UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Ana Caroline Tondo Bonafin

Fault Diagnosis of Discrete-Event Systems Modeled by a class of Labeled Petri Nets

Florianópolis - SC
2021

Ana Caroline Tondo Bonafin

Fault Diagnosis of Discrete-Event Systems Modeled by a class of Labeled Petri Nets

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do título de mestre Engenharia de Automação e Sistemas. Orientador: Prof. Felipe Gomes de Oliveira Cabral, Dr.

Florianópolis - SC
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Bonafin, Ana Caroline
Fault Diagnosis of Discrete-Event Systems Modeled by a class of Labeled Petri Nets / Ana Caroline Bonafin ; orientador, Felipe Gomes de Oliveira Cabral, 2021.
60 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2021.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Fault diagnosis. 3. Petri nets. 4. Discrete Event Systems. 5. Fault detection. I. Gomes de Oliveira Cabral, Felipe. II. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Engenharia de Automação e Sistemas. III. Título.

Ana Caroline Tondo Bonafin

Fault Diagnosis of Discrete-Event Systems Modeled by a class of Labeled Petri Nets

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Felipe Gomes de Oliveira Cabral, Dr
Universidade Federal de Santa Catarina

Prof. Gustavo da Silva Viana, Dr.
Universidade Federal do Rio de Janeiro

Prof. José Eduardo Ribeiro Cury, Dr.
Universidade Federal de Santa Catarina

Prof. Marcos Vicente Moreira, Dr.
Universidade Federal do Rio de Janeiro

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre Engenharia de Automação e Sistemas.

Documento assinado digitalmente
Werner Kraus Junior
Data: 12/11/2021 10:59:02-0300
CPF: 531.085.239-53
Verifique as assinaturas em https://v.ufsc.br

Prof. Werner Kraus Junior, Dr.
Coordenador do Programa de
Pós-Graduação

Documento assinado digitalmente
Felipe Gomes de Oliveira Cabral
Data: 11/11/2021 19:37:28-0300
CPF: 124.807.927-22
Verifique as assinaturas em https://v.ufsc.br

Prof. Felipe Gomes de Oliveira Cabral, Dr.
Orientador

Florianópolis - SC, 2021.

“ This work is dedicated to the few who live up to the opportunities they have, while many waste them and thousands never have them, but they always dreamed of them. ” (Unknown author)

ACKNOWLEDGEMENTS

The development of the approach here presented was a huge challenge to me, understand and deal with the complexity of the Petri nets formalism was not an easy task, but provided me a lot of knowledge and new analytic capacities for solving problems on daily basis. It can be said that, not only the objective of this work were reach, but mine personal development expectations as well.

With that being said, I am very grateful to everyone that has been with me in the last two years, specially my mom and dad that always worked hard so I could get this far in my career and, all my friends, in special Amanda, Dierli and Afonso for all the support when things got rough, I love you all and, without you, the process would be so much harder.

I am infinitely grateful to my advisor Felipe G. Cabral for all his support and empathy throughout the Masters. I would like to thank the examination board for accepting the invitation and saving some time to evaluate this work, especially Professor Marcos, for all the help during the development of this research.

RESUMO

Neste trabalho, um método de diagnóstico de falhas para Sistemas a Eventos Discretos (SEDs) modelados por redes de Petri rotuladas (RPRs) é proposto. Para tanto, nesta dissertação é suposto que algumas transições da rede de Petri são não observáveis, incluindo as transições de falha. O método de diagnóstico consiste em dois passos: primeiro, a detecção online de falhas é feita; e então, candidatos de falha são isolados com base na sequência de eventos observada. A detecção online é feita com base na construção de uma RPR a partir do modelo do comportamento livre de falha do sistema, chamada de rede de Petri do comportamento observável (RPCO), cujas transições são todas observáveis e cuja linguagem gerada é igual à linguagem observável do modelo livre de falha quando algumas condições são satisfeitas. Além disso, neste trabalho, é mostrado que a RPCO pode ser implementada para detecção de falha ao invés do grafo de alcançabilidade da rede que modela o sistema, ou mesmo parte dele, o que leva a um rápido método de detecção de falhas sem necessitar de um grande uso de memória. Um estudo de caso é também apresentado para ilustrar o método proposto

Palavras-chave: Detecção de falhas, Diagnóstico de falhas, Redes de Petri, Sistemas a eventos discretos.

RESUMO EXPANDIDO

Introdução

Sistemas automatizados estão sujeitos à ocorrência de falhas que podem alterar seu comportamento nominal, o que pode causar danos aos equipamentos e ferir operadores humanos. Portanto, se faz necessária a implementação de um sistema de diagnóstico eficiente que detecte e isole a ocorrência de falhas a fim de prevenir danos ao sistema. A fim de resolver esse problema, o diagnóstico de falhas em Sistemas a Eventos Discretos (SEDs) tem recebido bastante atenção na literatura, sendo as abordagens mais comuns desenvolvidas para sistemas modelados por autômatos e Redes de Petri (RP).

Neste trabalho, um método de diagnóstico de falhas para sistemas modelados por Redes de Petri Rotuladas (RPR) é proposto. Em um primeiro momento, a detecção da ocorrência de uma falha é feita e depois os candidatos à falha são isolados. Para isso, a sequência de eventos observada e o modelo completo do comportamento do sistema em RPR são utilizados para isolar o tipo de falha que ocorreu. O esquema de detecção de falhas é baseado na construção da chamada Rede de Petri do Comportamento Observável (RPCO), cujas transições são rotuladas somente por transições observáveis e à linguagem gerada é igual a linguagem livre de falha do sistema. A RPCO fornece todos os eventos observáveis do comportamento livre de falha do sistema que são factíveis depois da observação de uma sequência de eventos. A detecção da falha é feita caso um evento não factível na marcação atual da RPCO seja observado.

Além disso, neste trabalho, um estudo de caso de um sistema pesagem-mistura introduzido na norma internacional IEC 6084 é apresentado. Nesse sistema, foram considerados dois tipos de falha com o objetivo de ilustrar as etapas de detecção e isolamento em um sistema diagnosticável prático. Outro estudo de caso também é apresentado para comparar a eficiência do método proposto com outras abordagens relevantes da literatura. A metodologia proposta neste trabalho é justificável para sistemas que possuem grande comportamento paralelo, uma vez que o detector de falhas, a RPCO, é obtido utilizando-se o formalismo de redes de Petri, o que, geralmente, permite um grafo menor para esse tipo de sistema.

Objetivos

Este trabalho tem como objetivo o desenvolvimento de um método de diagnóstico de falhas para Sistemas a Eventos Discretos modelados por Redes de Petri, utilizando somente a estrutura da rede que modela o sistema, sem a necessidade de construir uma estrutura adicional. O método é desenvolvido para redes de Petri que possuem transições observáveis e não observáveis. As etapas de detecção e isolamento da falha são separadas para garantir maior velocidade na detecção, o que é fundamental para sistemas que demandam uma rápida resposta após a ocorrência de uma falha. O método também é comparado com outras abordagens propostas na literatura.

Metodologia

A metodologia do trabalho consiste do levantamento bibliográfico de métodos de diagnóstico de falhas em SEDs modelados por autômatos e redes de Petri. Além disso, trabalhos que abordam temas como identificação de sistemas, prognosticabilidade, controle super-

visório e diagnosticadores mínimos foram considerados com o objetivo de identificar os métodos propostos na literatura para esses problemas correlatos.

Após o levantamento bibliográfico, identificou-se o estado da arte do problema de diagnóstico de falhas em SEDs modelados por redes de Petri. A abordagem mais utilizada propõe converter a rede de Petri em um grafo orientado e um conjunto de marcações que podem ser usados para o diagnóstico de falhas. Entretanto, esse método tem um resultado computacional ruim para redes que possuem muito comportamento paralelo. Assim, seguiu-se o estudo de uma técnica de construção de um diagnosticador de falhas que preservasse a natureza distribuída das redes de Petri, sem a conversão para um novo grafo que não preserve essa propriedade.

Identificou-se, em seguida, hipóteses para o modelo do sistema que viabilizam a aplicação do método. Além disso, condições suficientes que garantem que a linguagem observada do sistema é igual à linguagem gerada da RPCO foram identificadas. Os resultados teóricos do trabalho foram provados e aplicações práticas foram conduzidas para comparação com o estado da arte.

Resultados e Discussões
Neste trabalho, um método para detecção de falhas em sistemas a eventos discretos modelados por redes de Petri rotuladas seguras é proposto. A abordagem consiste na remoção de todas as transições não observáveis do modelo do comportamento livre de falha do sistema, resultando na Rede de Petri do Comportamento Observável. Uma vez que a RPCO é construída, o processo de detecção pode ser realizado calculando os eventos observáveis factíveis para uma determinada marcação. Se um evento que não é factível na marcação atual é observado, a falha é detectada. Neste trabalho é suposto que o modelo em Rede de Petri do sistema é diagnosticável. A análise da diagnosticabilidade pode ser feita offline usando qualquer método proposto na literatura.

Dois estudos de caso são conduzidos para ilustrar a eficiência do método quando comparado a trabalhos clássicos propostos na literatura. A RPCO do primeiro estudo de caso, um sistema de pesagem-mistura introduzido na IEC 60848, possui 10 lugares e 7 transições, enquanto que o grafo de alcançabilidade do modelo do sistema tem 197 estados e 412 transições. O grafo de marcações base, usado como diagnosticador tradicional para sistemas modelados por redes de Petri, possui 51 estados e 116 transições. Para o segundo estudo de caso, um sistema de manufatura, a RPCO possui 22 lugares e 14 transições, enquanto que o grafo de alcançabilidade do modelo do sistema tem 30.880 estados e 140.748 transições e o grafo de marcações base possui 435 estados e 1.182 transições. Isso mostra a grande redução em memória que pode ser obtida ao se utilizar o método proposto neste trabalho.

Considerações Finais
Com a evolução dos sistemas automatizados, o tamanho dos modelos estão crescendo tanto em número de componentes quanto em complexidade, portanto, diagnosticar corretamente possíveis ocorrências de falhas que podem alterar o comportamento nominal desses sistemas, prevenindo danos a equipamentos e mantendo a segurança de operadores, pode não ser uma tarefa fácil. Neste trabalho, um esquema de diagnóstico de falhas para sistemas a eventos discretos modelados por uma classe de redes de Petri rotuladas é apresentado.

O método consiste de dois passos: (i) primeiro a ocorrência de um evento de falha é detectada e depois (ii) os eventos candidatos à falha são isolados. A principal contribuição deste trabalho é que o modelo do sistema usado para a detecção de falhas é, no geral, muito menor do que o grafo de alcançabilidade do sistema, o que permite o uso de menos memória para implementação do detector de falhas do que usando as técnicas tradicionais. Isso faz com que a falha possa ser detectada de forma rápida, sem o uso de qualquer grafo de alcançabilidade. O método proposto foi aplicado em dois estudos de caso com sucesso, em que a eficiência do detector de falhas proposto pode ser comprovada comparando-se o tamanho da RPCO calculada com os grafos obtidos de acordo com outras propostas da literatura.

Como trabalhos futuros, investiga-se a possibilidade de se relaxar algumas das hipóteses consideradas neste trabalho. Um método de detecção e diagnóstico de falhas para outras classes de redes de Petri que preserve a estrutura da rede que modela o sistema original também é um tema em aberto. Os resultados deste trabalho foram apresentados no Simpósio Brasileiro de Automação Inteligente (SBAI) de 2021 e uma contribuição foi submetida para publicação na *Control Engineering Practive Journal*.

Palavras-chave: Detecção de falhas, Diagnóstico de falhas, Redes de Petri, Sistemas a eventos discretos.

ABSTRACT

In this work, a fault diagnosis method for Discrete-Event Systems (DES) modeled as labeled Petri nets (LPN) is proposed. In order to do so, it is assumed that some transitions of the Petri net are unobservable, including the fault transitions. The diagnosis method consists of two steps: first, an online fault detection is carried out; and then, by using the observed sequence of events and the Petri net model, the fault candidates are isolated. The online fault detection is based on the construction of an LPN from the fault-free system behavior model, called observable behavior Petri net (OBPN), whose transitions are all observable, and whose generated language is guaranteed to be equal to the observable language of the fault-free system model when some conditions are satisfied. It is also shown that the OBPN can be used for fault detection instead of implementing the reachability graph, or even part of it, of the Petri net system model, which leads to a fast fault detection method without requiring the use of a large amount of memory. A case study is also presented in order to illustrate the proposed method.

Keywords: Fault detection, Fault diagnosis, Petri nets, Discrete Event Systems.

LIST OF FIGURES

LIST OF ABBREVIATIONS AND ACRONYMS

BRG	Basis Reachability Graph
DES	Discrete-Event System
LPN	Labeled Petri Net
OBPN	Observable Behavior Labeled Petri net
PND	Petri Net Diagnoser
PNs	Petri Nets
SFC	Sequential Function Chart
SMPN	State Machine Petri Net

LIST OF SYMBOLS

L	Language of the system
*	Kleene-closure operator
P	Set of places
T	Set of transitions
Pre	Function of ordinary arcs that connect places to transitions
$Post$	Function of ordinary arcs that connect transitions to places
$\mathcal{N}$	Petri net
x_0	Initial marking function
Σ	Set of events
σ_f	Fault event
$\mathcal{N}_l$	Labeled Petri net that models the system
S	Sequence of transitions
ℓ	Labeling function
Σ_u	Set of unobservable events
Σ_o	Set of observable events
Σ_f	Set of fault events
T_o	Set of observable transitions
T_u	Set of unobservable transitions
T_f	Set of fault transitions
$\underline{x_c}$	Current state of the system
Γ	Set of feasible events
σ_o	Obervable event
$\mathcal{N}_o$	Observable behavior Labeled Petri net of the system
t_u	Unobservable transition of $\mathcal{N}_N$
T_{N_u}	Set of unobservable transitions of $\mathcal{N}_N$
T_{N_o}	Set of observable transitions of $\mathcal{N}_N$

CONTENTS

1 INTRODUCTION

Automated systems are subject to fault occurrences that can alter their nominal behavior, which can cause equipment damages and potentially injury human operators. Thus, it is necessary to implement an automatic fault diagnosis system to efficiently detect and isolate the fault occurrence to prevent damages to the system. In order to address this problem, fault diagnosis of Discrete-Event Systems (DESs) has received considerably attention in the literature (PROCK, 1991; SAMPATH et al., 1995, 1996; CONTANT et al., 2004; MIYAGI; RIASCOS, 2006; GIUA et al., 2007; CABASINO et al., 2014; CABRAL et al., 2015; SANTORO et al., 2017; RAN et al., 2017; NUNES et al., 2018) where the most common approaches consider systems modeled as automata or Petri Nets (PNs).

In Sampath et al. (1995), a diagnoser for DESs modeled as automata is proposed, and a necessary and sufficient condition to verify language diagnosability, *i.e.*, the capability of detect and isolate the fault event occurrence after a bounded number of event observations, using the diagnoser automaton is presented. The diagnoser presented in Sampath et al. (1995) is based on the computation of an observer automaton of the system model, whose number of states can grow exponentially with the system state space cardinality. Since then, methods to avoid the use of observers for diagnosability verification, and have polynomial complexity with the system state space cardinality, have been proposed (QIU; KUMAR, 2006; MOREIRA et al., 2011, 2016).

In order to avoid the computation of an observer automaton for diagnosis, a Petri Net Diagnoser (PND) for DESs modeled by automata is proposed in Cabral et al. (2015). The PND is capable of estimating online the states of the fault-free system model, and can be obtained in polynomial time with respect to the system model size. If, after the observation of an event, the state estimate of the PND is equal to the empty set with respect to a specific fault, then this fault is diagnosed. Methods to convert the PND to Programmable Logic Controller languages, such as Ladder and Sequential Function Chart (SFC), are also proposed in Cabral et al. (2015).

Several fault diagnosis strategies for DESs modeled by PNs have also been proposed in the literature. In the PN modeling, events are usually associated with transitions, leading to the so-called Labeled Petri Net (LPN). The main advantage of modeling systems using a PN formalism instead of automata is the more complex structure that makes it more suitable to describe some structural information of the system, such as concurrency and synchronization, leading to more compact models. The fault diagnosis of bounded PN models can be carried out directly from the reachability graph of the net by applying the methods developed for systems modeled by automata. The main advantage of these methods is the fast tracking of the system states after the observation of events. However, if there are several subsystems operating concurrently, the reachability graph of the system may have a large number of states and transitions, which requires a large amount of memory

to store it. In order to circumvent the memory space problem, an alternative would be to compute on the fly the next state estimate of the Petri net, after the observation of an event. This can be done based on the current state estimate and the system model. The main disadvantage of this method is the time necessary to compute online the state estimate of the system. In this case, the system cannot execute any observable event until the next state estimate is computed, which can generate false diagnosis status restricting the application of this approach.

In order to avoid the construction of the complete reachability graph, recent works in fault diagnosis of systems modeled by PNs have been proposed. The so-called basis markings and minimal explanations have been presented in Cabasino et al. (2010) to avoid the construction of the complete reachability graph for PNs with unobservable transitions, called silent transitions (CABASINO et al., 2010). The introduction of the concept of minimal explanations, which provide the shortest unobservable firing sequence that explains, for a given marking, the observable transition that has fired, together with basis markings, which is a part of the reachability set related with the sequence that has fired are used in Cabasino et al. (2010) to provide a method for diagnosis of LPN. In the case of bounded LPN systems, Cabasino et al. (2010) also proposes a deterministic graph called Basis Reachability Graph (BRG), to perform most of the computations offline. However, in the worst-case, according to Yue et al. (2019), the number of nodes of the BRG is equal to the number of reachable states in the reachability tree. Hence, computing the BRG has the same complexity as computing the reachability graph in a worst-case scenario.

Lefebvre and Delherm (2007) proposed an approach based on state estimation, where it is considered that some places of the PN system model are observable and diagnosis techniques are provided by solving a set of constraints. This method avoids the construction of the whole reachability graph of the system, that must be modeled by ordinary, live and safe PNs. One of the main algorithms in Lefebvre and Delherm (2007), provides a list of the minimal sets of places that must be observed to perform the state estimation for a given firing sequence. It is important noticing that this algorithm is NP complete.

