COPPE
Instituto Alberto Luiz Coimbra de
Pós-Graduação e Pesquisa de Engenharia
UFRJ

DIAGNOSABILITY OF MODULAR DISCRETE EVENT SYSTEMS WITH UNOBSERVABLE COMMON EVENTS

Thiago Cabral de Souza

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

Orientador: João Carlos dos Santos Basilio

Rio de Janeiro
Abril de 2022

DIAGNOSABILITY OF MODULAR DISCRETE EVENT SYSTEMS WITH UNOBSERVABLE COMMON EVENTS

Thiago Cabral de Souza

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

Orientador: João Carlos dos Santos Basilio

Aprovada por: Prof. João Carlos dos Santos Basilio, Ph.D.
Prof. Gustavo da Silva Viana, D.Sc.
Prof. Patrícia Nascimento Pena, D.Eng.

RIO DE JANEIRO, RJ – BRASIL
ABRIL DE 2022

Souza, Thiago Cabral de

Diagnosability of modular Discrete Event systems with unobservable common events/Thiago Cabral de Souza. – Rio de Janeiro: UFRJ/COPPE, 2022.

IX, 63 p.: il.; $29,7$cm.

Orientador: João Carlos dos Santos Basilio

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2022.

Referências Bibliográficas: p. 61 – 63.

1. Discrete event systems. 2. Distributed systems. 3. Modular diagnosability. I. Basilio, João Carlos dos Santos. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

À minha avó, Sônia Cabral (In memoriam); à Pandora (In memoriam).

Agradecimentos

Agradeço e dedico essa dissertação à Sônia Cabral Pessanha (In memoriam), minha avó. Obrigado por todo o suporte e amor, obrigado por ter me incentivado na área da matemática, nada disso seria possível sem você. Obrigado por ter compartilhado todo o seu amor e por ter dividido tudo isso comigo, essa conquista é nossa. Eu te amo.

Obrigado ao meu melhor amigo, Mikael Mesquita Gualdi. Você é um irmão para mim, sou muito grato por tudo que a gente já passou e pelo que passaremos juntos. Sem você ao meu lado, esse trabalho não estaria aqui.

Obrigado à Isabella Valadares Silveira por todo o carinho e companherismo. Obrigado pelo que estamos vivendo, por estarmos um pelo outro, lado a lado nos momentos tristes e felizes, não somente nessa jornada, mas na vida.

Obrigado aos meus familiares, Rafael Cabral e Fernando Cabral pelo suporte, que me incentivaram e ajudaram a seguir esse caminho, em um momento triste para todos nós. O que fica aqui é a alegria e o nosso amor, conseguimos.

Obrigado ao meu orientador, João Carlos Basilio, que é um dos responsáveis pela conclusão deste trabalho. Obrigado por sempre acreditar em mim, por me conceder tantas oportunidades durante a pandemia. Compartilhar com você os momentos em conjunto, empolgados por estarmos fazendo e produzindo algo que amamos foi muito importante. Trabalhar ao seu lado foi e sempre será uma honra.

Obrigado aos meus amigos do Instituto Federal Fluminense, que estiveram comigo nessa caminhada. Obrigado ao Alexandre Leite por todos os seus ensinamentos, ao Paulo Victor Padrão por sua mentoria e ao Wladimir Pinheiro por ter sido um companheiro no mestrado, nas viagens semanais, matérias e trabalhos em conjunto.

Obrigado aos meus amigos e colegas do Laboratório de Controle e Automação (LCA), especialmente ao Lucas Antunes Floriano, Raphael Julio, Ryan Pitanga, Braian Igreja e ao Thiago Henrique.

Obrigado aos professores da COPPE/UFRJ, que sempre mostraram uma paixão pelo que fazem e o prazer de pensar junto e ensinar. Obrigado Celina Miraglia, Lilian Kawakami, Marcos Vicente, Gustavo Viana, Alessandro Jacoud.

Obrigado ao programa de Programa de Excelência Acadêmica - Brasil (CAPES), por ter apoiado esse trabalho.

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

DIAGNOSABILITY OF MODULAR DISCRETE EVENT SYSTEMS WITH UNOBSERVABLE COMMON EVENTS

Thiago Cabral de Souza

Abril/2022

Orientador: João Carlos dos Santos Basilio

Programa: Engenharia Elétrica

Abstract:
Previous works on modular diagnosability of discrete event systems assume that different modules can only share observable events. In this thesis, we remove this assumption and presents a general formulation for the modular diagnosability problem. The main contributions of the thesis are a necessary and sufficient condition for modular diagnosability of regular languages, and an automaton-based algorithm for modular diagnosability verification.

Resumo:
Trabalhos anteriores em diagnosticabilidade modular de sistemas modelados por eventos discretos, assumem que módulos podem somente compartilhar eventos observáveis. Nessa dissertação, essa suposição é removida e é apresentada uma formulação para o de diagnosaticabilidade modular. As principais contribuições dessa dissertação são uma condição necessária e suficiente para diagnosticabilidade modular com linguagens regulares, e um algoritmo, baseado em autômatos diagnosticadores, para verificação de diagnosticabilidade modular.

Contents

List of Figures

Chapter 1

Introduction

Since it was first introduced in the literature (LIN, 1994; SAMPATH *et al.*, 1995), fault diagnosis of discrete event systems (DES) has been an active area of research (see ZAYTOON e LAFORTUNE (2013) and the references therein). The tricky aspect of fault diagnosis, which has motivated a plethora of research works, is that a fault, when it happens, does not lead the system to an immediate halt, but may spoil a whole production, before it has been noted, or, when, not timely detected, may cause serious financial losses and harmful accidents. Saying that a fault can be diagnosed means that, after a finite number of event observations, the diagnosis system is undoubtedly sure that the fault has occurred. However, applying this idea to real DES is not so simple, and, as such, throughout these years, the two main issues in the research of fault diagnosis are: *(i)* on-line diagnosis (SAMPATH *et al.*, 1995; GENC e LAFORTUNE, 2007; RAMIREZ-TREVINO *et al.*, 2011; CABRAL *et al.*, 2015); and *(ii)* offline diagnosability analysis of the system (SAMPATH *et al.*, 1995; YOO e LAFORTUNE, 2002; QIU e KUMAR, 2006; MOREIRA *et al.*, 2011; CLAVIJO e BASILIO, 2017; VIANA e BASILIO, 2019)

Early research works considered diagnosability analysis and on-line diagnosis assuming a monolithic (also referred to as centralized or global) architecture (SAMPATH *et al.*, 1995; YOO e LAFORTUNE, 2002; SAMPATH *et al.*, 1996). The main advantage of centralized diagnosis is the conceptual simplicity and the diagnosis veracity. On the other hand, its main disadvantages are the computational complexity and the combinatorial explosion, because a monolithic diagnoser may become too large when dealing with a large-scale system.

In order to overcome the aforementioned problems associated with the monolithic approach, the architectures of decentralized diagnosis (DEBOUK *et al.*, 2000), modular diagnosis (CONTANT *et al.*, 2006) and distributed diagnosis (SU *et al.*, 2002) were proposed. Their main objective is similar, which is to achieve the same diagnosis performance with the monolithic approaches without building a monolithic diagnoser. It is meaningless to discuss which architecture is better, since each one has its own scope of use based on different assumptions and different types of models. The main differences between these architectures are as follows:

- **Decentralized architecture**. In this architecture, the input system model is still the monolithic system model. The system is partitioned into several sites, each site having full knowledge of the global model but has only local observation of the system. A local diagnoser is built based on the local observation of the whole system. One of the protocols proposed in DEBOUK *et al.* (2000) has led to a generalization of the monolithic approach, leading to the notion of codiagnosability. In that approach, local diagnosers cannot communicate directly with each other, but they provide their local decision to a coordinator, which issues the final diagnostic decision: a fault is issued when at least one of the local diagnosers is sure that the fault has occurred, and remains quiet otherwise.

- **Modular architecture**. In this architecture, the input system model is a collection of module models. Different modules can have common events, and the monolithic model can be obtained by building the parallel composition of all modules. The idea is to see if an apparently non-diagnosable monolithic system can be diagnosable (modularly) by leveraging the synchronization provided by the common events to identify the non-existing ambiguous traces responsible for the apparent non-diagnosability. If the system is found modularly diagnosable, it is enough to build a diagnoser for the module that contains the fault.

In this dissertation, we focus on the modular architecture and address the problem of modular diagnosability verification. The concept of modular diagnosability, as proposed in CONTANT *et al.* (2006), considers a DES with m modules, each module being modeled by an automaton G_k, $k \in H = \{1, 2, \ldots, m\}$, satisfying the following assumptions: *(i)* the whole systems is $G_H = \|_{k \in H} G_k$; *(ii)* for any subset $I = \{i_1, i_2, \ldots, i_n\} \subseteq H$, where $n \leq m$, the language $L(G_I)$ generated by $G_I := \|_{i_k \in I} G_{i_k}$ is live; *(iii)* the common events of different modules are observable. In CONTANT *et al.* (2006), the local diagnosability of each module is analyzed by using the diagnoser approach, and, for each module $k \in I$ that is not locally diagnosable with respect to its own set of fault events Σ_{f_k}, and observable events, there must exists an F-indeterminate cycle (SAMPATH *et al.*, 1995) in the local diagnoser G_{d_k}. The idea of CONTANT *et al.* (2006) is to perform parallel compositions of diagnoser G_{d_k} and the local diagnosers of other modules in order to verify if the F-indeterminate cycle survives. If there exists a set of local diagnosers such that, after building the parallel composition of G_{d_z} and those diagnosers, the $F-$indeterminate cycle does not survive, the system is modularly diagnosable.

