence. Since the number of tokens of p_4 grows indefinitely in this case, $\mathcal{N}$ is an unbounded Petri net. However, even though $\mathcal{N}$ is unbounded, the resulting CT of $\mathcal{N}$ is finite. This happens because after the occurrence of the transition sequence $t_1t_2t_3$, which would normally result in the marking vector $\vec{m}' = [0\ 1\ 0\ 6]^T$, the algorithm finds the predecessor $\vec{m} = [0\ 1\ 0\ 5]^T$ of $\vec{m}'$, which is such that $\vec{m}' \geq \vec{m}$ and $\vec{m}'(p_4) > \vec{m}(p_4)$, which causes

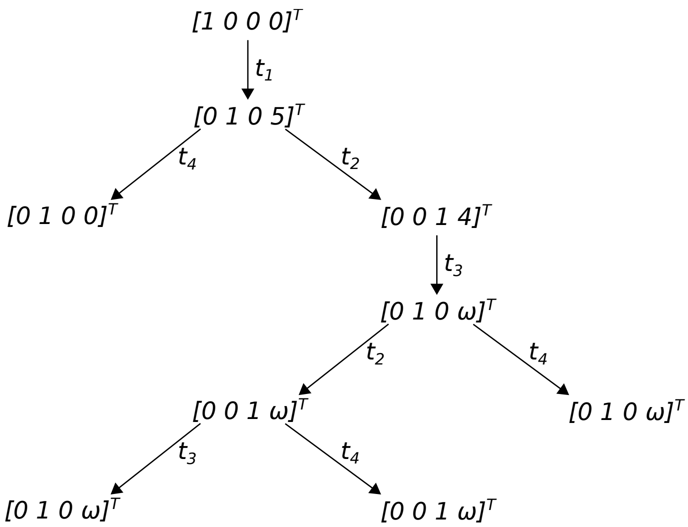


Figure 2.8: Tree generated by the CT algorithm of the Petri net of Figure 2.7.

the marking of p_4 to be replaced by ω, preventing the tree from growing indefinitely.

Chapter 3

Fault diagnosis of discrete event systems

Fault diagnosis of DESs consists in the study of the capability to infer the occurrence of some event of interest of the system, usually the fault event [4]. Notice that if the fault event is not observable, its occurrence can only be inferred by analyzing the occurrence of other events that are observable. In this regard, one of the reasons why this study is important is because it allows the detection of malfunctions during the operation of a system without the need of sensors that directly detects it.

Let $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$ be a partition of the set of events into the observable and unobservable event sets, respectively, and let $\Sigma_f \subseteq \Sigma_{uo}$ be the set of fault events associated with a language L. It can be assumed, without the loss of generality, that there is only one fault event, *i.e.*, $\Sigma_f = \{\sigma_f\}$ [10]. Let L denote the language generated by the system and let $\Psi(\Sigma_f) = \{s \in L : (\exists u \in \Sigma^*)[s = u\sigma_f]\}$ be the set of all sequences of L that end with fault event σ_f. We say that language L is diagnosable with respect to the fault event σ_f if, for every sequence $s_f \in \Psi(\Sigma_f)$, the fault event can be detected without any doubt after the occurrence of an arbitrarily long event sequence s_a by analyzing the observable events of sequences s_f and s_a. Notice that, for the detection of σ_f to be successful, there cannot exist another sequence $s_n \in L$ that does not possess a fault event and generates the same sequence of observation as sequence $s_y = s_f s_a$. The formal definition of language diagnosability is as follows [4].

Definition 3.1 (Diagnosability of a language)**.** *A prefix-closed and live language L is diagnosable with respect to Σ_f and the projection $P_o : \Sigma^* \rightarrow \Sigma_o^*$ if the following condition holds true:*

$$(\forall s \in \Psi(\Sigma_f))(\exists n_s \in \mathbb{N})(\forall t \in L/s)$$

$$((|t| \geq n_s) \Rightarrow ((\forall \omega \in P_o^{-1}(st) \cap L)(\sigma_f \in \omega))).$$

In the upcoming sections of this chapter, we elaborate the problem of fault diagnosability and online diagnosis of DESs modeled by labeled Petri nets. In Section 3.1, we present the changes that we must consider in order to define the diagnosability of Petri net systems. Section 3.2 introduces the concept of online diagnosis of Petri nets systems, and we also elaborate the online diagnoser of Petri nets system proposed by [23].

3.1 Fault diagnosability of DESs modeled by labeled Petri nets

In the context of labeled Petri nets, the language $L(\mathcal{N})$, generated by a labeled Petri net $\mathcal{N}$, is the language to be diagnosed. However, when Definition 3.1 is used to determine whether language $L(\mathcal{N})$ is diagnosable, some problems may occur. Even when $L(\mathcal{N})$ is live and diagnosable with respect to Definition 3.1, the dynamics governed by the transitions of $\mathcal{N}$ may prevent the fault event from being diagnosed. For example, consider the Petri net $\mathcal{N}_1$ depicted in Figure 3.1, whose generated language is $L(\mathcal{N}_1) = \{\epsilon, w, wb, wbb, \dots, \sigma_f, \sigma_f a, \sigma_f aa, \dots\}$, and let the sets of observable, unobservable and fault events be $\Sigma_o = \{a, b\}$, $\Sigma_{uo} = \{w\sigma_f\}$ and $\Sigma_f = \{\sigma_f\}$, respectively. It is trivial to conclude that language $L(\mathcal{N}_1)$ is both live and diagnosable, since events a and b can occur indefinitely after events w and σ_f

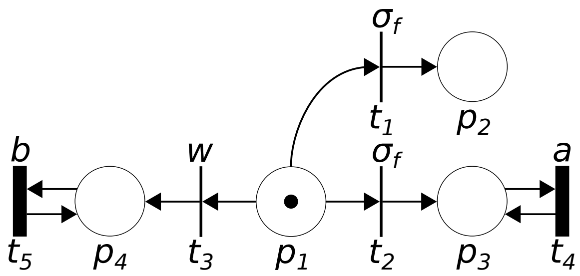


Figure 3.1: Labeled Petri net that generates a language that is live and diagnosable, but is not deadlock-free and prevents the diagnosability of event σ_f.

occur and we are always able to assert that the fault event σ_f has occurred after event a is observed. However, the Petri net $\mathcal{N}_1$ is not deadlock-free, since the firing of transition t_1 moves the token from place p_1 to p_2, which does not enable any transition. Furthermore, notice that transition t_1 models the occurrence of the fault event σ_f; therefore, when t_1 fires, we are not able to diagnose the occurrence of σ_f, since no transitions fire in the sequel.

Thus, in order to be able to refer to the diagnosability of systems modeled by labeled Petri nets without the occurrences of problems such as the aforementioned one, a new definition of diagnosability that is used specifically for systems modeled by labeled Petri nets has been proposed [14]. In [14], the event sequences s and t of Definition 3.1 are replaced by transition sequences r and u, as follows [14].

Definition 3.2 (Diagnosability of a Petri net)**.** *A prefix-closed and live language $L(\mathcal{N})$ generated by a deadlock-free labeled Petri net $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ is diagnosable with respect to Σ_f and the projection $P_o : \Sigma^* \to \Sigma_o^*$ if the following condition holds true:*

$$(\forall r \in LT(\mathcal{N}))(\exists n_r \in \mathbb{N})(\forall u \in T^*)$$

$$(((\ell(r) \in \Psi(\Sigma_f)) \wedge (ru \in LT(\mathcal{N})) \wedge (|u| \geq n_r)) \Rightarrow D),$$

$$\textit{where } D = ((\forall \omega \in P_o^{-1}(\ell(ru)) \cap L(\mathcal{N}))(\sigma_f \in \omega)).$$

In words, Definition 3.2 states that a labeled Petri net system is diagnosable if the following condition is true: for every transition sequence r of the labeled Petri net

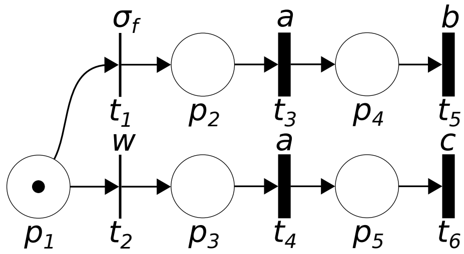


Figure 3.2: Example of a labeled Petri net that is not deadlock-free, but can be diagnosed.

that can be fired from the initial marking and that ends with a fault event, there is a natural number n_r such that for all sequences of transitions u longer than or with the same length as n_r that can be fired after r, is such that all event sequences ω that have the same projection over Σ_o^* as event sequence $\ell(ru)$ have event σ_f inside it.

Remark 3.1. *Although Definition 3.2 requires the labeled Petri net to be deadlock-free, it does not imply that a labeled Petri net with deadlocks cannot be diagnosed. Consider the labeled Petri net $\mathcal{N}$ depicted in Figure 3.2, in which σ_f is the fault event. After the observation of event a, it is not possible to be sure whether the fault event σ_f has occurred or not, since it cannot be confirmed which one of the transition sequences t_1t_3 or t_2t_4 fired. However, if either events b or c are observed, we become certain of which one of the above transition sequences fired before the occurrence of events b or c. If event c is observed, it means that a token was added to place p_5 by the firing transition sequence t_2t_4 to enable transition t_6, whereas if b is observed, it means that a token was added to place p_4 by the transition sequence t_1t_3 to enable transition t_5. Therefore, since the occurrence of the fault event can be inferred without doubt after the occurrence of a finite number of event observations, $\mathcal{N}$ is diagnosable. With this remark, we can conclude that for a Petri net with deadlocks to be diagnosable, the parts of the Petri net that do not contain deadlocks must satisfy the properties of Definition 3.2 and the situations in which the deadlock occur must be such that we are able to confirm whether the fault event has occurred or not.*

The problem of verifying whether a fault event of a labeled Petri net is diagnosable has been studied by several works such as [14], in which is proposed a computational procedure based on linear programming to solve the problem of diagnosability for potentially unbounded labeled Petri nets. Notice, however, that the focus of the present work is to develop of an online diagnoser that detects the fault event of DESs modeled by diagnosable labeled Petri nets; therefore, the online diagnoser proposed here will not verify if the Petri nets to be diagnosed are diagnosable. We will thus assume diagnosability a priori.

3.2 Online diagnosis of labeled Petri nets

Whenever a labeled Petri net is diagnosable, we may be able to infer the occurrence of a fault event during its operation by using an online diagnoser, where it is an algorithm that is able to detect whether a fault event has occurred by observing the occurrence of each observable event of the DES modeled by the labeled Petri net during its operation.

Since the online diagnoser runs simultaneously with the physical plant, it is imperative that the online computation involved in the decision process to detect the occurrence of the fault event given the occurrence of an observable events runs as fast as possible in order to keep up with the occurrence of events in the plant. To this end, online diagnosers usually move most of their burdensome computation to the offline part of the algorithm.

Between the works mentioned in Chapter 1 that computes the online diagnosis of Petri net systems, notice that the online diagnosers of [16, 17] consider the observation of the number of tokens of some of the places of the Petri net, whereas the online diagnosers of [19, 20] are limited to the diagnosis of acyclic or reversible Petri net systems, [15] is limited to bounded Petri nets and [21, 22] require that the Petri net system to be diagnosed does not have two or more observable events that share the same label. Therefore, among the works [15–23], only [18, 23] propose online diagnosers that are able to diagnose the language of any Petri net system that does not possess cycles of unobservable transitions, *i.e.*, the T_{uo}-induced Petri net of the Petri net system is acyclic.

Since the online diagnoser to be proposed in this work has the same assumptions as [18, 23], the online diagnoses of both works are better suited to be compared to the one to be implemented here. However, the online diagnoser proposed in [23] is more advanced that the one proposed in [18], since the online diagnoser of [18] detects the fault events of a Petri net system by enumerating all possible marking vectors that may be reached in the Petri net with the firing of all possible transition sequences labeled by the observed event sequence, whereas the online diagnoser of [23] only considers the basis markings and justifications that are consistent with the event observation, which results in less elements to be analyzed by the online diagnoser. Therefore, we choose the online diagnoser presented in [23] to be compared with the online diagnoser to be proposed here. For this comparison to be successful, we present a brief explanation about the online diagnoser designed for unbounded Petri net systems proposed by [23].

The method for the construction of an online diagnoser capable of diagnosing fault events on Petri nets proposed in [23] is based on the notion of basis markings and justifications. Two types of diagnosers are proposed in [23]: the first one that is able to diagnose unbounded Petri nets whose T_{uo}-induced Petri nets are acyclic and another one that is limited to bounded Petri nets. Although the latter moves most of the burdensome part of the procedure to offline computation, which increases the speed of the online diagnosis, the former is able to diagnose a wider class of Petri net, including unbounded Petri nets. Since the diagnoser proposed here is also able to diagnose unbounded Petri nets, we will only review the online diagnoser designed in [23] for unbounded Petri nets.

Let $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ be a Petri net whose sets of observable and unobservable events are Σ_o and Σ_{uo}, respectively. Given a sequence of observable events $s_o \in P_o(L(\mathcal{N}))$ observed during the dynamic evolution of $\mathcal{N}$, we say that a basis marking $\vec{m}_b$ is a marking that may be reached from the initial marking $\vec{m}_0$

after the firing of a transition sequence $s \in LT(\mathcal{N})$ whose event observation is s_o and unobservable transitions are strictly necessary to enable the observable transitions of s. The group of unobservable transitions of s forms a so-called justification $\vec{j}$, which is a minimal group of unobservable transitions that explains how the observable transitions labeled by the events of s_o are able to change the Petri net marking from $\vec{m}_0$ to $\vec{m}_b$. In this regard, $\vec{j}$ is a vector whose components are either the number of repetitions of each unobservable transition in the sequence, or zero, if the unobservable transition does not appear in the sequence. Notice that s_o and $\vec{m}_b$ may be associated with multiple justifications, due to the Petri net being able to possess multiple transitions associated with the same label and the possibility of multiple sets of unobservable events justifying the firing of the same transition.

Since the T_{uo}-induced Petri net of the Petri net to be diagnosed is assumed to be acyclic, we are able to enumerate the groups of unobservable transitions that may strictly justify each observable sequence of the Petri net. Therefore, the online diagnosis of a Petri net may consist of computing the possible pairs of basis markings and justifications that the Petri net may reach after each event observation, so that the online diagnoser is able to verify the occurrence of a fault event by checking whether those justifications contain fault transitions.

Before the occurrence of any observable events, the online diagnoser considers that the only possible basis marking that the Petri net may be in is the initial marking $\vec{m}_0$, and the justification for that basis marking is associated with an empty transition sequence λ. After the observation of an observable event σ_o, for each current possible basis marking $\vec{m}_b$ and its associated justification $\vec{j}$ that the diagnoser is currently in, the diagnoser computes all possible basis marking $\vec{m}'_b$ that may be reached after firing a sequence of transitions $s_{uo}t_o$, where s_{uo} is an unobservable transition sequence that strictly justifies the change from marking $\vec{m}_b$ to $\vec{m}'_b$ after the firing of a transition t_o labeled by σ_o. The diagnoser considers that all markings $\vec{m}'_b$ are the new possible basis markings that the Petri net may be in after the firing of σ_o, and the justification $\vec{j}\,'$ associated with $\vec{m}'_b$ is equal to the previous justification

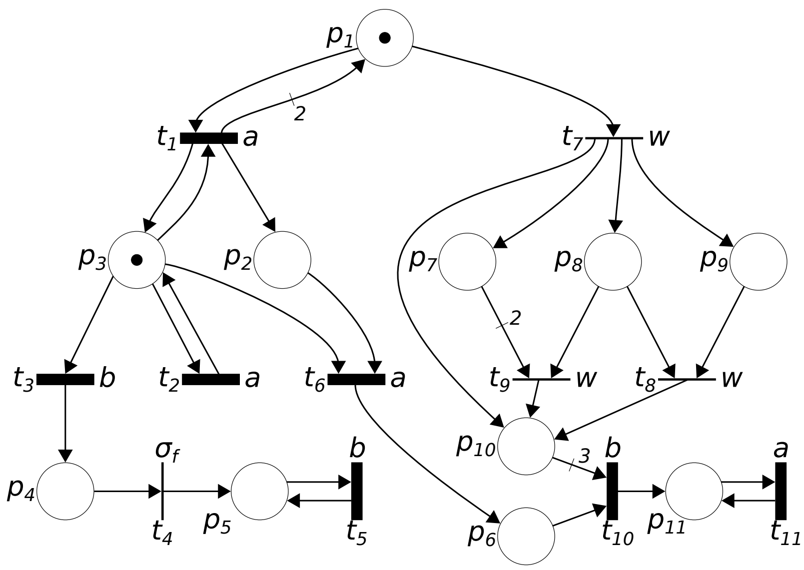


Figure 3.3: Petri net of Example 3.1.

$\vec{j}$, which is associated with $\vec{m}_b$, with the addition of the transitions of s_{uo}.

After computing a new set of basis markings and justifications based on the observation of an event σ_o, the algorithm verifies if the fault event has certainly occurred by verifying whether every justification contains a fault transition.

The following example illustrates the operation of the online diagnoser proposed in [23].

Example 3.1. *Consider the Petri net of Figure 3.3, where* $\Sigma_o = \{a, b\}$, $\Sigma_{uo} = \{w, \sigma_f\}$ *and* $\Sigma_f = \{\sigma_f\}$. *Before the observation of any events, the only basis marking that the Petri net may be in is* $\vec{m}_1$, *and the corresponding justification is* $\vec{j}_1$, *which are shown in Tables 3.1 and 3.2, respectively.*

After the observation of event a, either transitions t_1 *or* t_2 *could have fired from* $\vec{m}_1$ *without the need of the firing of any unobservable transition. Therefore, the diagnoser changes its current basis markings to* $\vec{m}_1$ *and* $\vec{m}_2$, *both of which are associated with justification* $\vec{j}_1$, *since both markings may be reached without the need to fire any unobservable transitions. If event a is observed again, we obtain, from basis marking* $\vec{m}_1$, *basis markings* $\vec{m}_1$ *and* $\vec{m}_2$, *which are associated with* $\vec{j}_1$. *However, we also obtain, from* $\vec{m}_2$, *basis markings* $\vec{m}_2$, $\vec{m}_3$ *and* $\vec{m}_4$, *also associated with* $\vec{j}_1$, *since* $\vec{m}_2$ *enables transitions* t_1, t_2 *and* t_6. *Therefore, the basis markings to be considered*

Table 3.1: The basis markings considered in the example.

Label	Basis marking
$\vec{m}_1$	$[1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$
$\vec{m}_2$	$[2\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$
$\vec{m}_3$	$[3\ 2\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$
$\vec{m}_4$	$[2\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0]^T$
$\vec{m}_5$	$[1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$
$\vec{m}_6$	$[2\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$
$\vec{m}_7$	$[3\ 2\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$
$\vec{m}_8$	$[0\ 0\ 0\ 0\ 0\ 0\ 2\ 1\ 1\ 0\ 1]^T$
$\vec{m}_9$	$[0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 2\ 0\ 1]^T$
$\vec{m}_{10}$	$[1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]^T$
$\vec{m}_{11}$	$[2\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]^T$
$\vec{m}_{12}$	$[3\ 2\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]^T$

Table 3.2: The justifications considered in the example.

Label	t_4	t_7	t_8	t_9
$\vec{j}_1$	0	0	0	0
$\vec{j}_2$	0	2	1	0
$\vec{j}_3$	0	2	0	1
$\vec{j}_4$	1	0	0	0

by the diagnoser changes to $\vec{m}_1$, $\vec{m}_2$, $\vec{m}_3$ *and* $\vec{m}_4$, *all of which are associated with* $\vec{j}_1$.

Remark 3.2. *As shown in Example 3.1, the number of possible basis markings that the diagnoser considers after each observation of event* a *grows with respect to the previous number of possible basis markings, since the firing of transition* t_2 *does not change the Petri net marking, whereas the firing of transition* t_1 *adds tokens to places* p_1 *and* p_2. *Therefore, if the diagnoser keeps observing event* a, *the number of basis markings that the diagnoser has to consider will grow indefinitely, forcing the diagnoser to evaluate a considerable amount of basis markings. As a consequence, there will be an increase in the computational time of the online diagnoser after each observation. This drawback that occurs with the Petri net of Figure 3.3 will not be present in the approach we will propose in Chapter 4.*

If we consider that the currently possible basis markings are $\vec{m}_1$, $\vec{m}_2$, $\vec{m}_3$ *and* $\vec{m}_4$, *all of them associated with* $\vec{j}_1$, *and event* b *is observed, then either transition* t_3 *has fired from* $\vec{m}_1$, $\vec{m}_2$ *or* $\vec{m}_3$, *resulting in the basis markings* $\vec{m}_5$, $\vec{m}_6$ *or* $\vec{m}_7$, *all of them associated with* $\vec{j}_1$, *or sequences* $t_7t_7t_8t_{10}$ *or* $t_7t_7t_9t_{10}$ *has fired from* $\vec{m}_4$. *Sequence* $t_7t_7t_8t_{10}$ *results in the basis marking* $\vec{m}_8$ *associated with justification* $\vec{j}_2$ *and* $t_7t_7t_9t_{10}$ *results in the basis marking* $\vec{m}_9$ *associated with justification* $\vec{j}_3$. *Notice that justification* $\vec{j}_2$ *(resp.* $\vec{j}_3$*) is created by adding the unobservable transitions* t_7, t_7 *and* t_8 *(resp.* t_7, t_7 *and* t_9*) to* $\vec{j}_1$. *It is worth remarking that if the current marking is either* $\vec{m}_5$, $\vec{m}_6$ *or* $\vec{m}_7$, *transition* t_4, *which is associated with the fault event* σ_f, *is enabled; therefore, the fault event may occur between the first observation of event* b *and the next event observation. The diagnoser proposed in [23] is able to detect if transition* t_4 *could have fired from those markings before another event observation by analyzing the unobservable transition sequences that are enabled by them, therefore allowing the diagnoser to conclude that the fault event could have occurred before the next event observation. For a more complete explanation on the aforementioned procedure, readers are referred to [23].*

Based on the above discussion, $\vec{m}_5$, $\vec{m}_6$, $\vec{m}_7$, $\vec{m}_8$ *are the current possible basis* *markings after the first observation of event b, where the first three basis markings are associated with* $\vec{j}_1$*,* $\vec{m}_8$ *is associated with* $\vec{j}_2$ *and* $\vec{m}_9$ *is associated with* $\vec{j}_3$*. In this case, two scenarios are possible: (i) if event b is observed again, then* t_4t_5 *is the only sequence that could have fired; (ii) if event a is observed, then the only transition that could have fired is* t_{11}*. Notice that, in case (i), sequence* t_4t_5 *can only fire from* $\vec{m}_5$*,* $\vec{m}_6$ *or* $\vec{m}_7$*, updating the current basis markings to* $\vec{m}_{10}$*,* $\vec{m}_{11}$ *or* $\vec{m}_{12}$*, all of them associated with the justification* $\vec{j}_4$ *that contains the transition* t_4*. Since all current justifications contain transition* t_4*, which is associated with the fault event, the diagnoser is sure that the fault event has occurred. In case (ii), transition* t_{11} *could only have fired from the basis markings* $\vec{m}_8$ *and* $\vec{m}_9$*, and since* t_{11} *does not change the Petri net marking after firing, the new possible basis markings are reduced to* $\vec{m}_8$ *and* $\vec{m}_9$*, which are associated with justifications* $\vec{j}_2$ *and* $\vec{j}_3$*, respectively. Notice that both* $\vec{j}_2$ *and* $\vec{j}_3$ *do not contain a transition associated with a fault event, and together with the fact that it is not possible to fire transition* t_4 *from the markings* $\vec{m}_8$ *or* $\vec{m}_9$*, the diagnoser is sure that no fault events has occurred in this scenario.*

Chapter 4

Online diagnoser of labeled Petri nets based on λ-free labeled priority Petri nets

This work approaches the computation and usage of online diagnosers of labeled Petri nets by creating a λ-free labeled priority Petri net, called diagnoser Petri net, that has a similar behavior as the Petri net to be diagnosed without using unobservable transitions, *i.e.*, whenever an observable transition t_o of the Petri net to be diagnosed fires after the firing of an unobservable transition sequence s_{uo} whose added tokens contributed to the firing of t_o, one of the transitions t'_o of the diagnoser Petri net that has the same label as t_o also fires, causing the same exchange of tokens in the diagnoser Petri net as the firing of sequence $s_{uo}t_o$ does in the Petri net to be diagnosed. By analyzing the resulting reachable markings of the diagnoser Petri net of the sequences of transitions consistent with the event observation of the Petri net to be diagnosed, we will be able to give a verdict about the fault occurrence.