An interpreted diagnoser based on the online solution of programming problems is proposed in Basile et al. (2009). The method avoids the computation of all possible reachable markings of the net after the observation of a sequence of events. However, it is assumed that the net does not have two different observable transitions labeled with the same event.

In all fault diagnosis approaches mentioned above, fault detection and isolation are carried out at the same time, *i.e.*, the fault is identified only after it is distinguished from all other fault event types. A different strategy is presented in Roth et al. (2011), Moreira and Lesage (2019) and Souza et al. (2020), where the fault diagnosis is carried out in two

steps, firstly the fault is detected and then, fault candidates are isolated. This strategy is suitable for systems where the fault must be detected as soon as possible, stopping the plant execution, in order to avoid the loss of products and ensure the security of equipment and system operators. As a consequence, when the fault is detected, only fault candidates are provided in the isolation step and not the specific type of fault that has occurred.

In this work, a fault diagnosis method for systems modeled by LPNs is proposed. As in Roth et al. (2011), Moreira and Lesage (2019) and Souza et al. (2020), the fault detection is carried out first, and then, fault candidates are isolated. In order to do so, the observed sequence of events and the Petri net model of the complete system behavior are used to compute the fault candidates. This step is performed offline, and thus, the observed reachability graph can be computed without interfering in the fault detection. In this work, the fault detection scheme is based on the construction of an LPN from the fault-free behavior system model, called Observable Behavior Labeled Petri net (OBPN), whose transitions are all labeled with observable events and whose generated language is equal to the observable fault-free language of the system. The OBPN provides all observable events of the fault-free system behavior that are feasible after the observation of a sequence of events. Thus, the detection is carried out by analyzing the marking of the OBPN after the observation of a sequence of events. If an event that does not label any enabled transition of the OBPN is observed, then the fault is diagnosed.

Differently from Lefebvre and Delherm (2007), the fault detection is performed only by observing system events. In this regard, it is assumed that the system may have unobservable events, and the fault is diagnosed when its occurrence can be detected after a bounded number of event occurrences. Differently from Basile et al. (2009), the Petri net model of the system may have more than one observable transition labeled with the same event. Differently from the works presented in Cabasino et al. (2010, 2011), our method is not based on the construction of reachability graphs to perform detection. It is important to remark in most cases that the number of places and transitions of the OBPN is smaller than or equal to the number of places and transitions of the LPN that models the fault-free system behavior. Thus, the memory required to implement the diagnoser is smaller than the memory needed to implement other PN diagnosers proposed in the literature.

Moreover, in this work, an example of a weighing-mixing system, introduced in the international standard IEC 60848, with two fault types is constructed to illustrate the detection and isolation of a fault in a diagnosable practical system. Other case study is presented to show the efficiency of the proposed method compared to other approaches of the literature. Both study cases illustrate the class of PNs covered with the method proposed in this work. In the next section, we present the general and specific objectives of this work.

1.1 OBJECTIVES

The aim of this work is to develop a fault diagnosis method for DESs modeled as PNs, using only the structure of the net that models the system, without the need to construct additional structures as reachability graphs, basis markings, minimal explanations or solving a set of constraints.

Firstly, we explore fault detection methods in DESs modeled as PNs with a set of observable transitions, then we develop a fault detection technique using only the model structure to construct the diagnoser. After the fault is detected, the observation of the generated sequence of the system is then used to isolate fault candidates.

1.1.1 Specific objectives

The specific objectives of the presented work are enumerated in the following.

1. Investigate fault diagnosis methods of DESs modeled by PNs with a set of observable transitions;
2. Construct the Observable Behavior Petri net (OBPN);
3. Propose a method to detect the fault occurrence;
4. Propose a method to isolate the fault occurrence that was prior detected;
5. Apply the developed approach to a practical system.

1.2 ORGANIZATION OF THE WORK

Chapter 2 presents a literature review about DESs, PNs and their main properties to a better understanding of the developed approach. Also, a brief explanation about the difference between the diagnosis problem and the verification of diagnosability problem is shown with an example. The problem considered in this work is formulated in Chapter 3, where the algorithms for the computation of the OBPN are proposed. In this chapter, a running example is used to illustrate the results. A case study is presented in Chapter 4, where the practical system is the weighing-mixing introduced in the international standard IEC 60848 (IEC:60848, 2002) with two fault types considered. In Chapter 5 conclusions are drawn.

2 BACKGROUND

A Discrete-Event System (DES) is a system whose state space is a discrete set and the state transition mechanism is event-driven, *i.e.*, the evolution of the system depends on events that occur, typically, asynchronously over time. Events cause the state transition of the system and can be thought as an instantaneous action that leads the system to another state. For example, the press of a button by a human operator, can happen for any reason in any point of time, causing the system to instantaneously evolve to a different state. Thus, due to the nature of DESs, differential or difference equations are not appropriate to describe their behavior. Therefore, mathematical modeling formalisms to develop appropriate models that describe the behavior of these systems and provide a framework for analysis techniques are needed.

In order to present the mathematical formalisms used to model the behavior of many systems, it is important to know how these behaviors can be represented. In this regard, when considering the evolution of a DES, we need to know the sequence of events that the system has generated and, associate it with the change in system states. To correct represent the behavior of the system and provide analysis techniques, mathematical modeling formalisms as automata and Petri nets (PNs) were developed (CASSANDRAS; LAFORTUNE, Stephane, 2009; MURATA, 1989).

The most ordinary class of DES models is formed by automata, they are a intuitive and easy to use tool but, in some cases, modeling complex systems may lead to a very large state space due to its lack of structure. On the other hand, Petri nets have more structure, they are commonly used to represent systems with synchronous, asynchronous, concurrent and parallel activities and, most of them, in a more compact way than automata. Both PNs and automata are defined and represented graphically as graphs. In this work, we only consider systems modeled with PNs and we propose a PN based diagnoser method that keep track with the observable fault free behavior of the system.

With that in mind, in the next section we formally define languages and then, in the following sections, we present the structure, the main properties and modeling formalisms of Petri nets. Lastly, in section 2.7 the diagnosability definition of systems modeled by PNs is presented.

2.1 LANGUAGES

In this work, we use the symbol Σ to represent a set of events of a give system model and the symbol σ to represent a generic event. A sequence of events is represented here as s and its length is denoted by $||s||$. We can also have an empty sequence, whose length is equal to zero, denoted as ε. The formal definition of language is presented in the following.

Definition 1 (Language). *A Language L defined over a set of events Σ is a set of finite length sequences of events from Σ.*

Therefore, a language L generated by a system represents all sequences of events that can occur in the system. For example, the language $L = \{\varepsilon, a, ab, baa, aab\}$ defined over $\Sigma = \{a, b\}$ is formed by five sequences, including the empty sequence ε. The symbol $\star$ is used in this work to represent the Kleene-closure operation, where $\Sigma^\star$ denotes the set of all possible finite length sequences formed with elements of Σ.

The language of a system that can always evolve from any generated sequence of events is called live. The formal definition of a live language is presented bellow in Definition 2 (CASSANDRAS; LAFORTUNE, Stephane, 2009).

Definition 2 (Live language). *A language L is said to be live if for all $s \in L$, there exists σ such that $s\sigma \in L$.*

In order to create and modify languages, we can apply operations such as concatenation, the Kleene-closure, and the projection. These operations are defined in the following in definitions 3, 4, 5, and 6, respectively. It is also important to remark that, as languages are sets, therefore all set operations can be applied to them (CASSANDRAS; LAFORTUNE, Stephane, 2009; MURATA, 1989).

Definition 3 (Concatenation). *Let $L_1, L_2 \subseteq \Sigma^\star$, then the concatenation L_1L_2 is defined as:*

$$L_1L_2 = \{s = s_1s_2 : (s_1 \in L_1) \ \textit{and} \ (s_2 \in L_2)\}$$

A sequence s is in L_1L_2 if it is formed by the concatenation of a sequence $s_1 \in L_1$ and $s_2 \in L_2$. For example, let $L_1 = \{\varepsilon, a, ab, baa\}$ and $L_2 = \{cd, cdd\}$ then, the concatenation of L_1 and L_2 is given by $L_1L_2 = \{cd, cdd, acd, acdd, abcd, abcdd, baacb, baacdd\}$.

Definition 4 (Kleene-closure). *Let $L \subseteq \Sigma^\star$, then*

$$L^\star = \{\varepsilon\} \cup L \cup LL \cup \ldots$$

An element of $L^\star$ is formed by the concatenation of elements of L. By definition, the empty sequence ε is also an element of $L^\star$, because it represents the concatenation of zero elements. For example, the kleene-closure of language L_1 is given by $L_1^\star = \{\varepsilon, a, ab, baa, aab, abaa, aa, abab, baabaa, \ldots\}$.

It is necessary to define prefix and suffix of a sequence s, in order to present the formal definition of prefix-closure. To do so, let $s = tuv$, where $t, u, v \in \Sigma^\star$, then t is a prefix of s, u is a subsequence of s, and v is a suffix of s. The prefix-closure of a language L is formally defined as follows (CASSANDRAS; LAFORTUNE, Stephane, 2009).

Definition 5 (Prefix closure). *Let $L \subset E^\star$, then*

$$\bar{L} := \{s \in E^\star : (\exists t \in E^\star)\,[st \in L]\}$$

i.e., the prefix closure of language L is the language denoted by $\bar{L}$ and consisting of all the prefixes of all sequences in L.

For example, the prefix-closure of $L = \{abc\}$ is given by $\bar{L} = \{\varepsilon, a, ab, abc\}$.

Another important operation that can be applied to sequences or languages is the projection operation, defined as follows.

Definition 6 (Natural projection)**.** *The projection $P_s^l : \Sigma_l^\star \to \Sigma_s^\star$, where $\Sigma_s \subset \Sigma_l$, is defined recursively as follows:*

$$P_s^l(\varepsilon) = \varepsilon$$
$$P_s^l(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_s \\ \varepsilon, & \text{if } \sigma \in \Sigma_l \backslash \Sigma_s, \end{cases}$$
$$P_s^l(s\sigma) = P_s^l(s)P_s^l(\sigma), \text{ for all } s \in \Sigma_l^\star, \sigma \in \Sigma_l,$$

where $\backslash$ denotes the set difference operation.

According to Definition 6, the natural projection operation erases all events $\sigma \in \Sigma_l \backslash \Sigma_s$ from the sequences $s \in \Sigma_l^\star$. For example, let $\Sigma_l = \{a, b, c\}$ and $\Sigma_s = \{a, c\}$, consider the language $L = \{a, b, ac, abb, acbc, bcc\}$ then, the projection P_s^l of $L \in \Sigma_l^\star$ is given by $P_s^l(L) = \{\varepsilon, a, ac, cc, acc\}$.

The language of a DES is the set that contains all the information about all possible sequences that a system is capable to generate. As one can notice, to describe a DES only by its generated language can be a difficult task. Then, structures that are capable of representing languages and can be manipulated using operations as the ones defined above, allowing the construction and analysis of system with complex behavior are necessary. In the next section, we finally define the formalism of PNs, that is used in this work to represent languages.

2.2 PETRI NETS

Petri nets (PNs) are a graphical and mathematical modeling tool firstly developed by Carl Adam Petri in the early sixties. A PN can be enunciated as a direct, weighted, bipartite graph with an initial state, called initial marking. Its graph consist of two kinds of nodes called places and transitions, where arcs can connect either places to transitions or transitions to places. A marking assigns to each place p a positive integer, then we can say that p is marked with k tokens. Graphically, places are drawn as empty circles, transitions as bars and tokens as black dots.

In a PN, events are associated with transitions and, to a transition fire, a set of conditions must be satisfied. The information regarding these conditions are held in the places, that can be input and/or output places of a transition. Input places are associated with the necessary conditions required for this transition fire and output places

can be seen as the new conditions affected by the occurrence of this transition (MURATA, 1989; CASSANDRAS; LAFORTUNE, Stephane, 2009). In the following subsection, the structure of a Petri net graph and its relations are defined.

2.2.1 Petri net graph

A Petri net graph is composed of two types of nodes, places and transitions, and arcs connecting them. This can be seen as relationships that define the basic components of a Petri net structure. As a PN is a bipartite graph, arcs do not connect nodes of the same type. The formal definition of a Petri net graph is given as follows (CASSANDRAS; LAFORTUNE, Stephane, 2009; MURATA, 1989).

Definition 7 (Petri net graph)**.** *A Petri net graph is a weighted bipartite graph*

$$(P, T, Pre, Post)\,,$$

where P is the set of places, T is the set of transitions, $Pre : (P \times T) \to \mathbb{N} = \{0, 1, 2, \ldots\}$ is the function of ordinary arcs that connect places to transitions, and $Post : (T \times P) \to \mathbb{N}$ is the function of ordinary arcs that connect transitions to places.

The set of finite places is denoted by $P = \{p_1, p_2, \ldots, p_n\}$ and the finite set of transitions is denoted by $T = \{t_1, t_2, \ldots, t_m\}$. Therefore, $|P| = n$, $|T| = m$, where $|.|$ denotes set cardinality. The set of input places (resp. input transitions) of a transition $t_j \in T$ (resp. place $p_i \in P$) is denoted as $I_p(t_j)$ (resp. $I_t(p_i)$), and is formed by the places $p_i \in P$ (resp. transitions $t_j \in T$) such that $Pre\left(p_i, t_j\right) > 0$ (resp. $Post(t_j, p_i) > 0$). Analogously, the set of output places (resp. output transitions) of a transition $t_j \in T$ (resp. place $p_i \in P$) is denoted as $O_p\left(t_j\right)$ (resp. $O_t\left(p_i\right)$), and is formed of the places $p_i \in P$ (resp. transitions $t_j \in T$) such that $Post\left(t_j, p_i\right) > 0$ (resp. $Pre\left(p_i, t_j\right) > 0$).

The functions Pre and $Post$ determine the number of arcs, or weight of the arc, that connect places to transitions and transitions to places. The values of these functions are shown in the graph only if they are greater than 1. In the sequel, an example of the graph of a Petri net is presented, and then, subsection 2.2.2 presents how conditions are described in a PN graph. The PN examples of this section were based on Cabral (2017) thesis.

Example 1. *Let the graph of a Petri net, presented in Figure 1, be defined as $P = \{p_1, p_2\}$, $T = \{t_1\}$, $Pre(p_1, t_1) = 1$, $Post(t_1, p_2) = 2$. In this example, $I_p(t_1) = \{p_1\}$ and $I_t(p_2) = \{t_1\}$, $O_t(p_1) = \{t_1\}$ and $O_p(t_1) = \{p_2\}$.*

2.2.2 Petri net marking

As mentioned above, in a Petri net, transitions are associated with events driving a DES and places describe conditions under which these transitions can occur. So, in this

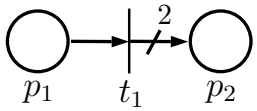


Figure 1 – Petri net graph of example 1.

circumstances, token is the name given to the element that indicates if these conditions are met, therefore, when tokens are assigned to places, we have a Petri net called marked Petri net. Formally, the number of tokens assigned to a place p_i is given by $x(p_i)$, where $x : P \to \mathbb{N}$ is a marking function. The marking of a PN is represented by the column vector $\underline{x} = \begin{bmatrix} x(p_1) & x(p_2) & \dots & x(p_n) \end{bmatrix}^T$ formed of the number of tokens in each place p_i, for $i = 1, \dots, n$.

Graphically a token is drawn as a dark dot or a number in the assigned place. In the following we present the formal definition of a marked Petri net.

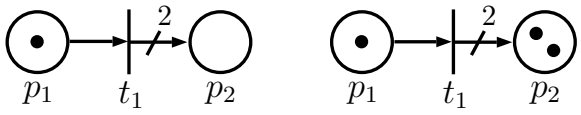
Definition 8 (Petri net)**.** *A marked Petri net $\mathcal{N}$ is a five-tuple*

$$\mathcal{N} = (P, T, Pre, Post, x_0)\,,$$

where $(P, T, Pre, Post)$ is a Petri net graph and $x_0 : P \to \mathbb{N}$ is the initial marking function.

In a marked Petri net, or simply Petri net, the marking vector $\underline{x}$ represents the system state and for each new reachable state, the PN reaches a new marking. In the sequel, we present an example of a marked PN. The initial marking of a PN is denoted as x_0.

Example 2. *Let $\mathcal{N}$ be the Petri net graph defined in Example 1. Figure 2 shows two possible initial markings, $x_0^1 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$ and $\underline{x}_0^2 = \begin{bmatrix} 1 & 2 \end{bmatrix}^T$, for Petri net $\mathcal{N}$.*



(a) Initial marking $\underline{x}_0^1$ (b) Initial marking $\underline{x}_0^2$

Figure 2 – Petri Net $\mathcal{N}$ of example 2 with two different initial markings.

In a Petri net, a transition $t_j \in T$ is said to be enabled when the number of tokens assigned to each input place of t_j is greater or equal to the weight of the arcs that connect the places of $I_p(t_j)$ to transition t_j. The mathematical definition of an enabled transition is presented in the sequel.

Definition 9 (Enabled transition)**.** *A transition $t_j \in T$ is said to be enabled if*

$$x\left(p_i\right) \geq Pre\left(p_i, t_j\right), \forall p_i \in I_p\left(t_j\right).$$

In the following subsection, the dynamics of a Petri net graph is presented.

2.2.3 Petri net dynamics

In a Petri net, when a transition is enabled it can fire. The state transition function of a PN is defined through the change in the marking of the places due to the firing of an enabled transition. If a transition t_j is enabled, for a given marking $\underline{x}$, and t_j fires, a new marking $\underline{x}'$ is reached according to the following state transition equation

$$x'(p_i) = x(p_i) - Pre(p_i, t_j) + Post(t_j, p_i), \text{ for } i = 1, \ldots, n. \tag{1}$$

According to Equation (1), if p_i is an input place of t_j and t_j fires, the number of tokens that are equal to the weight of the arc that connects p_i to t_j, $Pre(p_i, t_j)$, are removed. Similarly, if p_i is an output place of t_j, it receives the amount of tokens as the weight of the arc that connects t_j to p_i, $Post(t_j, p_i)$. Example 3 shows the evolution of a Petri net due to the firing of a transition.