Let us now analyze the computational complexity of the Modular Diagnosability Algorithm (MDA) proposed in CONTANT *et al.* (2006). Assuming that $|X_z|$ is the maximal number of states of one module and $|\Pi_{fz}|$ is the maximal number of fault classes in one module, in the worst case, the complexity of constructing the local diagnoser is $\mathcal{O}(2^{|X_z| \times 2^{|\Pi_{fz}|}})$. Notice that, in the worst case, the parallel composition of all the local diagnosers are built, and thus, the computational complexity of the approach in CONTANT *et al.* (2006) is $\mathcal{O}((2^{|X_z| \times 2^{|\Pi_{fz}|}})^m)$, or, equivalently, $\mathcal{O}(2^{m|X_z| \times 2^{|\Pi_{fz}|}})$. Thus, the worst case computational complexity is exponential, which justifies the search for a new algorithm that has polynomial computational complexity.

In MYADZELETS e PAOLI (2013) and SCHMIDT (2013), two approaches were proposed for specification-based modular diagnosability analysis, *i.e.*, when the system has a modular structure and the fault is modeled by a set of a specific language. These approaches are also applicable for event-based modular diagnosability analysis, *i.e.*, the fault of the system is modeled by a set of unobservable fault events. One problem of these two approaches is that a monolithic model is necessarily built. In MYADZELETS e PAOLI (2013), the idea is similar to the approach in CONTANT *et al.* (2006), but the verifier approach in YOO e LAFORTUNE (2002) is applied. In the worst case, the whole system must be built. Assuming that the number of states of the monolithic model is $|X_z|^m$, the number of the events is at most $|\Sigma| = | \cup_{z \in H} \Sigma_z|$, the number of fault classes of the monolithic system is $|\Pi_{fH}|$, then, the complexity of the approach in MYADZELETS e PAOLI (2013) is $\mathcal{O}(|X_z|^{2m} \times |\Sigma| \times |\Pi_{fH}|)$. In SCHMIDT (2013), the module models are simplified by using an abstraction-based technique before analyzing the modular diagnosability. In the worst case, the complexity to analyze an event-based modular diagnosability is linear in the number of states and second order polynomial in the number of transition of monolithic model. Therefore, the complexity of this approach is $\mathcal{O}(|X_z|^m \times (|X_z|^m \times |\Sigma|)^2)$, i.e., $\mathcal{O}(|X_z|^{3m} \times |\Sigma|^2)$.

More recently, LI *et al.* (2017) has proposed a modular verification algorithm based on the algorithm developed by MOREIRA *et al.* (2011). Although the computational complexity of the algorithm proposed in LI *et al.* (2017) has been found to be $\mathcal{O}(|X_z|^{m+1} \times |\Sigma|)$, thus polynomial, the algorithm still assumes, as in CONTANT *et al.* (2006), that common events between modules are observable.

In this dissertation, we remove all assumptions regarding the observability of common events and presents a general approach for modular diagnosability. The main contributions of this dissertations are as follows:

(i) we present a necessary and sufficient condition for modular diagnosability of regular languages;

(ii) we propose an automaton-based algorithm for modular diagnosability verification, and;

(iii) we develop a diagnoser that relies on the observation of the events of faulty module.

The methodology proposed here relies on the test automaton proposed in VIANA e BASILIO (2019) and therefore has the advantage of searching for strongly connected components, which is polynomial, but requires the construction of a diagnoser for the faulty module, therefore it is exponential on the state size of the faulty module; see also CLAVIJO e BASILIO (2017) for a comparison between the average state sizes of diagnosers and verifiers.

This thesis is structured as follows. In Chapter 1, we present some preliminary notions used throughout this thesis. In Chapter 2, we presents some background on Discrete Event System necessary for the remaining chapter. In Chapter 3, we present a new approach for modular diagnosability analysis. Finally, in Chapter 4, we present the conclusion and outline some future perspectives for this work.

Chapter 2

Theoretical Background

This chapter intends to present a brief review of Discrete Event Systems (DES) theory, and to recall the basic concepts for the reader to have a better understanding of the dissertation. Thus, we present the neccessary background on DES, which is based on CASSANDRAS e LAFORTUNE (2009), and the theoretical background on fault detection and modular diagnosability. Chapter 2 is structured as follows. In Section 2.1 we present the basic background on Discrete Event Systems. In Section 2.2, we show and define different types of automata, and operations regarding them. In Section 2.3, we present some background on fault diagnosis of discrete event systems. Finally, in Section 2.4, we present diagnosability verification and its relation in different architectures.

2.1 Discrete Event Systems

Discrete Event Systems (DES) are dynamical systems whose dynamic evolves on asynchronous occurrence of events over time. This implies that the system states, usually denoted by X, can be described by either numerical values belonging to set $\mathbb{N}$ or $\mathbb{Z}$ e.g. $X = \{0, 1, 2, ...\}$, or even symbolical values, such $X = \{open, closed, stuck\}$. On the other hand, the system events are described by instantaneous actions that are responsible for triggering the evolution of the system, passing from one state to the next one, or to a collection to states. The system behavior is driven by the ocurrence of events. All possible sequences of the events that can be generated by a given DES describe the language of this system, which is defined over a set of events (alphabet) of the system. These concepts and more, are described in detail in the next sections.

2.1.1 Language

One formal way to study the logical behavior of a DES is through the theories of language and automata. The starting point is the fact that any DES has an associated event set Σ, whose cardinality, denoted as $|\Sigma|$, is assumed to be finite. The set Σ is thought of as the "alphabet" of a language and event sequences are thought of as "words" in that language. In the literature, the sequences of events σ of Σ are also called traces, strings or sequences; the term sequences will be adopted throughout the thesis. The length of a sequence s is the number of events of s, and is denoted as $|s|$. The set of all sequences formed with the events of Σ is called Kleene-closure of Σ and is denoted as Σ^*. By definition, ε is a sequence that does not contain events, being refered to as the empty sequence, and $|\varepsilon| = 0$. In addition, $\sigma \in s$ means that event σ belongs to sequence s. We denote as 2^Σ the power set of Σ or, namely the set that contains all the subsets of Σ; including the empty set $\varnothing$ and Σ itself.

Definition 1 (Language). *A language L defined over an event set Σ is a set of sequences with finite length formed with events of Σ.*

From Definition 1, since a language of a DES belongs to the set of all the finite length sequences of elements of Σ, we may say that $L \subseteq \Sigma^*$. Notice that, $\varnothing$, Σ and Σ^* are also languages defined over Σ. In addition, ε may be an element of L, although $\varepsilon \notin \Sigma$.

Example 2.1. *Let $\Sigma = \{a, b, c\}$ be a set of events. We may define over Σ, the following languages $L_1 = \{a, ab, ac\}$ consists of only three sequences; language $L_2 = \{\varepsilon, a, aa, ab, abc\}$ which contains five sequences; and $L_3 = \{s \in \Sigma^* : |s| \leq 2\} = \{\varepsilon, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc\}$.*

2.1.2 Operations on Languages

The usual set operations, such as union, intersection, difference, and complement with respect to another language, are applicable to languages, since languages are sets. In addition, the following operations can be defined for languages: concatenation, Kleene-closure, prefix-closure, post-language, natural projection and inverse projection.

Definition 2. *(Concatenation) Let $L_a, L_b \subseteq \Sigma^*$. Then, the contatenation L_aL_b is defined as:*

$$L_aL_b = \{s \in \Sigma^* : (s = s_as_b), (s_a \in L_a) \text{ and } (s_b \in L_b)\}.$$

According to Definition 2 sequence s is in L_aL_b if it is formed by the concatenation of a sequence $s_a \in L_a$ and $s_b \in L_b$ in this order.

Definition 3 (Kleene-closure)**.** *Let $L \subseteq \Sigma^*$. Then, the Kleene-closure of L, denoted by L^*, is defined as:*

$$L^* := \{\varepsilon\} \cup L \cup LL \cup LLL \dots$$

.

Before we introduce some operations on language we will define the prefix of a sequence. Given a sequence $s \in \Sigma^*$, we say that a sequence $t \in \Sigma^*$ is prefix of a sequence $s \in \Sigma^*$ if there exists a sequence $v \in \Sigma^*$ such that $tv = s$. Also, both s and ε are prefixes of s.

Definition 4 (Prefix-closure)**.** *The prefix-closure of a language L consists of all prefixes of all sequences in L. The prefix-closure of L, denoted as $\bar{L}$, is defined as:*

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

The language L is considered prefix closed if $L = \bar{L}$. Thus, the language L is prefix-closed if all prefixes of every sequence in L are also elements of L.

Example 2.2. *Let* $\Sigma = \{a, b, c\}$*, and consider languages* $L_1 = \{\varepsilon, a, ab, abc\}$ *and* $L_2 = \{b\}$ *defined over* Σ*. Note that* L_2 *is not prefix-closed, since* $\bar{L}_2 = \{\varepsilon, b\} \neq L_2$*, but* L_1 *is prefixed-closed, since* $L_1 = \bar{L}_1$*.*

Definition 5 (Post-Language)**.** *Let* $L \subseteq \Sigma^*$*, and* $s \in L$*. Then the post-language of* L *after* s*, denoted by* L/s*, in the language:*

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

By definition, if $s \notin L$*, then* $L/s = \varnothing$*.*

Another type of operation performed on sequences and languages is the so-called natural projection, or simply projection.