Let $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ be the labeled Petri net to be diagnosed, where $T = T_o \dot\cup T_{uo}$ and $\Sigma = \Sigma_o \dot\cup \Sigma_{uo}$. In order for the computation of the diagnoser Petri net $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ proposed in this work to be successful, the following assumptions are made besides Assumption A1:

A2. The Petri net is diagnosable;

A3. The T_{uo}-induced subnet of $\mathcal{N}$ is acyclic.

Notice that assumption **A2** is important because the computation of the diagnoser does not verify the diagnosability of the Petri net, and assumption **A3** is a consequence of one of the algorithms used to create the diagnoser Petri net, which is unable to compute cycles of unobservable transitions.

In order for to analyze multiple reachable markings of the diagnoser Petri net $\mathcal{N}\mathcal{D}$ to give a verdict about the fault occurrence, we will propose a new approach for the state estimation of λ-free labeled Petri nets, where we modify the diagnoser Petri net to solve the event conflicts of $\mathcal{N}\mathcal{D}$ in such a way that each sequence of events of $\mathcal{N}\mathcal{D}$ will only label one sequence of transitions, and by firing this transition sequence, we obtain a marking vector that is able to represent multiple markings that $\mathcal{N}\mathcal{D}$ could be in after the firing of transitions consistent with the event observations before the solution of the event conflicts. Therefore, after solving the event conflicts of $\mathcal{N}\mathcal{D}$, we will only require the analysis of one marking vector in order to infer the fault occurrence. It is worth remarking that we can either solve the event conflicts of $\mathcal{N}\mathcal{D}$ during the online diagnosis, where the event conflicts are solved as they occur due to the event observations, or we can solve all event conflicts of $\mathcal{N}\mathcal{D}$ during the offline computation of the diagnoser. Although the former approach results in a slower online diagnosis, it may not be possible to solve all event conflicts of $\mathcal{N}\mathcal{D}$ using our approach, whereas the latter approach can only be used for a class of diagnoser Petri nets, as will be shown in Section 4.3.

In Section 4.1, we propose two algorithms that obtain the initial diagnoser Petri net. After that, Section 4.2 presents algorithms that modify the diagnoser Petri net in order to solve its event conflicts, which allows us to estimate multiple states of the original diagnoser Petri net by analyzing a single state of the modified diagnoser Petri net and, based on that state, make some conclusion about the occurrence of a fault event. Lastly, in Section 4.3, we make an additional assumption with respect to the diagnoser Petri net that allows us to solve all event conflicts of the diagnoser

Petri net during the offline computation of the diagnoser Petri nets, thus optimizing the online diagnosis of the original Petri net using the diagnoser Petri net.

4.1 Obtaining the diagnoser Petri net

Given a labeled Petri net $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ to be diagnosed and the sets Σ_o, Σ_{uo}, Σ_f of observable, unobservable and fault events, respectively, the first step in the construction of the diagnoser Petri net is to generate a λ-free labeled priority Petri net $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$, whose behavior is similar to $\mathcal{N}$, but instead of having unobservable transitions, $\mathcal{ND}$ has multiple instances of each observable transition of $\mathcal{N}$ in such a way that the set of sequences of transitions that may fire in $\mathcal{ND}$ are equivalent to the set of observable transition sequences that may fire in $\mathcal{N}$. The algorithm also creates a special set of transitions T_{fv}, which are labeled by a new event σ_{fv} and allows the diagnoser to verify whether a fault event could have occurred before the occurrence of another observable event that confirms it by checking whether one of the transitions of T_{fv} is enabled.

The computation of the transitions of the diagnoser Petri net relies on the computation of special markings of Petri net $\mathcal{N}$, termed minimal markings, which are presented in the following subsection.

4.1.1 Minimal markings

We define a marking vector $\vec{m}_{min}$ of a labeled Petri net $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$, whose set of unobservable events is Σ_{uo}, as a minimal marking of a given transition $t \in T$ if $\vec{m}_{min}$ only has the tokens strictly necessary to enable transition t after the firing of a minimal sequence of unobservable transitions, which is formally defined as follows.

Definition 4.1 (Minimal marking). *Let $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ be a labeled Petri net, where $T = T_o \dot{\cup} T_{uo}$, and let $t \in T$ be a transition. A marking vector $\vec{m}_{min} \in \mathbb{Z}_+^{n_P}$ of $\mathcal{N}$ is a minimal marking of t if there exists a minimal unobservable* *transition sequence $s_{uo} \in T_{uo}^*$ such that $\vec{m}_{min}[s_{uo}t\rangle$ and the following is true:*

$$(\forall \vec{m}_b \in \mathbb{Z}_+^{n_P})(\forall s_b \in T_{uo}^*),$$

$$(((\vec{m}_b \leq \vec{m}_{min}) \wedge (s_b \lhd s_{uo}) \wedge (\vec{m}_b[s_b t\rangle)) \implies (\vec{m}_b = \vec{m}_{min} \wedge s_b = s_{uo}))$$

In words, each minimal marking $\vec{m}_{min}$ is associated with a minimal unobservable transition sequence s_{uo} whose firing enables t. Additionally, $\vec{m}_{min}$ and s_{uo} are such that for all marking vectors $\vec{m}_b$ that are less than or equal to $\vec{m}_{min}$ and all unobservable transition sequences s_b that are equal to s_{uo} except for some or none of its transitions removed, with $\vec{m}_b$ enabling t through the firing of s_b, then, either $\vec{m}_b$ and s_b are equal to $\vec{m}_{min}$ and s_{uo}, respectively. Additionally, notice that a minimal marking does not necessaryly need to be a reachable marking of the Petri net, meaning that minimal markings are only associated with the structures of the Petri nets, *i.e* the places and transitions of the Petri nets.

In order to make the definition of minimal markings cleaner, we present the following example.

Example 4.1. *Consider the Petri net of Figure 4.1, whose observable and unobservable events are $\Sigma_o = \{a, b\}$ and $\Sigma_{uo} = \{w, \sigma_f\}$, respectively. Both marking vectors $\vec{m}_1 = [1\ 0\ 0\ 0\ 0\ 0]^T$ and $\vec{m}_2 = [0\ 1\ 0\ 0\ 0\ 0]^T$ are minimal markings of the observable transition t_2, being associated with unobservable transition sequences t_1 and λ, respectively. Notice that marking vector $\vec{m}_3 = [1\ 1\ 0\ 0\ 0\ 0]^T$ cannot be considered a minimal marking of transition t_2, since its minimal unobservable transition sequence whose firing enables t_2 is λ, and the marking vector $\vec{m}_2$, which is less than $\vec{m}_3$, is already a minimal marking associated with λ.*

Remark 4.1. *Since $\vec{m}_{min}$ is defined over $\mathbb{Z}_+^*$ instead of $R(\mathcal{N})$, it is possible that a minimal marking of a transition is not in the reachable markings of a Petri net. For example, marking $\vec{m} = [0\ 0\ 0\ 0\ 1\ 1]^T$ is a minimal marking that enables transition t_5 of the Petri net of Figure 4.1 and is associated with the empty sequence λ, but $\vec{m}$ is not a reachable marking of the Petri net.*

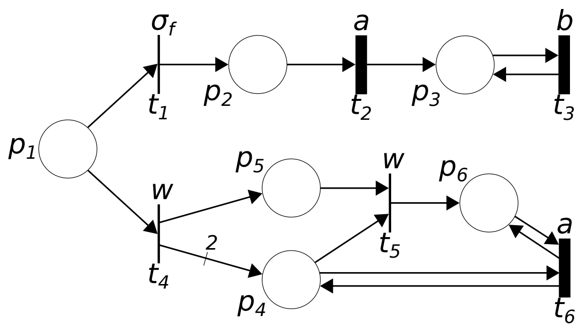


Figure 4.1: Petri net considered in Example 4.1.

If we find all minimal markings that enables a transition t after the firing of minimal unobservable transition sequences, those minimal markings will represent all possible minimal combinations of tokens that allows us to fire transition t after the firing of minimal unobservable transition sequences. This allow us to compute all combinations of tokens that can be used to fire the observable transitions of the Petri net to be diagnosed, which is an information that can be used along the unobservable transition sequences associated with the minimal markings to create the transitions of the diagnoser Petri net.

We now propose algorithm *FAM* (Algorithm 2) to compute all minimal markings and their corresponding unobservable transition sequences for a given transition $t \in T$ of a Petri net $\mathcal{N}$. Since function *FAM* has a complex structure, readers may prefer to follow Example 4.2 to have a better understanding about the way function *FAM* works.

All possible minimal markings $\vec{m}_{min}$ found by the function are stored in matrix $mark_C$, where each column represents a possible minimal marking of t and each row represents each place $p \in P$. The function also creates matrix $mark_S$, whose i-th column is associated with the i-th column of $mark_C$, and whose rows are associated with the transitions of T. For each minimal marking stored in the i-th column of $mark_C$, the function stores the number of repetitions of each transition $t_{uo} \in T$ in the minimal unobservable transition sequence s_{uo} associated with the minimal marking by making $mark_S(t_{uo}, i)$ equal to the number of times that t_{uo} fires in s_{uo}. It is worth remarking that, although the function does not directly process s_{uo}, we are able to

Algorithm 2 Algorithm for the function FAM that obtains all possible minimal markings that eventually enable transition t together with the groups of transitions of the minimal unobservable transition sequences that enable t after firing from the minimal markings

Inputs:

- $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$: labeled Petri net model
- Σ_{uo}: set of unobservable events
- t: transition $t \in T$

Outputs:

- $mark_C$: matrix whose columns represent all minimal markings that eventually enable t after the firing of unobservable transitions
- $mark_S$: matrix whose columns are associated with the columns of $mark_C$ and represent the minimal groups of transitions that compose the minimal unobservable transition sequences s_{uo} that enable t after firing from their associated minimal markings represented in $mark_C$. Each element of a column of $mark_S$ is associated with a transition of T and is equal to the number of times that transition fires in s_{uo}

Set $mark_C \leftarrow [Pre(:, t)]$
Set $mark_S \leftarrow [\vec{0}_{n_T \times 1}]$
Set $T_{b,t} \leftarrow \{t_{uo} \in T : (\ell(t_{uo}) \in \Sigma_{uo}) \wedge (O(t_{uo}) \cap I(t) \neq \emptyset)\}$
Set $\overrightarrow{mul}_{max}$ as a vector that initially associates each transition of $T_{b,t}$ with zero
For each $t_{uo} \in T_{b,t}$ do
    Set $P_{com} \leftarrow O(t_{uo}) \cap I(t)$
    Set $mul \leftarrow 0$
    Do
        Set $mul \leftarrow mul + 1$
        For each possible combination of the repetitions of each transition $t_b \in T_{b,t}$ varying from zero to $\overrightarrow{mul}_{max}(t_b)$, whose repetitions are represented by vector $\overrightarrow{rep}_b$ that associates each element of $T_{b,t}$ with a non negative integer, do
            Set $\vec{r}_b \leftarrow \vec{0}_{n_T \times 1}$
            Set $\vec{r}_b(t_{uo}) \leftarrow mul$
            For each $t_b \in T_{b,t} \backslash \{t_{uo}\}$, set $\vec{r}_b(t_b) \leftarrow \overrightarrow{rep}_b(t_b)$
            Set $flag \leftarrow True$
            For each $t_b \in T_{b,t}$ such that $(\overrightarrow{rep}_b(t_b) > 0) \vee (t_b = t_{uo})$ do
                Set $\vec{r}_{-t_b} \leftarrow \vec{r}_b$
                Set $\vec{r}_{-t_b}(t_b) \leftarrow \vec{r}_{-t_b}(t_b) - 1$
                If $positive(Pre(:, t) - positive((Post - Pre)\vec{r}_b)) = positive(Pre(:, t) - positive((Post - Pre)\vec{r}_{-t_b}))$, set $flag \leftarrow False$ and break from the for loop
            If $flag$ is $True$
                Set $\mathcal{NT} = (P, T_T, Pre_T, Post_T, \vec{m}_0, \Sigma_T, \ell_T) \leftarrow \mathcal{N}$
                Create transition t_N and add it to T_T
                Set $Pre_T(:, t_N) \leftarrow positive((Pre - Post)\vec{r}_b)$
                Set $Post_T(:, t_N) \leftarrow \vec{0}_{n_P \times 1}$
                Set $\Sigma_{uo,T} \leftarrow \Sigma_{uo}$
                Create new event σ_N and add it to $\Sigma_{uo,T}$ and Σ_T
                Set $\ell_T(t_N) \leftarrow \sigma_N$
                Set $[\sim, markT_S] \leftarrow FAM(\mathcal{NT}, \Sigma_{uo,T}, t_N)$
                For each column i of $markT_S$ do
                    Set $markT_S(t_{uo}, i) \leftarrow markT_S(t_{uo}, i) + mul$
                    For each $t_b \in T_{b,t}$, set $markT_S(t_b, i) \leftarrow markT_S(t_b, i) + \overrightarrow{rep}_b(t_b)$
                    Set $mark_C \leftarrow [mark_C, positive(Pre(:, t) - (Post - Pre)markT_S(T, i))]$
                    Set $mark_S \leftarrow [mark_S, markT_S(T, i)]$
    While $min(mul \times Post(P_{com}, t_{uo}) - Pre(P_{com}, t)) < 0$
    Set $\overrightarrow{mul}_{max}(t_{uo}) \leftarrow mul$
For each column i of $mark_C$ and $mark_S$ do
    If there exists a column j of $mark_C$ and $mark_S$ such that $mark_C(:, j) \leq mark_C(:, i)$ and $mark_S(:, j) \leq mark_S(:, i)$, remove column i from $mark_C$ and $mark_S$

assert that there is an unobservable transition sequence enabled by the minimal marking that may be composed of the transitions represented in the columns of $mark_S$ since the T_{uo}-induced Petri net of $\mathcal{N}$ is acyclic and a i-th column of $mark_S$ is such that $mark_C(:,i) - (Post - Pre)mark_S(:,i) \geq 0$. It is also important to notice that matrix $mark_C$ can have two or more identical columns depending on whether there exists more than one unobservable transition sequence that leads the minimal marking to transition t.

In steps 1–2 of Algorithm 2, the function finds the trivial minimal marking $\vec{m}_{min}$ associated with sequence λ, which represents the case in which transition t is enabled without the firing of any other unobservable transitions. In order to directly enable transition t, the minimal marking $\vec{m}_{min}$ must be equal to $Pre(:,t)$.

In order to find the other minimal markings that enable transition t, the function analyzes all transitions whose firings directly add tokens to the input places of t, which are grouped into the set $T_{b,t}$ in step 3. Notice that all minimal unobservable transition sequences whose firings contribute to the firing of t must end with transitions of $T_{b,t}$, since the firing of others transitions afterwards would not contribute to the firing of t; therefore, in the steps 4–36, the function finds all minimal markings that enable transition t that are associated with unobservable transition sequences $s_{uo} \in T_{uo}^*$ that end with sequences $s_b \in T_{b,t}^*$. It is worth remarking that function *FAM* only analyzes the sequences $s_b \in T_{b,t}^*$ whose transitions directly contribute to the firing of transition t, meaning that s_b does not have a transition whose firing does not contribute to the firing of t after the firing of other transitions of the sequence. Furthermore, the function analyzes each unobservable transition sequence s_b using the vector $\overrightarrow{rep}_b$, which is created in step 10 and whose elements are associated with the transitions of T and indicate the number of occurrences of each transition in s_b.

For each sequence s_b, the function finds, in steps 11–32, all marking vectors that enable transition t after the firing of unobservable transition sequences that end with s_b by finding all transition sequences $s_a \in T_{uo}^*$ whose firing contribute to the firing of sequence s_b, where the transitions of each sequence s_a is denoted in the function by a column of matrix $markT_S$. In order to find the sequences s_a, function *FAM* creates, in steps 20–26, a temporary Petri net $\mathcal{NT}$ that is a copy of $\mathcal{N}$ but containing a sink transition t_N[1], whose firing consumes the same tokens as the firing of sequence s_b would. Using Petri net $\mathcal{NT}$, the function makes a recursion call of itself in step 27 to find all minimal marking and their associated minimal unobservable transition sequences s_a that enable transition t_N after firing, which are also minimal unobservable transition sequences whose firing contributes to the firing of sequence s_b; therefore, by combining sequences s_a and s_b, we find the unobservable transition sequences $s_{uo} = s_a s_b$ whose firing contribute to the firing of transition t. Finally, the function computes the minimal marking $\vec{m}_{min}$ as the marking vector that contains the tokens strictly necessary to enable transition t after the firing of the unobservable transition sequence s_{uo}.

In the last two steps 35 and 36, the function finds and deletes every marking vectors and their associated sequences of the outputs of the function that either are repeated in the output or are not minimal when compared with others elements, ensuring that all the elements of both outputs are minimal and are valid with respect to Definition 4.1.

We now present example 4.2, which uses function *FAM* to compute all minimal markings of a given transition of a Petri net.

Example 4.2. *Consider the Petri net of Figure 4.2, whose observable and unobservable events are* $\Sigma_o = \{a, b\}$ *and* $\Sigma_{uo} = \{w, \sigma_f\}$*, respectively. We will show how function FAM finds all the minimal markings and groups of unobservable transition sequences associated with the observable transition* t_6*. The minimal markings and their corresponding sequences found by the function are listed in Table 4.1.*

Function FAM first executes steps 1 and 2 to directly compute the trivial minimal marking $\vec{m}_1$*, which is associated with the empty unobservable transition sequence* λ*, meaning that no unobservable transitions fire to enable transition* t_6*. In order to enable transition* t_6 *without the firing of other transitions,* $\vec{m}_1$ *must contain the*

[1] A sink transition does not have any output places.

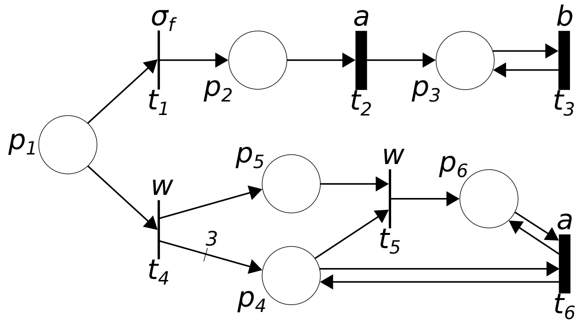


Figure 4.2: Petri net considered in Example 4.1.

Table 4.1: List of minimal markings and their corresponding unobservable transition sequence associated with transition t_6 that are found by function *FAM*.

Label	Minimal marking ($mark_C$)	Unobservable transition sequence ($mark_S$)
$\vec{m}_1$	$[0\ 0\ 0\ 1\ 0\ 1]^T$	$\lambda = [0\ 0\ 0\ 0\ 0\ 0]^T$
$\vec{m}_2$	$[1\ 0\ 0\ 0\ 0\ 1]^T$	$t_4 = [0\ 0\ 0\ 1\ 0\ 0]^T$
$\vec{m}_3$	$[0\ 0\ 0\ 2\ 1\ 0]^T$	$t_5 = [0\ 0\ 0\ 0\ 1\ 0]^T$
$\vec{m}_4$	$[1\ 0\ 0\ 0\ 0\ 0]^T$	$t_4t_5 = [0\ 0\ 0\ 1\ 1\ 0]^T$

tokens necessary to directly enable transition t_6, i.e., $\vec{m}_1$ must be equal to $Pre(:, t_6)$.

After that, the function detects that places p_4 and p_6 are input places of transition t_6 and are also output places of unobservable transitions t_4 and t_5, respectively. Since transitions t_4 and t_6 are the only unobservable transitions whose firing directly contribute to the firing of t_6, then all minimal markings other than the trivial minimal marking must be associated with unobservable transition sequences that ends with either transition t_4, t_5 or with a combination of both. In order to find those minimal markings, the function execute step 3 to add transitions t_4 and t_5 to the set of transition $T_{b,t}$ to analyze them, and then it executes steps 11–32 to compute all minimal markings associated with sequences of unobservable transitions that end with sequences composed by the combinations of transitions t_4 and t_5.

In order to analyze the minimal markings associated with the sequences that end with transition t_4, the function executes steps 11–27 to find all minimal unobservable transition sequences whose firing contributes to the firing of transition t_4, since their firing would also indirectly contribute to the firing of transition t_6 by firing transition

t_4. To find those sequences, the function creates a temporary Petri net $\mathcal{NT}_1$, which is equal to $\mathcal{N}$ except for an additional sink transition t_{N1} that consumes the same tokens as t_4, which consumes one token from place p_1. Since t_{N1} requires the same tokens as t_4 to be enabled, then all minimal unobservable transition sequences whose firing enable t_{N1} also enable t_4; therefore, the function finds all minimal unobservable transition sequences that enable transition t_4 by making a recursive call of itself in step 27 to find all the minimal markings and their associated minimal unobservable transition sequences whose firing enable transition t_{N1}, which also enable transition t_4. The function finds that the only minimal unobservable transition sequence that enables transition t_4 is the empty sequence λ; thus, the only minimal marking that the function can generate that enables transition t_6 after the firing of a minimal unobservable transition sequence that ends with t_4 is t_4 itself. After the execution of steps 28–32, the function finds that the minimal marking associated with t_4 is equal to $\vec{m}_2$, since $\vec{m}_2$ has one token in place p_1 to enable t_4 and another token in place p_6 to enable transition t_6 after the firing of t_4.

Although, according to Algorithm 2, the function would analyze minimal markings associated with sequences that end with transition t_5 after analyzing t_4, for the sake of clarity, we will, in the example, first show how it generates the minimal markings associated with unobservable transition sequences that end with both transitions t_4 and t_5. In order to generate those minimal markings, the function will find the minimal unobservable transition sequences whose firing contribute to the firing of both transitions t_4 and t_5. However, notice that the firing of transition t_4 adds tokens to places p_4 and p_5, which can be consumed by transition t_5; therefore, we only require one token in place p_1 to enable the firing of both transitions t_4 and t_5. To find the minimal unobservable transition sequences whose firing contribute to the firing of both transitions t_4 and t_5, the function again create a temporary Petri net $\mathcal{NT}_2$ that contains an additional transition t_{N2}, where $Pre(:, t_{N2})$ models the tokens required to fire transitions t_4 and t_5 by consuming one token from place p_1. Since t_{N2} is equal to t_{N1} from the previously created temporary Petri net $\mathcal{NT}_2$, the recursion *call of function FAM with respect to $\mathcal{NT}_2$ and t_{N2} results in the same output as the one for $\mathcal{NT}_1$ and t_{N1}, which is a minimal marking associated with sequence λ, which means that there are no other minimal unobservable transition sequences whose firing contribute to the firing of transitions t_4 and t_5. Therefore, the only minimal marking that enables t_6 after the firing of an unobservable transition sequences that ends with both transitions t_4 and t_5 that the function finds is the marking vector $\vec{m}_4$, which has only one token in place p_1, which is enough to enable sequence $t_4t_5t_6$.*

If we execute the aforementioned steps in order to analyze transition t_5, we will find that the the minimal markings that enables t_6 after the firing of minimal unobservable transition sequences that end with transition t_5 are $\vec{m}_3$ and $\vec{m}_4$, which are associated with sequences t_5 and t_4t_5, respectively. Notice that the second minimal marking $\vec{m}_4$ was already found during the analysis of the combination of transitions t_4 and t_5. In order to prevent function FAM from adding repeated or invalid minimal markings and their associated sequences to its output, steps 35 and 36 removes all minimal markings and their associated sequences of the output that are either repeated or are not valid with respect to Definition 4.1 when comparing it to the other minimal markings found by the function.