Example 3. *Consider the PN depicted in Figure 3 (a). Notice that transition t_1 is enabled for the initial marking $\underline{x}_0 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$ and it can fire. When t_1 fires, one token is removed from p_1, since the weight of the input arc of t_1 is 1, and two tokens are added in p_2, since the weight of the arc that connects t_1 to p_2 is 2. The resulting marking due to the fire of t_1 is $\underline{x} = \begin{bmatrix} 0 & 2 \end{bmatrix}^T$, as shown in Figure 3 (b).*

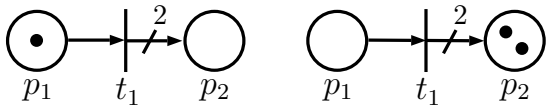


Figure 3 – Petri Net $\mathcal{N}$ of Example 3 with transition t_1 enabled (a) and the new marking reached after the firing of t_1 (b).

In this work, we represent the transition function of the Petri net as $f : \mathbb{N}^n \times T \to \mathbb{N}^n$. Thus, the firing of an enabled transition t_j in state $\underline{x}$ leads to the new marking $\underline{x}' = f(\underline{x}, t_j)$. The domain of f can be extended to $\mathbb{N}^n \times T^\star$, where $\star$ denotes the Kleene-closure operator, to consider a sequence of transition firings as $f(\underline{x}, \theta) = \underline{x}$, where θ denotes the empty sequence of transitions, and $f(\underline{x}, st_j) = f(f(\underline{x}, s), t_j)$, for all $s \in T^\star$ and $t_j \in T$.

In the sequel, we present the construction of a reachability graph, where its structure is commonly used for analysis techniques, such as the fault diagnosis proposed in this work.

2.2.4 Reachability graph of a Petri net

The reachability graph of a Petri net, represents all possible states reached by the initial marking of the PN graph. At each new state reached by the fire of a transition, a new node is added to the graph. If the fire of a transition leads to a state that already exists on the reachability graph, then, its transition is drawn back to the respective node. In the following we illustrate this method with an example.

Example 4. *Consider the Petri net graph depicted in Figure 4 (a), where the initial marking is given by* $\underline{x}_0 = \begin{bmatrix} 2 & 0 & 0 \end{bmatrix}^T$. *So, the initial state of the reachability graph depicted in Figure 4 (b), is given by the initial state* $[2, 0, 0]$. *Now, we need to examine all transitions that can fire from the initial state and define these states as new nodes in the graph. In this example, the only transition that is enable to fire due to the initial marking of the PN is* t_0, *which leads to the new state* $[1, 1, 0]$, *in Figure 4 (b) represented by state* 1.

From state 1, *transitions* t_0, t_1 *and* t_2 *are enabled. If* t_1 *fires we reach state* 0, *that is the initial state of the graph. If* t_0 *fires, we reach state* 2, *given by* $[0, 2, 0]$. *If* t_2 *fires, we reach state* 3, *given by* $[1, 0, 1]$.

Suppose we are in state 2, *transitions* t_1 *and* t_2 *are enabled, if* t_1 *fires, we reach state* 1 *again, if transition* t_2 *fires, state* 4, *given by* $[0, 1, 1]$, *is reached. In state* 4, *it is possible to go back to state* 2 *with the firing of transition* t_3, *and then, state* 3 *can be reached with the firing of transition* t_1, *or state* 5 *can be reached with the firing of transition* t_2. *We follow discovering all the states and transitions that are possible from the initial marking until there are no more new nodes or new connections between nodes. Therefore, the reachability graph resulting from the Petri net presented in Figure 4 (a) is the reachability graph presented in Figure 4 (b).*

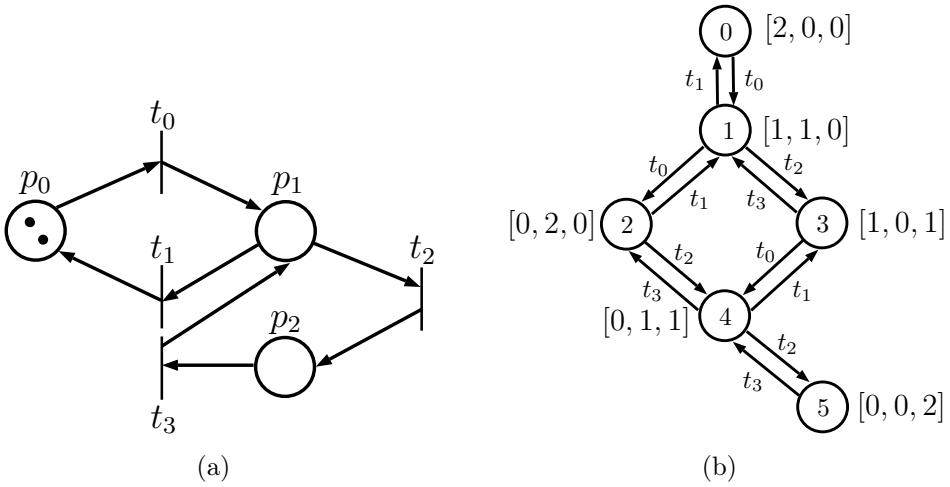


Figure 4 – Petri net graph, 4 (a) , of Example 4 and its reachability graph 4 (b).

In the sequence we present some important Petri net properties.

2.3 PETRI NET PROPERTIES

In this section, important concepts are briefly explained, so that the reader can understand some properties used to developed the approach presented in Chapter 3.

2.3.1 Deadlock free

A PN is said to be deadlock free if no reachable marking is a deadlock, *i.e.*, all reachable markings enable at least one transition of the net (DAVID; ALLA, 2005).

2.3.2 T'-induced subnet

Definition 10 presents the T'-induced subnet defined by Cabasino et al. (2012).

Definition 10 (T' - induced subnet)**.** *Let T' be a subset of T. The T'-induced subnet of a Petri net $\mathcal{N} = (P, T, Pre, Post, x_0)$ is the subnet $\mathcal{N}' = (P, T', Pre', Post', x_0)$, where $Pre' : P \times T' \to \mathbb{N}$ and $Post' : T' \times P \to \mathbb{N}$ with $Pre'\left(p_i, t_j\right) = Pre\left(p_i, t_j\right)$ and $Post'\left(t_j, p_i\right) = Post\left(t_j, p_i\right)$, for all $t_j \in T'$ and $p_i \in P$.*

Therefore, a T'-induced net contains all places of the original net $\mathcal{N}$, but only the transitions in the subset T' and their related arcs.

In the next section we define the formalism of Labeled Petri nets to establish a correspondence between events and transitions of a Petri net graph.

2.4 LABELED PETRI NET

In order to use Petri nets to model Discrete-Event systems and represent languages, we associate events to each transition of the Petri net. This is carried out by a labeling function, leading to the so-called Labeled Petri nets (LPNs) (CASSANDRAS; LAFORTUNE, Stephane, 2009), defined as follows.

Definition 11 (Labeled Petri net)**.** *A labeled Petri net is a seven-tuple*

$$\mathcal{N}_l = (P, T, Pre, Post, x_0, \Sigma, \ell)\,,$$

where $(P, T, Pre, Post, x_0)$ is a Petri Net, Σ is the set of events, and $\ell : T \to \Sigma$ is the labeling function that assigns events of Σ to transitions in T.

An enabled transition t_j in a Labeled Petri net, only fires when the event associate it occurs. Example 5 presents this dynamic.

Example 5. *Consider the LPN presented in Figure 5, where $P = \{p_1, p_2\}$, $T = \{t_1, t_2, t_3, t_4\}$, $Pre(p_1, t_2) = Pre(p_2, t_3) = Pre(p_2, t_4) = 1$, $Post(t_1, p_1) = Post(t_2, p_2) = 1$, $\underline{x}_0 = \begin{bmatrix} 1 & 0 \end{bmatrix}^T$, $\Sigma = \{a, b, c, d\}$, $\ell(t_1) = \{a\}$, $\ell(t_2) = \{b\}$, $\ell(t_3) = \{c\}$ and $\ell(t_4) = \{d\}$. In this*

example, transitions t_1 and t_2 are enabled and can fire as soon as events a or b occurs, respectively. Suppose we change the initial marking of the net to $\underline{x}_0 = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$, then the transitions t_1, t_3 and t_4 would be enabled and could fire when events a or c, or d occurs, respectively.

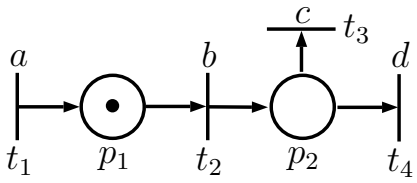


Figure 5 – Labeled Petri Net $\mathcal{N}_l$ of example 5.

The domain of the labeling function ℓ can be extended to consider sequences of transition firings, *i.e.*, $\ell : T^\star \to \Sigma^\star$, as $\ell(\theta) = \varepsilon$, where ε denotes the empty sequence of events, and $\ell(st) = \ell(s)\ell(t)$, for all $s \in T^\star$ and $t \in T$. The labeling function can also be applied to a set of sequences of transitions $B \subseteq T^\star$ as $\ell(B) = \cup_{s \in B} \ell(s)$.

In the sequel we present sequences and languages generated by Labeled Petri nets.

2.5 PETRI NET LANGUAGES

The language $L(\mathcal{N})$ generated by a Labeled Petri net $\mathcal{N}$, or simply L, represents all sequences of transitions labels that are obtained by all possible sequences of transitions firings. The language L of a Petri net $\mathcal{N}$ is defined over a set of events Σ, where Σ is a set of finite length sequences of transitions labeled with the function ℓ. Then, before explaining the subject matter of the presented work in section 2.7, we first present the concepts of fault and fault free sequences in order to define faulty and fault free languages.

Definition 12 (Fault and fault free sequences)**.** *A fault sequence is a sequence of events s such that σ_f is one of the events that form s. On the other hand, a fault free sequence is one that does not contain the fault event σ_f.*

Definition 13 (Fault and fault free languages)**.** *The fault free language $L_N \subset L$ is the set of all fault free sequences of L and the set of all sequences generated by the PN model of the system that contain σ_f is $L_F = L \setminus L_N$.*

The next section present some Petri net classes used in this work.

2.6 PETRI NET CLASSES

In this section, classes of Petri nets considered in the development of this work are formally presented.

2.6.1 Safe Petri net

A place $p_i \in P$ is said to be safe, if $x(p_i) \leq 1$ for all reachable markings of the net. A Petri net is said to be safe for a given initial marking $\underline{x}_0$, if all places are safe for all reachable markings of the net (DAVID; ALLA, 2005).

2.6.2 Ordinary Petri net

An ordinary Petri net is a PN such that $Pre(p_i, t_j) \leq 1$ and $Post(t_j, p_i) \leq 1$, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$(MURATA, 1989).

2.6.3 State machine Petri net

A particular class of ordinary Petri nets is the so-called State Machine Petri Net (SMPN). An SMPN is an ordinary PN in which each transition has exactly one input place and one output place. SMPNs allow the representation of conflicts, but not the synchronization of concurrent activities (MURATA, 1989). In the following, example 6 shows an SMPN graph.

Example 6. *Figure 6 graphically represent an state machine Petri net, where each transition has only one input and one output place.*

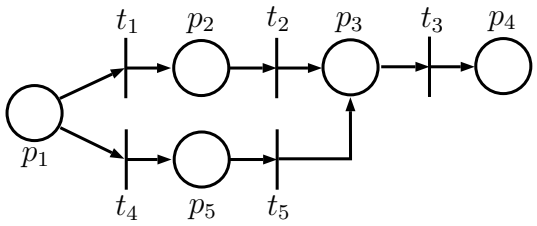


Figure 6 – Petri Nets of class SMPN of Example 6 .

In the following section, we present the diagnosability of DESs modeled as LPN.

2.7 DIAGNOSABILITY OF DISCRETE EVENT SYSTEMS MODELED BY PETRI NETS

The diagnosis subject can be divided into two different problems: the diagnosis and the verification of diagnosability. A diagnosis problem is the one concerned with determining which faults, if any, explain certain sequences of observable events with respect to the system model. Whereas, a diagnosability problem is to determine if, once a fault has occurred, a diagnoser can detect its occurrence in a finite number of events occurrences. Therefore, the system needs to be diagnosable, so a fault diagnosis method can be applied. The works of Bakalara et al. (2020), Cabral (2017), Cabral et al. (2015), Cabasino et al. (2013, 2011, 2010), Dotoli et al. (2009), Ru and Hadjicostis (2009), Basile

et al. (2009) and Lefebvre and Delherm (2007), and Genc and Stéphane Lafortune (2003) propose approaches to diagnosis, while Cabasino et al. (2014, 2009, 2012) and Moreira et al. (2011), and Sampath et al. (1995) are concerned with verifying the diagnosability. In these frameworks, faults are usually modeled as unobservable events, otherwise its diagnosis would be straightforward.

The first relevant approach of diagnosis and diagnosability of DESs were developed within the automata framework in the late nineties. In the pioneer work of Sampath et al. (1995, 1996) the basic and fundamental concepts are defined, and then, an approach based on the system model, where the fault diagnosis could be done following only the current diagnoser state is presented. The mathematical definition of diagnosability proposed by Sampath et al. (1995) is shown in Definition 14.

Definition 14 (Language diagnosability)**.** *Let L and $L_N \subset L$ be the prefix-closed and live languages generated by the system and the fault free model of the system, respectively. Let $L_F = L \backslash L_N$. Then, L is said to be diagnosable with respect to the projection $P_o : \Sigma^\star \to \Sigma_o^\star$ and Σ_f if the following holds*

$$(\exists z \in \mathbb{N})\,(\forall s \in L_F)\,(\forall st \in L_F)\,(\|t\| \geq z \Rightarrow (P_o(st) \notin P_o\,(L_N)))$$

where $\|.\|$ denotes the length of a sequence.

According to Definition 14, L is diagnosable with respect to the projection P_o, and the set of fault events Σ_f, if, and only if, for all fault sequences st with arbitrarily long length after the occurrence of a fault event, there does not exist a fault free sequence $s_N \in L_N$, such that $P_o(st) = P_o(s_N)$. Thus, if L is diagnosable, then it is always possible to identify the occurrence of a fault event after a bounded number of observations of events. It is important to remark that in order to this definition be stated, it is assumed that the language generated by the system is live.

At a later time, the diagnosis problem were addressed to PNs due to the well known fact that PN models of systems that have behaviors such as concurrency and synchronization usually are more compact than their correspondent automata representations. For this reason, PNs started to be considered suitable due to its analytical capabilities and its distributed nature, that allows the reduction of the computational complexity when solving diagnosability problems by avoiding the construction of the entire reachability graph (FANTI; SEATZU, 2008; ZAYTOON; LAFORTUNE, Stéphane, 2013; GIUA; SILVA, 2018). Before we present the notion of diagnosability of LPN modeled systems, we first define some important notation.

Let $\mathcal{N}_l$ be an LPN that models the system. The set of all finite-length sequences of transitions that can fire from the initial marking $\underline{x}_0$ is given by $S = \{s \in T^\star : f(\underline{x}_0, s)!\}$, where ! denotes *is defined*. The language generated by $\mathcal{N}_l$, L, is obtained using the labeling function ℓ and is defined as $L = \ell(S)$.

The set of events Σ can be partitioned as $\Sigma = \Sigma_u \dot{\cup} \Sigma_o$, where Σ_u and Σ_o denote, respectively, the sets of unobservable events and observable events. The set of fault events is denoted by $\Sigma_f \subseteq \Sigma_u$. In this work, for simplicity, we consider that the system has a unique fault event, *i.e.*, $\Sigma_f = \{\sigma_f\}$. The sets of transitions labeled by events of Σ_o, Σ_u and Σ_f are denoted by T_o, T_u and T_f, respectively.

Definition 15 (Mask function)**.** *The observation mask function $M : T^\star \to \Sigma_o^\star$, is recursively defined as $M(t_j) = \sigma$, if $t_j \in T_o$, where $\sigma = \ell(t_j)$, $M(t_j) = \varepsilon$, if $t_j \in T_u$, $M(st_j) = M(s)M(t_j)$, for all $s \in T^\star$ and $t_j \in T$, and $M(\theta) = \varepsilon$.*

The inverse mask function $M^{-1} : \Sigma_o^\star \to 2^S$ is defined as $M^{-1}(\rho) = \{s \in S : M(s) = \rho\}$.

Let S_F be the set formed of all sequences of transitions of S that have at least one fault transition $t_f \in T_f$. The following definition of diagnosability of an LPN can be stated.

Definition 16. *A deadlock free LPN $\mathcal{N}_l$ is diagnosable with respect to $M : T^\star \to \Sigma_o^\star$ and T_f if*

$$(\exists z \in \mathbb{N})(\forall \nu \in S_F)(\forall \nu\mu \in S_F)(\|\mu\| \geq z \Rightarrow \\ \forall \rho \in M^{-1}(M(\nu\mu)), \rho \in S_F). \tag{2}$$

□

The set of all sequences of transition firings that does not contain any transition from T_f is denoted by S_N. Thus, $S_N = S \setminus S_F$. The LPN that models S_N is denoted in this work as $\mathcal{N}_N = (P_N, T_N, Pre_N, Post_N, x_{0,N}, \Sigma_N, \ell_N)$, where $\Sigma_N = \Sigma \setminus \Sigma_f$. Note that T_N can be partitioned as $T_N = T_{N_o} \dot{\cup} T_{N_u}$, where $T_{N_o} \subset T_o$ and $T_{N_u} \subset T_u$ denote the sets of observable and unobservable transitions of $\mathcal{N}_N$, respectively. In the following, an example is presented in order to illustrate the notion of language diagnosability for labeled Petri nets.