Definition 6 (Projection)**.** *The projection takes a sequence formed from the larger event set* Σ_l *and erases the events in it that do not belong to the smaller event set* Σ_s*. We start by defining the projection* P *for sequences:*

$$P_{\Sigma_l,\Sigma_s} : \Sigma_l^* \to \Sigma_s^*.$$

where

$$P_{\Sigma_l,\Sigma_s}(\varepsilon) := \varepsilon$$

$$P_{\Sigma_l,\Sigma_s}(e) := \begin{cases} e & \textit{if } e \in \Sigma_s \\ \varepsilon & \textit{if } e \notin \Sigma_s \end{cases}$$

$$P_{\Sigma_l,\Sigma_s}(se) := P_{\Sigma_l,\Sigma_s}(s) P_{\Sigma_l,\Sigma_s}(e) \quad \textit{(for } s \in \Sigma_l^*, e \in \Sigma_l.\textit{)}$$

As can be seen from the Definition 6, the projection operation takes a sequence formed from the larger event set (Σ_l) and erases events in it that do not belong to the smaller event set (Σ_s). We will also be working with the corresponding inverse map, $P^{-1}_{\Sigma_s,\Sigma_l} : \Sigma_s^* \to 2^{\Sigma^*}$, defined as the following.

$$P^{-1}(t) := \{s \in \Sigma^* : P(s) = t\}.$$

The projection P_{Σ_l,Σ_s} and its inverse $P^{-1}_{\Sigma_s,\Sigma_l}$ are extended to languages by simply applying them to all sequences in the language. Let $L \subseteq \sigma_\ell^*$

$$P_{\Sigma_l,\Sigma_s}(L) = \{t \in \Sigma_s^* : (\exists s \in L)[P(s) = t]\}.$$

For $L_s \subseteq \Sigma_s^*$

$$P^{-1}_{\Sigma_l,\Sigma_s}(L_s) : \{s \in \Sigma^* : (\exists t \in L_s)[P(s) = t]\}.$$

In order to demonstrate these operations, consider the following example.

Example 2.3. *Let us consider the set of events $\Sigma = \{a, b, c\}$ and the language $L_1 = \{\varepsilon, b\}$ and the language $L_2 = \{a, bb, c\}$. :*

$$L_1^* = \{\varepsilon, b, bb, bbb, ...\}$$

$$L_1L_2 = \{a, bb, c, ba, bbb, bc\}$$

$$L_2L_1 = \{a, ab, bb, bbb, c, cb\}$$

If we define the projection $P_{\Sigma_l,\Sigma_s} : \Sigma_l^* \to \Sigma_s^*$, such that $\Sigma_l = \{a, b, c\}$ and $\Sigma_s = \{a, b\}$, then:

$$P(abc) = ab$$

$$P^{-1}(ab) = \{c\}^*\{a\}\{c\}^*\{b\}\{c\}^*$$

$$P^{-1}(\varepsilon) = \{c\}^*$$

$$P(L_2) = \{a, bb, \varepsilon\}$$

$$P^{-1}(P(L_2)) = \{\{c\}^*a\{c\}^*, \{c\}^*b\{c\}^*b\{c\}^*, \{c\}\}$$

2.2 Automata

Automata are devices that are capable of representing a language according to well-define rules, using a state transition structure, that determines events can occur at each state of the system. The simplest representation of automaton is its directed graph, or state transition diagram. In the following, we formally define a deterministic automata:

Definition 7 (Deterministic Automata). *A deterministic automaton, denoted by G, is a six-tuple:*

$$G = (X, \Sigma, f, \Gamma, x_0, X_m),$$

where:

- X is the set of states of the automata;
- Σ is the finite set of events associated with G;
- $f : X \times \Sigma \to X$ is the state transition function of the states, $f(x, \sigma) = y$ means that there is a transition rotulates by event σ from state x to state y; generally, f is a partial function in its own domain;
- x_0 is the initial state of the system;
- $\Gamma : X \to 2^{\Sigma}$ such that $\Gamma(x) = \{\sigma \in \Sigma : f(x, \sigma)!\}$, where $f(x, \sigma)!$ denotes that σ is defined in x, i.e., there exists $y \in X$ such that $f(x, \sigma) = y$;
- $X_m \subseteq X$ is the set of marked states.

The transition function, f is a partial function in its own domain, as can be seen in the following example

Example 2.4. *Let G be an automaton whose state transition diagram is depicted in Figure 2.1. Based on the diagram, we have that:*

- $X = \{x_0, x_1, x_2\}$;
- $\Sigma = \{a, b, c, d\}$;

- The state transition function of G is given by $f(x_0,a) = f(x_2,d) = x_0$, $f(x_0,b) = f(x_1,b) = x_1$, $f(x_1,a) = f(x_1,c) = f(x_2,b) = x_2$;
- The feasible event sets of each state are given by $\Gamma(x_0) = \{a,b\}$, $\Gamma(x_1) = \{a,b,c\}$,$\Gamma(x_2) = \{b,d\}$;
- The initial state of G is $x_0 = \{x_0\}$;
- The set of marked states of G is $X_m = \{x_0,x_1\}$

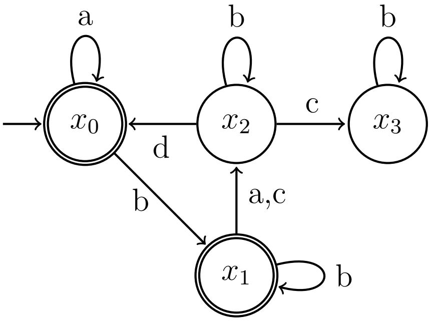


Figure 2.1: State transition diagram for Example 2.4

The transition function f has its domain extended from $X \times \Sigma$ to $X \times \Sigma^*$ as follows:

- $f(x,\varepsilon) = x$;
- $f(x,s\sigma) = f(f(x,s),\sigma)(\forall x \in X)(\forall s \in \Sigma^*)(\forall \sigma \in \Sigma) \quad [(f(x,s) = z)(f(z,\sigma)!)]$.

Thus, the language generated and marked by an automaton, can be define as follows.

Definition 8 (Generated and marked language)**.** *The language generated by automaton G is defined as $L(G) := \{s \in \Sigma^* : f(x_0,s)!\}$ and the language marked by automaton G is defined as $L_m(G) := \{s \in L(G) : f(x_0,s) \in X_m\}$.*

Note that $L_m(G)$ will always be a subset of $L(G)$, since $L_m(G)$ is composed by all sequences s such that $f(x_0,s) \in X_m$, and $L(G)$ is prefix-closed. Languages $L(G)$ and $L_m(G)$ always satisfy the following relation:

$$L_m(G) \subseteq \overline{L_m(G)} \subseteq L(G).$$

2.2.1 Operations on Automata

In order to analyze DES modeled by automata, we first need to review the set of operations capable of properly modifying the state transition diagram of an automaton, usually refered to as unary operation: accessibility or accessible part. The product and the parallel compositions, which are fundamental to obtain the fault diagnosers, that we intend to propose in this work will also be presented.

Unary Operations

- **Accessible Part**

A state $x \in X$ of an automaton G is accessible if $\exists s \in \Sigma^*$ such that $f(x_0, s) = x$. From the definitions of L(G) and $L_m(G)$, one may notice that we can erase all the states of G that are not accessible or reachable, starting from x_0, by some sequence in $L(G)$, without affecting not only the generated language of G, but also its marked language. When we remove a state, we also remove all transitions linked to that state. The accessible part of an automaton is defined as:

Definition 9 (Accessible Part). *Let $G = (X, \Sigma, f, x_0, X_m)$ be an automaton. The accessible parte or accessibility of G, denoted by $Ac(G)$, is defined as*

$$Ac(G) := (X_{ac}, \Sigma, f_{ac}, x_0, X_{ac,m}).$$

where:

(i) $X_{ac} = \{x \in X : (\exists s \in \Sigma^*)[f(x_0, s) = x]\}$;

(ii) $X_{ac,m} = X_m \cap X_{ac}$;

(iii) $f_{ac} = f|_{X_{ac} \times \Sigma \to X_{ac}}$

Item (iii) means that we are restricting f to the smaller domain of the accessible states X_{ac}. Note that the event set of $A_c(G)$ remains equal to the original event set of G, even if some event of the set no longer appears in the state transition diagram of $A_c(G)$. Thus, the accessible does not changes $L(G)$ and $L_m(G)$.

- **Coaccessible Part**

A state $x \in X$ of an automata is coaccessible if $\exists s \in \Sigma^*$ such that $f(x,s) \in X_m$. Otherwise, x is a non-coaccessible state. We denote the operation of deleting all the states of G that are not coaccessible by $CoAc(G)$, where $CoAc$ stands for "coaccessible" part. The automaton denoted by $CoAc(G)$, is defined as

$$CoAc(G) := (X_{coac}, \Sigma, f_{coac}, x_{0,coac}, X_m),$$

where:

(i) $X_{coac} = \{x \in X : (\exists s \in \Sigma^*)[f(x,s) \in X_m]\}$,

(ii) $x_{0,coac} := \begin{cases} x_0 & \text{if, } x_0 \in X_{coac}, \\ \text{undefined,} & \text{otherwise,} \end{cases}$

(iii) $f_{coac} = f|_{X_{coac} \times \to X_{coac}}$.