Remark 4.2. *Although it is difficult to find all minimal markings associated with a transition t of a Petri net $\mathcal{N}$ with function FAM, the idea behind the function is intuitive. If we want to find the minimal markings that enable transition t_6 of the Petri net depicted in Example 4.2, the first one that can easily be found is the trivial minimal marking $\vec{m}_1$, which is associated with the empty transition sequence λ and models the tokens required in places p_4 and p_5 to directly enable t_6. In order to find the other minimal markings, we need to find the unobservable transitions whose firing add tokens to both places p_4 and p_5, since their firing add tokens that are required to fire transition t_6.*

Among the unobservable transitions of the Petri net, only the firing of transitions t_4 and t_5 add tokens to places p_4 and p_5; therefore, we can obtain the other minimal markings of t_6 by analyzing the minimal markings that enable transitions t_4 and t_5.

Since the minimal marking $\vec{m}_{t1} = [1\ 0\ 0\ 0\ 0\ 0]^T$ *enables* t_4 *without the need to fire another transition, it also enables transition* t_6 *after the firing of transition* t_4 *with the addition of a token of place* p_6 *that the firing of* t_4 *does not generate, resulting in the minimal marking* $\vec{m}_2$*. We would also need to verify if there are other unobservable transitions that enable transition* t_4 *after firing. However, there are no transitions that add tokens to place* p_1*, which means that there are no unobservable transition whose firing contributes to the firing of transition* t_4*.*

The same verification can be executed for the combination of transitions t_4 *and* t_5*, which require one token from place* p_1 *to fire since the tokens that are added by the firing of* t_4 *can be consumed by the firing of transition* t_5*. Thus, we find that the minimal marking* $\vec{m}_{t2} = [1\ 0\ 0\ 0\ 0\ 0]^T$ *enables both transitions* t_4 *and* t_5*, and since the firing of both transitions generate enough tokens to enable transition* t_6*, the marking vector* $\vec{m}_4$*, which is equal to* $\vec{m}_{t2}^T$*, is also a minimal marking of transition* t_6*.*

Finally, if we do the same aforementioned steps for transition t_5*, we will also find that the marking vector* $\vec{m}_3$ *is a minimal marking of transition* t_6*.*

4.1.2 Obtaining the diagnoser Petri net

Let $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ be the Petri net to be diagnosed, which is associated with the set of observable events Σ_o, unobservable events Σ_{uo} and fault events Σ_f. The λ-free labeled priority diagnoser Petri net $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ is computed by following the steps of Algorithm 3, which uses function *FAM* to compute all minimal markings and their corresponding unobservable transition sequences associated with every observable transition of the Petri net, which are used by the algorithm to compute all the transitions of the diagnoser Petri net $\mathcal{ND}$ that are required to model the behavior of $\mathcal{N}$ without the use of unobservable transitions. The algorithm also uses the minimal markings of every fault transition obtained by function *FAM* to create the transitions associated with event σ_{fv}, which allows the diagnoser to infer that the fault event could have occurred before the observation of another event by being enabled.

Algorithm 3 Algorithm that obtains the diagnoser Petri net of a labeled Petri net

Inputs:

- $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$: labeled Petri net model
- Σ_o, Σ_{uo}, Σ_f: sets of observable, unobservable and fault events, respectively

Outputs:

- $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$: Diagnoser Petri net

1: Set $\mathcal{ND} \leftarrow (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ as an empty labeled priority Petri net
2: Set $P_D \leftarrow P \cup \{p_f\}$
3: Set $\Sigma_D \leftarrow \Sigma_o \cup \{\sigma_{fv}\}$
4: Set $\vec{m}_{0,D}(P) \leftarrow \vec{m}_0$
5: Set $\vec{m}_{0,D}(p_f) \leftarrow 0$
6: For each $t_o \in T$ s.t. $\ell(t_o) \in \Sigma_o$ do
7:     Set $[mark_C, mark_S] \leftarrow FAM(\mathcal{N}, \Sigma_{uo}, t_o)$
8:     For each corresponding i-th column of $mark_C$ and $mark_S$ do
9:         Create transition t'_o and add it to T_D
10:         Set $Pre_D(P, t'_o) \leftarrow mark_C(:, i)$
11:         Set $Pre_D(p_f, t'_o) \leftarrow 0$
12:         Set $Post_D(P, t'_o) \leftarrow mark_C(:, i) + Post(:, t_o) - Pre(:, t_o) + (Post - Pre)mark_S(:, i)$
13:         Set $Post_D(p_f, t'_o)$ as 1, if $(\exists t_f \in T)[(\ell(t_f) \in \Sigma_f) \wedge (mark_S(t_f, i) > 0)]$, or 0, otherwise
14:         Set $\ell_D(t'_o) \leftarrow \ell(t_o)$
15:         If $(\exists t \in T_D \backslash \{t'_o\})[(Pre_D(:, t'_o) = Pre_D(:, t)) \wedge (Post_D(:, t'_o) = Post_D(:, t)) \wedge (\ell(t'_o) = \ell(t))]$
16:             Remove t'_o from $\mathcal{ND}$
17: For each $t_f \in T$ s.t. $\ell(t_f) \in \Sigma_f$ do
18:     Set $[mark_{f,C}, \sim] \leftarrow FAM(\mathcal{N}, \Sigma_{uo}, t_f)$
19:     Remove all repeated columns of $mark_{f,C}$
20:     For each i-th column of $mark_C$ do
21:         Create transition t_{fv} and add it to T_D
22:         Set $Pre_D(P, t_{fv}) \leftarrow mark_C(:, i)$
23:         Set $Pre_D(p_f, t_{fv}) \leftarrow 0$
24:         Set $Post_D(:, t_{fv}) \leftarrow Pre_D(:, t_{fv})$
25:         Set $\ell_D(t_{fv}) \leftarrow \sigma_{fv}$

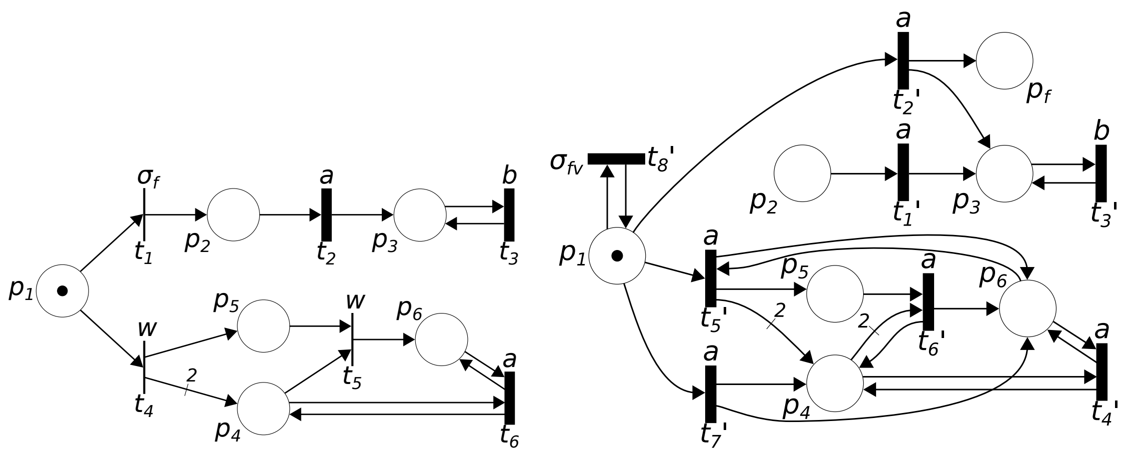
The resulting Petri net $\mathcal{ND}$ contains of all the places and their initial number of tokens of $\mathcal{N}$ and a new place p_f, whose marking indicates that a fault event has occurred, *i.e.*, the diagnoser will be able to confirm the occurrence of a fault event during the system operation whenever $\vec{m}_D(p_f) > 0$. We will assume that place p_f is the last place of the matrices associated with the places of the diagnoser Petri net.

The Algorithm 3 starts by creating in steps 1–5 the diagnoser Petri net $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ as a Petri net that has all the places and their initial marking of the Petri net $\mathcal{N}$, with the addition of a special place p_f, whose initial marking is zero. Additionally, the set of events of the diagnoser Petri net Σ_D has all the observable events of Σ_o and an additional special event σ_{fv}, *i.e.*,

$$\Sigma_D = \Sigma_o \cup \{\sigma_{fv}\}.$$

In order to compute the transitions of the diagnoser Petri net that model the behavior of $\mathcal{N}$, the algorithm executes steps 6–16, where, for each observable transition t_o, the algorithm finds all minimal markings and their associated sequences that enable t_o. Then, for each minimal marking $\vec{m}_{min}$ associated with a minimal unobservable transition sequence s_{uo} that enables t_o after firing, Algorithm 3 creates a transition t'_o in the diagnoser Petri net, that is labeled by the same event that labels transition t_o and whose firing causes the same exchange of tokens in $\mathcal{ND}$ as the firing of sequence $s_{uo}t_o$ causes in $\mathcal{N}$. After iterating every observable transition of $\mathcal{N}$, notice that the firing of the transitions of the diagnoser Petri net $\mathcal{ND}$ are able to model the firing of every possible minimal sequence $s_{uo}t_o$ of $\mathcal{N}$, where $s_{uo} \in T^*_{uo}$ and $t_o \in T_o$; therefore, after each firing of an observable transition in $\mathcal{N}$ after the occurrence of a minimal unobservable transition sequence s_{uo}, it is possible to fire a transition of $\mathcal{ND}$ that models the occurrence of $s_{uo}t_o$, causing $\mathcal{ND}$ to have a similar behavior to $\mathcal{N}$ after the firing of observable transitions. Furthermore, since the firing of each transition t'_o of $\mathcal{ND}$ models the firing of an unobservable transition sequence s_{uo} and an observable transition t_o of $\mathcal{N}$, whenever s_{uo} has a fault event and t'_o fires, we would be able to infer that the fault event has occurred. In order for the firing of t'_o to flag the fault occurrence, the algorithm makes the firing of transition t'_o add a token to place p_f whenever its associated sequence s_{uo} has a fault transition.

Although the transitions of $\mathcal{ND}$ are able to infer the occurrence of a fault event of $\mathcal{N}$ by adding tokens to place p_f after firing, they are unable to assert whether a fault transition could have occurred before the occurrence of a transition $t'_o \in T_D$ that is associated with an unobservable transition sequence that contains the aforementioned fault transition. In order to allow $\mathcal{ND}$ to infer the above possibility, the algorithm executes steps 17–25 to create the special transitions t_{fv}, which are labeled by the special event σ_{fv} and are such that whenever they are enabled, we are able to assert that a fault transition could have fired after the firing of a minimal



(a) (b)

Figure 4.3: Petri net considered in Example 4.3(a) and its resulting diagnoser Petri net(b).

unobservable transition sequence and before the firing of an observable transition that consumes tokens generated by the fault transition. In order to create those transitions, the algorithm finds all of the minimal markings that enables the fault transitions of $\mathcal{N}$, and for each minimal marking $\vec{m}_{min}$, the function creates a transition t_{fv} that is labeled by σ_{fv} and consumes as many tokens as $\vec{m}_{min}$. Notice that transitions t_{fv} will never fire, since we are only creating them to check if they are enabled; therefore, in order to facilitate the analysis of $\mathcal{ND}$ with other tools such as the coverability tree, the algorithm prevents the firing of t_{fv} from changing the Petri net marking by making its firing add the same tokens it consumes.

To further elucidate the construction of the diagnoser Petri net using Algorithm 3, we present the following example.

Example 4.3. *Let* $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ *be the Petri net depicted in Figure 4.3(a), whose observable, unobservable and fault events are* $\Sigma_o = \{a, b\}$, $\Sigma_{uo} = \{w, \sigma_f\}$ *and* $\Sigma_f = \{\sigma_f\}$*, respectively, and let the Petri net* $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ *of Figure 4.3(b) be the resulting diagnoser Petri net of* $\mathcal{N}$.

In order to generate the λ*-free labeled priority Petri net* $\mathcal{ND}$ *from* $\mathcal{N}$*, Algorithm 3 first executes steps 1–5 to create the diagnoser Petri net* $\mathcal{ND}$ *as a labeled priority*

Table 4.2: List of minimal markings, their corresponding unobservable transition sequences and the transition generated in $\mathcal{ND}$ with each minimal marking associated with transition t_2.

Label	Minimal marking	Associated sequence	Generated transition
$\vec{m}_1$	$[0\ 1\ 0\ 0\ 0\ 0]^T$	λ	t'_1
$\vec{m}_2$	$[1\ 0\ 0\ 0\ 0\ 1]^T$	t_1	t'_2

Table 4.3: List of minimal markings, their corresponding unobservable transition sequences and the transition generated in $\mathcal{ND}$ with each minimal marking associated with transition t_3.

Label	Minimal marking	Associated sequence	Generated transition
$\vec{m}_3$	$[0\ 0\ 1\ 0\ 0\ 0]^T$	λ	t'_3

Petri net that contains all the places of $\mathcal{N}$ and an additional place p_f. Notice that the places of $\mathcal{ND}$ that are copies of the places of $\mathcal{N}$ have the same initial tokens, i.e. place p_1 has one token, and place p_f starts with zero tokens.

After step 5, the algorithm executes steps 6–16 to process each observable transitions $t_o \in T$ of the $\mathcal{N}$ in order to create multiple transitions t'_o in the diagnoser Petri net, where each transition t'_o does the same exchange of tokens in $\mathcal{ND}$ as the firing of a possible minimal sequence $s_{uo}t_o$, such that $s_{uo} \in T^*_{uo}$, does in $\mathcal{N}$. In order to create those transitions, the algorithm executes function FAM (Algorithm 2) to compute all minimal markings $\vec{m}_{min}$ that enable t_o after the firing of minimal unobservable transition sequences s_{uo}, which can be used to compute the transitions t'_o in $\mathcal{ND}$ that model the firing of sequence $s_{uo}t_o$ in $\mathcal{N}$. All the minimal markings, their corresponding unobservable transition sequences and each transition generated in $\mathcal{ND}$ with each minimal marking associated with transitions t_2, t_3 and t_6 are shown in Tables 4.2, 4.3 and 4.4, respectively.

Notice that each different pair of minimal markings and sequences generates each transition of the diagnoser Petri net. In order to exemplify the construction of those transitions, we present the constructions of transitions t'_2 and t'_5, which are

Table 4.4: List of minimal markings, their corresponding unobservable transition sequences and the transition generated in $\mathcal{ND}$ with each minimal marking associated with transition t_6.

Label	Minimal marking	Associated sequence	Generated transition
$\vec{m}_4$	$[0\ 0\ 0\ 1\ 0\ 1]^T$	λ	t'_4
$\vec{m}_5$	$[1\ 0\ 0\ 0\ 0\ 1]^T$	t_4	t'_5
$\vec{m}_6$	$[0\ 0\ 0\ 2\ 1\ 0]^T$	t_5	t'_6
$\vec{m}_7$	$[1\ 0\ 0\ 0\ 0\ 0]^T$	$t_4 t_5$	t'_7

constructed from the minimal markings $\vec{m}_2$ and $\vec{m}_5$, respectively.

In order to construct transition t'_2 from the minimal marking $\vec{m}_2$ to model the firing of $t_1 t_2$, the algorithm makes the firing of t'_2 consume the same number of tokens as $\vec{m}_2$ and makes it add the same number of tokens as the firing of transition $t_1 t_2$ from the marking $\vec{m}_2$, which results in a marking that contains one token in place p_3. The algorithm also defines the label of transition t'_2 as the label of transition t_2, which is event a. Finally, since transition t_1 is labeled by the fault event, the algorithm makes the firing of t'_2 add a token to place p_f to flag the fault occurrence.

The process of constructing transition t'_5 is similar to the construction of t'_2. The algorithm labels t'_5 with event a, which is the same label of t_6, and makes the firing of t'_5 consume the same tokens that are in $\vec{m}_6$ and add the tokens of the resulting marking vector $\vec{m}' = [0\ 0\ 0\ 2\ 1\ 1]^T$ after firing sequence $t_4 t_6$ from $\vec{m}_5$. Since t_4 is not labeled by the fault event, transition t'_5 does not add a token to place p_f.

After iterating every observable transition of $\mathcal{N}$, Algorithm 3 executes steps 17–25 to iterate all transitions associated with fault events in order to create transitions that, when enabled, indicate that a fault transition may have fired before it contributed to the firing of an observable transition. As such, the algorithm iterates the only fault transition t_1 and finds all minimal markings that enable t_1 after the firing of an unobservable transition sequence by using function FAM (Algorithm 2). Since there are no unobservable transitions that contribute to the firing of t_1, the only minimal marking that enables t_1 is $\vec{m}_8 = [1\ 0\ 0\ 0\ 0\ 0]^T$, which is associated with

λ; therefore, Algorithm 3 creates transition t'_8, whose firing consumes and adds the same number of tokens as $\vec{m}_8$ in order to prevent the firing of t'_8 from changing the Petri net marking, since we will only check if t'_8 is enabled instead of firing it. Notice that t'_8 is labeled by the event σ_{fv} in order to differentiate it from other transitions.

The diagnoser Petri net $\mathcal{ND}$ generated by Algorithm 3 possesses several properties that contributes to the fault diagnosis of a Petri net $\mathcal{N}$. Since those properties relate the behavior of both nets $\mathcal{N}$ and $\mathcal{ND}$, we will define the mapping function $MT : T_D^* \rightarrow T_o^*$, which is defined by the following recursion:

$$MT(\lambda) = \lambda$$

$$MT(t'_o) = \begin{cases} t_o, \ \textit{if } t'_o \textit{ was generated by the observable transition } t_o \in T_o \\ \lambda, \ \textit{if } \ell(t'_o) = \sigma_{fv} \end{cases}$$

$$MT(s_o t'_o) = MT(s_o)MT(t'_o), \forall s_o \in T_D^*, \forall t'_o \in T_D.$$

In words, function MT transforms each transition t'_o of a transition sequence of $\mathcal{ND}$ in either the corresponding observable transition t_o that generated it, or an empty transition sequence, if t'_o is associated with the event σ_{fv}. Similar to the projection operation, we also extend MT to set of transition sequences by applying the function to each sequence of the set.

With the mapping MT defined, we are able to establish that the set of observable transition sequences and observed language of $\mathcal{N}$ are equivalent to the set of transition sequences and language of $\mathcal{ND}$, as shown in the following lemmas.

Lemma 4.1. *Let $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ be a Petri net . Then, the diagnoser Petri net $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ obtained in accordance with Algorithm 3 is such that $PT_o(LT(\mathcal{N})) = MT(LT(\mathcal{ND}))$.*

Proof. The equality $PT_o(LT(\mathcal{N})) = MT(LT(\mathcal{ND}))$ is proven by induction on the length of sequence $s_o \in T_o^*$

(Basis step) For the empty transition sequence $s_o = \lambda$, it is trivial that $s_o \in$

$PT_o(LT(\mathcal{N}))$ and $s_o \in MT(LT(\mathcal{ND}))$.

(Inductive step) Assuming that there is a sequence $s_o \in T_o^*$ such that $s_o \in PT_o(LT(\mathcal{N}))$ and $s_o \in MT(LT(\mathcal{ND}))$, it will be proven that, for every observable transition $t_o \in T_o$, if $s_o t_o \in PT_o(LT(\mathcal{N}))$, then $s_o t_o \in MT(LT(\mathcal{ND}))$ and if $s_o t_o \in MT(LT(\mathcal{ND}))$, then $s_o t_o \in PT_o(LT(\mathcal{N}))$, which proves that $PT_o(LT(\mathcal{N})) = MT(LT(\mathcal{ND}))$.

If $s_o = t_{o,1}, t_{o,2}, \ldots t_{o,k}$ is such that $s_o \in MT(LT(\mathcal{ND}))$ and $s_o \in PT_o(LT(\mathcal{N}))$, then, for each observable transition $t_{o,i}$, where $i = 1, 2, \ldots, k$, there is a minimal sequence of unobservable transitions $s_i \in T_{uo}^*$ and a minimal marking vector $\vec{m}_{min,i} \in \mathbb{Z}_+^{n_P}$ such that $\vec{m}_{min,i}[s_i t_{o,i}\rangle$. Additionally, due to step 12 of Algorithm 3, for a marking vector $\vec{m} \geq \vec{m}_{min,i}$, the firing of $s_i t_{o,i}$ generates the same number of tokens in every place of $\mathcal{N}$ as the transition $t'_{o,i} \in T_D$ associated with $s_i t_{o,i}$ generates in $\mathcal{ND}$, with the exception of place $p_f \in P_D$, which does not affect the Petri net dynamic since it does not possess output transitions. Therefore, since the corresponding places of $\mathcal{N}$ and $\mathcal{ND}$ have the same amount of initial tokens, the firing of sequence $s_1 t_{o,1} s_2 t_{o,2} \ldots s_k t_{o,k}$ in $\mathcal{N}$ generates the same marking vector $\vec{m}_f$ as the firing of sequence $t'_{o,1} t'_{o,2} \ldots t'_{o,k}$ in $\mathcal{ND}$ generates, with the exception of p_f.

If the firing of t_o is observed after $\mathcal{N}$ reached the marking vector $\vec{m}_f$, then there is a minimal marking vector $\vec{m}_{min,f} \leq \vec{m}_f$ and a minimal sequence of unobservable transitions $s_{uo} \in T_{uo}^*$ such that $\vec{m}_{min,f}[s_{uo} t_o\rangle$. Since $\vec{m}_{min,f}$ is a minimal marking that enables $s_{uo} t_o$, $\mathcal{ND}$ has a transition t'_o associated with $s_{uo} t_o$ that is also enabled by $\vec{m}_{min,f}$. Therefore, since both nets can be at the same marking $\vec{m}_f$ after the observation of s_o and transition t'_o, such that $MT(t'_o) = t_o$, is enabled in $\mathcal{ND}$, then $s_o t_o \in MT(LT(\mathcal{ND}))$ whenever $s_o t_o \in PT_o(LT(\mathcal{N}))$.

If $\mathcal{ND}$ current marking is $\vec{m}_f$ and t'_o, such that $MF(t'_o) = t_o$, is enabled, then there is a minimal marking $\vec{m}_{min,f} \leq \vec{m}_f$ and a sequence of unobservable transitions s_{uo} such that $\vec{m}_{min,f}[s_{uo} t_o\rangle$. Thus, $\vec{m}_f[s_{uo} t_o\rangle$, which implies that $s_o t_o \in PT_o(LT(\mathcal{N}))$ whenever $s_o t_o \in MT(LT(\mathcal{ND}))$.

Therefore, it was proven that, for every observable transition $t_o \in T_o$, if $s_o t_o \in$

$PT_o(LT(\mathcal{N}))$, then $s_o t_o \in MT(LT(\mathcal{ND}))$ and if $s_o t_o \in MT(LT(\mathcal{ND}))$, then $s_o t_o \in PT_o(LT(\mathcal{N}))$, which proves, by induction, that $PT_o(LT(\mathcal{N})) = MT(LT(\mathcal{ND}))$. ■

Lemma 4.2. *Let* $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ *be a diagnosable Petri net and let* $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ *be the diagnoser Petri net obtained in accordance with Algorithm 3. Then* $\mathcal{ND}$ *is such that* $P_o(L(\mathcal{N})) = P_{\sigma_{fv}}(L(\mathcal{ND}))$, *where* $P_{\sigma_{fv}} : \Sigma_D^* \to \Sigma_o^*$.