Example 7. *The Labeled Petri net system depicted in Figure 7 is formed by five places $P = \{p_1, p_2, p_3, p_4, p_5\}$ and six transitions $T = \{t_1, t_2, t_3, t_4, t_5\}$, labeled by the observable events $\Sigma_o = \{a, b\}$ and unobservable events $\Sigma_u = \{c, \sigma_f\}$, where σ_f is the fault event. The observable behavior of the fault free model of the system is given by the set of event sequences $\{a, ab, aba, abab, ...\}$. At each occurrence of the event a, it is expected that the next observation is event b, and thus, we form a pattern where after every event a, b is also observed. For example, sequence abab is an expected behavior of the system. After the occurrence of the fault σ_f, the behavior of the system is given by the set $a^\star$ and in this case, event b is no longer observed. Thus, the system is said to be diagnosable due to the fact that it is possible to identify the fault within a finite number of event occurrences, in*

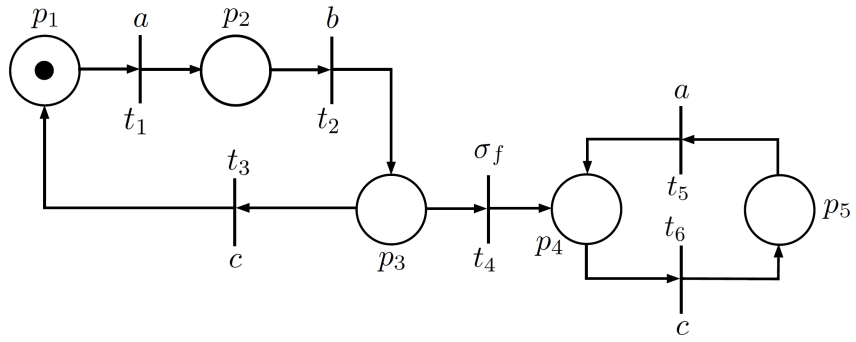


Figure 7 – Labeled Petri net for example 7.

this case, as soon as two events in sequence are observed. Thereby, to perform the fault diagnosis in this system, its just necessary to observe two occurrences of event a.

On the other hand, suppose that the observable event set changes to $\Sigma_o = \{a, c\}$*. The observable behavior of the fault free model of the system is given by the set of event sequences* $\{a, ac, aca, acac, ...\}$*. At each occurrence of event a, we expect to see event c. But after the fault occurrence,* σ_f*, the set of observable sequence of events is given by* $\{a, ac, aca, acac, ...\}$*, that is the same sequence expected in the fault free behavior of the system. Therefore, fault diagnosis methods cannot be applied in this system conditions, because the system is not diagnosable due to the fact that is not possible to distinguish between the fault free and the fault sequence behaviors.*

In the following chapter, we present the algorithms to perform fault diagnosis in systems modeled with Labeled Petri nets.

3 FAULT DIAGNOSIS OF DISCRETE EVENT SYSTEMS BY A CLASS OF PETRI NETS

In this chapter, we describe the approached problem and then present the algorithms and conditions to compute the observable behavior Petri net of the fault free system model.

3.1 PROBLEM FORMULATION

The fault diagnosis scheme in two steps proposed in this work is depicted in Figure 8. First, a comparison between the set of observable feasible events of the model at the current state $\underline{x}_c$, Γ $(\underline{x}_c)$, with the observed event σ_o generated by the plant, is carried out. If $\sigma_o \in \Gamma(\underline{x}_c)$, then the fault is not detected and a signal is emitted to the model player to update the current state $\underline{x}_c$ of the model and compute the new set of observable feasible events $\Gamma(\underline{x}_c)$. The fault isolation module also receives the observed event σ_o, and computes the next possible states considering the complete Petri net system model, including the post-fault behavior, and the current state estimate of the system. On the other hand, if $\sigma_o \notin \Gamma(\underline{x}_c)$, then the fault is detected and a signal is emitted to the fault isolation module to compute the fault candidates. The model player initiates providing the observable feasible events at the initial state x_0 of the model.

It is important to remark that, if the fault isolation module receives an observed event while it is updating its state estimate, then a first-in first-out queue is formed with the observed events. Thus, after the fault detection, it may be necessary to wait the fault isolation module to process offline all observed events before providing the fault candidates. This process can be performed using traditional techniques found in the literature. The advantage of this method is that it does not interfere with the fault detection and the system can be halted as soon as the fault is detected.

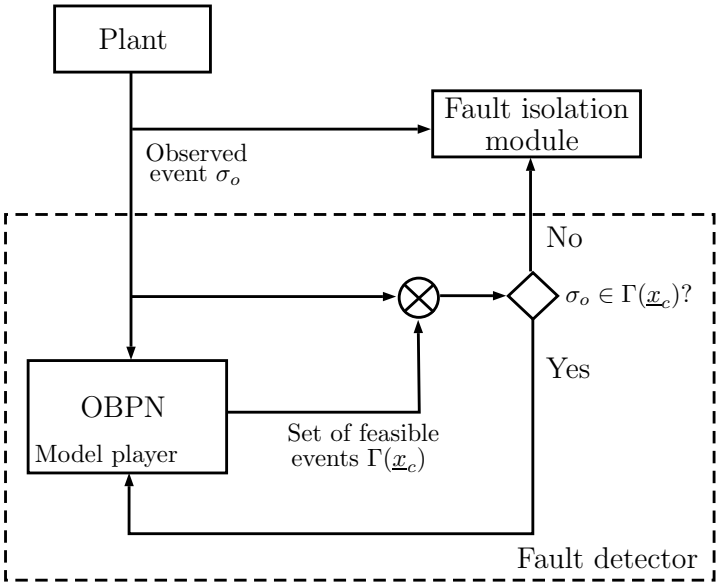


Figure 8 – Fault diagnosis scheme for LPN.

In order to perform an online fast computation of the set of observable feasible events of $\mathcal{N}_N$, it is necessary to compute from $\mathcal{N}_N$ an LPN that does not have unobservable transitions. By doing so, one can easily track the state of the new LPN in order to compute the observable feasible events, without the need for computing any unobservable reach. Let us call the Petri net obtained from $\mathcal{N}_N$ such that all transitions are observable, the observable behavior labeled Petri net (OBPN), denoted as $\mathcal{N}_o = (P_o, T_{N_o}, Pre_o, Post_o, x_{0,o}, \Sigma_N, \ell_o)$. In order to use $\mathcal{N}_o$ for fault detection, without increasing the detection delay, it is necessary to guarantee that its generated language is equal to the observable language of the fault-free system model. Thus, the objective of this work is to find a labeled Petri net $\mathcal{N}_o$ from $\mathcal{N}_N$ such that $M(S_N) = \ell_o(S_o)$, where $S_o = \{s \in T^\star_{N_o} : f_o(\underline{x}_{0,o}, s)!\}$ and $f_o : \mathbb{N}^n \times T^\star_{N_o} \to \mathbb{N}^n$ is the transition function of $\mathcal{N}_o$.

In order to characterize the class of labeled Petri nets that can be used in the fault diagnosis scheme proposed in this work, we need to consider the following assumptions regarding the system model:

A1. The LPN system model N_l is safe.

A2. There is no sequence of transition firings $s \in S$ such that $s = uv$, where $u \in T^\star$, $v \in T^\star_u$, and v has arbitrarily long length.

A3. Two or more transitions of $\mathcal{N}_N$ labeled with the same observable event cannot be simultaneously enabled for any marking estimate.

It is important to remark that several practical systems can be modeled by safe Petri nets (MURATA, 1989; LEFEBVRE; DELHERM, 2007), and thus Assumption **A1** holds true in several cases, as in the example presented in Chapter 4. Assumption **A2** requires that the system does not have cycles of unobservable events, which is a usual assumption considered in the fault diagnosis of DES (SAMPATH et al., 1995). Assumption **A3** implies that the observation of an event is related with the firing of only one observable transition of $\mathcal{N}_N$ that is feasible for all reachable markings of the current marking estimate. Thus, after the observation of an event generated by the system, one can uniquely identify which transition has fired. It is important to remark that Petri nets that do not have two different observable transitions labeled with the same event, as the PN considered in (BASILE et al., 2009), satisfy Assumption **A3**.

In the following, the steps to compute the Petri net $\mathcal{N}_o$ are presented.

3.2 COMPUTATION OF $\mathcal{N}_o$

The method proposed in this work, consists of modifying the fault free labeled Petri net model, $\mathcal{N}_N$, in order to compute the OBPN $\mathcal{N}_o$ by removing all unobservable transitions $t_u \in T_{N_u}$ of $\mathcal{N}_N$. The method is based on the merging of the input and output places of each transition t_u of $\mathcal{N}_N$ by following the steps of Algorithm 1.

Algorithm 1 Eliminating an unobservable transition t_u of $\mathcal{N}_N$.

Input: $\mathcal{N}_N = (P_N, T_N, Pre_N, Post_N, x_{0,N}, \Sigma_N, \ell_N)$ and $t_u \in T_{N_u}$

Output: $\mathcal{N}'_N = \left(P'_N, T'_N, Pre'_N, Post'_N, x'_{0,N}, \Sigma_N, \ell'_N\right)$

1: $T'_N = T_N \setminus \{t_u\}$
2: Define $P_u = \{p_{x,y} : p_x \in I_p(t_u) \wedge p_y \in O_p(t_u)\}$
3: $P'_N = (P_N \setminus (I_p(t_u) \cup O_p(t_u))) \cup P_u$
4: $Pre'_N(p_i, t_j) = Pre_N(p_i, t_j)$ for all $p_i \in P'_N \setminus P_u$ and $t_j \in T'_N$
5: $Post'_N(t_j, p_i) = Post_N(t_j, p_i)$, for all $t_j \in T'_N$ and $p_i \in P'_N \setminus P_u$
6: **for each** $p_{x,y} \in P_u$ **do**
7: $\quad Pre'_N(p_{x,y}, t_{out}) = 1$, $\forall t_{out} \in O_t(p_x) \setminus \{t_u\}$
8: $\quad Pre'_N(p_{x,y}, t_{out}) = 1$, $\forall t_{out} \in O_t(p_y)$
9: $\quad Post'_N(t_{in}, p_{x,y}) = 1$, $\forall t_{in} \in I_t(p_x)$
10: $\quad Post'_N(t_{in}, p_{x,y}) = 1$, $\forall t_{in} \in I_t(p_y) \setminus \{t_u\}$
11: $x'_{0,N}(p_{x,y}) = x_{0,N}(p_x)$, $\forall p_{x,y} \in P_u$
12: $x'_{0,N}(p_i) = x_{0,N}(p_i)$, $\forall p_i \in P'_N \setminus P_u$
13: $\ell'_N(t_j) = \ell_N(t_j)$, $\forall t_j \in T'_N$

In Algorithm 1, the modified Petri net $\mathcal{N}'_N$ is computed from $\mathcal{N}_N$, by merging the input places of t_u with its output places, and eliminating t_u from $\mathcal{N}_N$. Thus, in Line 1, t_u is removed from T_N, obtaining the new set of transitions T'_N. Then, in Line 2, set P_u is formed with the merging of all input places p_x and output places p_y of t_u. In Line 3, the new set of places P'_N is obtained. From Lines 4 to 10, the preconditions and the postconditions of each transition of T'_N are computed. Note that, except from the arcs connected to t_u, all other arcs of $\mathcal{N}_N$ are maintained, and that the input transitions of each $p_x \in I_t(p_x)$ or $p_y \in I_t(p_y) \setminus \{t_u\}$ are input transitions of $p_{x,y}$, and that the output transitions of each $p_x \in O_t(p_x) \setminus \{t_u\}$ or $p_y \in O_t(p_y)$ are output transitions of $p_{x,y}$. Finally, in Lines 11-12 we define the initial marking of the new net $\mathcal{N}'_N$, and in Line 13, we define the labeling function ℓ'_N.

Example 8. *Consider the Petri net $\mathcal{N}_N$ depicted in Figure 9(a). The set of observable and unobservable transitions of $\mathcal{N}_N$ are $T_{N_o} = \{t_2, t_4, t_5, t_6, t_9\}$ and $T_{N_u} = \{t_1, t_3, t_7, t_8\}$, respectively. Let us eliminate transition t_7 of Petri net $\mathcal{N}_N$ according to Algorithm 1 in order to compute Petri net $\mathcal{N}'_N$. The first step is to compute set $T'_N = T_N \setminus \{t_7\} = \{t_1, t_2, t_3, t_4, t_5, t_6, t_8, t_9\}$. Since places p_5 and p_8 are input places of t_7 and p_9 is an output place of t_7, we can compute set $P_u = \{p_{5,9}, p_{8,9}\}$ according to Line 2 of Algorithm 1. The set of places of $\mathcal{N}'_N$ is computed in Line 3 and it is equal to $P'_N = \{p_1, p_2, p_3, p_4, p_{5,9}, p_6, p_7, p_{8,9}, p_{10}\}$. Then, in Lines 4-10, the Pre'_N and $Post'_N$ are calculated, and since transition t_7 does not belong to $\mathcal{N}'_N$ and places $p_{5,9}$ and $p_{8,9}$ were created, we define the arcs related to the places $p_{5,9}$ and $p_{8,9}$ as $Pre'_N(p_{8,9}, t_8) = Pre'_N(p_{5,9}, t_8) = Post'_N(t_6, p_{8,9}) = Post'_N(t_4, p_{5,9}) = 1$. Finally, in Lines 11, 12, and 13, the initial marking is defined and the labeling function ℓ'_N is established from the labeling function of $\mathcal{N}_N$,*

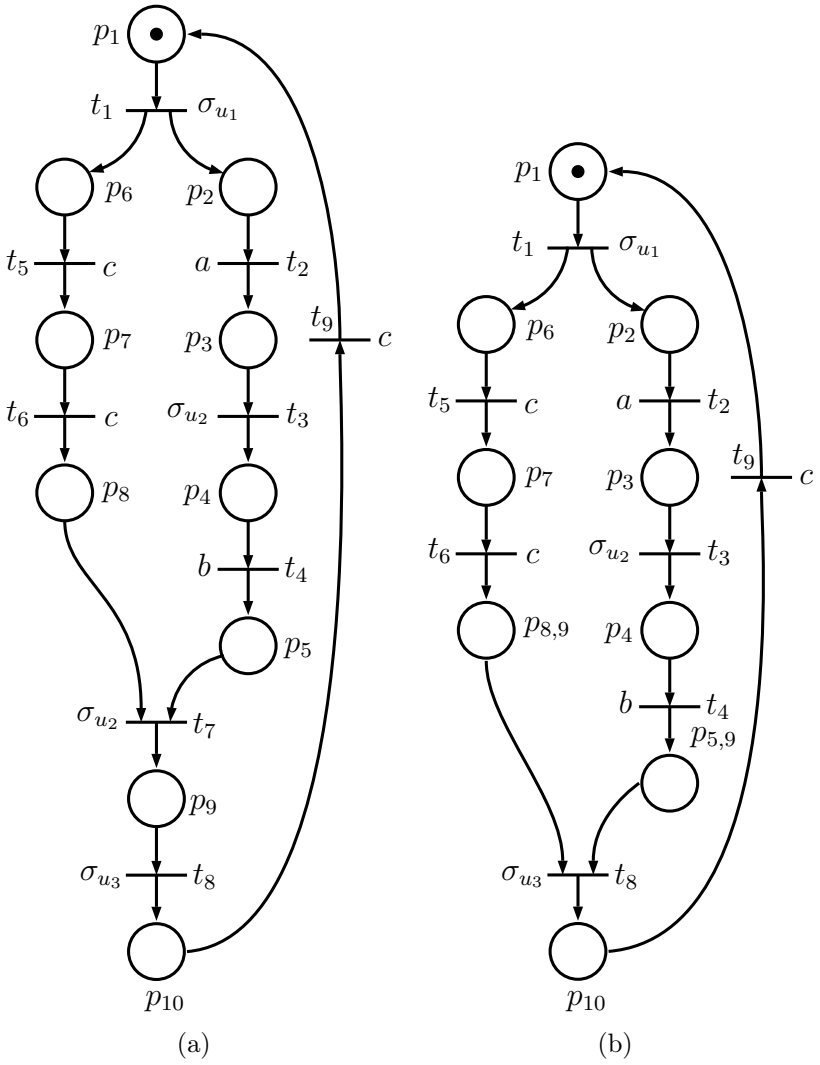


Figure 9 – Petri net $\mathcal{N}_N$ of Example 8 (a); and the reduced Petri net $\mathcal{N}'_N$ after the elimination of the unobservable transition t_7 according to Algorithm 1 (b).

ℓ_N. *The resulting Petri net* $\mathcal{N}'_N$ *is presented in Figure 9(b).* □

In order to obtain conditions to guarantee that the generated language of $\mathcal{N}_o$, computed by applying recursively Algorithm 1 to remove all unobservable transitions of $\mathcal{N}_N$, be equal to the observable language of $\mathcal{N}_N$, *i.e.*, $\ell_o(S_o) = M(S_N)$, it is first necessary to find all weakly connected components (CORMEN et al., 2009) of $\mathcal{N}_N$, $\mathcal{N}_k = \big(P_k, T_k, Pre_k, Post_k, x_{0,k}, \Sigma_N, \ell_k\big)$, such that $T_k \neq \emptyset$ and $T_k \subseteq T_{N_u}$, for $k = 1, \ldots, \rho$, where ρ denotes the number of weakly connected components of $\mathcal{N}_N$ of this kind. The procedure to obtain $\mathcal{N}_k$ is presented in Algorithm 2.

Algorithm 2 Computation of all $\mathcal{N}_k$, $k = 1, \ldots, \rho$, of $\mathcal{N}_N$

Input: $\mathcal{N}_N = \big(P_N, T_N, Pre_N, Post_N, x_{0,N}, \Sigma_N, \ell_N\big)$
Output: $\mathcal{N}_k = \big(P_k, T_k, Pre_k, Post_k, x_{0,k}, \Sigma_N, \ell_k\big)$, $k = 1, \ldots, \rho$
1: Compute the T_{N_u}-induced subnet of $\mathcal{N}_N$, denoted as $\mathcal{N}_u$
2: Find all weakly connected components $\mathcal{N}_k$ of $\mathcal{N}_u$ such that $T_k \neq \emptyset$

In Line 1 of Algorithm 2, the T_{N_u}-induced subnet of $\mathcal{N}_N$, $\mathcal{N}_u$, is computed. Since T_{N_u} is formed only of unobservable transitions, $\mathcal{N}_u$ may be formed of several disconnected graphs. Thus, in Line 2 of Algorithm 2, maximal weakly connected subgraphs of $\mathcal{N}_u$ that has at least one transition are obtained, forming the weakly connected components $\mathcal{N}_k$.