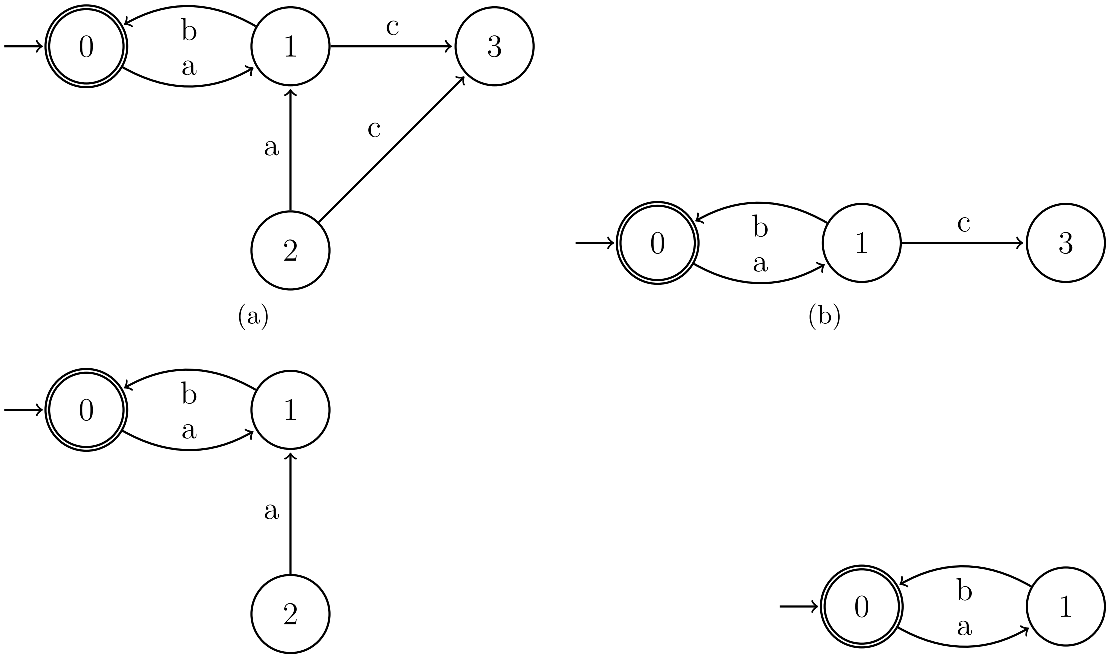
Notice that, the coaccessible part of an automaton may generate an different language, but the marked language remains the same.

- **Trim Operation**

An automaton that is both accessible and coaccessible is said to be trim. We define the Trim operation to be

$$Trim(G) := CoAc[Ac(G)] = Ac[CoAc(G)].$$

Example 2.5. *Let the state transition diagram of automaton G_1 depicted in Figure 2.2a. To obtain Ac(G), it suffices to delete state 2 and the transitions $(f(2,c) : [c \in \Gamma(2)])$; the resulting automaton is depicted in Figure 2.2b. In order to obtain CoAc(G), we need to identify the states of G that are not coaccessible to the marked state 0, which is state 3. We, then, delete this state and the transition labeled by event c and a Γ that satisfy: $c \in \Gamma(3)$ and $c \in \Gamma(x) : f(x,\Gamma) = 3$, to obtain CoAc(G)* *depicted in Figure 2.2c. Finally the trim automaton, Trim(G), is depicted in Figure 2.2d.*



(a) (b) (c) (d)

Figure 2.2: Automaton G (a); Ac(G) (b); CoAc(G) (c), and, Trim(G) (d) of Example 2.5

Operations with two or more automata

- **Product Composition**

The product composition, denoted by ×, is a composition of two automata that allows only the occurrence of events that are common to both. The following definition describes this operation mathematically [13].

Definition 10 (Product composition). *Consider the following automata:*

$$G_1 = (X_1, \Sigma_1, f_1, x_{0,1}, x_{m_1}),$$

$$G_2 = (X_2, \Sigma_2, f_2, x_{0,2}, x_{m_2}).$$

The product composition between G_1 and G_2 leads to the following automaton:

$$G_1 \times G_2 = Ac(X_1 \times X_2, \Sigma_1 \times \Sigma_2, f_{1\times 2}, \Gamma_{1x2}, (x_{0,1}, x_{0,2}), X_{m_1} \times X_{m_2}).$$

where $f_{1\times 2}$, and $\Gamma_{1\times 2}$ is defined as:

(i) $f((x_1, x_2), \sigma) := \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)), & \text{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2), \\ \text{undefined}, & \text{otherwise;} \end{cases}$

(ii) $\Gamma_{1\times 2}(x_1, x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2)$ is the feasible event function of $G_1 \times G_2$,

Considering the righthand side of the definition 10 of $G_1 \times G_2$, it can be noted that we are only interested in the accessible part of the automaton. In other words, an event only occurs in $G_1 \times G_2$ if it occurs in both G_1 and G_1. In the product operation, the transitions of two automata should always be synchronized with a common event. Then, we can say that both generated and marked languages by the product $G_1 \times G_2$ can be given by:

$$L(G_1 \times G_2) = L(G_1) \cap L(G_2),$$

$$L_m(G_1 \times G_2) = L_m(G_1) \cap L_m(G_2).$$

- **Parallel Composition**

As stated in [13], the more commonly method to building the complete model of a system from its individual components is through parallel (or synchronous) composition of automata, being each automaton a local component (or subsystem) of the global system. The definition of parallel composition is as follows.

Definition 11 (Parallel Composition)**.** *Consider automata* $G_1 = (X_1, \Sigma_1, f_1, x_{0,1}, x_{m_1}), G_2 = (X_2, \Sigma_2, f_2, x_{0,2}, x_{m_2})$. *The product composition be-*

tween G_1 and G_2 will be given by the following automaton:

$$G_1 \parallel G_2 = Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\|2}, \Gamma_{1\|2}, (x_{0,1}, x_{0,2}), X_{m_1} \times X_{m_2}),$$

where $f_{1\|2}$ is defined as follows:

$$\textit{(i)}\ \ f((x_1,x_2),\sigma) := \begin{cases} (f_1(x_1,\sigma), f_2(x_2,\sigma)), & \textit{if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2), \\ (f_1(x_1,\sigma), x_2), & \textit{if } \sigma \in \Gamma_1(x_1) \setminus \Sigma_2, \\ (x_1, f_2(x_2,\sigma)), & \textit{if } \sigma \in \Gamma_2(x_2) \setminus \Sigma_1, \\ \textit{undefined,} & \textit{otherwise.} \end{cases}$$

It can be noted that a common event of the automata G_1 and G_2 can only occur when the event is in both active event sets of the states that compose, the new state of $G_1 \parallel G_2$. Private events, i.e., $\sigma \in (\Sigma_1 \setminus \Sigma_2) \cup (\Sigma_2 \setminus \Sigma_1)$, are able to be executed as long as possible. Therefore, the parallel composition synchronizes only common events to both G1 and G2. In order to define the generated and marked languages of automaton $G_1 \| G_2$, let $\Sigma = \Sigma_1 \cup \Sigma_2$ and define the following projection:

$$P_{\Sigma,\Sigma_i} : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_i^* \text{ for } i = 1,2.$$

Considering the use of projections, we now can obtain the resulting languages of the parallel composition between G_1 and G_2 that are given by:

$$L(G_1 \| G_2) = P^{-1}_{\Sigma,\Sigma_1}[L(G_1)] \cap P^{-1}_{\Sigma,\Sigma_2}[L(G_2)],$$

$$L_m(G_1 \| G_2) = P^{-1}_{\Sigma,\Sigma_1}[L_m(G_1)] \cap P^{-1}_{\Sigma,\Sigma_2}[L_m(G_2)].$$

Example 2.6. *Consider automaton G_1 and G_2 depicted in Figures 2.3a and 2.3b, with $\Sigma_1 = \{a,b,c\}$ and $\Sigma_2 = \{a,c\}$. As can be seen in Figure 2.4a the product composition didn't allow the event $\sigma = \{b\}$ to occur, since $b \notin \Sigma_2$; however in the parallel composition depicted in Figure 2.4b, the transition from state* $(0,0)$ *to* $(2,0)$ *is defined.*

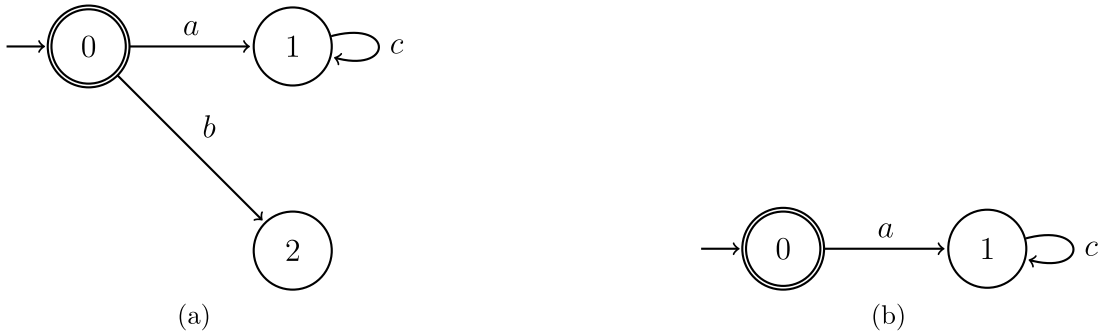


Figure 2.3: Automaton G_1 (a) and Automaton G_1 (b) of Example 2.6.