Proof. It is trivial that the observed language of $\mathcal{N}$, $P_o(L(\mathcal{N}))$, is equal to $\ell(PT_o(LT(\mathcal{N})))$, since both projections P_o and PT_o remove the unobservable events and transitions from the language and the set of transition sequences of $\mathcal{N}$, respectively.

Since each mapping executed by MT associates the transitions labeled by σ_{fv} with empty transitions and transitions that are not labeled by σ_{fv} with the transitions of $\mathcal{N}$ that generated them in such a way that the label of the generated transition in $\mathcal{ND}$ is equal to the label of the original transition in $\mathcal{N}$, the labels of the transition sequences of $MT(LT(\mathcal{ND}))$ are equal to the labels of the transition sequences of $LT(\mathcal{ND})$ when disregarding event σ_{fv}. Therefore, we can assert that $P_{\sigma_{fv}}(L(\mathcal{ND})) = \ell(MT(LT(\mathcal{ND})))$.

In Lemma 4.1, it was proven that $PT_o(LT(\mathcal{N})) = MT(LT(\mathcal{ND}))$; thus, we can also assert that $\ell(PT_o(LT(\mathcal{N}))) = \ell(MT(LT(\mathcal{ND})))$. Finally, based on this equality and the equalities $P_o(L(\mathcal{N})) = \ell(PT_o(LT(\mathcal{N})))$ and $P_{\sigma_{fv}}(L(\mathcal{ND})) = \ell(MT(LT(\mathcal{ND})))$, we are able to affirm that $P_o(L(\mathcal{N})) = P_{\sigma_{fv}}(L(\mathcal{ND}))$. ■

Lemma 4.2 shows that the observed generated language of the original Petri net $\mathcal{N}$ is equal to the language generated by the diagnoser Petri net $\mathcal{ND}$ when special event σ_{fv} is disregarded; therefore, for each sequence of observable events $s_o \in P_o(L(\mathcal{N}))$ that may occur in $\mathcal{N}$, there is always a sequence of transitions $s' \in LT(\mathcal{ND})$ that can fire in $\mathcal{ND}$ and is labeled by s_o. Furthermore, the possible sequences s' model all possible minimal sequences of transitions $s \in LT(\mathcal{N})$ that could have fired in $\mathcal{N}$ to model the observation of s_o, and if s contains a fault event, then s' adds at least one token to the special place p_f. Based on the aforementioned notion, we may state the following theorem.

Theorem 4.1. *Let $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ be a diagnosable Petri net and let $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ be the diagnoser Petri net obtained in accordance with Algorithm 3. Then, for all sequences of observable events $s_o \in P_o(L(\mathcal{N}))$ such that $(\forall s \in P_o^{-1}(s_o) \cap L(\mathcal{N}), \sigma_f \in s)$, the firing of any transition sequence $s_D \in LT(\mathcal{ND})$ in $\mathcal{ND}$, where $\ell_D(s_D) = s_o$, results in a marking vector in $\mathcal{ND}$ for which place $p_f \in P_D$ has at least one token.*

Proof. Let $s_o \in P_o(L(\mathcal{N}))$ be a sequence of observable events such that $(\forall s \in P_o^{-1}(s_o) \cap L(\mathcal{N}), \sigma_f \in s)$. Since Lemma 4.2 states that $P_o(L(\mathcal{N})) = P_{\sigma_{fv}}(L(\mathcal{ND}))$, there may be multiple sequences of transitions $s_D \in LT(\mathcal{ND})$ such that $\ell_D(s_D) = s_o$, in which each sequence s_D is associated with a transition sequence $s_c \in LT(\mathcal{N})$ such that $P_o(\ell(s_c)) = s_o$. If $(\forall s \in P_o^{-1}(s_o) \cap L(\mathcal{N}))(\exists s_c \in LT(\mathcal{N}))[(\sigma_f \in s) \wedge (P_o(\ell(s_c)) = s_o)]$, then we can also say that $\sigma_f \in \ell(s_c)$. Furthermore, all sequences s_c must contain at least one fault transition whose firing is necessary to enable an observable transition, otherwise, we would be able to create a transition $s_c \in LT(\mathcal{N})$ that would have the observation s_o and would not have any fault transitions; therefore, since s_c is associated with s_D, then, due to the execution of step 13 on Algorithm 3, at least one of the transitions $t'_o \in s_D$ has place $p_f \in P_D$ as an output place. Since place p_f has no output transitions, the firing of s_D in $\mathcal{ND}$ results in a marking in which p_f has at least one token; thus, the firing of any possible corresponding sequence $s_D \in LT(\mathcal{ND})$, such that $\ell_D(s_D) = s_o$, results in a marking in $\mathcal{ND}$ such that place $p_f \in P_D$ has at least one token.

■

Due to the result of Theorem 4.1, we are able to confirm the occurrence of a fault event during the operation of $\mathcal{N}$ by checking whether the observed event sequence $s_o \in P_o(L(\mathcal{N}))$ is such that all transition sequences $s' \in LT(\mathcal{ND})$ of the diagnoser Petri net that have the same observation as s_o results in a marking in $\mathcal{ND}$ such that place $p_f \in P_D$ contains at least one token.

It is also desirable for the diagnoser to be able to infer that the fault event could have occurred before the observation of an event that confirms its occurrence. When analyzing the transition sequences of the diagnoser Petri net that are consistent with the observation of the event sequence $s_o \in P_o(L(\mathcal{N}))$ in the original Petri net, there are two possible scenarios that allow us to infer that the fault event could have occurred:

- *(i)* There is a transition sequence $s \in LT(\mathcal{ND})$ such that $\ell_D(s) = s_o$ and whose firing results in a marking vector that enables a transition labeled by σ_{fv}.
- *(ii)* There are two possible transition sequences $s_1, s_2 \in LT(\mathcal{ND})$ such that $\ell_D(s_1) = \ell_D(s_2) = s_o$ and the firing of s_1 results in a marking vector that contains a token in p_f whereas the firing of s_2 results in a marking that does not.

In case *(i)*, the fault event could have occurred because a transition labeled by σ_{fv} is enabled whenever it is possible to fire a fault transition after the firing of a minimal unobservable transition sequence, whereas in case *(ii)*, the fault event could have occurred because sequences s_1 and s_2 models two different scenarios of transition sequences that could have fired in the original Petri net, where the fault event only occurred in one of them. In order to prove that those cases allow the diagnoser to infer that the fault event could have occurred, we present the following results.

Lemma 4.3. *Let $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ be a diagnosable Petri net and let $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ be the diagnoser Petri net obtained in accordance with Algorithm 3. Then, for all transition sequences $s \in LT(\mathcal{N})$ such that $\ell(s) \in \Psi(\Sigma_f)$, there is a transition sequence $s' \in LT(\mathcal{ND})$ in $\mathcal{ND}$ such that $PT_o(s) = MT(s')$, $\sigma_{fv} \notin \ell_D(s')$ and the firing of s' in $\mathcal{ND}$ results in a marking vector for which a transition $t_{fv} \in T_D$ labeled by event σ_{fv} is enabled.*

Proof. Let $s \in \Psi(\Sigma_f)$ be such that $s_1 t_{o,1} s_2 t_{o,2} \dots s_k t_{o,k} s_f t_f$, where, for $i = 1, 2, \dots, k$, $s_i \in T_{uo}^*$ and $t_{o,i} \in T_o$, whereas $s_f \in T_{uo}^*$ and $t_f \in T_{uo}$ is such that $\ell(t_f) = \sigma_f$.

Notice that each unobservable transition sequence s_i may contain transitions that do not directly or indirectly contribute to the firing of $t_{o,i}$, *i.e.*, transitions whose firings do not add tokens that are spent by $t_{o,i}$ or another transition of s_i whose generated tokens contribute to the firing of $t_{o,i}$. Let $s_{un,i}$ be the sequence of the transitions of s_i that do not contribute to the firing of $t_{o,i}$ in the same order as they appear in s_i, and let $s_{uo,i}$ be the sequence of the transitions of s_i that contributes to the firing of $t_{o,i}$. Since the tokens generated by the firing of the transitions of $s_{un,i}$ are not used by the firing of either $s_{uo,i}$ or $t_{o,i}$, it is possible to fire $s_{un,i}$ after the firing of sequence $s_{uo,i}t_{o,i}$ without changing the resulting marking; therefore, sequence $s_i t_{o,i}$ may be changed to sequence $s_{uo,i}t_{o,i}s_{un,i}$. Additionally, by concatenating $s_{un,i}$ with s_{i+1} to s'_{i+1}, we may repeat the same process for sequence $s'_{i+1}t_{o,i+1}$, resulting in the sequence $s_{uo,i+1}t_{o,i+1}s_{un,i+1}$.

If the process described above were repeated for all $i = 1, 2, \ldots, k$, we would obtain sequence $s_{uo,1}t_{o,1}s_{uo,2}t_{o,2} \ldots s_{uo,k}t_{o,k}s_{un,k}s_f t_f$, whose firing results in the same marking vector as s. The unobservable transition sequences $s_{un,k}$ and s_f may also be concatenated into sequence s'_f, which can be separated into sequences $s_{f,uo}$ and $s_{f,un}$, in which the former transitions contributes to the firing of t_f, whereas the latter transitions do not contribute to the firing of t_f. Thus, we are able rearrange s further in to obtain sequence $s_{min} = s_{uo,1}t_{o,1}s_{uo,2}t_{o,2} \ldots s_{uo,k}t_{o,k}s_{f,uo}t_f s_{f,un}$.

Since s_{min} is such that each sequence $s_{uo,i}$, for $i = 1, 2, \ldots, k$, contains only transitions that contribute to the firing of the observable transition $t_{o,i}$, each sequence $s_{uo,i}$ is associated with a minimal marking that enables $t_{o,i}$ after firing it. Due to all transitions of $\mathcal{ND}$ being created to model the firing of each observable transition after the firing of a sequence of unobservable transitions associated with minimal markings, there is a sequence $s' \in LT(\mathcal{ND})$ such that $s' = t'_{o,1}t'_{o,2} \ldots t'_{o,k}$, where, for $i = 1, 2, \ldots, k$, $t_{o,i}$ is associated with $s_{uo,i}$, which makes $PT_o(s_{min}) = MT(s')$ and is such that $\sigma_{fv} \notin \ell_D(s')$. Since the order in which the observable transitions of s and s_{min} are organized are the same, $PT_o(s_{min}) = PT_o(s)$. Thus, s' is also such that $PT_o(s) = MT(s')$

Given the way that $\mathcal{ND}$ was constructed and by excluding place p_f of $\mathcal{ND}$, the firing of s' in $\mathcal{ND}$ results in the same marking vector $\vec{m}_f$ as the firing of sequence $s_{uo,1}t_{o,1}s_{uo,2}t_{o,2}\dots s_{uo,k}t_{o,k}$ in $\mathcal{N}$, with the exception of place p_f.

Due to the steps 17–25 of Algorithm 3, if $s_{f,uo}$ is associated with a minimal marking that enables the fault transition t_f after firing, then $\mathcal{ND}$ contains a transition t_{fv} associated with the event σ_{fv} that is enabled whenever sequence $s_{f,uo}t_f$ is enabled in $\mathcal{N}$. Therefore, since s_{min} is such that sequence $s_{f,uo}t_f$ is enabled after the firing of $s_{uo,1}t_{o,1}s_{uo,2}t_{o,2}\dots s_{uo,k}t_{o,k}$ on $\mathcal{N}$, which results in the same marking vector as the firing of s' in $\mathcal{ND}$, with the exception of place p_f, then there is a transition $t_{fv} \in T_D$ associated with the event σ_{fv} that is enabled after the firing of s'.

■

Theorem 4.2. *Let* $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ *be a diagnosable Petri net and let* $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ *be the diagnoser Petri net obtained in accordance with Algorithm 3. Then, for all sequences of observable events* $s_o \in P_o(L(\mathcal{N}))$ *such that* $(\exists s \in P_o^{-1}(s_o) \cap L(\mathcal{N}))[\sigma_f \in s]$*, there exists a transition sequence* $s_D \in LT(\mathcal{ND})$ *in* $\mathcal{ND}$ *such that* $\ell_D(s_D) = s_o$ *and whose firing results in a marking vector that either enables a transition* $t_{fv} \in T_D$ *labeled by event* σ_{fv} *or is such that place* p_f *has at least one token.*

Proof. If the transition sequence $s \in LT(\mathcal{N})$ is such that $\ell(s) = s_o$ and $\sigma_f \in \ell(s)$, then there is a fault transition $t_f \in T$ such that $(\sigma_f \in \ell(t_f)) \wedge (t_f \in s)$, which may be in one of the following situations:

- *(i)* t_f does not contribute to the firing of any observable transition of s.
- *(ii)* t_f contributes to the firing of an observable transition $t_o \in s$.

If the situation of t_f is *(i)*, then t_f and all unobservable transitions whose firings are justified by the tokens added by the firing of t_f may be moved within sequence s in a similar manner that transitions were moved in Lemma 4.3, creating an equivalent transition sequence s' whose firing results in the same marking vector and observation as s, but is such that t_f and all unobservable transitions that fired due to the firing of t_f are located after the last observable transition. Notice that s' can be further modified into a new transition sequence s'' by removing the unobservable transitions that fire in s' after t_f, in which $PT_o(s') = PT_o(s'')$. Since s'' last transition is t_f, which is a fault transition, the label of s'' is such that $\ell(s'') \in \Psi(\Sigma_f)$. Therefore, since $PT_o(s) = PT_o(s') = PT_o(s'')$ and $s'' \in \Psi(\Sigma_f)$, according to Theorem 4.3, there is a transition sequence $s_D \in LT(\mathcal{ND})$ such that $PT_o(s) = MT(s_D)$ and $\sigma_{fv} \notin \ell_D(s_D)$, which implies that $\ell_D(s_D) = s_o$, and its firing in $\mathcal{ND}$ results in a marking vector that enables a transition $t_{fv} \in T_D$ associated with the event σ_{fv}.

If the situation of t_f is *(ii)*, then t_f contribute to the firing of at least one observable transition t_o of s. Thus, similar to the conclusions shown in theorem 4.1, there is a transition sequence $s_D \in LT(\mathcal{ND})$ such that $\ell(s_D) = s_o$, in which s_D contains a transition t'_o associated with t_o and an unobservable transition sequence s_{uo} that contains the fault transition t_f, which means that t'_o adds a token to p_f, according to Algorithm 3.

Finally, in both cases, it was proved that, for the transition sequence $s \in LT(\mathcal{N})$ with a fault transition, there is a transition sequence $s_D \in LT(\mathcal{ND})$ such that $\ell_D(s_D) = s_o$ and its firing in $\mathcal{ND}$ results in a marking vector that either enables a transition $t_{fv} \in T_D$ associated with the event σ_{fv} or is such that place p_f has at least one token.

$\blacksquare$

Using the results of Theorem 4.2, we can confirm that the fault event could have occurred during the operation of the system that $\mathcal{N}$ models if the observed event sequence $s_o \in P_o(L(\mathcal{N}))$ is such that there is a sequence $s' \in LT(\mathcal{ND})$ labeled by s_o whose firing results in a marking vector that either enables a transition labeled by event σ_{fv} or has a token in place p_f. Finally, if the observed event sequence $s_o \in P_o(L(\mathcal{N}))$ of $\mathcal{N}$ does not satisfy the conditions of both Theorems 4.1 and 4.2, then we are able to assert that no fault event has occurred, since there would not exist a transition sequence $s \in LT(\mathcal{N})$ that would be labeled by s_o and have a fault transition in it.

In order to summarize the online diagnosis of a system modeled by a Petri net $\mathcal{N}$ that we may do using the diagnoser Petri net $\mathcal{ND}$, we enumerate the possible scenarios of the diagnoser, given the observation of an event sequence $s_o \in P_o(L(\mathcal{N}))$:

- C_F: The diagnoser is sure that a fault event has occurred if, for all transition sequences $s' \in LT(\mathcal{ND})$ such that $\ell_D(s') = s_o$, the marking $\vec{m}_f$ reached at $\mathcal{ND}$ after firing s' is such that $\vec{m}_f(p_f) > 0$.

- C_D: The diagnoser is sure that a fault event may have occurred before another event observation after s_o if the condition described in C_F is false and if there is at least one transition sequence $s' \in LT(\mathcal{ND})$ such that $\ell_D(s') = s_o$ and the marking $\vec{m}_f$ reached at $\mathcal{ND}$ after firing s' is such that $\vec{m}_f(p_f) > 0$ or $\vec{m}_f$ enables a transition in $\mathcal{ND}$ labeled by event σ_{fv}.

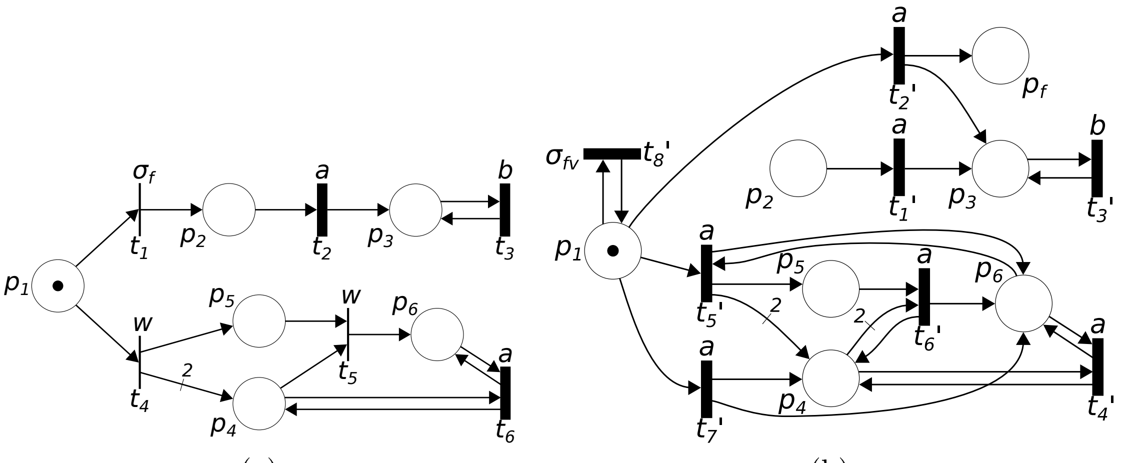
- C_N: The diagnoser is sure that no fault events have occurred if both conditions described in C_F and C_D are false.

In order to show how the diagnoser Petri net can be used for the fault diagnosis, we present the following example that does the online diagnosis of the Petri net presented in Example 4.3.

Example 4.4. *Let* $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ *be the Petri net depicted in Figure 4.4(a), whose observable, unobservable and fault events are* $\Sigma_o = \{a, b\}$, $\Sigma_{uo} = \{w, \sigma_f\}$ *and* $\Sigma_f = \{\sigma_f\}$*, respectively, and let the Petri net* $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ *of Figure 4.4(b) be the resulting diagnoser Petri net of* $\mathcal{N}$*.*

Let both Petri nets $\mathcal{N}$ *and* $\mathcal{ND}$ *have initial states where place* p_1 *has a token. Notice that the transition* t_8' *of* $\mathcal{ND}$ *labeled by* σ_{fv} *is enabled. This means that a fault event could have occurred before any observations, which is true, since transition* t_1 *may fire in* $\mathcal{N}$*. Therefore, in this case, the diagnoser state is equal to* C_D*.*

If event a *is observed, then either sequences* t_1t_2 *or* $t_4t_5t_6$ *fired in* $\mathcal{N}$*, where the former generates the marking vector* $\vec{m}_1 = [0\ 1\ 0\ 0\ 0\ 0]^T$ *and the latter generates the marking vector* $\vec{m}_2 = [0\ 0\ 0\ 1\ 0\ 1]^T$*. Observe that if* t_2' *fires in* $\mathcal{ND}$ *to justify*



(a) (b)

Figure 4.4: Petri net considered in Example 4.4(a) and its resulting diagnoser Petri net(b).

the observation of event a, the resulting marking is equal to $\vec{m}_1$ *with the addition of a token in place* p_f*, i.e.* $\vec{m}'_1 = [0\ 1\ 0\ 0\ 0\ 0\ 1]^T$*, which replicates the behavior of the firing of sequence* t_1t_2 *and indicates the occurrence of the fault event* σ_f*. If* t'_7 *fires instead, the generated marking vector is equal to* $\vec{m}'_2$*, which is equal to* $\vec{m}_2$ *with the exception of place* p_f*, which has zero tokens in* $\vec{m}'_2$*. Therefore, both possible transition sequences that explain the occurrence of event a in* $\mathcal{N}$ *can be explained by transitions of* $\mathcal{ND}$*. Since both marking vectors are consistent with the observation of event a, the diagnoser is not able to assert whether the fault event occurred or not because one of them has a token in place* p_f*, whereas the other does not, meaning that the diagnoser is still at state* C_D*.*

If event b is observed after the observation of event a, the only possible sequence of transitions that could have fired in $\mathcal{N}$ *to justify the observation of event sequence ab is transition sequence* $t_1t_2t_3$*, which contains a fault transition. Furthermore, the only transition sequence that we can fire in* $\mathcal{ND}$ *to justify sequence ab is transition sequence* $t'_2t'_3$*, which generates a marking in* $\mathcal{ND}$ *that has a token in place* p_f*. Since the only marking vector that can be reached in the diagnoser Petri net that is consistent with the observation of sequence ab has a token in place* p_f*, the diagnoser is able assert that a fault event has occurred; thus, the state of the diagnoser changes to* C_F*. The reverse case happens when event a is observed again instead of event* *b, where the only consistent marking vector of $\mathcal{ND}$ does not contain any tokens in place p_f, meaning that the fault event has not occurred and that the diagnoser state changes to C_N.*

Since the fault diagnosis with the diagnoser Petri net $\mathcal{ND}$ proposed here consists of checking every possible marking vector it may reach from an observed sequence of events, we reduce the problem of diagnosability to the problem of state estimation of the λ-free diagnoser Petri net $\mathcal{ND}$. This problem will be addressed in Section 4.2, where we will further modify the diagnoser Petri net so that each event sequence will only be associated with one transition sequence in $\mathcal{ND}$ while maintaining the main properties of $\mathcal{ND}$ that are required for the fault diagnosis.

4.2 State estimation of the λ-free diagnoser Petri net $\mathcal{ND}$ by solving its event conflicts

In order to diagnose the occurrence of a fault event after the observation of a sequence of observable events $s_o \in \Sigma_o^*$ in a labeled Petri net $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ with the λ-free diagnoser Petri net $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ that was generated in Section 4.1, we need to estimate the states that $\mathcal{ND}$ may reach after firing transition sequences labeled by s_o in such a way that we can verify if those states are such that place $p_f \in P_D$ contains tokens or enable a transition $t_{fv} \in T_D$ such that $\ell_D(t_{fv}) = \sigma_{fv}$.

According to the findings in [26], a possible approach to estimate the markings that a λ-free labeled Petri net may archive is to check every possible marking that is consistent with the observed event sequence. The work also proves that the maximum number of possible markings that may be reached after the observation of an event sequence grows polynomially with the length of the transition. However, since the number of feasible marking may grow indefinitely during the system operation while using this approach, we will not use it.

Another work that involves the state estimation of λ-free labeled Petri net is the one proposed in [24], where the set of markings consistent with an observed event sequence is described by a linear system with a fixed structure that does not depend on the length of the sequence. Although this approach avoids the problem of a structure that grows indefinitely, it assumes that the λ-free labeled Petri net is such that every set of transitions labeled by the same event is contact-free, which means that two transitions labeled by the same event cannot have a place that is in the input or output place of both transitions, *i.e.*, for two transitions $t_1, t_2 \in T$ such that $\ell(t_1) = \ell(t_2)$, we have that $(I(t_1) \cup O(t_1)) \cap (I(t_2) \cup O(t_2)) = \emptyset$. However, the requirement for all transitions that share labels in the diagnoser Petri net $\mathcal{ND}$ to be contact-free is too restrictive for this work, since multiple transitions $t'_o \in T_D$ of $\mathcal{ND}$ may be created from a single transition $t_o \in T$ of $\mathcal{N}$ during the construction of the Petri net $\mathcal{ND}$ in Algorithm 3, and those transitions have the same output places and label as t_o, which usually renders the resulting Petri net $\mathcal{ND}$ as not contact-free.