In the sequel, we present conditions to guarantee that $\ell_o(S_o) = M(S_N)$.

Theorem 1. *Let $\mathcal{N}_o$ be the LPN obtained by applying recursively Algorithm 1 to $\mathcal{N}_N$, eliminating all the unobservable transitions in T_{N_u}. Then, $\ell_o(S_o) = M(S_N)$, if all the following conditions are satisfied for each $\mathcal{N}_k$, $k = 1, \ldots, \rho$:*

*C*1. *(i) $\mathcal{N}_k$ is an SMPN with a unique source place; or (ii) all places of $\mathcal{N}_k$ have at most one input transition and one output transition, and all the output places (resp. input places) of the unobservable transitions of $\mathcal{N}_k$ do not have tokens if there is at least one token in one of its input places (resp. output places), for all reachable markings of $\mathcal{N}_k$;*

*C*2. *All places $p_i \in P_k$ that are not source places in $\mathcal{N}_k$ cannot have observable input transitions in $\mathcal{N}_N$, i.e., $Post_N(t_j, p_i) = 0$, for all $t_j \in T_{N_o}$;*

*C*3. *The places of $\mathcal{N}_k$ that are not source places cannot have a token in the initial marking of $\mathcal{N}_N$.*

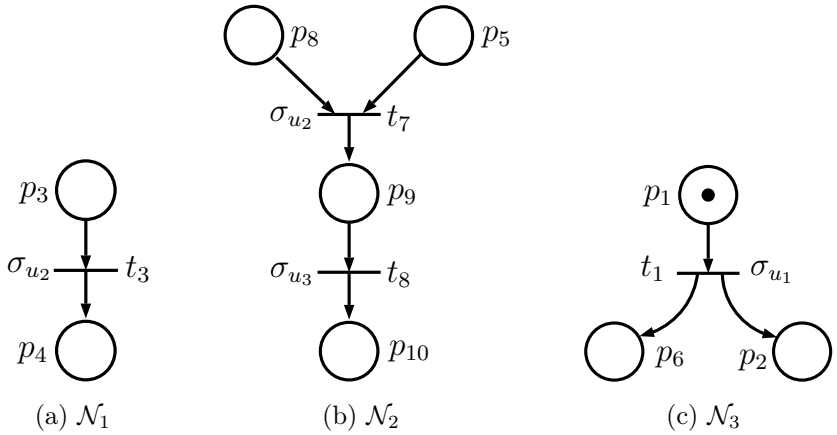
Proof. Let us consider first Condition $C1.(i)$, *i.e.*, $\mathcal{N}_k$ is an SMPN with a unique source place. Then, since $\mathcal{N}_N$ is safe, the sum of the number of tokens of all the places of $\mathcal{N}_k$ is at most equal to one for all reachable markings of the net. Thus, only one place $p_i \in P_k$ of $\mathcal{N}_k$ can have a token at a time, and after the firing of an observable output transition of p_i, the token is removed from $\mathcal{N}_k$. After applying Algorithm 1 to eliminate all transitions of $\mathcal{N}_k$, all places of $\mathcal{N}_k$ are merged into a unique place p with the same observable output transitions as the places of P_k, which corresponds to the observed behavior of the SMPN. Since it is considered, according to Condition $C2$, that only the source place of $\mathcal{N}_k$ can have an observable input transition, and, according to Condition $C3$, only the source place can have an initial token, then an observable transition t_o of $\mathcal{N}_N$ that is an output transition of a place of $\mathcal{N}_k$, is enabled if, and only if, t_o is enabled in the modified net obtained by removing all unobservable transitions of the SMPN $\mathcal{N}_k$. Thus, Conditions $C2$ and $C3$ guarantee that the observed behavior of the modified net obtained by removing all unobservable transitions of $\mathcal{N}_k$ is the same as the observed behavior of $\mathcal{N}_N$.

Let us consider now Condition $C1.(ii)$, *i.e.*, all places of $\mathcal{N}_k$ have at most one input transition and one output transition. Let t_u be an unobservable transition of $\mathcal{N}_k$ with set of input places $I_p(t_u)$ and set of output places $O_p(t_u)$. Since it is assumed that all the output places (resp. input places) of t_u do not have tokens if there is at least one token in one of its input places (resp. output places), for all reachable markings of $\mathcal{N}_k$, then, it is not possible that an output transition of a place in $O_p(t_u)$ and an output transition of

a place in $I_p(t_u)$ be both enabled at the same time. Thus, after applying Algorithm 1 to eliminate t_u, the set of places P_u is formed, and the firing of an output transition t_o of any place $p_{x,y} \in P_u$ corresponds to the firing of the same output transition t_o of $p_x \in I_p(t_u)$ or $p_y \in O_p(t_u)$, since according to Algorithm 1, $\mathcal{N}'_N$ and $\mathcal{N}_N$ have the same observable transitions. Since both p_x and p_y cannot be enabled at the same time, then the removal of a token of $p_{x,y}$ of $\mathcal{N}'_N$ after the firing of t_o leads to the same observed behavior as the firing of t_o in $\mathcal{N}_N$.

In addition, if Conditions $C2$ and $C3$ are true, then a transition t_o in $\mathcal{N}'_N$ is enabled if, and only if, t_o is enabled in $\mathcal{N}_N$, and, consequently, the removal of t_u according to Algorithm 1 does not alter the observed behavior of $\mathcal{N}'_N$ with respect to the observed behavior of $\mathcal{N}_N$. It is also important to remark that the elimination of an unobservable transition t_u does not change the characteristics of the modified net, *i.e.*, each place of $\mathcal{N}'_k$ still has at most a unique input transition and a unique output transition. Thus, the observed language of $\mathcal{N}_k$ is the same as the observed language of the modified net obtained by removing recursively all unobservable transitions of $\mathcal{N}_k$. ■

Example 9. *Consider again the LPN model $\mathcal{N}_N$ depicted in Figure 9(a). If we apply Algorithm 2 to $\mathcal{N}_N$, the weakly connected components $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$, depicted in Figure 10, are obtained. Note that all conditions C1-C3, presented in Theorem 1, are satisfied in $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$, and thus, we can apply Algorithm 1 recursively, eliminating all unobservable transitions of $\mathcal{N}_N$, in order to compute the Petri net $\mathcal{N}_o$ depicted in Figure 11.* □



(a) $\mathcal{N}_1$ (b) $\mathcal{N}_2$ (c) $\mathcal{N}_3$

Figure 10 – Weakly connected components of $\mathcal{N}_N$: $\mathcal{N}_1$ (a), $\mathcal{N}_2$ (b), and $\mathcal{N}_3$ (c) of Example 9.

Note that the elimination of an unobservable transition t_u leads to the creation of $|I_p(t_u)| \times |O_p(t_u)|$ places. Thus, the size of the Petri net $\mathcal{N}_o$ depends on the number of unobservable transitions of each $\mathcal{N}_k$, and the number of their input and output places. Let n_{p_k} denote the maximum number of input or output places of all unobservable transitions of

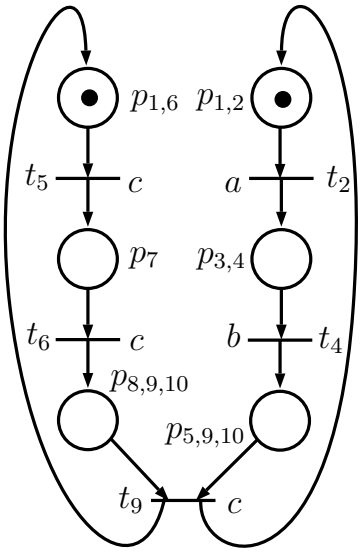


Figure 11 – Observable Behavior Petri Net $\mathcal{N}_o$ of Example 9.

$\mathcal{N}_k$. Then, in the worst-case, the number of places of $\mathcal{N}_o$ grows with complexity $O(|P_N| + \sum_{k=1}^{\rho} n_{p_k}^{|T_k|+1})$. It is also important to remark that the worst-case is, in general, not observed in real-world systems and, in several examples there is a reduction in the number of places of $\mathcal{N}_o$ in comparison with $\mathcal{N}_N$.

In the next example we show that the conditions presented in Theorem 1 are only sufficient, *i.e.*, the OBPN $\mathcal{N}_o$, obtained by using Algorithm 1 to eliminate all the unobservable transitions of $\mathcal{N}_N$, may be such that $\ell_o(S_o) = M(S_N)$ when the conditions are not satisfied.

Example 10. *Consider the LPN $\mathcal{N}$ presented in Figure 12, where $T_u = \{t_2, t_3, t_4\}$ and $T_o = \{t_1, t_5, t_6, t_7, t_8\}$. Notice that, the unique weakly connected component of $\mathcal{N}$ obtained by using Algorithm 2, $\mathcal{N}_1$, depicted in Figure 13, does not satisfy Conditions C1.(i) and C1.(ii) of Theorem 1, i.e., it is not an SMPN and p_2 has more than one unobservable output transition. However, if Algorithm 1 is used recursively to eliminate all the transitions of $\mathcal{N}_1$, then the modified LPN $\mathcal{N}_o$ of Figure 14 is obtained. As it can be seen, the observable language of $\mathcal{N}$ is equal to the language generated by $\mathcal{N}_o$.* □

It is important to remark that the assumption that the SMPN $\mathcal{N}_k$ has a unique source place, in Condition C1.(i) of Theorem 1, can be relaxed. In order to do so, it is necessary to modify $\mathcal{N}_N$, generating a new fault-free behavior Petri net $\mathcal{N}_N^a$, such that each $\mathcal{N}_k$ that is a SMPN of $\mathcal{N}_N$ with number $n_s > 1$ of source places, is replaced with n_s SMPN in $\mathcal{N}_N^a$ where each one has a unique source place. In this case, the conditions of Theorem 1 can be verified in $\mathcal{N}_N^a$.

In Algorithm 3 we present a method for modifying $\mathcal{N}_N$ when $\mathcal{N}_k$ is an SMPN with more than one source place p_{s_q}, for $q = 1, 2, \ldots, n_s$. Firstly, in Line 1, we create set P_s formed of all source places p_{s_q} of $\mathcal{N}_k$, and then, in Line 2, we remove from P_N all places

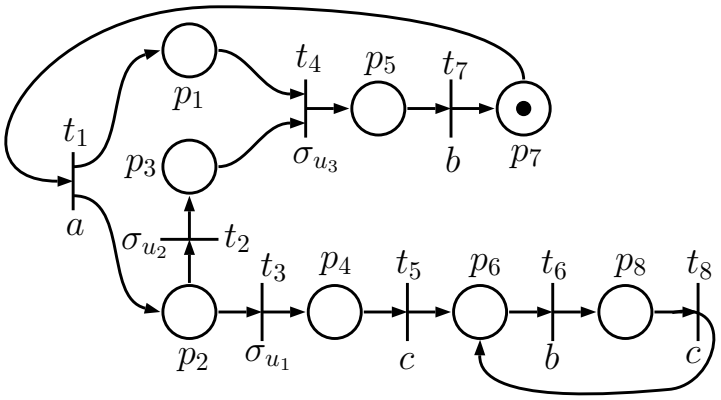


Figure 12 – LPN model $\mathcal{N}$ of Example 10.

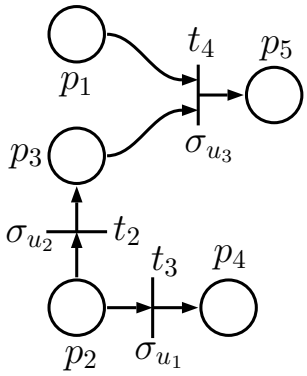


Figure 13 – $\mathcal{N}_1$ of Example 10.

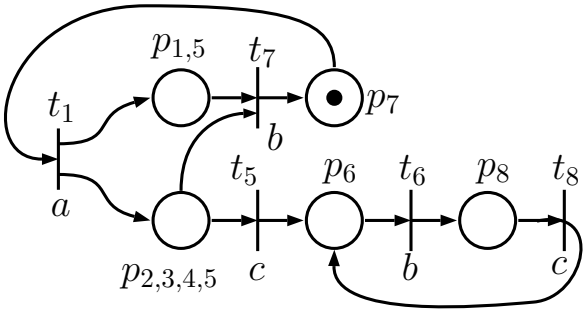


Figure 14 – $\mathcal{N}_o$ of Example 10.

of $\mathcal{N}_k$, except the source places of P_s, forming set P_N^a, and in Line 3, we remove from T_N all transitions of T_k and the observable output transitions of the the places in P_k, forming set T_N^a. Then, in Lines 4 and 5, we define the arcs from places to transitions and from transitions to places, respectively, for all places and transitions of P_N^a and T_N^a, of $\mathcal{N}_N^a$. Then, for each source place $p_{s_q} \in P_S$, we perform a search for all nodes that are reachable in P_k from the source place. In order to do so, we use Algorithm 4, called $REACH$, that systematically explores the nodes (places and transitions) of a Petri net graph, starting from each source place p_{s_q}. The output of the $REACH$ algorithm are the set of places P_q, formed of all places reached from p_{s_q} in $\mathcal{N}_k$ without considering p_{s_q} itself, and the

set of transitions T_q, reached from p_{s_q}, including the observable output transitions of the places in P_k. After that, from Lines 9 to 12 new places p_i^q and transitions t_j^q are created for each place $p_i \in P_q$ and transition $t_j \in T_q$, and the sets P_N^q and T_N^q are formed. The arcs connecting the places and transitions of P_N^q and T_N^q are created in Lines 13 to 16. In Lines 17 and 18, the sets of places and transitions of $\mathcal{N}_N^a$ are updated by adding the places and transitions, respectively, of P_N^q and T_N^q. Finally, in Lines 19 and 21, the initial state $x_{0,N}^a$ is defined, and in Lines 20 and 22, the labeling function l_N^a is defined.

Algorithm 3 Modification algorithm when $\mathcal{N}_k$ is an SMPN with more than one source place

Input: $\mathcal{N}_N = \big(P_N, T_N, Pre_N, Post_N, x_{0,N}, \Sigma_N, \ell_N\big)$,
$\mathcal{N}_k = \big(P_k, T_k, Pre_k, Post_k, x_{0,k}, \Sigma_N, \ell_k\big)$
Output: $\mathcal{N}_N^a = \Big(P_N^a, T_N^a, Pre_N^a, Post_N^a, x_{0,N}^a, \Sigma_N, \ell_N^a\Big)$

1: Let P_s be the set formed of all source places p_{s_q} of N_k
2: Define $P_N^a = (P_N \setminus P_k) \cup P_s$
3: Define $T_N^a = T_N \setminus (T_k \cup O_t(P_k))$
4: $Pre_N^a(p_i, t_j) = Pre_N(p_i, t_j)$, for all $p_i \in P_N^a$ and $t_j \in T_N^a$
5: $Post_N^a(t_j, p_i) = Post_N(t_j, p_i)$, for all $p_i \in P_N^a$ and $t_j \in T_N^a$
6: **for each** $p_{s_q} \in P_s$ **do**
7: Define $P_N^q = \emptyset$ and $T_N^q = \emptyset$
8: $(P_q, T_q) = REACH(\mathcal{N}_k, \mathcal{N}_N, p_{s_q})$
9: **for each** $p_i \in P_q$ **do**
10: Create place p_i^q and add it to P_N^q
11: **for each** $t_j \in T_q$ **do**
12: Create transition t_j^q and add it to T_N^q
13: $Pre_N^a(p_{s_q}, t_j^q) = Pre_N(p_{s_q}, t_j)$, for all $t_j^q \in T_N^q$
14: $Pre_N^a(p_i^q, t_j^q) = Pre_N(p_i, t_j)$, for all $p_i^q \in P_N^q$ and $t_j^q \in T_N^q$
15: $Post_N^a(t_j^q, p_i^q) = Post_N(t_j, p_i)$, for all $p_i^q \in P_N^q$ and $t_j^q \in T_N^q$
16: $Post_N^a(t_j^q, p_i) = 1$, for $t_j^q \in T_N^q$ and for all $p_i \in O_p(t_j) \cap P_N^a$, and $Post_N^a(t_j^q, p_i) = 0$, for all $p_i \notin O_p(t_j) \cap P_N^a$ and $p_i \notin P_N^q$
17: $P_N^a \leftarrow P_N^a \cup P_N^q$
18: $T_N^a \leftarrow T_N^a \cup T_N^q$
19: $x_{0,N}^a(p_{s_q}) = x_{0,N}(p_{s_q})$, and $x_{0,N}^a(p_i^q) = 0$, for all $p_i^q \in P_N^q$
20: $l_N^a(t_j^q) = l_N(t_j)$, for all $t_j^q \in T_N^q$
21: $x_{0,N}^a(p_i) = x_{0,N}(p_i)$, for all $p_i \in P_N \setminus P_k$
22: $l_N^a(t_j) = l_N(t_j)$, for all $t_j \in T_N \setminus (T_k \cup O_t(P_k))$

In the sequel, we present an example to illustrate the use of Algorithm 3.