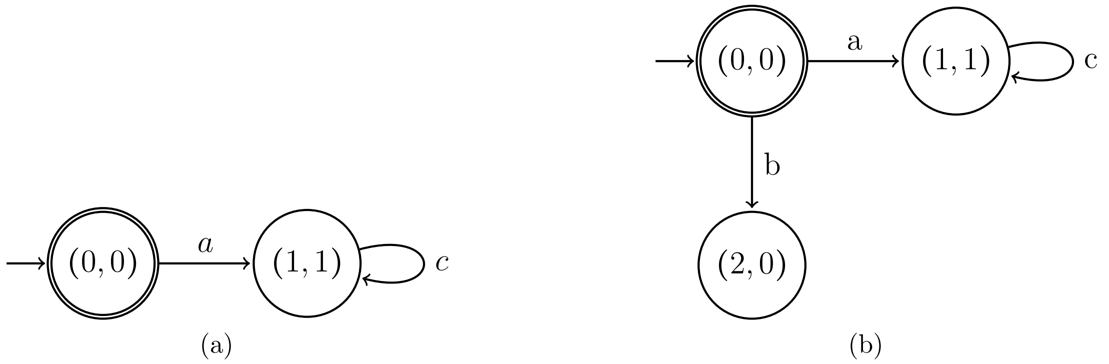


Figure 2.4: Automaton $G_1 \times G_2$ (a) and Automaton $G_1 || G_2$ (b) of Example 2.6.

2.2.2 Nondeterministic Automata

A nondeterministic automaton, denoted by G_{nd}, is a six-tuple $G_{nd} = (X, \Sigma_{nd}, f_{nd}, \Gamma_{nd}, X_0, X_m)$, where the elements of G_{nd} have the same meaning as in the deterministic automaton G, with the exception that the transition function can be nondeterministic, i.e., (i) Σ is a set of events , (ii) $\varepsilon \in \Sigma_{nd}$, (iii) $f_{nd} : X \times \Sigma_{nd} \rightarrow 2^X$; and the initial state can be defined as a set $x_0 \subseteq X$.

In order to define languages generated and marked by a nondeterministic automaton, it is necessary to extend f_{nd} to the domain $X \times \Sigma^*$, resulting in the extended transition function f^e_{nd}. To this end let $\varepsilon R(x)$ denote the ε-reach of a state x, i.e., the set of states reached from x by following transitions labeled with the empty string ε, including state x. The ε-reach can be extended to a set of states $B \subseteq X$ as

$$\varepsilon R(B) = \cup_{x \in B} \varepsilon R(x).$$

The extended nondeterministic transition function $f^e_{nd} : X \times \Sigma^* \rightarrow 2^X$, can be defined recursively as follows: $f^e_{nd}(x,\varepsilon) = \varepsilon R(x)$, and $f^e_{nd}(x,s\sigma) = \varepsilon R(x)[\{z : z \in f_{nd}(y,\sigma) : [(y \in X) \wedge (y \in f^e_{nd}(x,s))]$. Thus, the language generated by G_{nd} can be defined as:

$$L(G_{nd}) = \{s \in \Sigma^* : (\exists x \in x_0)[f^e_{nd}(x,s)!]\},$$

and the language marked by G_{nd} can be defined as:

$$L_m(G_{nd}) = \{s \in \Sigma^* : (\exists x \in x_0)[f^e_{nd}(x,s) \cap X_m \neq \varnothing]\}.$$

Example 2.7. *Consider the state transition diagram of the nondeterministic automaton $G_{1_{nd}}$ depicted in Figure 2.5:*

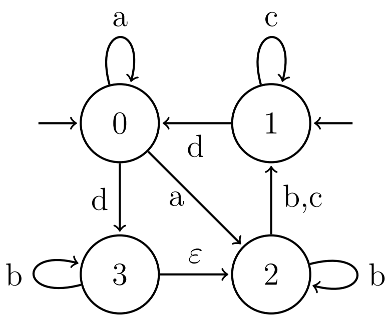


Figure 2.5: State transition diagram of Example 2.5.

The nondeterministic automaton $G_{1_{nd}}$ depicted in Figure 2.5, has two initial states $X_0 = \{0,1\}$. Note that the transition function assumes values in 2^x, for $x \in X$, for example $f_{nd}(0,a) = \{0,2\}$, $f_{nd}(2,b) = \{1,2\}$; also $f_{nd}(3,b\varepsilon b) = f_{nd}(3,bb) = \{3,2,1\}$ which suggests an uncertainty in the dynamic evolution of the system.

2.2.3 Deterministic Automata With Unobservable Events

In some cases, there might exist some events in a given system whose occurrence cannot be recorded by a sensor or because the event occurs at a remote location but is not communicated to the site where they must be processed, in this case, the system is said to be a partially-observed DES. An assumption needed for a deterministic automata nth unobservable events, is to identify which events are observable, and those that are not. To this end, we partition the set of events as:

$$\Sigma = \Sigma_o \ \dot{\cup} \ \Sigma_{uo},$$

where:

- Σ_o is the set of observable events of the system;

- Σ_{uo} is the set of unobservable events of the system.

We can build from the automaton with unobservable an automaton, with only observable events, called observer, which describes the behavior of G, denoted as $Obs(G, \Sigma_o)$. The construction of $Obs(G, \Sigma_o)$ requires the notion of unobservable reach of an state $x \in X$, which can be defined as:

$$UR(x, \Sigma_{uo}) = \{y \in X : (\exists t \in \Sigma_{uo}^*)[f(x,t) = y)]\}.$$

From the above definition, it is clear that $x \in UR(x, \Sigma_{uo})$. The unobservable reach can be extended to a set of states $A \in 2^X$ as:

$$UR(A) = \bigcup_{x \in A} UR(x),$$

with that in mind, the observer of G, $Obs(G, \Sigma_o)$, can be defined as follows:

$$Obs(G, \Sigma_o) = (X_{obs}, \Sigma_o, f_{obs}, \Gamma_{obs}, x_{0_{obs}}, x_{m_{obs}}),$$

where:

- $X_{obs} \in 2^X$;

- $f_{obs}(x_{obs}, \sigma) = UR(\{y \in X : (\exists x \in x_{obs})[f(x,\sigma) = y]\}, \Sigma_{uo})$;

- $\Gamma_{obs}(x_{obs}) = \bigcup_{x \in x_{obs}} \Gamma(x)$;

- $X_{m_{obs}} = \{x_{obs} \in X_{obs} : x_{obs} \cap X_m \neq \varnothing\}$.

Algorithm 1: Construction of automaton $Obs(G;\Sigma_o)$

```
    Input: $G = (X, \Sigma, f, \Gamma, x_0, x_m), \Sigma_o$
    Output: $Obs(G, \Sigma_o) = (X_{obs}, \Sigma_o, f_{obs}, \Gamma_{obs}, x_{0_{obs}}, x_{m_{obs}})$
STEP 1: Set:
    STEP 1.1: $\Sigma_{uo} := \Sigma \setminus \Sigma_o$
    STEP 1.2: $x_{0_{obs}} := UR(x_0, \Sigma_{uo})$
    STEP 1.3: $X_{obs} := \{x_{0_{obs}}\}$
    STEP 1.4: $\tilde{X}_{obs} := X_{obs}$
STEP 2 Set:
    STEP 2.1 $\hat{X}_{obs} := \tilde{X}_o bs$
    STEP 2.2 $\tilde{X}_{obs} := \varnothing$
STEP 3: For $B \in \hat{X}_{obs}$
    STEP 3.1 $\Gamma_{obs}(B) = \bigcup_{x \in B} \Gamma(x)) \cap \Sigma_0$
    STEP 3.2 For $\sigma \in \Gamma_{obs}(B)$:
        STEP 3.2.1 $f_{obs}(B, \sigma) := UR(\{x \in X : (\forall y \in B)[x = f(y, \sigma)]\}, \Sigma_{uo})$
        STEP 3.2.2 $\tilde{X}_{obs} := \tilde{X}_{obs} \cup f_{obs}(B, \sigma)$
STEP 4 Set $X_{obs} := X_{obs} \cup \tilde{X_{obs}}$
STEP 5 Repeat Step 2 to 4 until the entire accessible part of $Obs(G;\Sigma_o)$ has been
       constructed
STEP 6 Set $X_{m_{obs}} := \{B \in X_{obs} : B \cap X_m \neq \varnothing\}$
```

The construction of $Obs(G, \Sigma_0)$ can be made in accordance with Algorithm 1.

Using Algorithm 1, it can be concluded that the resulting language generated by $Obs(G, \Sigma_o)$ is:

$$L(Obs(G, \Sigma_o)) = P_{\Sigma, \Sigma_o}[L(G)].$$

2.2.4 Strongly Connected Components

Definition 12 (Strongly Connected Components)**.** *strongly connected component of an automaton $G_1 = (X, \Sigma, f, \Gamma, x_0, x_m)$ is a maximal set of states $X_{scc} \subseteq X$ such that for every pair of states $u, v \in X_{scc}$,there is a path formed by events in Σ from u*

to v and from v to u; that is, every states u and v in X_{scc} are reachable from each other, and X_{scc} is maximal.