Since the state estimation of previous works are not suitable for the generated diagnoser Petri net $\mathcal{ND}$, we propose a new approach for the state estimation of λ-free labeled Petri nets. Starting from the diagnoser Petri net generated by Algorithm 3, which from now on will be referred to as $\mathcal{ND}_0$, this new approach alters the diagnoser Petri net to solve its event conflicts in such a way that each sequences of events $s_o \in L(\mathcal{ND}) \cap \Sigma_o^*$ that may occur in the diagnoser Petri net will only label a single transition sequence $s' \in LT(\mathcal{ND})$, which can fire from the initial marking vector of diagnoser Petri net and result in a marking vector that may represent multiple possible marking vectors of the original diagnoser Petri net $\mathcal{ND}_0$ that may be obtain by firing transition sequences labeled by s_o in it. The resulting diagnoser Petri net $\mathcal{ND}$ also has the properties shown in Theorems 4.1 and 4.2 with respect to the original Petri net $\mathcal{N}$, which ensures that $\mathcal{ND}$ can be used to diagnose the fault occurrence after the modifications. In order to modify $\mathcal{ND}$ so that it has the aforementioned properties, we use the function NOC, which solves every event conflict involving a given set of transitions that share the same label.

4.2.1 Function *NOC*

Function *NOC* is an algorithm that adds new places and transitions to the diagnoser Petri net $\mathcal{ND}$ so that all event conflicts that exclusively involve all transitions of a given set of transitions $T_C \subseteq T_D$ and are labeled by the same event $\sigma_C \in \Sigma_o$ are solved, *i.e.*, those event conflicts never occur during the modified diagnoser Petri net operation. It is worth remarking that each iteration of function *NOC* only solves the event conflicts involving the set of transitions T_C; therefore, in order to solve multiple event conflicts involving different set of transitions, function *NOC* is executed multiple times for each set.

The main idea of function *NOC* is that it creates a new transition t_C that has a higher priority to fire than the transitions of T_C and whose firing models that one of the transitions of T_C fired without specifying which one actually fired. Furthermore, the firing of t_C consumes the tokens required to enable all transitions of T_C; therefore, since t_C also has a higher priority to fire than the transitions of T_C, whenever the Petri net marking has enough tokens to enable all transitions of T_C, only transition t_C will be enabled, solving the event conflict.

Remark 4.3. *Although creating transition t_C solves the events conflicts involving a set of transitions T_C, it does not necessarily prevent the transitions of T_C from firing for all reachable markings of the diagnoser Petri net. Some of the transitions of T_C could be enabled by the Petri net without enabling all of the transitions of T_C, and since t_C is only enabled by a marking vector that enables all the transitions of T_C, it would not be enabled in this case, allowing the firing of transitions of T_C.*

Since transition t_C models the firing of multiple transitions whose firings can result in different marking vectors, function *NOC* also creates places that represent multiples possibilities of tokens that could have been generated after the firing of one of the transitions of T_C in $\mathcal{ND}_0$. In order to record the places created by function *NOC*, after each execution, the new places that were created by the function are stored into the set of places $P_p \subset P_D$, and the possible group of tokens that the tokens of the places of P_p represent are stored into the list $Poss_{cache}$, wherein each element is associated with a place of P_p and is a matrix in which each row is associated with each place of $\mathcal{ND}_0$ and each column represents a possible scenario of tokens in $\mathcal{ND}_0$ that one token in a place of P_p models.

The pseudocode of function *NOC* is shown in Algorithm 4, where the inputs are: the diagnoser Petri net $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$, the event $\sigma_C \in \Sigma_D$, the set of transitions $T_C \subseteq T_D$ that compose the event conflicts that the function is going to solve, the set of places P_p and its list of possibilities $Poss_{cache}$ that were created by previous iterations of function *NOC*. The output of the function returns the modified diagnoser Petri net $\mathcal{ND}$ and the elements related to the places created by the function, *i.e.*, P_p and $Poss_{cache}$. It is worth remarking that we consider the elements P_p and $Poss_{cache}$ as empty for the first iteration of function *NOC*. As function *FAM*, function *NOC* also has a complex structure; therefore, readers may prefer to follow Example 4.6 in order to better understand about the main ideas of function *NOC*.

In steps 1–7, function *NOC* creates transition t_C for the diagnoser Petri net as a transition that is labeled by σ_C and is involved in the priority relation (t, t_C) for all transitions $t \in T_C$. Since transition t_C models the firing of one of the transitions of T_C, the function makes t_C have the same priority relations as other transitions of the Petri net have with the transitions of T_C. The function further defines that the firing of transition t_C consumes the same tokens required to enable all transitions of T_C.

In order to define the tokens that the firing of transition t_C adds, the function must first compute the possible tokens of $\mathcal{ND}_0$ that the firing of each one of the transitions of T_C would add after firing from a marking that has the same number of tokens that transition t_C consumes, which would indicate the possible resulting tokens after the firing of one of the transitions of the event conflicts. The function does this computation by executing steps 8–33, where the function initially creates the element *Poss* as a matrix whose rows are associated with the places of P_D and whose columns represent each possibility of tokens added by each transition of T_C.

Algorithm 4 Algorithm for the function *NOC* that solves event conflicts caused by a set of transitions associated with a same event of the diagnoser Petri net

Inputs:

- $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$: diagnoser Petri net of $\mathcal{N}$
- σ_C and T_C: Event and set of transitions, respectively, of the event conflicts that are solved by the function
- P_p: set of places created for each group of possibilities
- $Poss_{cache}$: list in which each element is a matrix associated with a place of P_p and represents the possible group of markings that the associate place of P_p models

Outputs:

- $\mathcal{ND}$: diagnoser Petri net of $\mathcal{N}$ that does not have the event conflict $\langle \sigma_C, T_C \rangle$
- P_p: same meaning as the input P_p
- $Poss_{cache}$: same meaning as the input $Poss_{cache}$

Create transition t_C and add it to T_D
Set $\ell_D(t_C) \leftarrow \sigma_C$
For all $t \in T_C$, add (t, t_C) to ρ_D
For each $t \in T_D \backslash (T_C \cup \{t_C\})$ do
  If $(\exists t_c \in T_C)[(t_c, t) \in \rho_D]$, add (t_C, t) to ρ_D
  Else if $(\exists t_c \in T_C)[(t, t_c) \in \rho_D]$, add (t, t_C) to ρ_D
Set $Pre_D(:, t_C) \leftarrow max(Pre_D(:, T_C), column)$
Set $Poss \leftarrow Post_D(:, T_C) + Pre_D(:, t_C) - Pre_D(:, T_C)$
For each place $p_p \in P_p$ do
  While there is a column i such that $Poss(p_p, i) \geq 1$ do
    Set $Poss(p_p, i) \leftarrow Poss(p_p, i) - 1$
    Expand $Poss(:, i)$ to $\left[(Poss(P_D \backslash P_p, i) + Poss_{cache}(p_p))^T, (Poss(P_p, i)\ \vec{1}_{1 \times col(Poss_{cache}(p_p))})^T\right]^T$
Remove every row of $Poss$ that is associated with a place of P_p
Remove all repeated columns of $Poss$
For each place $p_p \in P_p$ do
  Do
    Set $Poss_{com}$ as an empty matrix
    For each i-th column of $Poss_{cache}(p_p)$ do
      Set $Poss_i \leftarrow Poss - Poss_{cache}(p_p)(:, i)$
      Remove all columns of $Poss_i$ that contain at least one negative element
      If $Poss_{com}$ is empty
        Set $Poss_{com} \leftarrow Poss_i$
      Else
        Set $Poss_{com}$ as a matrix that contains the columns that are both in $Poss_{com}$ and $Poss_i$
      Remove the repeated columns of $Poss_{com}$
    Set $Poss_{re}$ as an empty matrix
    For each column i of $Poss_{com}$ and each column j of $Poss_{cache}(p_p)$ do
      Set $Poss_{re} \leftarrow [Poss_{re}, Poss_{com}(:, i) + Poss_{cache}(p_p)(:, j)]$
    Remove the repeated columns of $Poss_{re}$
    If $Poss_{re}$ contains the same columns as $Poss$
      Set $Poss \leftarrow Poss_{com}$
      Set $Post_D(p_p, t_C) \leftarrow Post_D(p_p, t_C) + 1$
  While $Poss = Poss_{com}$
Set $Post_D(P_D \backslash P_p, t_C) \leftarrow min(Poss, column)$
Set $Poss \leftarrow Poss - min(Poss, column)$
Remove the repeated columns of $Poss$
If $Poss$ contains an element greater than zero
  Create place p_p and add it to P_D and P_p
  Set $\vec{m}_{0,D}(p_p) \leftarrow 0$
  Set $Poss_{cache}(p_p) \leftarrow Poss$
  Set $Post_D(p_p, t_C) \leftarrow 1$
  Set $\mathcal{ND} \leftarrow AOT(\mathcal{ND}, p_p, Poss)$

Since we are trying to find the possibilities of tokens in the initial diagnoser Petri net $\mathcal{ND}_0$, which does not have the places of set P_p, for each token of a place $p_p \in P_p$ in each possibility, the function removes that token from the possibility and expands the possibility using matrix $Poss_{cache}(p_p)$, expanding the column of the possibility into multiple columns that are the result of the sum of the possibility with the possibilities modeled by the columns of matrix $Poss_{cache}(p_p)$.

After computing matrix $Poss$, the function verifies, by executing steps 15–33, if it is possible to reduce the possibilities of $Poss$ with each matrix of possibilities $Poss_{cache}(p_p)$ associated with a places of $p_p \in P_p$ that previous iterations of function NOC created. In order to verify if $Poss$ can be reduced by $Poss_{cache}(p_p)$, the function generated a candidate for reduction $Poss_{com}$. Then, it checks if it is possible to obtain matrix $Poss$ by adding the columns of $Poss_{cache}(p_p)$ with the columns of $Poss_{com}$. If this combination results in matrix $Poss$, then we can use matrices $Poss_{cache}(p_p)$ and $Poss_{com}$ to represent the possibles tokens generated by the transitions of the event conflict instead of using $Poss$. Furthermore, since each token of place p_p represents the possibilities of $Poss_{cache}(p_p)$, instead of creating a new place to represent the possibilities of $Poss_{cache}(p_p)$, the function makes the firing of transition t_C add a token to place p_p, causing $Poss_{com}$ to be the only remaining matrix of possibilities that still requires to be modeled by a place. In order to verify if $Poss_{com}$ can be further reduced by other possibilities of $Poss_{cache}$, the function defines $Poss$ as $Poss_{com}$.

After doing all the reductions of matrix $Poss$ with respect to the matrices of $Poss_{cache}$, if there is a place $p \in P_D \backslash P_p$ such that each column of matrix $Poss$ has a number of tokens that is greater than or equal to a positive number $x \in \mathbb{N}$, then no matter which transition of T_C fired, all of them would have added x tokens to place p, meaning that we would be sure that place x would receive x tokens after firing t_C. Based on the previously mentioned idea, the function executes steps 34–36, where, for each place $p \in P_D \backslash P_p$, the function removes the number of tokens of place p of each column of $Poss$ by the minimum number of tokens in place p among the columns of $Poss$ and adds this same number to $Post_D(p, t_C)$, which makes the firing of transition t_C directly add those tokens to place p.

If matrix $Poss$ still has tokens after the aforementioned reductions, *i.e.*, $Poss$ has at least one element different from zero, then the function executes steps 38–42 to create a new place p_p, where each token in p_p indicates that the Petri net has the tokens of one column of $Poss$. The function also adds p_p to the set P_p and defines $Poss_{cache}(p_p)$ as $Poss$. Since the tokens of place p_p represent the possibilities of $Poss$ that were generated by the event conflict between the transitions T_C, the function makes the firing of transition t_C add a token to place p_p. Finally, function NOC executes function AOT to compute the output places of place p_p.

The pseudocode of function AOT is shown in Algortihm 5. Although function AOT is large, its execution simply creates the output transitions of a place p_p that was recently created by function NOC to represent the possibilities of matrix $Poss$.

In steps 1–6, function AOT verifies if there are two columns in matrix $Poss$ where one has tokens in place p_f and the other does not. If there are two columns that satisfies this condition, then a token in place p_p indicates that there could be a token in place p_f in the diagnoser Petri net, which means that we are able to infer that the fault event could have occurred whenever place p_p has a token. In order to flag this occurrence to the online diagnoser, function AOT creates a transition t_{fv}, which is labeled by the special event σ_{fv} and whose firing consumes and adds a token to place p_p.

Since each token in place p_p represents that the diagnoser Petri net has the tokens of one column of matrix $Poss$, some transitions of the diagnoser Petri net may be able to fire by consuming some tokens from the columns of matrix $Poss$, where those transitions would consume tokens from place p_p to indicate that they are consuming tokens from the possibilities of matrix $Poss$. In order to make the diagnoser Petri net able to fire those transitions by consuming tokens from place p_p, function AOT executes steps 7–42, where, for each possible group of possibilities of $Poss$ and each transition $t_r \in T_D$ that may fire by consuming tokens from this

Algorithm 5 Algorithm for the function AOT that creates all possible output transitions of place p_p

Inputs:

- $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$: diagnoser Petri net
- P_p : set of places created for each group of possibilities
- p_p : place of P_p that models multiple possibilities that are depicted in $Poss$
- $Poss$: matrix in which each column represents a possible marking that p_p models

Outputs:

- $\mathcal{NR} = (P_R, T_R, Pre_R, Post_R, \vec{m}_{0,R}, \Sigma_R, \ell_R, \rho_R)$: diagnoser Petri net that contains p_p and its output transitions

Set $\mathcal{NR} = (P_R, T_R, Pre_R, Post_R, \vec{m}_{0,R}, \Sigma_R, \ell_R, \rho_R) \leftarrow \mathcal{ND}$
If $Poss$ has two columns i and j such that $Poss(p_f, i) = 1$ and $Poss(p_f, j) = 0$
Create transition t_{fv} and add it to T_R
Set $Pre_R(p_p, t_{fv}) \leftarrow 1$
Set $Post_R(p_p, t_{fv}) \leftarrow 1$
Set $\ell_R(t_{fv}) \leftarrow \sigma_{fv}$
For each $t_r \in T_D$ do
Set $\overrightarrow{mul}_{max}$ as a vector that initially associates each column of $Poss$ with 0
For each column j of $Poss$ do
Set $P_{com} \leftarrow \{p \in P_D \backslash P_p : (Poss(p, j) > 0) \wedge (Pre_D(p, t_r) > 0)\}$
If $P_{com} \neq \emptyset$
Set $mul \leftarrow 0$
Do
Set $mul \leftarrow mul + 1$
For each possible combination of the repetitions of each column k of $Poss$, each varying from zero to $\overrightarrow{mul}_{max}(k)$, whose repetitions are represented by vector $\overrightarrow{rep}_{Poss}$ that associates each column of $Poss$ with a non negative integer, do
Set $flag \leftarrow True$
Set $\vec{c} \leftarrow \vec{0}_{|P_D \backslash P_p| \times 1}$
Set $\vec{c}(P_{com}) \leftarrow min([mul \times Poss(P_{com}, j), Pre_D(P_{com}, t_r)], column)$
Set $\vec{c}_{extra} \leftarrow Pre_D(P_D \backslash P_p, t_r) - \vec{c}$
Set $\vec{r} \leftarrow mul \times Poss(:, j) - \vec{c}$
For each column l of $Poss$ s.t. $\overrightarrow{rep}_{Poss}(l) \geq 1$ do
Set $P_{comL} \leftarrow \{p \in P_D \backslash P_p : (Poss(p, l) > 0) \wedge (\vec{c}_{extra}(p) > 0)\}$
Set $\vec{c}_L \leftarrow \vec{0}_{|P_D \backslash P_p| \times 1}$
Set $\vec{c}_L(P_{comL}) \leftarrow min([\overrightarrow{rep}_{Poss}(l) \times Poss(P_{comL}, l), \vec{c}_{extra}(P_{comL})], column)$
If $\vec{c}_L(P_{comL}) = min([(\overrightarrow{rep}_{Poss}(l) - 1) \times Poss(P_{comL}, l), \vec{c}_{extra}(P_{comL})], column)$
Set $flag \leftarrow False$ and break from the for loop
Set $\vec{c}_{extra} \leftarrow \vec{c}_{extra} - \vec{c}_L$
Set $\vec{r} \leftarrow \vec{r} + \overrightarrow{rep}_{Poss}(l) \times Poss(:, l) - \vec{c}_L$
If $flag$ is $True$
Create transition t_R and add it to T_R
Set $\ell_R(t_R) \leftarrow \ell_R(t_r)$
Add (t_t, t_R) and (t_R, t_t) to ρ_R for all (t_t, t_r) and (t_r, t_t) in ρ_R, respectively
Set $Pre_R(P_R \backslash P_p, t_R) \leftarrow \vec{c}_{extra}$
Set $Pre_R(P_p \backslash \{p_p\}, t_R) \leftarrow Pre_D(P_p \backslash \{p_p\}, t_r)$
Set $Pre_R(p_p, t_R) \leftarrow sum(\overrightarrow{rep}_{Poss}) + multi$
If $\ell_R(t_r) = \sigma_{fv}$
Set $Post_R(:, t_R) \leftarrow Pre_R(:, t_R)$
Else
Set $Post_R(:, t_R) \leftarrow Post_D(:, t_r)$
Set $Post_R(P_R \backslash P_p, t_R) \leftarrow Post_R(P_R \backslash P_p, t_R) + \vec{r}$
While $min(mul \times Poss(P_{com}, j) - Pre_D(P_{com}, t_r)) < 0$
Set $\overrightarrow{mul}_{max}(j) \leftarrow mul$

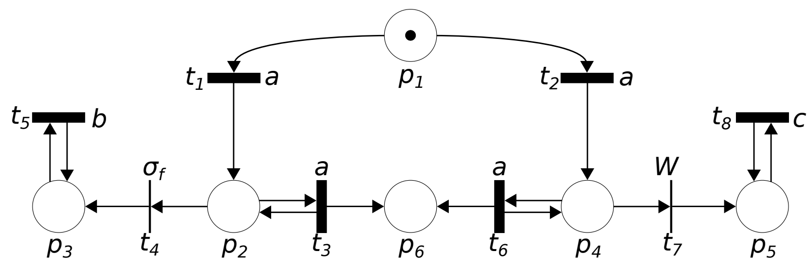


Figure 4.5: Petri net of Example 4.5.

group of possibilities, the function creates a copy transition t_R, which models the firing of t_r, but instead of consuming the same tokens as t_r when firing, transition t_R consumes some tokens required to enable transition t_r that are not in the group of possibilities and other tokens from place p_p to indicate that it is consuming tokens from the group of possibilities instead of other places of the diagnoser Petri net. Notice that transition t_R has the same label and priority relations as t_r, and if t_R is labeled by event σ_{fv}, then its firing adds the same tokens as it consumes, since the firing of transitions labeled by σ_{fv} should not change the diagnoser Petri net marking. If t_R is not labeled by σ_{fv}, then its firing adds the same tokens as the firing of transition t_r plus the tokens of the group of possibilities that would not be consumed by the firing of transition t_r.

We present the following example to show how function *NOC* solves two event conflicts of a diagnoser Petri net.

Example 4.5. *Consider the Petri net $\mathcal{N}$ of Figure 4.5 and its resulting diagnoser Petri net $\mathcal{ND}_0$ of Figure 4.6 after the execution of Algorithm 3. We will first use function NOC to solve the event conflict $\langle a, \{t'_1, t'_2\}, \vec{m}_{0,D}\rangle$ of $\mathcal{ND}_0$. Since this is the first execution of function NOC, we assume that the set of places P_p and the list of matrices $Poss_{cache}$ are initially empty.*

In order to change $\mathcal{ND}_0$ into a new Petri net $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ in which t'_1 are t'_2 cannot be simultaneously enabled, function NOC creates a new transition t_C, also named t'_{10}, which has a higher priority to fire than t'_1 and t'_2, i.e. both priority relations (t'_1, t'_{10}) and (t'_2, t'_{10}) are added to ρ_D. Furthermore, since the firing of transition t'_{10} is supposed

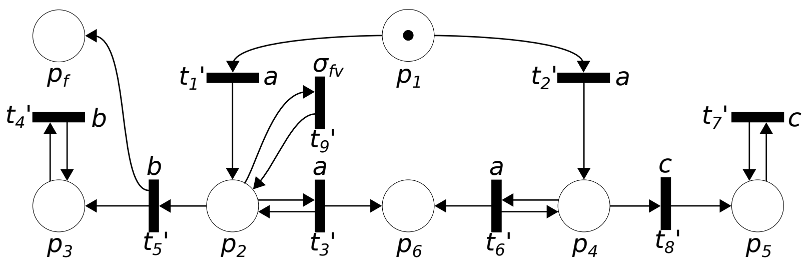


Figure 4.6: Diagnoser Petri net of the Petri net of Figure 4.5.

to model the firing of one of the transitions of T_C without specifying which one fired, it should consume the same number of tokens that either t'_1 or t'_2 consumes; therefore, the firing of transition t'_{10} also consumes a token from place p_1. Notice that whenever place p_1 has a token, only transition t'_{10} will be enabled, since it has a higher priority to fire than transitions t'_1 and t'_2; thus, the event conflict between transitions t'_1 and t'_2 will never occur in the modified diagnoser Petri net.

Since t'_{10} models multiple transitions, function NOC creates matrix Poss to store the possible tokens that either transition t'_1 or t'_2 generates after firing from a marking vector that has a token in p_1. Notice that the firing of t'_1 generates a token in place p_2, whereas the firing of t'_2 generates a token in place p_4; thus, the resulting matrix Poss that models both cases is as follows:

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

Notice that Poss cannot be reduced by other elements of $Poss_{cache}$ due to it being empty. Additionally, there are no rows of Poss whose minimal values are different from zero; therefore, the aforementioned Poss is the resulting matrix after the execution of step 36 of function NOC.

Since the resulting matrix Poss represents different possibilities of tokens, we cannot use the current places of the diagnoser Petri net to represent that we either *have one token in place p_2 or place p_4 with a single marking vector after the firing of t'_{10}. In order for $\mathcal{ND}$ to have a place whose tokens indicates that we either have one token in place p_2 or p_4, the function creates a new place p_7 for $\mathcal{ND}$, where each token of p_7 indicates that we either have one token in place p_2 or p_4, and the function also makes the firing of transition t'_{10} add a token to place p_7. Additionally, the function adds place p_7 to the set of places P_p and matrix Poss to the list $Poss_{cache}$ so that future iterations of function NOC may reduce their matrices Poss with the current matrix Poss.*

Notice that a token in place p_2 of $\mathcal{ND}_0$ enables the transitions t'_3, t'_5 and t'_9, whereas a token in place p_4 enables the transitions t'_6 and t'_8. Since place p_7 models two possibilities wherein one has a token in p_2 and the other has a token in p_4, a token in p_7 may contribute to the firing of one of the five previously mentioned transitions. Therefore, function AOT creates transitions t'_{11}, t'_{12}, t'_{13}, t'_{14} and t'_{15} as copies of transitions t'_3, t'_5, t'_6, t'_8 and t'_9, respectively, as shown in the resulting Petri net $\mathcal{ND}$ Figure 4.7. Notice that they all share place p_7 as their input place and their firing generates the same tokens as the transitions that generated them, with the exception of transition t'_{15}, whose firing adds a token to place p_7, since it is a transition labeled by σ_{fv}, which means that its firing cannot change the Petri net marking.

Now we show that we can execute function NOC again to solve the event conflict between transitions t'_{11} and t'_{13}, where both are labeled by event a. It is worth remarking that for this execution of function NOC, the set of places P_p has place p_7 and $Poss_{cache}(p_7)$ is equal to the matrix Poss of the previous iteration of function NOC.