Example 11. *Consider the Petri net $\mathcal{N}_N$ depicted in Figure 15(a), where the unique weakly connected component $\mathcal{N}_1$ is formed of the set of places $P_k = \{p_1, p_2, p_3, p_4\}$ and the unobservable transitions $T_k = \{t_1, t_2, t_3\}$. Note that $\mathcal{N}_1$ is an SMPN with two source places, p_1 and p_2.*

Algorithm 4 $(P_q, T_q) = REACH(\mathcal{N}_k, \mathcal{N}_N, p_{s_q})$

1: Compute all the places and transitions that are reachable from p_{s_q} in $\mathcal{N}_k$, using a standard breadth-first search algorithm (CORMEN et al., 2009).
2: Separate the nodes computed in Line 1 into the set of places P_q and the set of transitions T_q
3: Add to T_q all observable output transitions of the places $p_q \in P_q$ obtained from $\mathcal{N}_N$
4: $P_q \leftarrow P_q \setminus \{p_{s_q}\}$

In Line 1 of Algorithm 3, set $P_s = \{p_1, p_2\}$ is created. Then, in Line 2, all places that are not source places are removed from $\mathcal{N}_N$ and in Line 3, all transitions are removed from $\mathcal{N}_N$. Then, in Line 6, for the source place $p_{s_1} = p_1$, the first reach is computed, using Algorithm 4, leading to sets $P_1 = \{p_3, p_4\}$ and $T_1 = \{t_1, t_3, t_4, t_5\}$. In Lines 9 to 12, the sets $P_N^1 = \{p_3^1, p_4^1\}$ and $T_N^1 = \{t_1^1, t_3^1, t_4^1, t_5^1\}$ are computed, and in Lines 13 to 16, the arcs connecting places p_1, p_2, p_3^1, p_4^1 and transitions t_1^1, t_3^1, t_4^1, t_5^1, depicted in Figure 15(b), are created. In Line 17 and 18, respectively, sets P_N^a and T_N^a are updated to $P_N^a = \{p_1, p_2, p_3^1, p_4^1\}$ and $T_N^a = \{t_1^1, t_3^1, t_4^1, t_5^1\}$. The initial marking of the source place p_1 is computed in Line 19, and all transitions of T_N^1 are labeled in Line 20 with the same event of the corresponding transition of T_N. Algorithm 3 continues for the source place p_2, generating the complete Petri net depicted in Figure 15(b). □

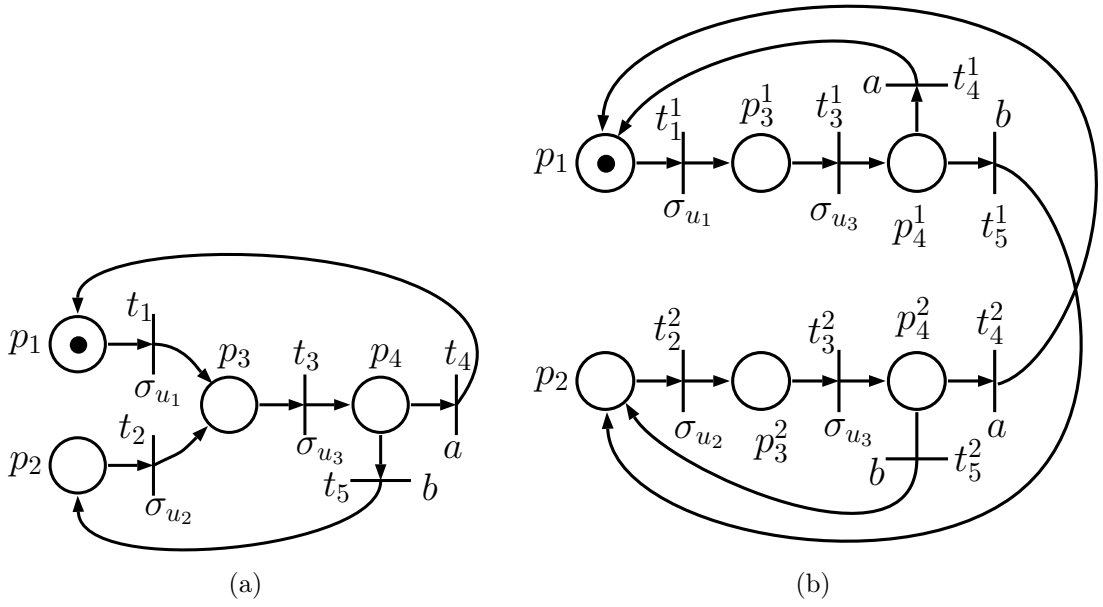


Figure 15 – Petri nets $\mathcal{N}_N$ (a) and $\mathcal{N}_r$ (b) of Example 11.

3.3 FINAL REMARKS

In this chapter we propose a new method for the detection of faults of discrete event systems modeled by safe Labeled Petri Nets. The method consists of removing the

non-observable transitions of the fault free behavior model of the system resulting in the Observable Behavior Petri Net, which is, in general, a smaller graph than the original fault free behavior Petri net model. Once the OBPN is computed, the fault detection process can be carried out by computing the feasible observable events for a given marking. If an event that is not feasible for the current marking is observed, the fault is detected.

In order to implement the method described in this chapter we suppose that the Petri net system model is diagnosable. The diagnosability analysis can be carried out offline by using any method proposed in the literature (CABASINO et al., 2012).

In the following Chapter we present two case studies to illustrate the use of the proposed method for fault diagnosis in practical systems. These examples are also used to compare the efficiency of the proposed method with others approaches presented in the literature.

4 CASE STUDIES

4.1 WEIGHING-MIXING SYSTEM

4.1.1 Description of the system

The case study we consider in this work is the weighing-mixing system introduced in the international standard IEC 60848 (IEC:60848, 2002), whose scheme is shown in Figure 16. In this system, products A and B are mixed together with two soluble bricks. In order to do so, valve **VA** is opened and the product A is firstly poured in a weighing balance until the mark w_a is reached, then valve **VA** is closed and valve **VB** is opened to provide product B to the weighing balance until mark w_b is reached. When it occurs, valve **VB** is closed and valve **VC** is opened, pouring products A and B into the mixer, until mark w_z is reached in the weighing balance. Then, valve **VC** is closed. We consider here that the flow of valve **VC** is greater than the flow of valves **VA** and **VB** when completely opened. In parallel, soluble bricks are fed into the mixer by a conveyor belt that is controlled using motor **BM**. The correct mixture is achieved with two bricks, that are counted using a transit detector (**TD**).

Once all necessary raw material (two bricks, and products A and B) are in the mixer, motor **MR** is turned on in order to start the mixing process. When the mixture is done, which is achieved after a predetermined time, motor **TM** is turned on in order to tilt the mixture container until the position of sensor **S1**, causing the mixture to be poured down. After reaching position **S1**, motor **MR** is turned off and motor **TM** is actuated to

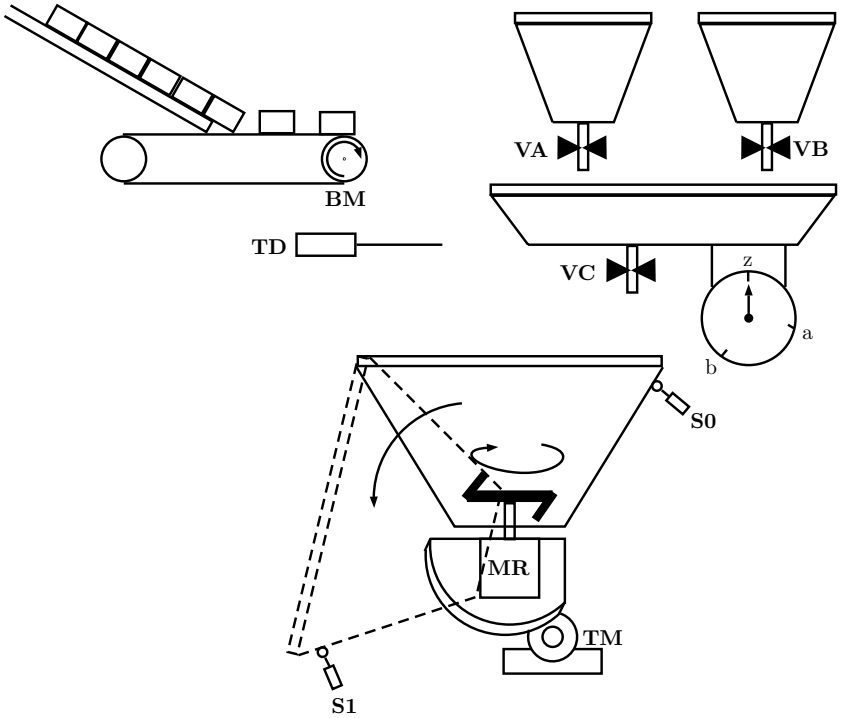


Figure 16 – Scheme of the weighing-mixing system, adapted from (IEC:60848, 2002).

turn on in the opposite direction until position **S0** is reached, which allows a new cycle to be initiated. The production cycle begins when a cycle start button (not represented in Figure 16) is pressed.

4.1.2 System model

We first present the fault-free behavior model of the system $\mathcal{N}_N$, which is represented by the black subnet of Figure 17. Place p_1 starts with one token representing that the system is ready to begin a production cycle, which is initialized by event c_s that models the pressing of the cycle start button. When c_s occurs, both subsystems, the conveyor belt and the weighing unit, start to operate, which is modeled by the parallel paths starting with places p_2 and p_7 in Figure 17. Transition t_2 is labeled with event b_{on}, that represents the command to turn on the conveyor belt motor **BM**. Transitions t_3 and t_4 are labeled with event d that models the rising edge of the **TD** signal, and thus, after two occurrences of event d, place p_5 receives a token, indicating that the mixing unit have been fed with two bricks. After that, the conveyor belt motor is commanded to turn off, modeled by event b_{off}. In parallel, products A and B are provided to the mixing unit. First, the command to open valve **VA**, va_o, is issued by the controller, and then, when the weight w in the balance is greater than value w_a, event a occurs. After observing the occurrence of a, the controller sends a command to close valve **VA**, va_c, and open valve **VB**, vb_o. When the weight w in the balance is greater than w_b, represented by event b, the controller issues the command vb_c to close valve **VB**, and then, command vc_o to open valve **VC** to empty the weighing balance. When the weight w is smaller than or equal to w_z, represented by event z, the balance is empty, modeled by place p_{15}. After that, valve **VC** is closed, represented by event vc_c.

When both places p_6 and p_{16} have tokens, the synchronizing transition t_{15} is enabled and immediately fires, which indicates that the mixing process has started, represented by places p_{17} and p_{20}. Transition t_{16} is associated with a timer. Since the value of the timer is considered in this work unobservable, t_{16} is labeled with the unobservable event σ_u that occurs when the accumulated value of the timer reaches a predetermined value. Then, the controller sends the command t_d to actuate the tipping motor **TM** to tilt down. The mixing motor is actuated by command m_{on}, and when both places p_{19} and p_{21} have tokens, the synchronizing transition t_{19} immediately fires, adding a token to p_{22}. When the mixer reaches the position of sensor **S1**, represented by event s_1, the mixer motor is turned off (m_{off}), and then the mixer is commanded to tilt up (t_u), until reaching the position of sensor **S0**, represented by event s_0.

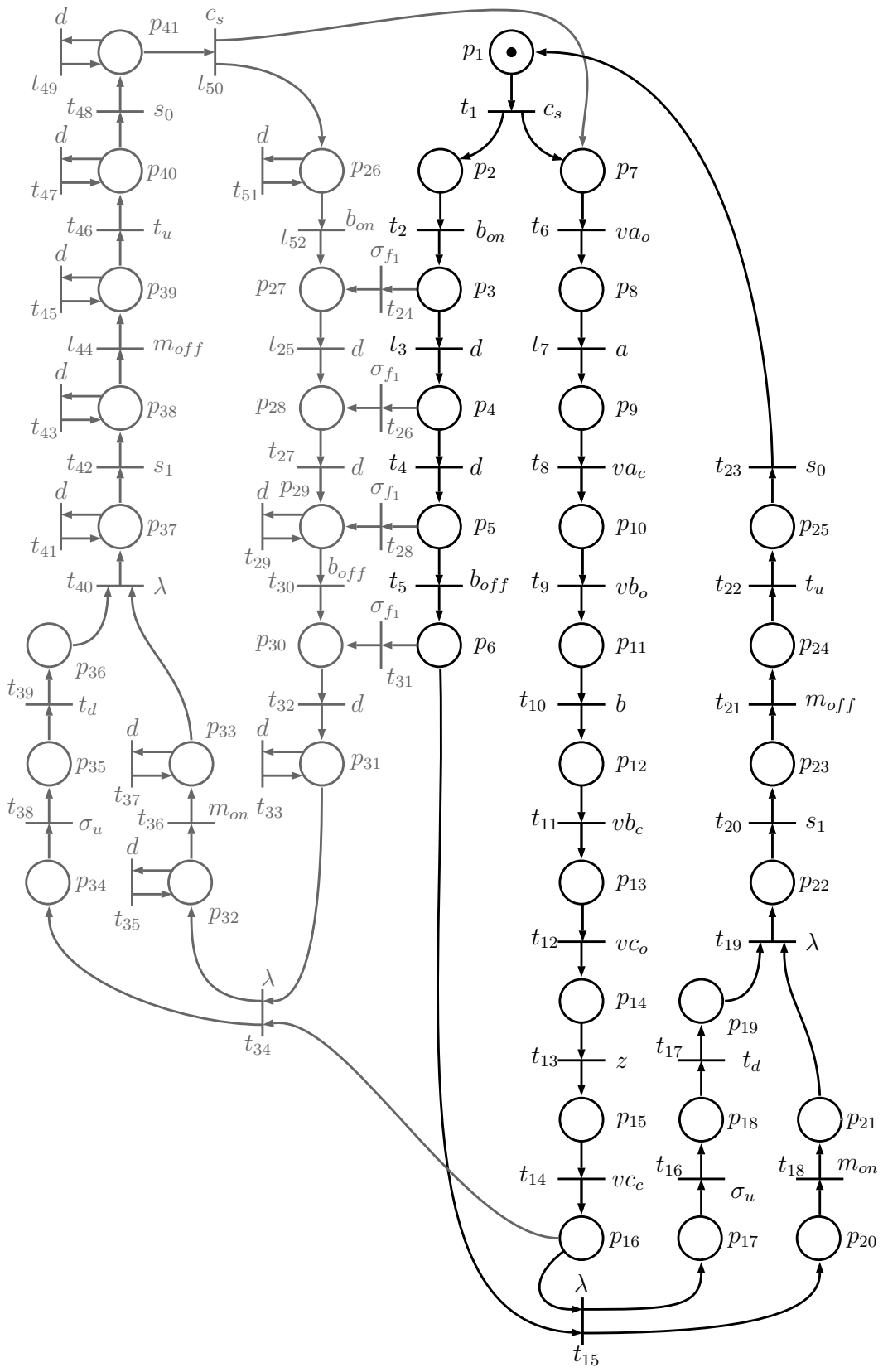


Figure 17 – Complete labeled Petri net $\mathcal{N}_{l_1}$ of the weighing-mixing system considering fault σ_{f_1}.

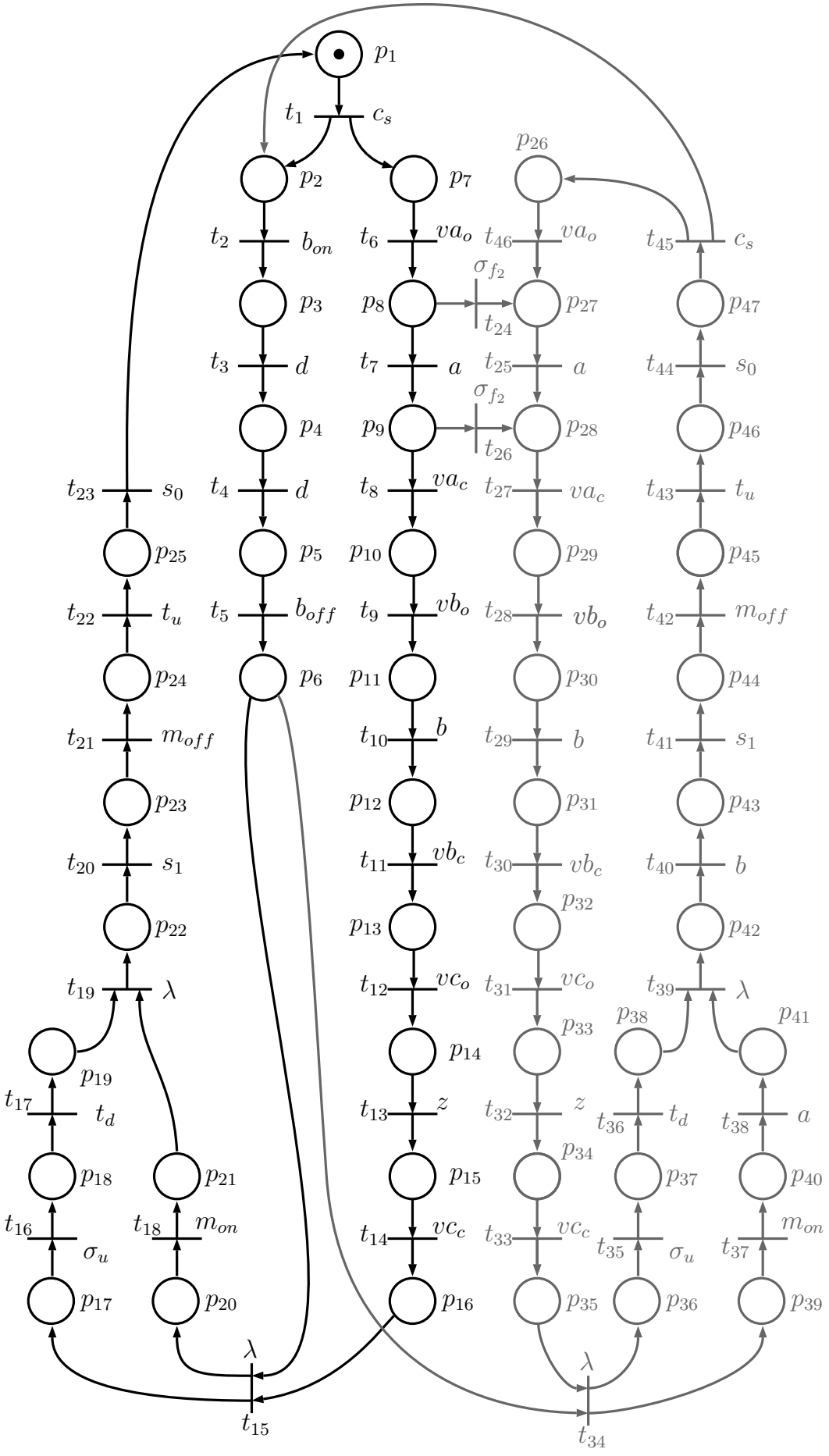


Figure 18 – Complete labeled Petri net $\mathcal{N}_{l_2}$ of the weighing-mixing system considering fault σ_{f_2}.