The search for strongly connected components has a simpler computational complexity than the search for cycles, since the complexity of the search for SSCs is $O(|n| + |E|)$ (CORMEN *et al.*, 2009), where E is the number of transitions; meanwhile, the search for cycles is, in the worst case, worse than exponencial, $\Sigma_{n=1}^{n-1} = \binom{n-i+1}{n}(n-1)!$.

2.3 Fault Diagnosis of Discrete Event Systems

In many applications where the system model contains unobservable events, we may be interested in determining if certain unobservable events could have occurred or must have occurred in the sequence of events executed by the system. If these unobservable events of interest model faults of system components, then knowing that one of these events has occurred is very important when monitoring the performance of the system. The main objective in the process of detection and fault diagnosis is to identify the cause of poor system functioning, i.e., the occurrence of these unobservable events (SAMPATH *et al.*, 1995).

2.3.1 Diagnosability of DES

Diagnosability is a property of a system related with the capacity of the language being able to distinguish faulty sequences, sequences that contains at least one fault event, from normal ones. In order to define language diagnosability (SAMPATH *et al.*, 1995, 1996), the following assumptions are made: (i) Language L is live, i.e., $\Gamma(x) \neq \varnothing$, $\forall x \in X$; (ii) let $\Sigma_f \subseteq \Sigma_{uo}$ be a set of events associated with failures of the system; (iii) There exists only one fault event, i.e., $\Sigma_f = \{\sigma_f\}$.

The third assumption, (iii), has been made for the sake of simplicity, since methods developed regarding diagnosability are the same applied to one fault event. In order to understand and define the concept of fault behavior, the definition of the fault-free behavior, which is presented as follows;

Definition 13 (Fault-free behavior)**.** *Let $L(G) = L$ be the language generated by automaton G and L_N the prefix-closed language formed by all the sequences of L that do not have any fault event from the set Σ_f. Then, the fault-free behavior of the system given by G, with respect to $\Sigma_f = \{\sigma_f\}$, will be modeled by subautomaton of G, G_N, that generates language L_N.*

The definition of diagnosabiliy is as follows (SAMPATH *et al.*, 1995):

Definition 14 (Diagnosability)**.** *Let L be the live and prefix-closed language generated by the system, and $L_N \in L$ be the fault-free language of L. $P_o = P_{\Sigma,\Sigma_o} = \Sigma^* \rightarrow \Sigma_o^*$ be a projection operation. Then, L is said to be diagnosable with respect to projection P_o and Σ_f, if*

$$(\exists z \in \mathbb{N})(\forall s \in L \setminus L_N)(\forall st \in L \setminus L_N, |t| \geq z) \Rightarrow (\forall \omega \in P_o^{-1}(P_o(st)) \cap L, \omega \in L \setminus L_N).$$

According to Definition 14, L is not diagnosable with respect to P_o and Σ_f if, and only if, there exists an arbitrarily long length faulty sequence st with the same observation than a fault-free sequence ω in L_N, i.e., $P_o(st) = Po(\omega)$.

2.3.2 Diagnoser Automaton

A diagnoser, in the context of DES, is an automaton that performs the detection of the fault event of a diagnosable language. In order to build a diagnoser, let A_ℓ be the automaton depicted in Figure 2.6, which can be defined as follows:

$$A_\ell = (X_\ell, \Sigma_f, f_\ell, x_{0,\ell}),$$

where: $X_\ell = \{N, Y\}$, $x_{0,\ell} = N$, $f_\ell(N, \sigma_f) = Y$, and $f_\ell(Y, \sigma_f) = Y$.

Then, the diagnoser automaton is obtained as follows:

$$G_d = Obs(G_\ell, \Sigma_o) = Obs(G \parallel A_\ell, \Sigma_o) = (X_d, \Sigma_o, f_d, x_{o_d}).$$

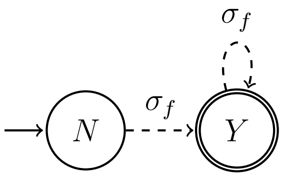


Figure 2.6: Label automata A_ℓ.

From the construction of the diagnoser, which is an observer, it can be concluded that $L(G_d) = P_{\Sigma,\Sigma_o}(L(G \parallel A_\ell)) = P_{\Sigma,\Sigma_o}(P^{-1}_{\Sigma,\Sigma}(L(G)) \cap P^{-1}_{\Sigma,\Sigma_f}(L(A_\ell))) = P_{\Sigma,\Sigma_o}(L(G) \cap \Sigma^*) = P_{\Sigma,\Sigma_o}(L(G))$. In addition, notice that all states of G_d have the form $x_d = \{(x_1, \ell_1), \dots, (x_n, \ell_n)\}$, and so, regarding fault indication, they can be classified as:

- Faulty state, if $\ell_i = Y$, for $i = 1, \dots, n$
- Normal state, if $\ell_i = N$, for $i = 1, \dots, n$
- Uncertain state, if $(\exists i \neq j)[(\ell_i = N) \wedge (\ell_j = Y)]$

When the diagnoser is in a faulty state, it is certain that a fault has occurred. On the other hand, if the diagnoser is in a normal state, it is sure that the fault has not occurred. However, if the diagnoser is in a uncertain state, it is not sure if the fault event has occurred or not. In order to better understand further concepts about fault diagnosis of discrete event systems, some definitions are required as follows:

Definition 15 (Path). *A path in G is a sequence $(x_1, \sigma_1, x_2, \dots, x_n)$, where $x_i \in X$, $\sigma_i \in \Sigma$, and $f(x_i, \sigma_1) = x_{i+1}$, $i = 1, 2, \dots, n-1$.*

Definition 16 (Cycle). *A cycle of G is the set formed of the states of a cyclic path $(x_k, \sigma_1, x_{k+1}, \sigma_2, \dots, \sigma_1, x_{k+1})$, where $x_{k+1} = x_k$.*

Definition 17 (Indeterminate Cycle). *An indeterminate cycle of G_d is a set of states $\{x_{d_1}, x_{d_2}, \dots, x_{d_p}\} \subseteq X_d$ formed of uncertain states, satisfying the following conditions:*

(i) $x_{d_1}, x_{d_w}, \dots, x_{d_p}$ form a cycle in G_d;

(ii) for all $\ell \in \{1, 2, \dots, p\}$, there exists states $(x_\ell^{k_\ell}, Y), (\tilde{x}_\ell^{r_\ell}, Y)$ in x_{d_ℓ}, with $x_\ell^{k_\ell}$ not necessarily distinct from $\tilde{x}_\ell^{r_\ell}$, where; $k_\ell = 1, 2, \dots, m_\ell$ and $r_\ell = 1, 2, \dots, \tilde{m}_l$ in such a way that the sequence of states $\{x_\ell^{k_\ell}\}, \ell = 1, 2, \ldots, p,\ k_\ell = 1, 2m \ldots, m_\ell$ and $\{\tilde{x}_\ell^{r_\ell}\}$, $\ell = 1, 2, \ldots, p$, $r_\ell = 1, 2, \ldots, \tilde{m}_\ell$ form cycles in G;

(iii) there exist two sequences $s = s_1 s_2 \ldots s_p \in \Sigma^*$ and $\tilde{s} = \tilde{s}_1 \tilde{s}_2 \ldots \tilde{s}_p \in \Sigma^*$, such that $P_o(s) = P_o(\tilde{s})$, where $s_\ell = \sigma_{\ell,1}\sigma_{\ell,2} \ldots \sigma_{\ell,m_\ell - 1}$, $f(x_\ell^j, \sigma_{\ell,j}) = x_\ell^{j+1}$, $j = 1, 2, \ldots, m_\ell - 1$, $f(x_\ell^{m_\ell}, \sigma_{\ell+1,0}) = x_{\ell+1}^1$, and $f(x_p^{m_p}, \sigma_{\ell,0}) = x_1^1$, and similarly for $\tilde{s}_\ell$.

In SAMPATH *et al.* (1995), the authors make the following assumptions on the system under investigation:

H1. The language L generated by G is live. This means that there is at least one transition defined at each state $x \in X$, i.e., the system cannot reach a point at which no event ocurrence is possible.

H2. There does not exist in G any cycle of unobservable events. In SAMPATH *et al.* (1995), a necessary and sufficient condition for diagnosability of regular languages is presented as follows.

Theorem 2.1. *(SAMPATH* et al.*, 1995) The language L generated by automaton G is diagnosable with respect to projection P_o and $\Sigma_f = \{\sigma_f\}$ if, and only if, its diagnoser G_d has no indeterminate cycles.*

According to Theorem 2.1 if there exists a cycle formed only with uncertain states, where the diagnoser can remain forever, it will be impossible for it to diagnose the fault occurrence. On the other hand, if it is possible for the diagnoser to leave this uncertain states cycle, then this cycle is not indeterminate. Therefore, in order to verify the system diagnosability using a diagnoser, it is necessary to search for indeterminate cycles in G_d (SAMPATH *et al.*, 1995, 1996).