In order to model the firing of both transitions t'_{11} and t'_{13}, function NOC creates transition t'_{16}, which has a higher priority than both transitions t'_{11} and t'_{13} and consumes a token from place p_7. Additionally, the function calculates the possible tokens that the firing of each transition may result, which is equal to the following matrix:

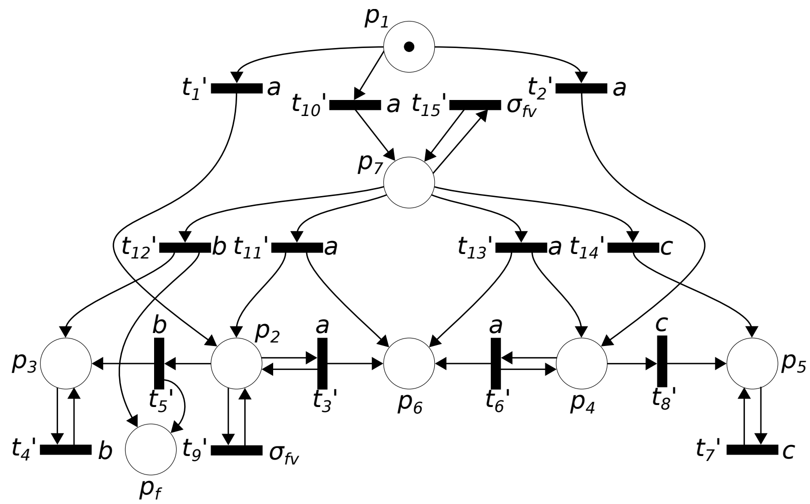


Figure 4.7: Resulting diagnoser Petri net after the execution of function NOC with respect to event a and the set of transitions $T_C = \{t'_1, t'_2\}$, where $\rho_D = \{(t'_1, t'_{10}), (t'_2, t'_{10})\}$.

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

Different from the previous iteration of function NOC, the elements P_p and $Poss_{cache}$ are not empty during this execution of NOC, which means that matrix Poss may be reduced by matrix $Poss_{cache}(p_7)$. In order to check whether that reduction is possible, the function creates a candidate for reduction matrix $Poss_{com}$, wherein each column is obtainable by reducing any column of Poss by a column of $Poss_{cache}(p_7)$. The only column that we are able to obtain in this manner is $Poss_{com} = [0, 0, 0, 0, 0, 1, 0, 0]^T$, which is the result of either the difference between the first columns of Poss and $Poss_{cache}(p_7)$ or the difference between the second columns of Poss and $Poss_{cache}(p_7)$. Notice that if we sum each column of $Poss_{cache}(p_7)$ with $Poss_{com}$, we obtain Poss again. In other words, if we combine the possibilities of $Poss_{com}$ with the possibilities of $Poss_{cache}(p_7)$, we are able to obtain the possibilities modeled in Poss; therefore, we are able to reduce Poss to $Poss_{com}$ by making the

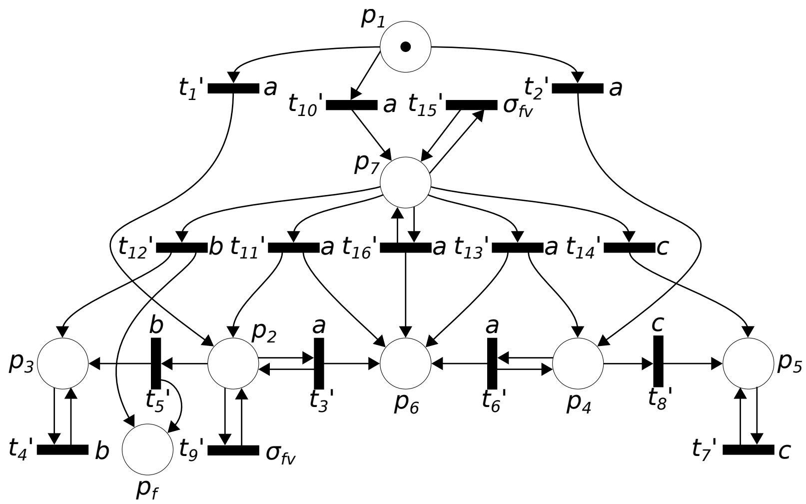


Figure 4.8: Diagnoser Petri net after the second iteration of function NOC, where $\rho_D = \{(t'_1, t'_{10}), (t'_2, t'_{10}), (t'_{11}, t'_{16}), (t'_{13}, t'_{16})\}$.

firing of transition t'_{16} add a token to place p_7 and replace $Poss$ with $Poss_{com}$ for the remainder of the function.

If $Poss = [0, 0, 0, 0, 0, 1, 0, 0]^T$, then we are sure that the firing of either transitions t'_{11} or t'_{13} adds a token to place p_6. The function detects and solves this behavior by making the firing transition t'_{16} add the tokens that are common among the columns of $Poss$, and since there is only one column in the matrix, the function makes the firing of transition t'_{16} add a token to place p_6. Additionally, since the firing of t'_{16} already adds a token to p_6, the function removes this token from $Poss$, making $Poss$ equal to a vector of zeros. Since $Poss$ only contain zeros after those operations, it does not model any possibilities of tokens, which means that the function does not need to create a new place to model it; therefore, the function ends its iteration without creating a new place, and the resulting diagnoser Petri net is depicted in Figure 4.8.

4.2.2 Using the diagnoser Petri net $\mathcal{ND}$ after the execution of function NOC to diagnose a labeled Petri net $\mathcal{N}$

As stated in Section 4.2.1, function NOC changes the diagnoser Petri net $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$ in such a way that the event conflicts that are related to a given set of transitions T_C are solved. Furthermore, the Petri net marking is able to model multiple markings of the original diagnoser Petri net $\mathcal{ND}_0$. In this section, we formally describe the properties of the resulting diagnoser Petri net $\mathcal{ND}$ with respect to a previous diagnoser Petri net $\mathcal{ND}_b$ after one execution of function *NOC* with $\mathcal{ND}_b$ as an input, which are similar to the Theorems 4.1 and 4.2 and are shown below.

Theorem 4.3. *Let $\mathcal{ND}_b = (P_{Db}, T_{Db}, Pre_{Db}, Post_{Db}, \vec{m}_{0,Db}, \Sigma_{Db}, \ell_{Db}, \rho_{Db})$ be a diagnoser Petri net obtained from a diagnosable Petri net $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ and let $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$, P_p and $Poss_{cache}$ be obtainable from Algorithm 4 assuming the diagnoser Petri net $\mathcal{ND}_b$, the set of transitions $T_C \subseteq T_{Db}$ labeled by event σ_C, the previous set of places P_{pb} that were created by previous iterations of function NOC and its associated list $Poss_{cache,b}$ are the inputs. If, for all sequences of observable events $s_o \in P_o(L(\mathcal{N}))$ such that $(\forall s \in P_o^{-1}(s_o) \cap L(\mathcal{N}), \sigma_f \in s)$, the firing of every transition sequence $s_{Db} \in LT(\mathcal{ND}_b)$ in $\mathcal{ND}_b$, satisfying $\ell_{Db}(s_{Db}) = s_o$, results in a marking vector in $\mathcal{ND}_b$ for which place $p_f \in P_{Db}$ has at least one token, then the firing of every transition sequence $s_D \in LT(\mathcal{ND})$ in $\mathcal{ND}$, satisfying $\ell_D(s_D) = s_o$, results in a marking vector at $\mathcal{ND}$ for which place $p_f \in P_D$ has at least one token.*

Proof. If the property described in Theorem 4.1 is valid for the Petri net $\mathcal{ND}_b$, then for all sequences of observable events $s_o \in P_o(L(\mathcal{N}))$ such that $(\forall s \in P_o^{-1}(s_o) \cap L(\mathcal{N}), \sigma_f \in s)$, the firing of any possible corresponding transition sequence $s_{Db} \in LT(\mathcal{ND}_b)$, where $\ell_{Db}(s_{Db}) = s_o$, results in a marking vector in $\mathcal{ND}_b$ where place $p_f \in P_{Db}$ has at least one token. If those possible transition sequences s_{Db} labeled by s_o are such that they do not contain a transition of T_C whose event observation σ_C causes an event conflict involving all transitions of T_C, then those transition sequences dynamics are the same as the dynamics of the transition sequences $s_D \in LT(\mathcal{ND})$ that are also labeled by s_o, since the transition $t_C \in T_D$ created in function *NOC* to solve event conflicts between the transitions of T_C is never used. This means that the transition sequences s_D are equal to the transition sequences s_{Db} and result in the same marking in both Petri nets. Therefore, in this case, the property described in Theorem 4.1 also applies to the Petri net $\mathcal{ND}$.

Whenever the possible transition sequences s_{Db} contain transitions of T_C and the observation of event σ_C causes event conflicts involving all transitions of T_C, the transition t_C that was created in $\mathcal{ND}$ by function *NOC* may fire in transition sequences s_D labeled by s_o, resulting in a marking vector that may be different from the one generated by s_{Db} by sequence s_{Db}. However, if the transition of s_{Db} that added a token to place p_f is such that it either was not involved in the conflict involving the transitions of T_C or was not enabled by a token that a transition of the event conflict generated, then the same transition fires in sequence s_D, meaning that s_D also adds a token in p_f. In order to demonstrate that the property of Theorem 4.1 is also true for the cases in which the transition of s_{Db} that adds a token to place p_f is either a transition of the event conflict involving the transitions T_C or is a transition that fires by consuming tokens generated by one of those transitions of the conflict, we separate them into two cases: *(i)* the event conflicts involving the transitions of T_C occur during the last event observation of s_o; *(ii)* the event conflicts involving the transitions of T_C occur before the last event observation of s_o.

During the last event observation of case *(i)*, if transition t_C is the last transition of the transition sequence s_D labeled by s_o, then its firing in $\mathcal{ND}$ may add tokens to the places of P_p to model different markings changing that each transition of T_C may cause with their firing. Furthermore, in step 34, the function makes it so that the firing of t_C directly generates the common tokens that all the transitions of T_C generate with their firing. Therefore, if all the transitions of T_C add tokens to place $p_f \in T_D$, then the firing of t_C also adds tokens to p_f; thus, in case *(i)*, the property of Theorem 4.1 also applies to $\mathcal{ND}$.

In case *(ii)*, other transitions may fire in s_D after the firing of transition t_C. If those transitions were not created by the last iteration of function *NOC*, then their dynamic would be the same in both Petri nets $\mathcal{ND}_b$ and $\mathcal{ND}$, making their firing cause the same change of tokens, including place p_f. However, if a transition

$t_R \in T_D$, which is different from t_C and was created during the last iteration of *NOC*, fires after t_C, then this transition was created by function *AOT*, whose firing consumes at least one token from the place $p_p \in P_D$ that was created by function *NOC* to model different markings that the firing of the transitions of T_C may result. Due to the way transition t_R was created in Algorithm 5, it is a copy of a transition $t_r \in T_{Db}$ of $\mathcal{ND}_b$ that consumes at least one token of the possibilities that place p_p models. Since t_R is a copy of t_r, its firing adds the same tokens to $\mathcal{ND}$ that t_r adds to $\mathcal{ND}_b$; thus, if t_r adds a token to place p_f, then t_R also adds a token to place p_f. Additionally, while creating transition t_R, function *AOT* assumes that each token that the firing of t_R consumes from place p_p is associated with a possibility that p_p models, and the tokens that t_r do not consume from that possibility are added to $\mathcal{ND}$ by t_R; therefore, if this possibility contains a token in p_f, then t_R also adds a token to p_f. Therefore, in case *(ii)*, the property of Theorem 4.1 also applies to $\mathcal{ND}$.

Since the property of Theorem 4.1 applies to all cases of s_D, then the property of Theorem 4.1 is also valid with respect to $\mathcal{ND}$. ∎

Theorem 4.4. *Let $\mathcal{ND}_b = (P_{Db}, T_{Db}, Pre_{Db}, Post_{Db}, \vec{m}_{0,Db}, \Sigma_{Db}, \ell_{Db}, \rho_{Db})$ be a diagnoser Petri net obtained from a diagnosable Petri net $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell)$ and let $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$, P_p and $Poss_{cache}$ be obtainable from Algorithm 4 assuming the diagnoser Petri net $\mathcal{ND}_b$, the set of transitions $T_C \subseteq T_{Db}$ labeled by event σ_C, the previous set of places P_{pb} that were created by previous iterations of function NOC and its associated list $Poss_{cache,b}$ are the inputs. If, for all sequences of observable events $s_o \in P_o(L(\mathcal{N}))$ such that $(\exists s \in P_o^{-1}(s_o) \cap L(\mathcal{N}))[\sigma_f \in s]$, there exists a transition sequence $s_{Db} \in LT(\mathcal{ND}_b)$, for which $\ell_{Db}(s_{Db}) = s_o$ and whose firing results in a marking vector in $\mathcal{ND}_b$ that either enables a transition $t_{fv} \in T_{Db}$ associated with event σ_{fv} or is such that place $p_f \in P_{Db}$ has at least one token, then there exists a transition sequence $s_D \in LT(\mathcal{ND})$, for which $\ell_D(s_D) = s_o$ and whose firing results in a marking vector in $\mathcal{ND}$ that either enables a transition $t_{fv} \in T_D$ associated with event σ_{fv} or is such that place $p_f \in P_D$*

has at least one token.

Proof. If the property described in Theorem 4.2 is valid for the Petri net $\mathcal{ND}_b$, then for all sequences of observable events $s_o \in P_o(L(\mathcal{N}))$ such that $(\exists s \in P_o^{-1}(s_o) \cap L(\mathcal{N}))[\sigma_f \in s]$, there is a corresponding transition sequence $s_{Db} \in LT(\mathcal{ND}_b)$ in $\mathcal{ND}_b$, where $\ell_D(s_{Db}) = s_o$ and whose firing results in a marking vector at $\mathcal{ND}_b$ that either enables a transition $t_{fv} \in T_{Db}$ associated with the event σ_{fv} or is such that place $p_f \in P_{Db}$ has at least one token. Given the sequence of observable events s_o that also satisfies the aforementioned condition, if there is a corresponded transition sequence $s_{Db} \in LT(\mathcal{ND}_b)$ labeled by s_o that satisfies the aforementioned condition and is such that it does not contain a transition of T_C involved in an event conflict that contains all transitions of T_C, then there is a transition sequence $s_D \in LT(\mathcal{ND})$ that is equal to s_{Db}. Therefore, s_D is also such that its resulting marking at $\mathcal{ND}$ either enables a transition $t_{fv} \in T_{Db}$ labeled by event σ_{fv} or is such that place p_f has at least one token.

If all corresponded transition sequences s_{Db} of $\mathcal{ND}_b$ that satisfies this theorem condition and are labeled by s_o contain transitions of T_C that are involved in event conflict composed of all transitions of T_C, then all associated transition sequences s_D of $\mathcal{ND}$ contain the transition t_C, which was created by the last iteration of function *NOC*. However, if one of those transition sequences s_{Db} is such that the resulting marking at $\mathcal{ND}_b$ after firing it either contains a token in place p_f or contain enough tokens to enable a transition labeled by event σ_{fv}, in which those tokens were not generated by a transition involved in the event conflict or a transition that consumes the tokens generated a transition of the event conflict, then those tokens are also generated in the associated transition sequence s_D of $\mathcal{ND}$ by transitions that were originally from $\mathcal{ND}_b$; therefore, in such a case, the condition of this theorem also applies to s_D.

If all the aforementioned transition sequences s_{Db} are such that either the transitions associated with the event conflict or the transitions whose firings consume tokens that a transition of the event conflict generates are the ones that cause the condition of the theorem to be true, then those sequences may be divided into two cases that vary according to the position of the event conflict in the sequence: *(i)* the event conflicts involving the transitions of T_C occur during the last event observation of s_o; *(ii)* the event conflicts involving the transitions of T_C occur before the last event observation of s_o.

Consider case *(i)*. If transition t_C is the last transition of the transition sequence s_D labeled by s_o, then it models the firing of multiple possible transition sequences s_{Db}, in which each ends with a different transition $t \in T_C$. Those modeled sequences may satisfy this theorem condition in three ways: *(i.i)* the firing of every transition of T_C adds a token to place p_f; *(i.ii)* the firing of at least one transition of T_C adds a token to place p_f and the firing of another transition of T_C does not; *(i.iii)* although the firing of any transitions of T_C does not add a token to place p_f, their resulting marking after the firing of a transition of T_C enables a transition labeled by event σ_{fv}. The case *(i.i)* is explained in Theorem 4.3, wherein all associated transitions s_D that ends with transition t_C have a token in p_f. Conversely, in case *(i.ii)*, the firing of t_C in s_D does not add a token to place p_f. Notice, however, that the firing of transition t_C in s_D models the firing of both transitions mentioned in such a way that the resulting marking models a possibility in which place p_f has a token and another possibility in which it does not. Therefore, due to steps 3–6 of the function AOT of Algorithm 5, the firing of transition t_C adds a token to a place $p_p \in P_p$ such that p_p has an output transition whose firing only consumes a token from it and is associated with event σ_{fv}. Lastly, in case *(i.iii)*, notice that the firing of transition t_C in place of a transition of T_C adds tokens to the places of P_p in such a way that $\mathcal{ND}$ has a copy transition for every transition of $\mathcal{ND}_b$ whose firing consumes tokens that could have been generated by transitions of T_C, which are modeled by the tokens of the places of P_p. Therefore, if the last transition of s_{Db} is a transition of T_C whose firing results in a marking vector that enables a transition $t_{fv} \in T_{Db}$ labeled by event σ_{fv}, then function AOT created a transition $t'_{fv} \in T_D$, which is a copy of t_{fv} and is enabled after the firing of sequence s_D by the resulting marking vector after the firing of t_C. Observe that, in all three cases, sequence s_D satisfied the property of this theorem; thus, for case *(i)*, the property of this theorem applies to s_D.

For case *(ii)*, if the firing of a transition $t_r \in T_{Db}$ in sequence s_{Db} after the firing of another transition $t \in T_C$ that is part of the event conflict consumes tokens from the tokens generated by t, then a similar process occurs in the associated transition sequence s_D, wherein transition t_C fires in place of t and a transition $t_R \in T_D$, which is a copy of transition t_r, fires in its place. Notice that the firing of transition t_r generates the same number of tokens as the firing of t_R, and those tokens are such that either place p_f has a token, or they contribute to make a transition $t_{fv} \in T_{Db}$ enabled, in which $\ell_{Db}(t_{fv}) = \sigma_{fv}$. Therefore, the tokens that the firing of transition t_R generates either add a token to place p_f, enables the same transition t_{fv}, but in $\mathcal{ND}$, or enables a transition which is a copy of t_{fv}, which is also labeled by event σ_{fv}. Therefore, in case *(ii)*, the condition of this theorem also applies to s_D.

It was shown that, for all cases of transition sequences $s_{Db} \in LT(\mathcal{ND}_b)$ that satisfies the theorem condition, there is a transition sequence $s_D \in LT(\mathcal{ND})$ that also satisfies the condition. ■

If the original diagnoser Petri net $\mathcal{ND}_0$ contain the properties of Theorems 4.1 and 4.2, which are needed for the fault diagnosis, then the resulting Petri net $\mathcal{ND}$ after the execution function *NOC* contain similar properties due to Theorems 4.3 and 4.4, which means that we are still able to diagnose the occurrence of the fault event using the resulting diagnoser Petri net. Additionally, if we further change $\mathcal{ND}$ with function *NOC* in order to solve more event conflicts, those properties remain valid for the resulting diagnoser Petri net. Therefore, a possible way of doing the online diagnosis using the diagnoser Petri net is use function *NOC* to solve the event conflicts of $\mathcal{ND}$ as they occur in $\mathcal{ND}$ while we try to fire the transitions of $\mathcal{ND}$ that are consistent with the event observations of the original Petri net $\mathcal{N}$, reducing the number of reachable states of $\mathcal{ND}$ that are consistent with the event observations of $\mathcal{N}$ to a single state.

After Algorithm 3 generates the initial diagnoser Petri net $\mathcal{ND}_0$, we are able to start the online diagnosis of $\mathcal{N}$, where, for each observed event $\sigma_o \in \Sigma_o^*$, we fire a transition $t_D \in T_D$ of $\mathcal{ND}$ labeled by σ_o that is enabled. If we are able to fire more than one transition labeled by σ_o in $\mathcal{ND}$, it means that $\mathcal{ND}$ current marking causes an event conflict involving transitions of a set of transitions $T_C \subseteq T_D$ and an event $\sigma_o \in \Sigma_o$, which we solve by executing function *NOC*; therefore, if σ_o causes an event conflict in $\mathcal{ND}$, we modify $\mathcal{ND}$ in such a way that this event conflict does not occur in the modified diagnoser Petri net. If we do that for all event conflicts that occur during the event observations, the event sequence $s_o \in \Sigma_o^*$ that was observed will label only one transition sequence $s_D \in LT(\mathcal{ND})$, whose firing results in a single marking vector $\vec{m}_D$. Since the resulting diagnoser Petri net contains the properties described in Theorems 4.3 and 4.4, we are able to enumerate the possible cases of the diagnoser after the observation of sequence s_o by analyzing the resulting marking vector $\vec{m}_D$, such that $\vec{m}_{0,D}[s_o\rangle\vec{m}_D$, as follows:

- C_F: The diagnoser is sure that a fault event has occurred if $\vec{m}_D(p_f) > 0$.
- C_D: The diagnoser is sure that a fault event may have occurred before the next event observation after s_o if there is a transition $t_{fv} \in T_D$ such that $\ell(t_{fv}) = \sigma_{fv}$ and $\vec{m}_D[t_{fv}\rangle$.
- C_N: The diagnoser is sure that no fault event has occurred if both conditions described in C_F and C_D are false.

Finally, we proposed the pseudocode of Algorithm 6 that does the diagnosis of the fault event of a Petri net $\mathcal{N}$ using the diagnoser Petri net $\mathcal{ND}$ that was generated by Algorithm 3. Notice that the algorithm shows the case in which the diagnoser currently is during its execution through the output C, whose value changes to one of the three cases C_F, C_D and C_N after each event observation. Furthermore, if one of the observed events causes an event conflict, the algorithm solves it by using function *NOC*, which makes the same observation result in the firing of only one transition.

Algorithm 6 Algorithm that executes the diagnosis of fault events of a Petri net using the diagnoser Petri net that was generated by Algorithm 3

Inputs:

- $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$: diagnoser Petri net

Outputs:

- C: Indicates the current case of the fault occurrence, whose values can be C_F, C_D or C_N

Create P_p as an empty set
Create $Poss_{cache}$ as an empty list of matrices
Set $\vec{m} \leftarrow \vec{m}_{0,D}$
If $(\exists t \in T_D)[(\ell(t) = \sigma_{fv}) \wedge (\vec{m}[t\rangle)]$
  Set $C \leftarrow C_D$
Else
  Set $C \leftarrow C_N$
Do
  Wait for the observation of an event $\sigma_C \in \Sigma_D \backslash \sigma_{fv}$
  Set $T_C \leftarrow \{t \in T_D : (\ell_D(t) = \sigma_C) \wedge (\vec{m}[t\rangle)\}$
  If $|T_C| \geq 2$
    Let $[\mathcal{ND}, P_{p,T}, Poss_{cache}] \leftarrow NOC(\mathcal{ND}, \sigma_C, T_C, P_p, Poss_{cache})$
    If $|P_{p,T}| > |P_p|$
      Set p_p as the only place of $P_{p,T} \backslash P_p$
      Set $\vec{m}(p_p) \leftarrow 0$
      Set $P_p \leftarrow P_{p,T}$
    Set $T_C \leftarrow \{t \in T_D : (\ell_D(t) = \sigma_C) \wedge (\vec{m}[t\rangle)\}$
  Set t_C as the only transition of T_C
  Set $\vec{m} \leftarrow \vec{m} + Post(:, t_C) - Pre(:, t_C)$
  If $\vec{m}(p_f) > 0$
    Set $C \leftarrow C_F$
  Else if $(\exists t \in T_D)[(\ell(t) = \sigma_{fv}) \wedge (\vec{m}[t\rangle)]$
    Set $C \leftarrow C_D$
  Else
    Set $C \leftarrow C_N$
While the online diagnoser is running

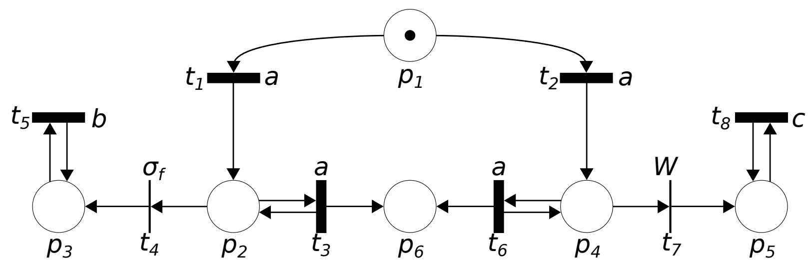


Figure 4.9: Petri net of Example 4.6.