We consider in this case study two fault types. For the sake of simplicity, and without loss of generality, we consider that only one fault may occur in one production cycle. Thus, the two different fault types lead to two possible models of the system, $\mathcal{N}_{l_1}$ and $\mathcal{N}_{l_2}$, represented in Figures 17 and 18, respectively. In Figure 17, we present the model of the system subject to fault σ_{f_1}, that represents a problem related with motor **BM** of the conveyor belt. When σ_{f_1} occurs, **BM** cannot be turned off anymore and the conveyor belt keeps running, wrongly feeding the mixing unit with new bricks. We consider that the fault may occur at any time between the occurrence of events b_{on} and b_{off}. After the occurrence of σ_{f_1}, bricks continue to be delivered to the mixer, being detected by the transit sensor **TD**, *i.e.*, an event d may occur at any time. It is important to remark that transition t_{32} models the fact that an event d is observed necessarily before the synchronization of the parallel activities of weighing products A and B, and feeding the mixer with bricks. This is due to the fact that the time to weigh products A and B, and then to empty the weighing balance is always greater than the time to deliver three bricks to the mixer.

In Figure 18, we present the model of the system subject to a fault in valve **VA**. Event σ_{f_2} models that valve **VA** stuck open when **VA** is already opened, which occurs in places p_8 and p_9. After valve **VA** stuck open, the weighing balance continues to be filled after the command to close the valve. Since the flow in valve **VC** is greater than the flow in valve **VA**, after opening valve **VC**, the weighing balance is emptied in a lower speed than when valve **VA** is not stuck open. However, after closing valve **VC**, it is filled again, leading to the post-fault behavior presented in Figure 18, where events a and b occur while the mixing process is taking place.

Note that faults σ_{f_1} and σ_{f_2} are related with the supply of raw material to the system, and therefore it is important to halt the system execution when any of these faults is detected so as not to lose material when it occurs.

In this example, we consider that only the events associated with the system sensors are observable. Thus, the set of observable events is $\Sigma_o = \{a, b, z, d, s_0, s_1\}$, and the unobservable event set is $\Sigma_u = \{\lambda, \sigma_u, \sigma_{f_1}, \sigma_{f_2}, c_s, b_{on}, b_{off}, va_o, va_c, vb_o, vb_c, vc_o, vc_c, m_{on}, m_{off}, t_d, t_u\}$. Table 1 presents the events of the system and their corresponding meanings.

The faults are separated into two fault classes $\Sigma_{f_1} = \{\sigma_{f_1}\}$ and $\Sigma_{f_2} = \{\sigma_{f_2}\}$, and the fault diagnosis is carried out in two steps. In the first step, by using the observable behavior Petri net $\mathcal{N}_o$, the fault is detected online, and, after detection, the system is stopped, and then the fault is isolated.

4.1.3 Diagnosis process

In order to implement the fault diagnosis strategy proposed in this work, it is first necessary to identify the weakly connected components $\mathcal{N}_k$, $k = 1, \ldots, \rho$, of $\mathcal{N}_N$ by using

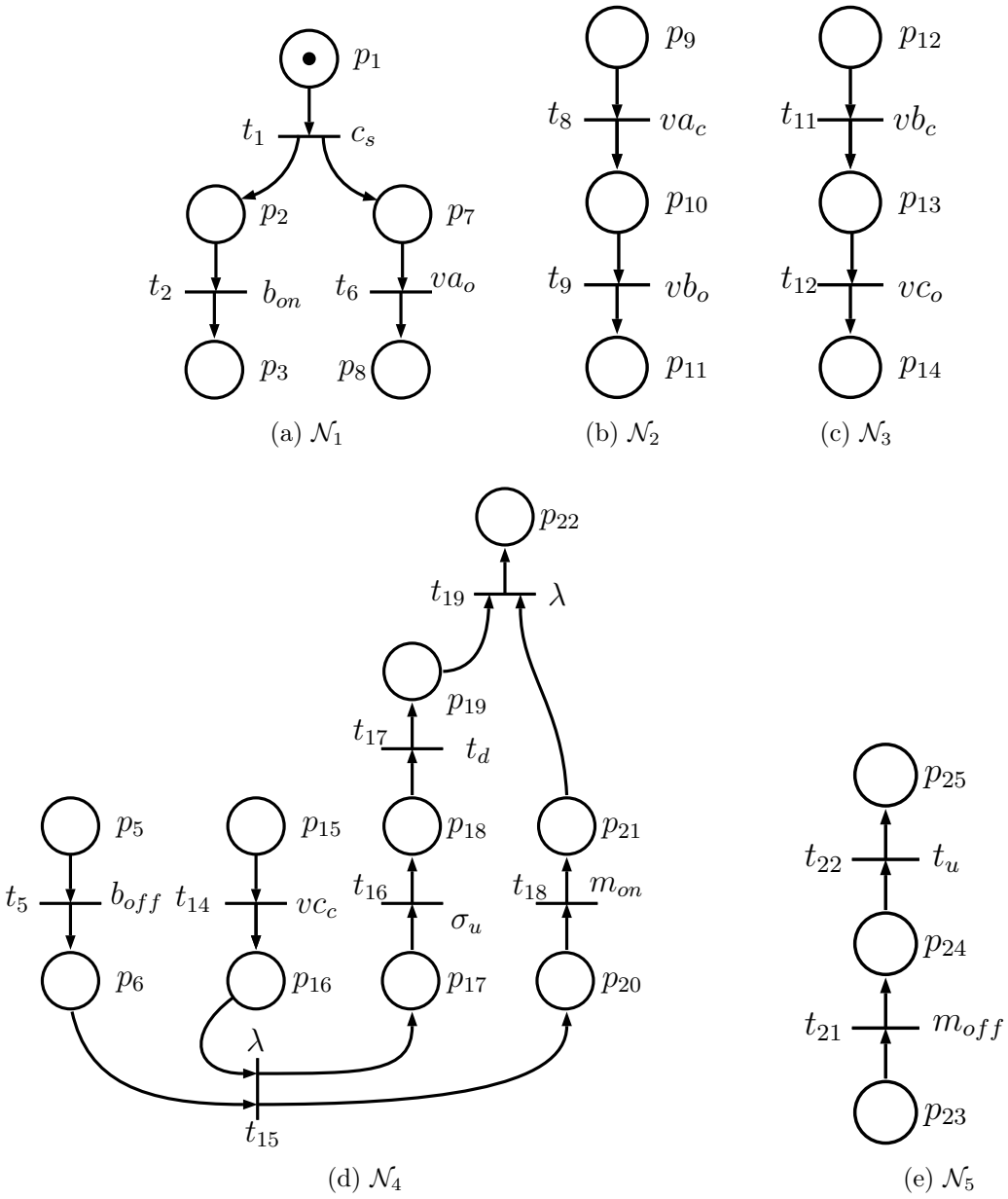
Table 1 – Events of the weighing-mixing system.

Event	Meaning
c_s	cycle start button is pressed
a	weight is greater than w_a
b	weight is greater than w_b
z	weight is smaller than or equal to w_z
d	detection of a brick with sensor **TD**
λ	synchronization of the end of tasks
σ_u	timeout of the mixing process
s_0	rising edge of sensor **S0**
s_1	rising edge of sensor **S1**
b_{on}	turn on the conveyor belt motor **BM**
b_{off}	turn off the conveyor belt motor **BM**
va_o	open valve **VA**
va_c	close valve **VA**
vb_o	open valve **VB**
vb_c	close valve **VB**
vc_o	open valve **VC**
vc_c	close valve **VC**
t_d	actuate **TM** to tilt down the mixer
t_u	actuate **TM** to tilt up the mixer
m_{on}	turn on the mixer motor **MR**
m_{off}	turn off the mixer motor **MR**
σ_{f_1}	**BM** remains permanently turned on
σ_{f_2}	**VA** stuck open

Algorithm 2. In Figure 19, we present the five weakly connected components of $\mathcal{N}_N$. As it can be seen from Figure 19, all $\mathcal{N}_k$, $k = 1, 2, \ldots, 5$, satisfy the conditions of Theorem 1. Thus, we can compute the OBPN $\mathcal{N}_o$ from the fault-free Petri net model $\mathcal{N}_N$, presented in the black subnet of Figure 17, applying recursively Algorithm 1 until all unobservable transitions of $\mathcal{N}_N$ are eliminated. In Figure 20, the OBPN $\mathcal{N}_o$ is depicted. Note that $\mathcal{N}_o$ has only 10 places and 7 transitions, and therefore, it can be implemented without requiring a large amount of memory to store it in a computer.

It is important to remark that the reachability graph of the Petri nets of Figures 17 and 18 have, respectively, 132 states and 298 transitions, and 126 states and 214 transitions, which shows that the diagnosers constructed using the reachability graphs of $\mathcal{N}_{l_1}$ and $\mathcal{N}_{l_2}$ would need much more memory space than the proposed fault detector based on $\mathcal{N}_o$. Moreover, it is important to remark that the BRG proposed in Cabasino et al. (2011) has 40 states and 100 transitions for the LPN $\mathcal{N}_{l_1}$, and 28 states and 46 transitions for the LPN $\mathcal{N}_{l_2}$.

Now, let us show how $\mathcal{N}_o$ can be used for fault detection. Let us suppose that the fault sequence associated with fault σ_{f_1}, $s = c_s\ b_{on}\ va_o\ \sigma_{f_1}\ d\ a\ va_c\ vb_o\ d\ b_{off}\ d$, has been executed by the system. Thus, its observation is given by $dadd$, which cannot be



(a) $\mathcal{N}_1$ (b) $\mathcal{N}_2$ (c) $\mathcal{N}_3$

(d) $\mathcal{N}_4$ (e) $\mathcal{N}_5$

Figure 19 – Weakly connected components of $\mathcal{N}_N, \mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4$ and $\mathcal{N}_5$ of the weighing-mixing system.

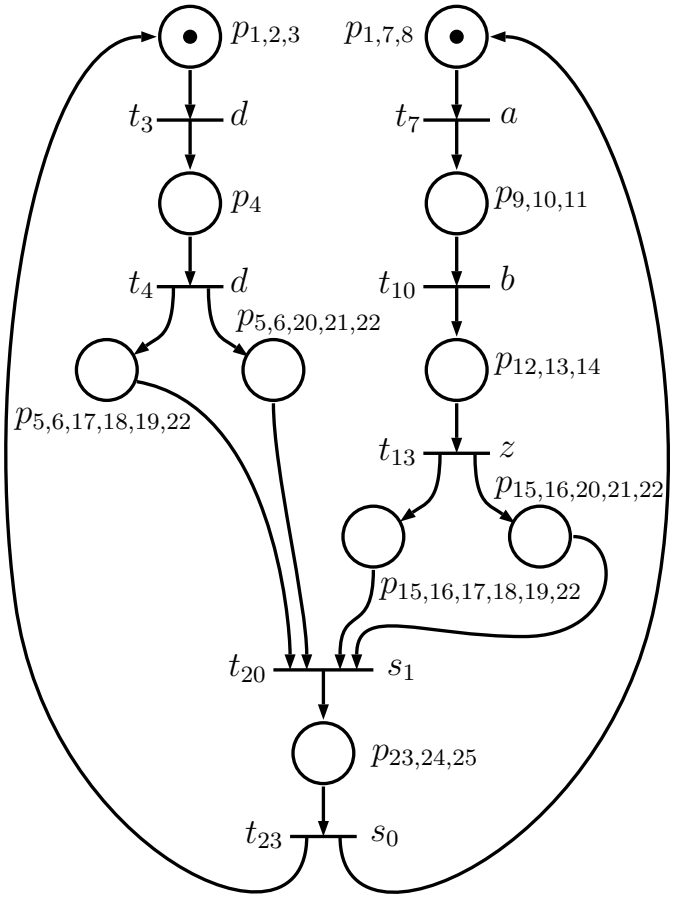


Figure 20 – Observable Behavior Petri Net $\mathcal{N}_o$ of the weighing-mixing system.

executed in $\mathcal{N}_o$, and the fault detection scheme indicates the occurrence of a fault. Since sequence *dadd* can only be observed if a transition labeled with σ_{f_1} is fired in $\mathcal{N}_{l_1}$, and it cannot be observed in $\mathcal{N}_{l_2}$, then the fault isolation module indicates that σ_{f_1} has occurred.

Let us consider now that sequence $s = c_s\ b_{on}\ va_o\ d\ a\ \sigma_{f_2}\ va_c\ vb_o\ d\ b_{off}\ b\ vb_c\ vc_o\ z\ vc_c\ m_{on}\ a$ is executed. In this case, the observation of s is *dadbza* which cannot be executed in $\mathcal{N}_o$. Thus, the fault detector indicates the fault occurrence. Since sequence *dadbza* can only be observed after the firing of a transition labeled with σ_{f_2} in $\mathcal{N}_{l_2}$, and it cannot be observed in $\mathcal{N}_{l_1}$, then the fault isolation module indicates that σ_{f_2} has occurred.

4.2 MANUFACTURING SYSTEM

Let us consider the manufacturing system depicted in Figure 21, obtained by adapting the example presented in Zhou and DiCesare (1993) and Cabasino et al. (2011). The system consists of four machines (M_1-M_4), four robots (R_1-R_4), and one automated guided vehicle (AGV), that produces two different types of products. Parts are delivered to the system to be processed, and robot R_1 transports them to Machine M_1 or M_3 alternately, starting with M_1. Each machine works on only one part at a time, and each robot arm can transport only one part from one machine to another at a time. When the

processing in machine M_1 (resp. M_3) ends, robot R_3 (resp. R_4) transports the processed part to machine M_2 (resp. M_4). Robot R_2 transports parts processed in machines M_2 and M_4 to the AGV, and the AGV delivers the final products to the output of the system. The complete Petri net system model is presented in Figure 22, where the fault-free behavior is depicted in black.

The gray parts of the Petri net model depicted in Figure 22 represents the post-fault behavior, where we have considered two fault events. Event σ_{f_1} models a fault in Robot R_1 that transports the part to the wrong machine, while event σ_{f_2} represents the fault in the AGV that is not capable of delivering final products to the output, becoming unavailable and emitting the same signal it provides when a part regularly exits the production line. In this example, the events associated with the robot arms picking up a part or delivering a part, and the event emitted by the AGV after delivering a part to the output are observable. All the other events are unobservable. Thus, $\Sigma_o = \{a, e, b, g, c, \ell, d, h, s_1, s_2\}$. In Table 2, the meaning of each event of the system is presented.

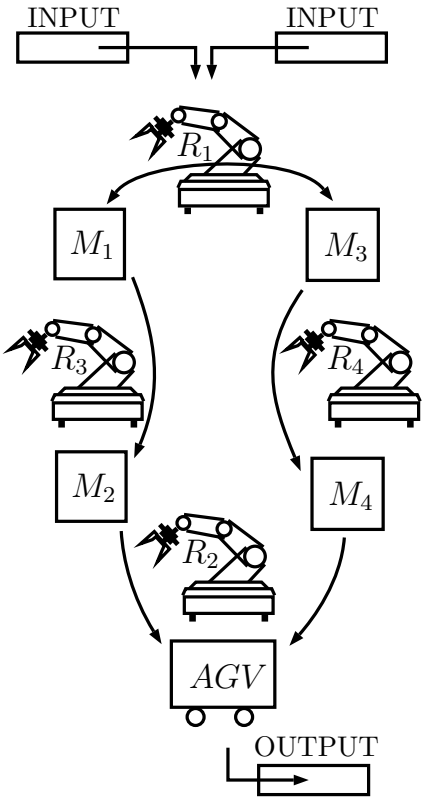


Figure 21 – Manufacturing system scheme.

In this case, it can be seen that Assumptions **A1**-**A3** and all conditions $C1$-$C3$ of Theorem 1 are satisfied. The OBPN $\mathcal{N}_o$ of this example can be seen in Figure 23. The reachability graph of the Petri net of Figure 22 has 30,880 states and 140,748 transitions, and the BRG, computed using the method presented in Cabasino et al. (2011), has 435 states and 1,182 transitions. The OBPN $\mathcal{N}_o$, depicted in Figure 23, has only 22 places and 14 transitions, which shows the great reduction in the model used for fault detection

Table 2 – Events of the manufacturing system.

Event	Meaning
a	R_1 picked up a part in Input 1
e	R_1 picked up a part in Input 2
s_1	R_1 delivered a part in machine M_1 and M_1 starts processing the part
s_3	R_1 delivered a part in machine M_3 and M_3 starts processing the part
e_i	end of service in M_i, for $i = 1, 2, 3, 4$
b	signal emitted by R_3 when picking up a part from M_1 or delivering it to M_2
g	signal emitted by R_4 when picking up a part from M_3 or delivering it to M_4
c	R_2 picked up a part in M_2
ℓ	R_2 picked up a part in M_4
h	R_2 delivered a part to the AGV
d	signal emitted by the AGV
σ_{f_1}	fault event in R_1
σ_{f_2}	fault event in AGV

using the method proposed in this work than using the other methods proposed in the literature.

In Table 3 we compare the size of the reachability graph, the basis reachability graph, and the OBPN for the weighing-mixing system and the manufacturing system, where we can see that the proposed method leads, in these examples, to a much smaller model than by using the other strategies. The numbers of states and transitions of the reachability graph have been computed using the Petri net tool TINA (`http://projects.laas.fr/tina/`), and the numbers of states and transitions of the basis reachability graph have been computed with the MATLAB toolbox used in (CABASINO et al., 2011) (`https://www.alessandro-giua.it/UNICA/TESI/09_Marco.Pocci/PN_DIAG.zip`).