Example 2.8. *Consider the DES modeled by automaton G, shown in Figure 2.7, where $\Sigma_o = \{a, b, c\}$ is the set of observable states, and $\Sigma_{uo} = \{\sigma_{uo}, \sigma_f\}$ is the set of unobservable states, and $\Sigma_f = \{\sigma_f\}$ is the set of fault event. In order to check the language diagnosability, we first compute the parallel composition between automaton G of Figure 2.8 and label automaton A_ℓ depicted in Figure 2.6, and thus, we obtain automaton G_ℓ depicted in Figure 2.8. Using automaton G_ℓ, we can* *now compute diagnoser automaton $G_d = Obs(G_\ell, \Sigma_o)$, depicted in Figure 2.9. If we examine G_d, we can check that its state set is formed with two normal state ($\{0N\}, \{2N\}$); three uncertain states ($\{2N, 3Y, 1N\}$,$\{2N, 3Y\}$, $\{2N, 2Y\}$); and two fault states ($\{3Y\}, \{2Y\}$). In addition, Figure 2.9 shows an indeterminate cycle in diagnoser state $\{2N, 2Y\}$. This cycle corresponds to the presence of two cycling traces in the automaton $G :$ (i) a normal trace $s_N = a\sigma_{uo}(bc)^n$, i.e., a trace without fault event; (ii) a fault trace $s_Y = a\sigma_f(bc)^*$, i.e., a trace that has a fault event. Since they have the same observation $P_{\Sigma,\Sigma_o}(s_Y) = P_{\Sigma,\Sigma_o}(s_N) = a(bc)^*$. Thus, according to Theorem 2.1, language $L(G)$ is not diagnosable with respect to projection P_{Σ,Σ_o} and $\Sigma_f = \{\sigma_f\}$.*

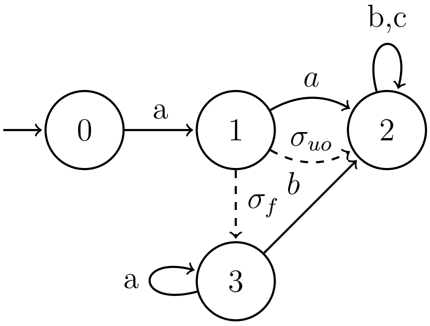


Figure 2.7: State transition diagram of G of example 2.8.

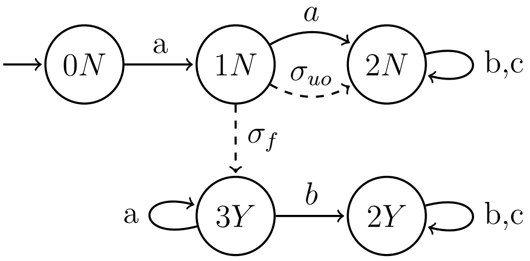


Figure 2.8: State transition diagram of G_ℓ of example 2.8.

2.4 Diagnosability Verification of Discrete Event Systems in different architectures

Diagnoser automaton can be used either off-line to check diagnosability or online by connecting it to the system to provide on-line diagnosis upon the occurrence of

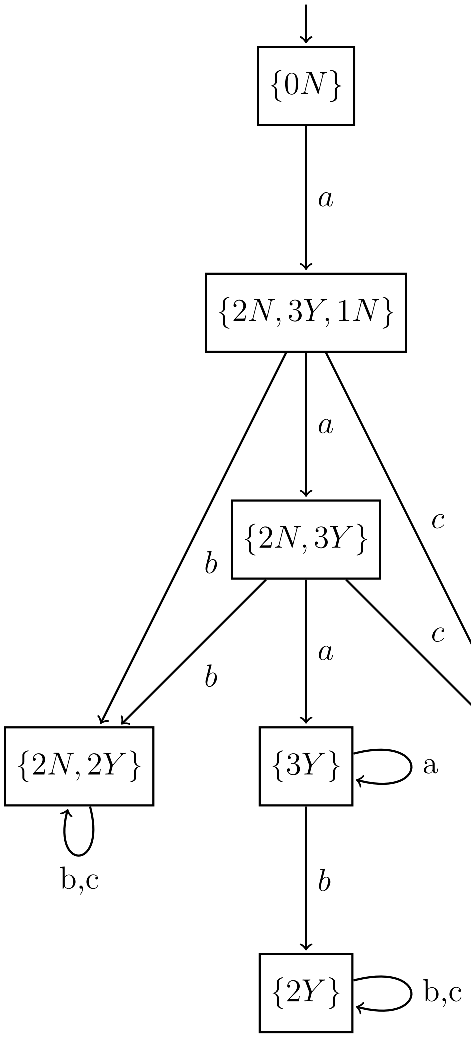


Figure 2.9: State transition diagram of G_d of example 2.8.

observable events. Nevertheless, since the construction of an entire diagnoser is in the worst case, exponential in the number of states of the plant automaton G, its use in diagnosability verification, according to Theorem 2.1, becomes harder. This limitation can be overcome by the use of verifier a automata (YOO e LAFORTUNE, 2002; QIU e KUMAR, 2006; MOREIRA *et al.*, 2011; JIANG *et al.*, 2001), whose corresponding verification algorithm requires polynomial time in the cardinality of the state space and the event set of G. Broadly speaking, the verifier automaton is built through the parallel composition of the system behavior with faults and the system behavior without faults, with a synchronization on the observable events.

2.4.1 Discrete Event Systems Architectures for Fault Diagnosis

Saying that a fault can be diagnosed (SAMPATH *et al.*, 1995; GENC e LAFORTUNE, 2007; RAMIREZ-TREVINO *et al.*, 2011; CABRAL *et al.*, 2015), means that, after a finite number of event observations (LIN, 1994; SAMPATH *et al.*, 1995; ZAYTOON e LAFORTUNE, 2013), the diagnosis system is undoubtedly sure that the fault has occurred. In this regard, the two main issues in the research of fault diagnosis are: (i) on-line diagnosis (SAMPATH *et al.*, 1995; GENC e LAFORTUNE, 2007; RAMIREZ-TREVINO *et al.*, 2011; CABRAL *et al.*, 2015); and (ii) offline diagnosability analysis of the system
Early research works consider diagnosability analysis and on-line diagnosis assuming a monolithic (also referred to as centralized or global) architecture (SAMPATH *et al.*, 1995; YOO e LAFORTUNE, 2002; SAMPATH *et al.*, 1996) The main advantage of centralized diagnosis is its conceptual simplicity and the diagnosis veracity. On the other hand, the computational complexity and the combinatorial explosion are its main disadvantages since, because a monolithic diagnoser may become too large when dealing with a large-scale system. In order to overcome the aforementioned problems, decentralized (DEBOUK *et al.*, 2000) and modular (CONTANT *et al.*, 2006) diagnosis architectures were proposed. Their main objectives of the proposed architectures are similar, namely is to achieve the same diagnosis performance as the monolithic approaches without building a monolithic diagnoser. It is meaningless to discuss which architecture is better, since each one has its own scope of use based on different assumptions and different types of models. The main features of each one of the above architectures are now presented.

1) Decentralized architecture

In this architecture, the input system model is still the monolithic system model. The system is partitioned into several sites, each site having full knowledge of the global model but has only local observation of the system. A local diagnoser is built based on the local observation of the whole system, as seen in Figure 2.10. Local sites $S_i, i = 1, 2, \ldots, N_s$, observe the system behavior based on the information provided by sensors connected to it; therefore, forming sets $\Sigma_{o_i}, i = 1, 2, \ldots, N_s$, of observable events for each site, and so, all events $\sigma \in \Sigma \setminus \Sigma_{o_i}$ are considered unobservable for site S_i. In the decentralized structure of Figure 2.10, each site processes the information received (an event observation) and can only communicate their diagnosis decision to the coordinator, which processes this information according to a predetermined rule and makes a decision regarding the fault occurrence; this process is called protocol. One of the protocols proposed in DEBOUK *et al.* (2000) has led to a generalization of the monolithic approach, being referred to as co-diagnosis. In this approach, local diagnosers cannot communicate directly with each other, but they provide their local decision to a coordinator, which issues the final diagnostic decision, namely a fault is issued when at least one of the local diagnosers is sure that the fault has occurred, and remains quiet otherwise.

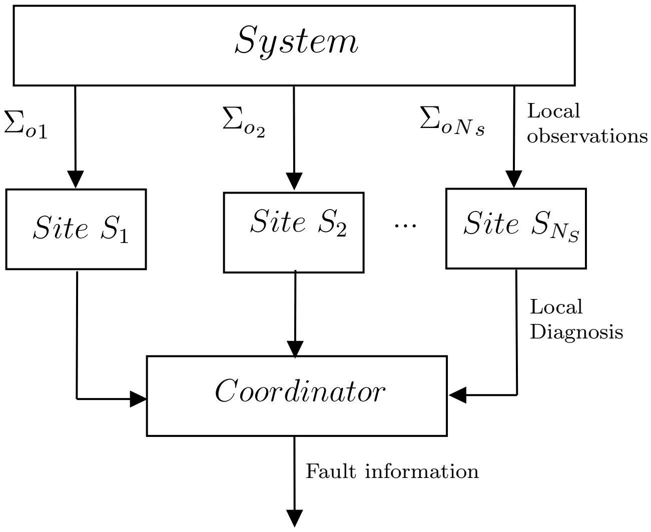


Figure 2.10: Block Diagram of a Decentralized Architectured System.

2) Modular architecture

In this architecture, the input system model is a collection of the models, which can be composed together via parallel composition to obtain the monolithic model. The different module behaviors are synchronized by the common events. It is assumed that the fault event under consideration appears in one module only.