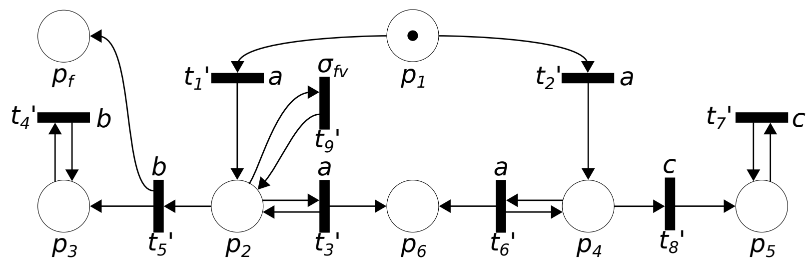


Figure 4.10: Diagnoser Petri net of the Petri net of Figure 4.9.

In order to show how the online diagnosis using the diagnoser Petri net and function *NOC* works, we present the following example.

Example 4.6. *Consider the Petri net* $\mathcal{N}$ *of Figure 4.9 and its resulting diagnoser Petri net* $\mathcal{ND}_0$ *of Figure 4.10 after the execution of Algorithm 3, which are the same Petri nets as the ones of Example 4.5.*

If event a is observed in $\mathcal{N}$ *while the current marking vector of* $\mathcal{ND}_0$ *is equal to its initial marking vector* $\vec{m}_{0,D}$*, either transitions* t'_1 *or* t'_2 *could fire to justify this observation, which means that the event conflict* $\langle a, \{t'_1, t'_2\}, \vec{m}_{0,D}\rangle$ *occurs in* $\mathcal{ND}_0$*. As shown in Example 4.5, we can use function NOC to solve this event conflict, modifying the diagnoser Petri net to the Petri net of Figure 4.11. After solving the above event conflict, the only transition of* $\mathcal{ND}$ *that we can fire to model the occurrence of event a is transition* t'_{10}*, which moves the token of place* p_1 *to place* p_7*. Since a token in place* p_7 *enables transition* t'_{15}*, which is labeled by* σ_{fv}*, we are able to conclude that the fault event could have occurred.*

If we observe event a for a second time, the transitions of $\mathcal{ND}$ *that are enabled by a token in place* p_7 *and whose firings can justify this observation are transitions* t'_{11} *and* t'_{13}*, which means that the Petri net is in another event conflict. In order to*

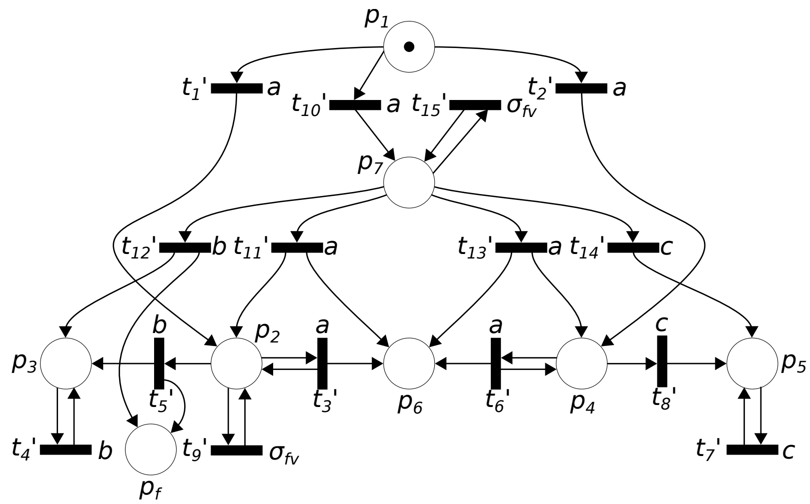


Figure 4.11: Resulting diagnoser Petri net after the execution of function NOC with respect to event a and the set of transitions $T_C = \{t'_1, t'_2\}$, where $\rho_D = \{(t'_1, t'_{10}), (t'_2, t'_{10})\}$.

solve such conflict, we execute function NOC which changes the diagnoser Petri net to the one of Figure 4.12

After solving the event conflicts involving transitions t'_{11} *and* t'_{13}*, the only transition that can justify the second observation of event* a *is transition* t'_{16}*, which adds one token to place* p_6*.*

If the event b *is observed after the first two occurrences of event* a*, the only transition of* $\mathcal{ND}$ *that can fire to correspond to this observation is* t'_{12}*, whose firing results in a marking vector that contains a token in place* p_f*, meaning that we are able to conclude that the a fault event has occurred. If the observed event is* c *instead, the corresponding transition that fires in* $\mathcal{ND}$ *is* t'_{14}*, whose firing results in a marking vector that does not contain a token in place* p_5 *and* p_6*. Since this marking vector does not have a token in place* p_f *and does not enable a transition labeled by event* σ_{fv}*, we are sure that no fault event has occurred after this event observation.*

4.2.3 Boundness of function *NOC*

We are able to diagnose the occurrence of a fault event of a Petri net $\mathcal{N}$ by analyzing the marking vectors that the diagnoser Petri net $\mathcal{ND}$ reaches after the firing of a transition sequence associated with the observed event sequence that occurs in $\mathcal{N}$.

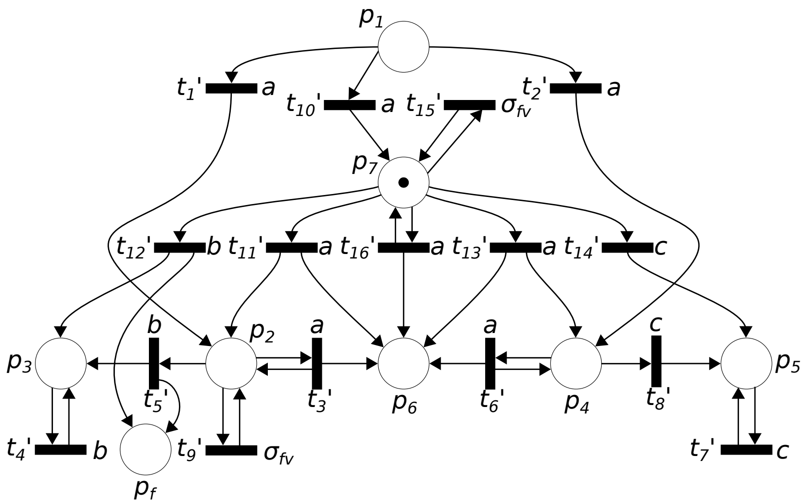


Figure 4.12: Diagnoser Petri net after the second iteration of function NOC, where $\rho_D = \{(t'_1, t'_{10}), (t'_2, t'_{10}), (t'_{11}, t'_{16}), (t'_{13}, t'_{16})\}$.

Furthermore, in order to limit the number of marking vectors that we analyze to one, whenever the observed event sequence causes an event conflict in $\mathcal{ND}$, function NOC solves this conflict by adding places and transitions to the diagnoser Petri net. However, the places and transitions that NOC adds to $\mathcal{ND}$ may cause more event conflicts, and, in some cases, by solving each event conflict that occurs due to event observations with function NOC, the number of places and transitions of $\mathcal{ND}$ may grow indefinitely.

To exemplify the aforementioned issue, consider the diagnoser Petri net $\mathcal{ND}$ that contains the places and transitions of Figure 4.13(a). The initial marking of $\mathcal{ND}$ causes the event conflict involving event a and transitions t_1 and t_2. If we execute execute function NOC with respect to this event conflict, we obtain the Petri net depicted in Figure 4.13(b), in which transitions t_1 and t_2 are hidden since they have lower priority than transition t_5 and are never enabled by the new Petri net reachable marking.

Although the first execution of function NOC solves the event conflicts between transitions t_1 and t_2, it also creates the event conflict involving transitions t_6 and t_7, which occurs after transition t_5 fires. If we solve the new event conflict with function NOC, we obtain the diagnoser Petri net of Figure 4.14, wherein transitions t_6 and t_7 are hidden. Notice that the resulting diagnoser Petri net contains a structure

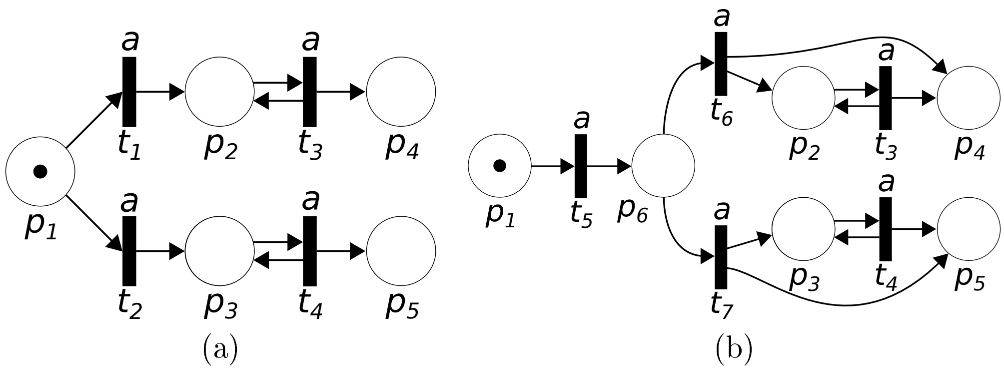


Figure 4.13: Part of a diagnoser Petri net $\mathcal{ND}$(a) and the resulting Petri net after the first execution of function *NOC* with respect to transitions t_1 and t_2.

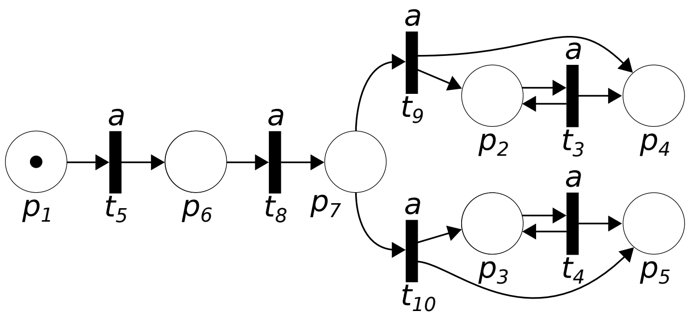


Figure 4.14: Part of a diagnoser Petri net $\mathcal{ND}$ after two executions of function *NOC*.

similar to the Petri net of Figure 4.13(b), but with the addition of one place and three transitions. Furthermore, the dynamics of transitions t_9 and t_{10} are similar to the dynamics of t_6 and t_{10}, which means that if we execute function *NOC* to solve the event conflict between t_9 and t_{10}, the resulting Petri net will have another event conflict similar to t_9 and t_{10}; therefore, this part of the diagnoser Petri net will grow indefinitely as we solve their event conflicts with function *NOC*.

It is worth remarking that the reason why this diagnoser Petri net grows indefinitely after each execution of function *NOC* is because each iteration has to create a new place to model different possibilities that cannot be reduced by other possibilities of the places that were created by previous iterations of *NOC*, and the output transitions of the created place cause more event conflicts, which causes the creation of more places. A token in place p_6 of the Petri net in Figure 4.14 models that the Petri net has a token in either p_2 or p_3, whereas p_7 is such that the first possibility has a token in p_2 and p_4, and the second possibility has a token in p_3 and p_5. Notice that the function is not able to reduce the possibilities modeled by place p_7 with the possibilities of place p_6 during the execution of steps 15–33, and the possibilities that future iterations of *NOC* cannot be reduced either.

Even though the structure of the Petri net of Figure 4.13(a) causes the diagnoser Petri net to grow indefinitely, there is a class of diagnoser Petri nets in which this growth does not occur due to the number of places that can be created by several iterations of function *NOC* being limited. Thus, in this section, we propose an additional assumption with respect to the diagnoser Petri net that allows the assertion that the diagnoser Petri net will not grow indefinitely as function *NOC* solves the event conflicts that may occur by the possible event observations.

In order to define the new assumption, let M_{s_o} be a matrix in which each column is a marking vector $\vec{m}$ such that there is a transition sequence $s \in LT(\mathcal{ND})$ labeled by the event sequence $s_o \in P_o(L(\mathcal{N}))$ such that $\vec{m}_{0,D}[s\rangle\vec{m}$, *i.e*, each column of M_{s_o} is a marking vector that may be reached after the firing of a transition sequence whose observation is equal to s_o. Furthermore, let M_{red,s_o} be M_{s_o} after the execution of the steps 35–36 of Algorithm 4, in which each row is reduced by their minimum value and all repeated columns of M_{s_o} after the above operation are removed.

Notice that M_{s_o} depicts the possible marking vectors the diagnoser Petri net may be in after the observation of event sequence s_o, whereas M_{red,s_o} only models the tokens that we are not certain that are in the resulting marking vector. Since the places created by function *NOC* also model these uncertain tokens, if we execute function *NOC* to solve the event conflicts caused by the observation of sequence s_o, all the tokens of M_{red,s_o} will be modeled by the places created by function *NOC*. Furthermore, the columns of matrix M_{red,s_o} may be reduced by matrices $M_{red,s}$, where sequence $s \in \overline{\{s_o\}}\backslash\{s_o\}$, *i.e.*, sequence s is a prefix of s_o different from s_o, in a similar manner that matrix *Poss* is reduced in steps 15–33 and step 35 of function *NOC*, and if this reduction results in a matrix of zeros, then the possibilities of M_{red,s_o} may be modeled by dividing it between previous possibilities that occurs before the occurrence of the last event of s_o.

If the aforementioned reduction is possible, then function *NOC* is also able to reduce the possibilities it considers in matrix *Poss* after the observation of the last event of s_o to a matrix of zeros, since matrix $M_{red,s}$ may modeled by places of P_p. If the function is able to completely reduce *Poss*, then, according to step 37, this iteration of the function does not create a new place; therefore, if there exists a natural number $k \in \mathbb{N}$ such that all event sequences $s_o \in P_o(L(\mathcal{N}))$ that contain k events or more result in matrices M_{red,s_o} that can be completely reduced by the matrices associated with the prefixes of s_o, then the execution of function *NOC* with respect to the diagnoser Petri net $\mathcal{ND}$ can only create a limited number of places. With this last conclusion, we are able to express an additional assumption that we can consider in order to be able to optimize the fault diagnosis, as follows:

A4. The diagnoser Petri net $\mathcal{ND}$ of $\mathcal{N}$ is such that there is a number $k \in \mathbb{N}$ such that for all event sequences $s_o \in P_o(L(\mathcal{N}))$, if $|s_o| \geq k$, then matrix M_{red,s_o} can be completely reduced by matrices $M_{red,s}$, where $s \in \overline{\{s_o\}} \backslash \{s_o\}$.

If we consider that Assumption **A4** is valid for the diagnoser Petri net, then the number of places that may be created by multiple iterations of function *NOC* is finite, which means that the number of transitions created by function *AOT*, which is only executed after the creation of a place, is also limited. Furthermore, each transition t_C that the function *NOC* creates to solve an event conflict is such that it does not increase the number of event conflicts in the Petri net, since its firing only replaces the firing of the transitions involved in a conflict; thus, Assumption **A4** guarantees that multiple iterations of function *NOC* are only able to create a limited number of places and transitions in the diagnoser Petri net $\mathcal{ND}$.

4.3 Using Assumption A4 to optimize the online diagnosis

If we consider that the generated diagnoser Petri net $\mathcal{ND}$ of a Petri net $\mathcal{N}$ satisfies Assumption **A4**, then we are able to solve all the event conflicts of $\mathcal{ND}$ during the offline computation of the diagnoser by using function *NOC* to solve those event conflicts before the observation of any events in $\mathcal{N}$. This allows the online diagnosis of $\mathcal{N}$ by using a diagnoser Petri net that does not require the execution of function *NOC* to solve the event conflicts during the online diagnosis.

In order to find all event conflicts of $\mathcal{ND}$, we use Algorithm 7, which is a modified version of Algorithm 1 that finds the extended coverability tree (ECT) of $\mathcal{ND}$, which allows the enumeration of all event conflicts within its input Petri net through its arcs. Notice that we are not able to use the CT of $\mathcal{ND}$ to enumerate all events conflicts because of two issues: *(i)* by replacing the number of tokens of a place within a node of CT with the symbol ω, an event conflict that only occurs when the number of tokens of that place is less than a threshold may be hidden; *(ii)* although a transition may remove tokens from a place whose number of tokens is ω after firing multiple times, the CT is not able to reduce the number of tokens from a place associated with ω, which means that an event conflict that occurs due to this reduction may be hidden. In order to show those issues and how Algorithm 7 solve them, we present two examples.

With respect to issue *(i)*, consider the Petri net of Figure 4.15(a) and its resulting CT of Figure 4.15(c). If transition t_1 fires once, then place p_1 gains a token, which enables transitions t_1 and t_2, and since both transitions are labeled by event a, the event conflict $\langle a, \{t_1, t_2\}, [1]^T \rangle$ occurs. However, this event conflict is not present in the resulting CT, since Algorithm 1 replaces the marking vector $[1]^T$ with $[\omega]^T$, which enables transitions t_1, t_2 and t_3 and hides the aforementioned event conflict. In order to solve this issue in the generated tree, Algorithm 7 executes the steps 15–21, in which, before the function replaces the value of the number of tokens of a place p of a

Algorithm 7 Algorithm ECT to obtain the extended coverability tree of a labeled priority Petri net

Inputs:

- $\mathcal{N} = (P, T, Pre, Post, \vec{m}_0, \Sigma, \ell, \rho)$: labeled priority Petri net model

Outputs:

- *Nodes*: $n_P \times l$ matrix whose columns are the l nodes of the tree
- *Arcs*: $l \times l$ matrix, wherein each element from the i-th row and j-th column is equal to the transition of $\mathcal{N}$ or the symbol r, which is associated with the connection between the i-th node and the j-th node, if such connection exists, or zero, otherwise

```
Set Nodes ← [m⃗_0]
Set l ← 1
Set Arcs ← [0]
Set nodesToCheck ← [1]
Set parents ← [0]
While nodesToCheck is not empty do
  Set currentNode ← nodesToCheck(1)
  Remove nodesToCheck(1) from nodesToCheck
  For each t such that Nodes(:, currentNode)[t⟩ do
    Set newNode ← Nodes(:, currentNode) + Post(:, t) − Pre(:, t)
    Set newNodeReg ← newNode
    Set currentParent ← currentNode
    While (currentParent ≠ 0) do
      If newNode ≥ Nodes(:, currentParent)
        Set T_enab ← {t_n ∈ T : newNode[t_n⟩}
        For each p ∈ P such that newNode(p) > Nodes(p, currentParent)
          Set newNodeW ← newNode
          Set newNodeW(p) ← ω
          Set T_enab,W ← {t_n ∈ T : newNodeW[t_n⟩}
          If T_enab = T_enab,W
            Set newNode(p) ← ω
            If (Post(p, :) ≥ Pre(p, :))
              Set newNodeReg(p) ← ω
      Set currentParent ← parents(currentParent)
    Set Nodes ← [Nodes, newNode]
    Set parents ← [parents, currentNode]
    Set Arcs ← [[Arcs, 0⃗_{l×1}]^T, 0⃗_{l+1×1}]^T
    Set l ← l + 1
    Set Arcs(currentNode,l) ← t
    Set flag ← True
    While (currentParent ≠ 0) and flag is True do
      If (Nodes(:, currentParent) = newNode)
        flag ← False
      If (Nodes(:, currentParent) = newNodeReg)
        Set newNodeReg ← newNode
      Set currentParent ← parents(currentParent)
    If flag is True
      Set nodesToCheck ← [nodesToCheck, l]
      If newNode ≠ newNodeReg
        Set Nodes ← [Nodes, newNodeReg]
        Set parents ← [parents, currentNode]
        Set Arcs ← [[Arcs, 0⃗_{l×1}]^T, 0⃗_{l+1×1}]^T
        Set l ← l + 1
        Set Arcs(l-1,l) ← r
```

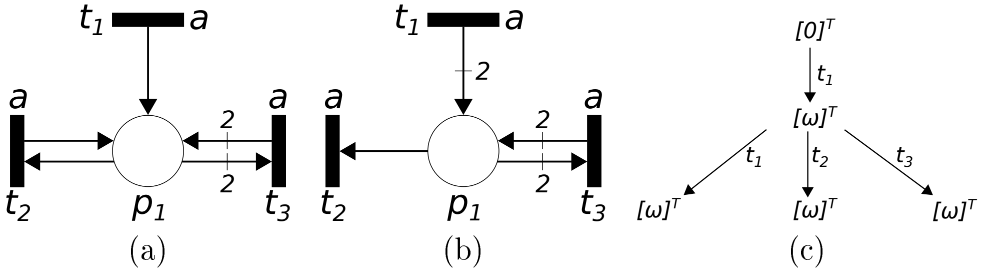


Figure 4.15: Petri nets of the examples of the first (a) and second (b) issues of Algorithm 1, and the resulting CT (c) of both Petri nets.

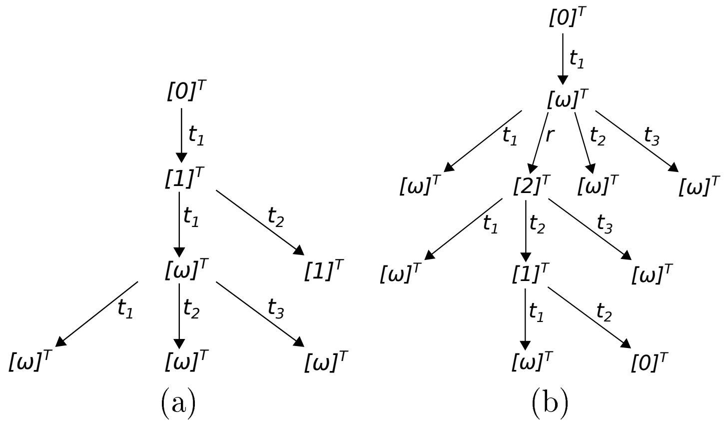


Figure 4.16: Resulting ECT (a) and (b) of the Petri nets of Figures 4.15(a) and 4.15(b), respectively.

node with ω, the function verifies if the marking vector of the new node enables the same transitions as the resulting marking vector after altering the number of tokens of p with ω. If the set of transitions enabled by the two are equal, then replacing the number of tokens of p with ω does not hide an event conflict; therefore, the algorithm replaces the number of tokens of p with ω. The resulting ECT of the Petri net is depicted in Figure 4.16(a), wherein the event conflict $\langle a, \{t_1, t_2\}, [1]^T \rangle$ is present. This event conflict occurs because even though the marking vector $[1]^T$ that is generated after the firing of t_1 from the first node is greater than $[0]^T$, which should cause $[1]^T$ to change to $[\omega]^T$, the algorithm verifies that $[1]^T$ does not enable transition t_3, which is enabled by $[\omega]^T$; therefore, the algorithm does no change $[1]^T$ to $[\omega]^T$.