Table 3 – Comparison between the sizes of the reachability graph (RG), the basis reachability graph (BRG), and the OBPN for the weighing-mixing system and the manufacturing system.

System	RG		BRG		OBPN	
	states	transitions	states	transitions	places	transitions
weighing-mixing (σ_{f_1})	132	298	40	100	10	7
weighing-mixing (σ_{f_2})	126	214	28	46	10	7
manufacturing system	30,880	140,748	435	1,182	22	14

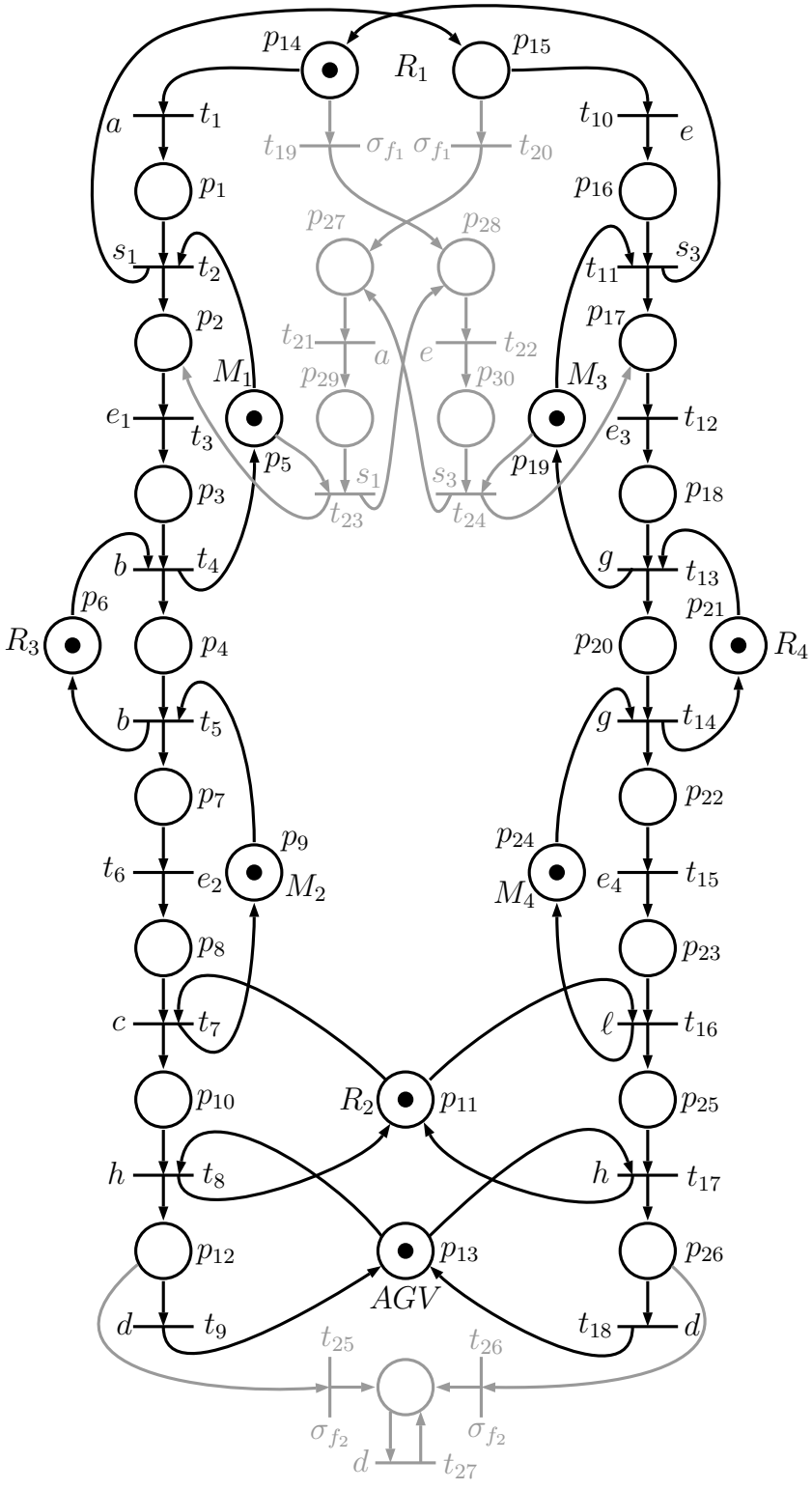


Figure 22 – Petri net system model.

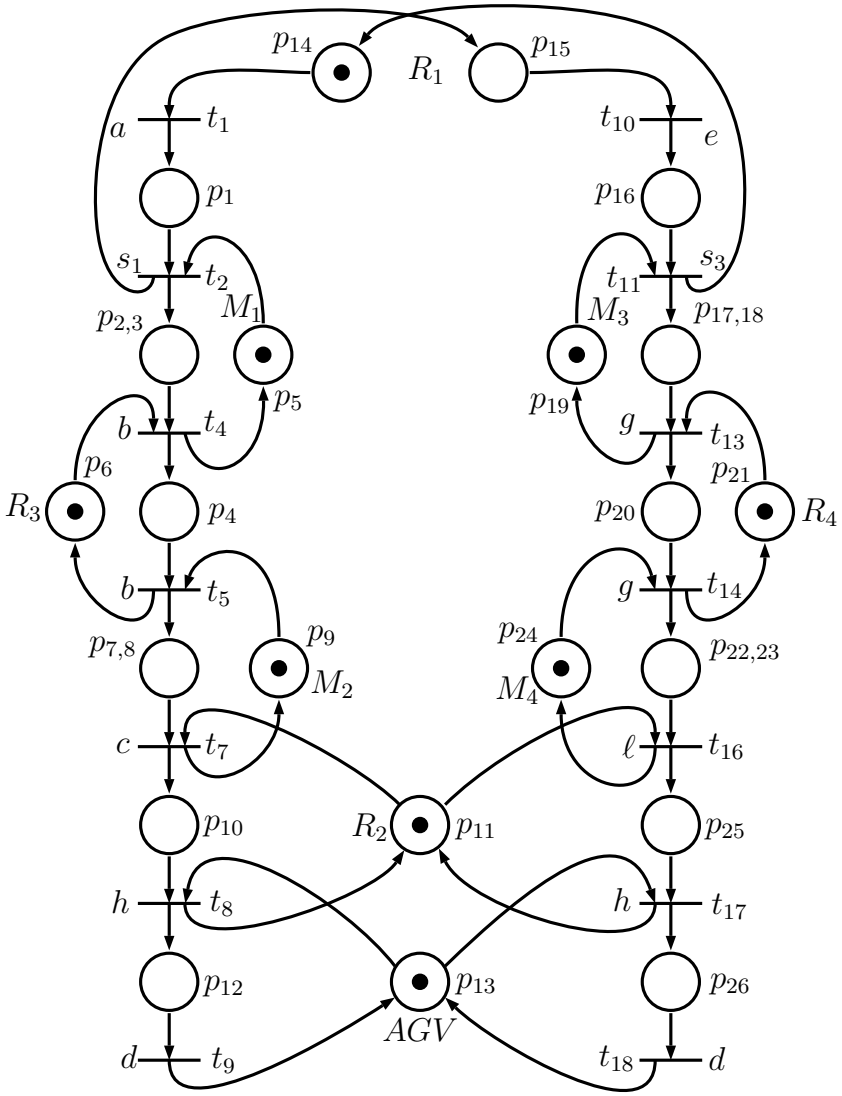


Figure 23 – $\mathcal{N}_o$ for the manufacturing system.

5 CONCLUSION

With the evolution of automated systems, the size of the models are growing in number of components and in complexity, then, it may not be an easy task to correctly diagnose possible faults occurrences that can alter the nominal behavior of these systems in order to prevent equipment damage and maintain the safety of human operators. We have presented in this work, a fault diagnosis scheme for discrete event systems modeled by a class of labeled Petri nets. The method is based on two steps, where in the first one we detect the occurrence of a fault event and then the candidates to these faults are isolated. The main advantage of the presented approach is that the model of the system used to fault detection is, in general, much smaller than the reachability graph of the system, which allows to use less memory for implementing the fault detector than using traditional strategies and, that the fault can be detected in a fast manner, without requiring the use of any reachability graph. The proposed method was consider for a practical weighing-mixing system introduced by the international standard IEC 60848 (IEC:60848, 2002).

In parallel with the writing of this work, significant improvements on the modification algorithms are being made in order to relax some of the proposed hypothesis. We are also currently investigating other classes of LPN that can be used avoiding the construction of complex structures for fault diagnosis. The future research themes are summarized as follows in section 5.1.

The results of this work are going to be presented in the next *Simpósio Brasileiro de Automação Inteligente* (SBAI 2021) and a contribution has been submitted for publication in the Control Engineering Practice Journal. These references are cited in the following.

- Bonafin et al. (2021a)
- Bonafin et al. (2021b)

5.1 FUTURE WORKS

- Implement the proposed diagnose method in a plant using programmable logic controllers
- Relax the assumptions and/or conditions proposed here in order to reach more Petri nets classes.

REFERENCES

BAKALARA, Johane; PENCOLÉ, Yannick; SUBIAS, Audine. How to use Model Checking for Diagnosing Fault Patterns in Petri nets. In: 15TH IFAC Workshop on Discrete Event Systems. [S.l.: s.n.], 2020. P. 269–274.

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, p. 748–759, 2009.

BONAFIN, Ana C.; CABRAL, Felipe G.; MOREIRA, Marcos V. Diagnose de falhas para uma classe de Redes de Petri. **Anais do XII Simpósio Brasileiro de Automação Inteligente-SBAI, Brasil**, 2021a.

BONAFIN, Ana C.; CABRAL, Felipe G.; MOREIRA, Marcos V. Fault Diagnosis of Discrete-Event Systems modeled as Labeled Petri Nets. **Control Engineering Practice**, 2021b. Submitted for publication.

CABASINO, Maria Paola; GIUA, Alessandro; LAFORTUNE, Stéphane; SEATZU, Carla. A new approach for diagnosability analysis of Petri nets using verifier nets. **IEEE Transactions on Automatic Control**, v. 57, n. 12, p. 3104–3117, 2012.

CABASINO, Maria Paola; GIUA, Alessandro; PAOLI, Andrea; SEATZU, Carla. Decentralized diagnosis of discrete-event systems using labeled Petri nets. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, v. 43, n. 6, p. 1477–1485, 2013.

CABASINO, Maria Paola; GIUA, Alessandro; POCCI, Marco; SEATZU, Carla. Discrete event diagnosis using labeled Petri nets. An application to manufacturing systems. **Control Engineering Practice**, v. 19, n. 9, p. 989–1001, 2011.

CABASINO, Maria Paola; GIUA, Alessandro; SEATZU, Carla. Diagnosability of bounded Petri nets. In: PROCEEDINGS of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference. [S.l.: s.n.], 2009. P. 1254–1260.

CABASINO, Maria Paola; GIUA, Alessandro; SEATZU, Carla. Diagnosability of discrete-event systems using labeled Petri nets. **IEEE Transactions on Automation Science and Engineering**, v. 11, n. 1, p. 144–153, 2014.

CABASINO, Maria Paola; GIUA, Alessandro; SEATZU, Carla. Fault detection for discrete event systems using Petri nets with unobservable transitions. **Automatica**, v. 46, n. 9, p. 1531–1539, 2010.

CABRAL, Felipe G. **Synchronous failure diagnosis of discrete-event systems**. 2017. PhD thesis – Universidade Federal do Rio de Janeiro.

CABRAL, Felipe G.; MOREIRA, Marcos V.; DIENE, Oumar; BASILIO, João Carlos. A Petri net diagnoser for discrete event systems modeled by finite state automata. **IEEE Transactions on Automatic Control**, v. 60, n. 1, p. 59–71, 2015.

CASSANDRAS, Christos G; LAFORTUNE, Stephane. **Introduction to discrete event systems**. [S.l.: s.n.], 2009.

CONTANT, Olivier; LAFORTUNE, Stéphane; TENEKETZIS, Demosthenis. Diagnosis of intermittent faults. **Discrete Event Dynamic Systems**, v. 14, n. 2, p. 171–202, 2004.

CORMEN, Thomas H; LEISERSON, Charles E; RIVEST, Ronald L; STEIN, Clifford. **Introduction to algorithms**. [S.l.: s.n.], 2009.

DAVID, René; ALLA, Hassane. **Discrete, continuous, and hybrid Petri nets**. [S.l.: s.n.], 2005. v. 1.

DOTOLI, Mariagrazia; FANTI, Maria Pia; MANGINI, Agostino Marcello; UKOVICH, Walter. On-line fault detection in discrete event systems by Petri nets and integer linear programming. **Automatica**, v. 45, n. 11, p. 2665–2672, 2009. ISSN 0005-1098. DOI: `https://doi.org/10.1016/j.automatica.2009.07.021`. Available from: `https://www.sciencedirect.com/science/article/pii/S0005109809003720`.

FANTI, Maria Pia; SEATZU, Carla. Fault diagnosis and identification of discrete event systems using Petri nets. In: IEEE. 2008 9th International Workshop on Discrete Event Systems. [S.l.: s.n.], 2008. P. 432–435.

GENC, Sahika; LAFORTUNE, Stéphane. Distributed diagnosis of discrete-event systems using Petri nets. In: SPRINGER. INTERNATIONAL Conference on Application and Theory of Petri Nets. [S.l.: s.n.], 2003. P. 316–336.

GIUA, Alessandro; SEATZU, Carla; CORONA, Daniele. Marking estimation of Petri nets with silent transitions. **IEEE Transactions on Automatic Control**, v. 52, n. 9, p. 1695–1699, 2007.

GIUA, Alessandro; SILVA, Manuel. Petri nets and automatic control: A historical perspective. **Annual Reviews in Control**, v. 45, p. 223–239, 2018.

IEC:60848. **GRAFCET specification language for sequential function charts**. 2nd. [S.l.: s.n.], 2002.

LEFEBVRE, Dimitri; DELHERM, Catherine. Diagnosis of DES with Petri net models. **IEEE Transactions on Automation Science and Engineering**, v. 4, n. 1, p. 114–118, 2007.

MIYAGI, PE; RIASCOS, LAM. Modeling and analysis of fault-tolerant systems for machining operations based on Petri nets. **Control Engineering Practice**, v. 14, n. 4, p. 397–408, 2006.

MOREIRA, Marcos V.; BASILIO, J. C.; CABRAL, F. G. "Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems" Versus "Decentralized Failure Diagnosis of Discrete Event Systems": A Critical Appraisal. **IEEE Transactions on Automatic Control**, v. 61, n. 1, p. 178–181, 2016.

MOREIRA, Marcos 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, p. 1679–1684, 2011.

MOREIRA, Marcos V.; LESAGE, Jean-Jacques. Fault diagnosis based on identified discrete-event models. **Control Engineering Practice**, v. 91, p. 104101, 2019.

MURATA, Tadao. Petri nets: Properties, analysis and applications. **Proceedings of the IEEE**, v. 77, n. 4, p. 541–580, 1989.

NUNES, Carlos E. V.; MOREIRA, Marcos V.; ALVES, Marcos V. S.; CARVALHO, Lilian K.; BASILIO, João Carlos. Codiagnosability of networked discrete event systems subject to communication delays and intermittent loss of observation. **Discrete Event Dyn Syst**, v. 28, p. 215–246, 2018.

PROCK, J. A new technique for fault detection using Petri nets. **Automatica**, v. 27, n. 2, p. 239–245, 1991.

QIU, Wenbin; KUMAR, Ratnesh. Decentralized failure diagnosis of discrete event systems. **IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans**, v. 36, n. 2, p. 384–395, 2006.

RAN, Ning; SU, Hongye; GIUA, Alessandro; SEATZU, Carla. Codiagnosability analysis of bounded Petri nets. **IEEE Transactions on Automatic Control**, v. 63, n. 4, p. 1192–1199, 2017.

ROTH, Matthias; LESAGE, Jean-Jacques; LITZ, Lothar. The concept of residuals for fault localization in discrete event systems. **Control Engineering Practice**, v. 19, n. 9, p. 978–988, 2011.

RU, Yu; HADJICOSTIS, Christoforos N. Fault diagnosis in discrete event systems modeled by partially observed Petri nets. **Discrete Event Dynamic Systems**, Springer, v. 19, n. 4, p. 551–575, 2009.

SAMPATH, Meera; SENGUPTA, Raja; LAFORTUNE, Stephane; SINNAMOHIDEEN, Kasim; TENEKETZIS, Demosthenis C. Failure diagnosis using discrete-event models. **IEEE transactions on control systems technology**, v. 4, n. 2, p. 105–124, 1996.

SAMPATH, Meera; SENGUPTA, Raja; LAFORTUNE, Stéphane; SINNAMOHIDEEN, Kasim; TENEKETZIS, Demosthenis. Diagnosability of discrete-event systems. **IEEE Transactions on automatic control**, v. 40, n. 9, p. 1555–1575, 1995.

SANTORO, Leonardo P.M.; MOREIA, Marcos V.; BASILIO, João C. Computation of minimal diagnosis bases of Discrete-Event Systems using verifiers. **Automatica**, v. 77, p. 93–102, 2017.

SOUZA, Ryan PC de; MOREIRA, Marcos V.; LESAGE, Jean-Jacques. Fault detection of Discrete-Event Systems based on an identified timed model. **Control Engineering Practice**, v. 105, p. 104638, 2020.

YUE, Hao; XU, Shulin; ZHOU, Guangrui; HU, Hesuan; GUO, Yiyun; ZHANG, Jihui. Estimation of Least-Cost Transition Firing Sequences in Labeled Petri Nets by Using Basis Reachability Graph. **IEEE Access**, v. 7, p. 165387–165398, 2019.

ZAYTOON, Janan; LAFORTUNE, Stéphane. Overview of methods for discrete event systems. **Annual Reviews in Control**, v. 37, n. 2, p. 308–320, 2013.

ZHOU, MengChu; DICESARE, Frank. **Petri net synthesis for discrete event control of manufacturing systems**. [S.l.]: Kluwer, 1993.