For the issue *(ii)*, consider the Petri net of Figure 4.15(b) and its resulting CT of Figure 4.15(c). If transition t_1 fires, two tokens are added to place p_1, which enables transition t_1, t_2 and t_3. If transition t_2 fires after the firing of t_1, a token is removed from place p_1, changing its marking to one, which only enables transitions t_1 and t_2. Since all transitions are labeled by event a, after the firing of sequence t_1t_2, the Petri net is in the event conflict $\langle a, \{t_1, t_2\}, [1]^T \rangle$. Although this event conflict occurs, it is not present in the CT of the Petri net due to Algorithm 1 replacing the marking of p_1 with ω after the firing of t_1, which prevents the firing of transition t_2 from removing tokens from it. In order to circumvent this issue, Algorithm 7 does a regression process through steps 11, 22–23, 34–35 and 38–42.

Before verifying if a newly created node associated with the marking vector *newNode* needs to have one of its markings replaced by ω, Algorithm 7 creates the vector *newNodeReg* as a copy of it, which will possibly be a regression of *newNode*. Whenever the algorithm changes a marking of a place p in *newNode* to ω and $Post(p,:) \geq Pre(p,:)$, the algorithm also changes the corresponding marking in *newNodeReg* to ω. If the comparison $Post(p,:) \geq Pre(p,:)$ is false, then it is possible that the firing of a transition reduces the number of tokens of p; therefore, the algorithm does not change the marking of p in *newNodeReg*. If the resulting *newNode* is different from *newNodeReg* and *newNodeReg* is not equal to a predecessor of *newNode*, then, after the creation of the node with respect to *newNode*, the algorithm creates an additional node with respect to *newNodeReg*, whose parent is *newNode* and the arc connecting *newNode* to *newNodeReg* is labeled by the symbol r. Notice that the addition of *newNodeReg* after *newNode* allows a transition to remove tokens from a place whose tokens were replaced by ω in *newNode*, which can reveal event conflicts that were hidden in the CT. If we execute Algorithm 7 with respect to the Petri net of Figure 4.15(b), we obtain the ECT of Figure 4.16(b), in which a regression occurs after the first firing of transition t_1. Notice that the regression changes the marking of p_1 from ω to two, which is the resulting marking after the firing of t_1. Furthermore, after firing transition t_2 from node $[2]^T$, we obtain the marking vector $[1]^T$, which shows the occurrence of the event conflict $\langle a, \{t_1, t_2\}, [1]^T \rangle$ that was hidden in the CT. It is worth remarking that the marking vector $[1]^T$ was not replaced by $[\omega]^T$ because both marking vectors enable different set of transitions, which is a problem related to the issue *(i)*, whose solution was already shown.

Since both issues *(i)* and *(ii)* are solved in Algorithm 7, we are able to find the event conflicts of $\mathcal{ND}$ by analyzing the groups of arcs of the ETC of $\mathcal{ND}$ that originate from a same node, in which each group is represented in a column of matrix *Arcs*. Therefore, we are able to enumerate an event conflict by selecting all the transitions of a column of *Arcs* that are labeled by a same event $\sigma \in \Sigma_D \backslash \sigma_{fv}$. Based on this logic, we execute Algorithm 8 to make all modifications with respect to the diagnoser Petri net $\mathcal{ND}$ during the offline computation, so that the resulting diagnoser Petri net does not contain any event conflicts involving the events of $\Sigma_D \backslash \sigma_{fv}$.

Algorithm 8 Algorithm that solves all event conflict of a diagnoser Petri net

---

Inputs:

- $\mathcal{ND} = (P_D, T_D, Pre_D, Post_D, \vec{m}_{0,D}, \Sigma_D, \ell_D, \rho_D)$: diagnoser Petri net of $\mathcal{N}$

Outputs:

- $\mathcal{ND}$: diagnoser Petri net of $\mathcal{N}$ that does not have any event conflict involving events that are different from σ_{fv}

Create P_p as an empty set
Create $Poss_{cache}$ as an empty list of matrices
Do
  Let $[Nodes, Arcs] \leftarrow ETC(\mathcal{ND})$
  For each i-th column of *Arcs* do
    For each $\sigma \in \Sigma_D \backslash \{\sigma_{fv}\}$
      Create T_C as an empty set
      For each j-th row of *Arcs* do
        If $Arcs(j,i) \in T_D \backslash T_C \wedge \ell(Arcs(j,i)) = \sigma$
          Add $Arcs(j,i)$ to T_C
      If $|T_C| \geq 2$
        Break from the for loop
    If $|T_C| \geq 2$
      Break from the for loop
  If $|T_C| \geq 2$
    Let $[\mathcal{ND}, P_p, Poss_{cache}] \leftarrow NOC(\mathcal{ND}, \sigma_C, T_C, P_p, Poss_{cache})$
While $|T_C| \geq 2$

If we execute Algorithm 6 after solving the event conflicts of the diagnoser Petri net with Algorithm 8, the steps 12–17 are never executed in Algorithm 6 during the online diagnosis, due to the diagnoser Petri net not having any event conflicts involving the events that may be observed during the system operation. By never executing those steps, the speed of the online diagnosis increases considerably, since the algorithm only does simple Petri net operations, such as changing and verifying the marking vector of the Petri net.

Remark 4.4. *Notice that the algorithms that are used to compute the diagnoser Petri net $\mathcal{ND}$ and solve its event conflicts can cause some of the transitions of $\mathcal{ND}$ to never be enabled by the reachable markings of $\mathcal{ND}$. However, we cannot remove those transitions after each execution of function NOC because some of them may be enabled by the reachable markings of $\mathcal{ND}$ after the execution of more iterations of function NOC. On the other hand, if we solve all event conflicts of $\mathcal{ND}$ during the offline computation using Algorithms 8, then there will not be another execution of function NOC; therefore, in this case, we can remove those transitions that are never enabled by the reachable markings of $\mathcal{ND}$, which can be found by verifying the transitions that are never enabled by the nodes of the extended coverability tree of $\mathcal{ND}$.*

In order to show that we can use Algorithms 8 and 6 to do the online diagnosis of a Petri net system using a limited structure that other works, such as the one of Cabasino et al.[23], can only diagnose using structures that may grow indefinitely, we present the following example.

Example 4.7. *Consider the Petri net $\mathcal{N}$ of Figure 4.17, which is the same Petri net of Example 3.1, wherein the Petri net was diagnosed using the online diagnoser proposed by [23]. In the Example 3.1, the number of considered markings that justified the observed event sequences could grow indefinitely, which means that the structure of the diagnoser could also grow with respect to the observed event sequence.*

If we execute Algorithm 3 with respect to $\mathcal{N}$, we obtain the initial diagnoser Petri net $\mathcal{ND}_0$ of Figure 4.18. Notice that $\mathcal{ND}_0$ is unbounded, since t'_1, which is labeled by event a, may fire indefinitely from the initial marking, increasing the number of tokens in place p_1. Furthermore, those marking vectors that cause the Petri net to be unbounded compose event conflicts involving event a and transitions t'_1, t'_2 and t'_6.

Consider the values of $M_{red,\epsilon}$, $M_{red,a}$, $M_{red,aa}$ and $M_{red,aaa}$ with respect to $\mathcal{ND}_0$, whose values are, respectively, as follows:

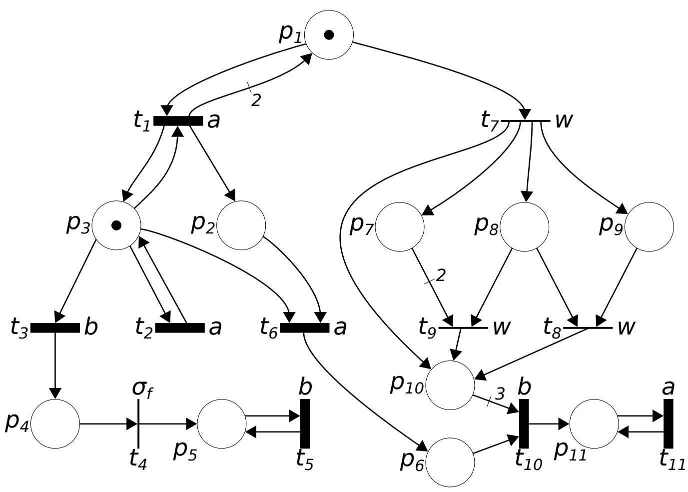


Figure 4.17: Petri net $\mathcal{N}$ of Example 4.7.

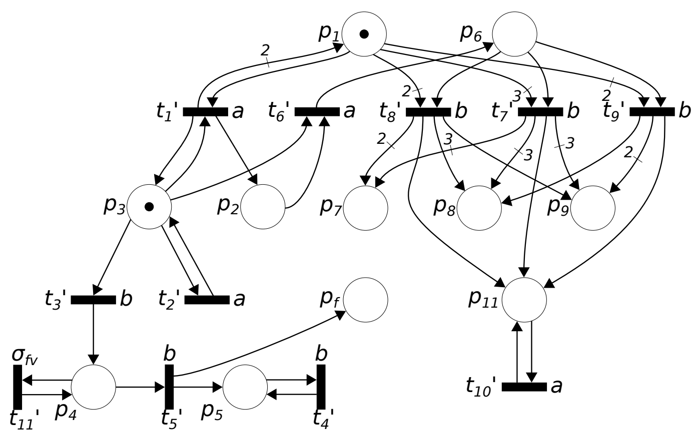


Figure 4.18: Initial diagnoser Petri net $\mathcal{ND}_0$ of the Petri net of Figure 4.17.

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 2 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 & 1 & 2 & 2 & 3 \\ 0 & 0 & 1 & 1 & 2 & 3 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

*By reducing $M_{red,aaa}$ with $M_{red,aa}$ while executing the steps 15–33 and step 35 of function NOC, we obtain the same matrix as $M_{red,a}$, which can also be completely reduced by $M_{red,a}$, i.e., if we reduce $M_{red,aaa}$ with $M_{red,aa}$ and $M_{red,a}$, we obtain a matrix with one column of zeros. Similar to the reduction of $M_{red,aaa}$, M_{red,s_o} can be completely reduced by its previous reduced set of markings for $s_o \in \{aaa\}\{a\}^*$. In addition, if event b is observed afterwards, the diagnoser Petri net marking can change to two possibilities: (i) the Petri net marking has a token in place p_4; (i) the Petri net marking has a token in place p_{11}, where, in both cases, neither place p_3 nor place p_6 have tokens. In case (i), only transition t'_5, which is labeled by event b, is enabled, and if it fires, it moves the token from place p_4 to p_5, which only enables transition t'_4, whose firing does not change the Petri net marking and is also labeled by event b. In the case (ii), the only transition that is enabled is t'_{10}, whose firing also does not change the Petri net marking and is labeled by event a. In both cases, the number of reachable markings is limited and they cannot be considered for the same observation, since cases (i) and (ii) occur due to the observations of events b and a, respectively, which means that they can be completely reduced after the observation of either events a or b. Therefore, since there is a value k for which all M_{red,s_o} can be completely reduced, in which $s_o \in \{\Sigma_D \backslash \{\sigma_{fv}\}\}^*$ and $|s_o| > k$, Assumption **A4** applies to $\mathcal{ND}_0$, which means that we are able to solve all of its event conflicts of $\mathcal{ND}_0$ with Algorithm 8.*

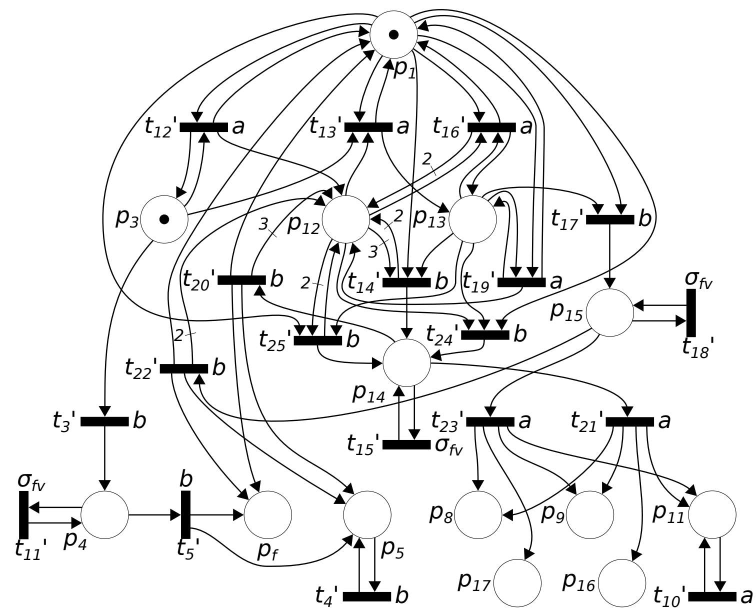


Figure 4.19: Diagnoser Petri net $\mathcal{ND}$ of the Petri net of Figure 4.17 after the execution of Algorithm 8.

The resulting diagnoser Petri net $\mathcal{ND}$ after the execution of Algorithm 8 to solve all of its event conflicts has a total of 18 places and 707 transitions. However, if we remove the places whose markings never change and transitions that never fire, the numbers of places and transitions are reduced to 14 and 19, respectively, as shown in Figure 4.19. Furthermore, the priority relations of ρ_D are shown below.

$$\rho_D = \{(t'_{12}, t'_{13}), (t'_{19}, t'_{13}), (t'_{17}, t'_{14}), (t'_{24}, t'_{14}), (t'_{25}, t'_{14}),$$

$$(t'_{12}, t'_{16}), (t'_{19}, t'_{16}), (t'_{17}, t'_{24}), (t'_{17}, t'_{25}), (t'_{24}, t'_{25})\}$$

If event a is observed an undetermined number of times, the transition sequence $s \in \{t'_{12}t'_{13}t'_{19}\}\{t'_{16}\}^*$ fires in $\mathcal{ND}$ to correspond to it. After the firing of sequence $t'_{12}t'_{13}t'_{19}$, $\mathcal{ND}$ reaches a marking vector $\vec{m}$ in which places p_1, p_{12} and p_{13} have one token each, which enables transition t'_{16}. Notice that the firing of transition t'_{16} only adds a token to place p_{12}; thus, if transition t'_{16} fires multiple times to correspond to the multiple observations of event a, place p_{12} gains multiple tokens while the Petri net structure remains the same, which allows the diagnoser to observe an undetermined number of observations of event a without the growth of the diagnoser

Petri net structure.

If event b is observed after the firing of an undetermined number of observations of event a that is greater than 5, transition t'_{14} *fires in* $\mathcal{ND}$ *to correspond it, removing token from places* p_1, p_{12} *and* p_{13} *and adding a token to place* p_{14}. *Notice that the transition* t'_{15}, *whose label is* σ_{fv}, *becomes enabled by the token in* p_{14}*; therefore, the diagnoser is able to assert that the fault event could have occurred after the observation of event b. If event b is observed again, transition* t'_{20} *fires to model it. Notice that the firing of* t'_{20} *adds a token to place* p_f*; thus, we are sure that the fault event has occurred after the second observation of event b. If event a is observed after the first observation of event b instead, transition* t'_{21} *fires in* $\mathcal{ND}$, *removing a token from place* p_{14} *and adding one token to places* p_8, p_9, p_{11} *and* p_{17}. *Observe that all tokens of the resulting marking vector do not enable transitions labeled by event* σ_{fv}, *and since place* p_f *does not have a token, we are able to confirm that the fault event has not occurred.*

Chapter 5

Conclusion and future works

We have presented in this work a new approach for the online diagnosis of fault events of discrete event systems modeled by labeled Petri net, which is able to use the reachable marking vectors of a diagnoser labeled priority and λ-free Petri net to determine the occurrence of fault events concurrently with the operation of a labeled Petri net system.

The presented approach is able to perform the online diagnosis of fault events of diagnosable labeled Petri net systems that do not contain cycles of unobservable transitions. Additionally, when Assumption **A4** is satisfied for the diagnoser Petri net, all computations regarding the diagnoser Petri net can be performed offline, allowing the execution of the online diagnosis of a Petri net with a structure that does not grow indefinitely. To the best of our knowledge, our approach is the first one in the literature that is able to execute the online diagnosis of systems modeled by labeled Petri nets such as the one of Example 4.7 without requiring structures that can possibly grow with respect to the observed event sequences, as it was shown to be the case for the online diagnoser of Cabasino et al.[23] in Examples 3.1.

Although the main objective of this work is online diagnosis of labeled Petri net systems, we have also indirectly contributed to other fields of studies. By using the extended coverability tree that is generated by Algorithm 7, instead of the normal coverability tree that is generated by Algorithm 1, we are able to obtain more information about the transitions that may fire in an unbounded labeled priority

Petri net. Furthermore, in order to use the diagnoser Petri net to diagnose the fault occurrences in systems, we proposed in Section 4.2 a new approach for the state estimation of λ-free labeled Petri nets that consists of solving the event conflicts of the Petri net using functions NOC and AOT. Although the approach for the state estimation proposed here is personalized for the fault diagnosis, it can also be adapted to a more general state estimation of λ-free labeled Petri nets. Finally, we have introduced the concept of event conflicts and shown possible solutions for it, which can be used in other problems of DESs, such as the supervisory control of Petri net systems.

Regarding future works, we list the following possible continuations of this work:

1. Further studies involving the diagnoser Petri nets may show that it is possible to use them to solve the problem of diagnosability of Petri net systems. A possible approach for deciding the diagnosability of a Petri net, for example, is to verify if there is a cycle of reachable markings in the diagnoser Petri net that causes the diagnoser to always conclude that the fault event could have occurred.

2. The construction of the diagnoser Petri net can be optimized to avoid the addition of transitions that are never enabled by the reachable markings.

3. The state estimation of the diagnoser Petri net using function NOC can be further improved by changing the type of the Petri net of the diagnosed Petri net to a more complex structure, such as labeled priority Petri net with variable arc weights. Preliminary work has shown that this change may allow the increase on the class of Petri nets whose event conflicts may be solved during the offline computation of the diagnoser.

4. Find a different approach for the state estimation of λ-free labeled Petri nets, which could be directly used to estimate the states of the diagnoser Petri net generated by Algorithm 3.

5. The same ideas presented in this work regarding the conversion of a Petri net to a λ-free labeled priority Petri net and the state estimation of λ-free labeled priority Petri nets can be applied to other studies involving DESs modeled by Petri nets, such as the study of opacity, fault prediction, and so forth.

References

[1] CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2 ed. New York, Springer, 2008.

[2] DAVID, R., ALLA, H. *Discrete, continuous, and hybrid Petri nets*, v. 1. Springer, 2010.

[3] LIN, F. "Diagnosability of discrete event systems and its applications", *Discrete Event Dynamic Systems*, v. 4, n. 2, pp. 197–212, 1994.

[4] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., et al. "Diagnosability of discrete-event systems", *IEEE Transactions on automatic control*, v. 40, n. 9, pp. 1555–1575, 1995.

[5] QIU, W., KUMAR, R. "Decentralized failure diagnosis of discrete event systems", *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, v. 36, n. 2, pp. 384–395, 2006.

[6] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C. "Generalized robust diagnosability of discrete event systems", *IFAC Proceedings Volumes*, v. 44, n. 1, pp. 8737–8742, 2011.

[7] CARVALHO, L. K., MOREIRA, M. V., BASILIO, J. C., et al. "Robust diagnosis of discrete-event systems against permanent loss of observations", *Automatica*, v. 49, n. 1, pp. 223–231, 2013.

[8] NUNES, C. E., MOREIRA, M. V., ALVES, M. V., et al. "Codiagnosability of networked discrete event systems subject to communication delays and intermittent loss of observation", *Discrete Event Dynamic Systems*, v. 28, n. 2, pp. 215–246, 2018.

[9] MOREIRA, M. V., JESUS, T. C., BASILIO, J. C. "Polynomial time verification of decentralized diagnosability of discrete event systems", *IEEE Transactions on Automatic Control*, v. 56, n. 7, pp. 1679–1684, 2011.

[10] SANTORO, L. P., MOREIRA, M. V., BASILIO, J. C. "Computation of minimal diagnosis bases of Discrete-Event Systems using verifiers", *Automatica*, v. 77, pp. 93–102, 2017.

[11] CABRAL, F. G., MOREIRA, M. V., DIENE, O., et al. "A Petri net diagnoser for discrete event systems modeled by finite state automata", *IEEE Transactions on Automatic Control*, v. 60, n. 1, pp. 59–71, 2014.

[12] DEBOUK, R., LAFORTUNE, S., TENEKETZIS, D. "Coordinated decentralized protocols for failure diagnosis of discrete event systems", *Discrete Event Dynamic Systems*, v. 10, n. 1, pp. 33–86, 2000.

[13] CONTANT, O., LAFORTUNE, S., TENEKETZIS, D. "Diagnosability of discrete event systems with modular structure", *Discrete Event Dynamic Systems*, v. 16, n. 1, pp. 9–37, 2006.

[14] CABASINO, M. P., GIUA, A., LAFORTUNE, S., et al. "A new approach for diagnosability analysis of Petri nets using verifier nets", *IEEE Transactions on Automatic Control*, v. 57, n. 12, pp. 3104–3117, 2012.

[15] LIU, B., GHAZEL, M., TOGUYÉNI, A. "On-the-fly and incremental technique for fault diagnosis of discrete event systems modeled by labeled petri nets", *Asian Journal of Control*, v. 19, n. 5, pp. 1659–1671, 2017.

[16] LEFEBVRE, D., DELHERM, C. "Diagnosis of DES with Petri net models", *IEEE Transactions on Automation Science and Engineering*, v. 4, n. 1, pp. 114–118, 2007.

[17] RAMÍREZ-TREVIÑO, A., RUIZ-BELTRÁN, E., RIVERA-RANGEL, I., et al. "Online fault diagnosis of discrete event systems. A Petri net-based approach", *IEEE Transactions on Automation Science and Engineering*, v. 4, n. 1, pp. 31–39, 2007.

[18] GENC, S., LAFORTUNE, S. "Distributed diagnosis of discrete-event systems using Petri nets". In: *International Conference on Application and Theory of Petri Nets*, pp. 316–336. Springer, 2003.

[19] AL-AJELI, A., BORDBAR, B. "Fourier-motzkin method for failure diagnosis in petri net models of discrete event systems". In: *2016 13th International Workshop on Discrete Event Systems (WODES)*, pp. 165–170. IEEE, 2016.

[20] PAIVA, P. R., DE FREITAS, B. I., CARVALHO, L. K., et al. "Online fault diagnosis for smart machines embedded in Industry 4.0 manufacturing

systems: A labeled Petri net-based approach", *IFAC Journal of Systems and Control*, p. 100146, 2021.

[21] DOTOLI, M., FANTI, M. P., MANGINI, A. M., et al. "On-line fault detection in discrete event systems by Petri nets and integer linear programming", *Automatica*, v. 45, n. 11, pp. 2665–2672, 2009.

[22] BASILE, F., CHIACCHIO, P., DE TOMMASI, G. "An efficient approach for online diagnosis of discrete event systems", *IEEE Transactions on Automatic Control*, v. 54, n. 4, pp. 748–759, 2009.

[23] CABASINO, M. P., GIUA, A., POCCI, M., et al. "Discrete event diagnosis using labeled Petri nets. An application to manufacturing systems", *Control Engineering Practice*, v. 19, n. 9, pp. 989–1001, 2011.

[24] GIUA, A., CORONA, D., SEATZU, C. "State estimation of λ-free labeled Petri nets with contact-free nondeterministic transitions", *Discrete Event Dynamic Systems*, v. 15, n. 1, pp. 85–108, 2005.

[25] BEST, E., KOUTNY, M. "Petri net semantics of priority systems", *Theoretical Computer Science*, v. 96, n. 1, pp. 175–215, 1992.

[26] RU, Y., HADJICOSTIS, C. N. "State estimation in discrete event systems modeled by labeled Petri nets". In: *Proceedings of the 45th IEEE Conference on Decision and Control*, pp. 6022–6027. IEEE, 2006.