Figure 2.11 shows a block diagram for a modular architecture, where it is assumed that each module k may only "observe" (has sensors) for one part of the system behavior only based on the information provided by the sensors connected to it; therefore, forming sets, Σ_{o_k}, $k = 1, 2, \ldots, M_s$, of observable events for each module, which implies that all events $\sigma \in (\Sigma \setminus \Sigma_{o_k})$ are considered unobservable for module k. The diagnosis decision in the modular architecture shown in Figure 2.11, is made in two levels. The first level is composed by a coordinator, which is responsible for deciding if the system is locally or modularly diagnosable. If the system is locally diagnosable, the coordinator sends the events from the module which posses a fault event, the first module by assumption, to the local diagnoser, which issues a diagnoser decision. Otherwise, the coordinator sends the information to the modular diagnoser together with the modules that are necessary in the synchronization with the local diagnoser. This information is determined by means of an off-line diagnosability analysis, in which the coordinator decides which modules present necessary information in order to built an effective modular diagnoser. The modular diagnoser is built by the information received from the coordinator and the necessary modules, and is responsible for determining if the system is either operating in normal or faulty mode.

2.4.2 Diagnosability verification for DES with decentralized architecture

Suppose that there are m local diagnosers, each one with observable event set $\Sigma_{o_i}, i = 1, 2, \ldots, m$. Let us initially assume, without loss of generality, that L is live. Additionally, let $\mathcal{L}(G_N) = L_N$ be the language associated with the non faulty

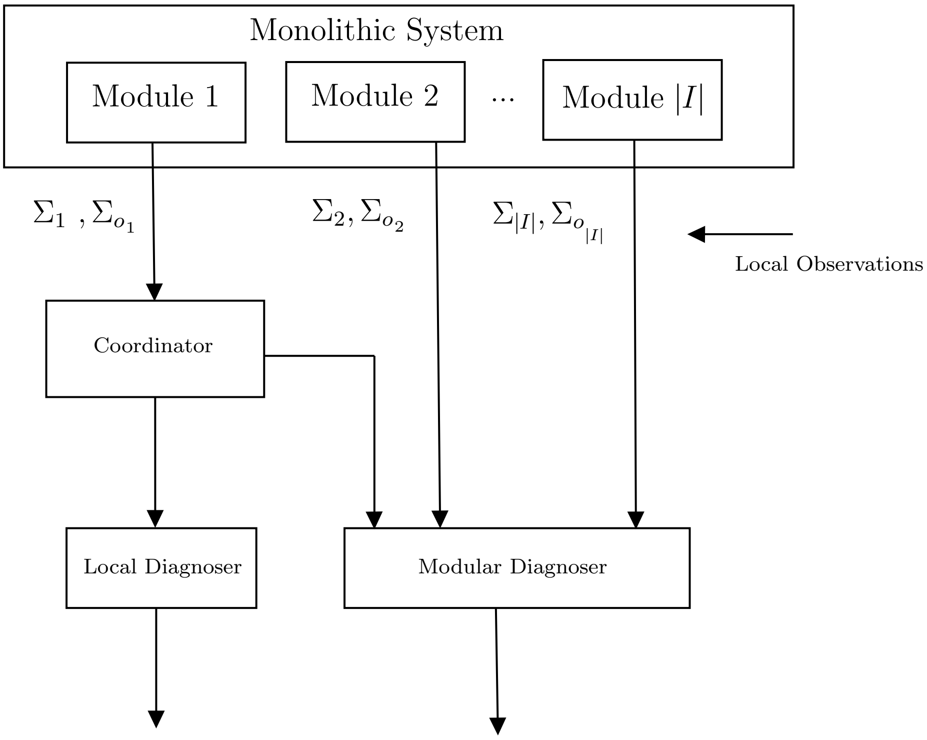


Figure 2.11: Block Diagram of a Modular Architecture System.

behavior of the system,*i.e.*, L_N is a prefix-closed language formed with all traces of L that do not contain any fault event from the set Σ_f.

Definition 18 (Codiagnosability). *Language L if codiagnosable with respect to projections $P_{\Sigma,\Sigma_{o_i}} : \Sigma^* \to \Sigma_{o_i}^*$ and fault event σ_f if*

$$(\exists n \in \mathbb{N})(\forall s \in L \setminus L_N)(\forall st \in L \setminus L_N)(|t| \geq n \Rightarrow D)$$

Where the diagnosability condition D is as follows:

$$(\exists i \in I_M)[\forall \omega \in P^{-1}_{\Sigma,\Sigma_{o_i}}(P_{\Sigma,\Sigma_{o_i}}(st)) \cap L, \omega \in L \setminus L_N]). \tag{2.1}$$

According to Definition 18, L is codiagnosable with respect to $P_{\Sigma,\Sigma_{o_i}}$, and Σ_f if there exists a trace st with arbitrarily long length after the occurrence of the fault event, and a trace $\omega \in L_N$, such that $P_{\Sigma,\Sigma_{o_i}}(s_i) = P_{\Sigma,\Sigma_{o_i}}(st)$ for all $i \in I_m$. When L is a regular language, codiagnosability can be verified using the following automata:

A diagnoser-based automaton with exponential time computacional complexity(VIANA e BASILIO, 2019).

Codiagnosability Verification using Diagnoser-based Automaton

In this section, we first state a necessary and sufficient condition for verification of codiagnosability of a system using a diagnoser-based automaton (VIANA e BASILIO, 2019), and in the sequel, we present an algorithm to verify the stated condition and provides a result for the diagnosability verification of a decentralized DES. The necessary and sufficient condition for verification, proved in VIANA e BASILIO (2019). The definition of the test automaton is present as follows:

- Automaton $G_{scc}^{Ns} = \|_{i=1}^{N_s} G_{d_1} \parallel G_\ell = (X_{scc}, \Sigma, f_{scc}, x_{0,scc})$, such that for all $x \in X_{scc}$, the states x presents the following structure $x = (x_{d_1}, x_{d_2}, \ldots, x_{d_s})$.

Theorem 2.2. *The language L generated by automaton G is codiagnosable with respect to projections $P_{o_i} : \Sigma^* \to \Sigma_{o_i}^*, i = 1, 2, \ldots, N_s$ and $\Sigma_f = \{\sigma_f\}$, if, and only if, G_{scc}^{Ns} does not have any nontrivial strictly connect components (SCC) formed with states $(x_{d_1}^1, x_{d_2}^1, \ldots, x_{d_s}^1, x_\ell^1), (x_{d_1}^1, x_{d_2}^1, \ldots, x_{d_s}^1, x_\ell^1), \ldots, (x_{d_1}^m, x_{d_2}^m, \ldots, x_{d_s}^m, x_\ell^m)$, such that, $\forall j \in I_m, x_{d_i}^j, i = 1, 2, \ldots, N_s$, is uncertain, and x_ℓ^j is a Y-labeled state.*

The work developed in VIANA e BASILIO (2019) has also proved that language generated by $L(G_{scc}) = L(G_\ell) = L(G)$, this fact can be proved straightforwardly since:

$$G_{scc} = G_d \parallel G_\ell = Obs(G_\ell, \Sigma_0) \parallel G_\ell \tag{2.2}$$

is a state partition automaton (CHO e MARCUS, 1989).

Example 2.9. *Consider a system modeled by automaton G, shown in Figure 2.12, where $\Sigma_o = \{a, b, c\}$ is the set of observable states, and $\Sigma_{uo} = \{\sigma_{uo}, \sigma_f\}$ is the set of unobservable states, and $\Sigma_f = \{\sigma_f\}$. In order to check the language diagnosability, we first compute the parallel composition between automaton G of Figure 2.12 and label automaton A_ℓ depicted in Figure 2.13a, and thus, we obtain automaton $G_\ell = G \parallel A_\ell$ depicted in Figure 2.13b. Using automaton G_ℓ, we can now compute the diagnoser automaton $G_d = Obs(G_\ell, \Sigma_o)$, depicted in Figure 2.14. Then, we compute*

```
Algorithm 2: Diagnosability Verification using automaton $G_{scc}^{N_s}$
    Input: $G, \Sigma_{o_i}, i = 1, 2, \ldots, N_s$, and $\Sigma_f = \{\sigma_f\}$
    Output: Codiagnosability Decision: Yes or No.
STEP 1: Compute automaton $G_{scc}^{N_s} = (\|_{i=1}^{N_s} G_{d_i}) \| G_\ell$.
STEP 2: Find all nontrivial SCCs of $G_{scc}^{N_s}$
STEP 3: Verify if there exists at least one nontrivial SCC formed with states
        $(x_{d_1}^1, x_{d_2}^1, \ldots, x_{d_s}^1, x_\ell^1), (x_{d_1}^2, x_{d_2}^2, \ldots, x_{d_s}^2, x_\ell^2), \ldots, (x_{d_1}^m, x_{d_2}^m, \ldots, x_{d_s}^m, x_\ell^m)$, such
        that, $\forall j \in I_m, x_{d_i}^j, i = 1, 2, \ldots, N_s$, is uncertain, and $x_\ell^j$ is a Y-labeled state.
STEP 4: If the answer is yes, then L is not codiagnosable with respect to projections
        $P_{o_i} : \Sigma^* \to \Sigma_{o_i}^*, i = 1, 2, \ldots, N_s$, and, $\Sigma_f = \{\sigma_f\}$. Otherwise, L is codiagnosable.
```

$G_{scc} = G_d \| G_\ell$, which is shown in Figure 2.15. Since G_{scc} has a nontrivial SCC formed with state $(\{2Y, 2N\}, 2Y)$, then, language L, generated by automaton G, is not diagnosable with respect to projections P_{Σ,Σ_o} and Σ_f.

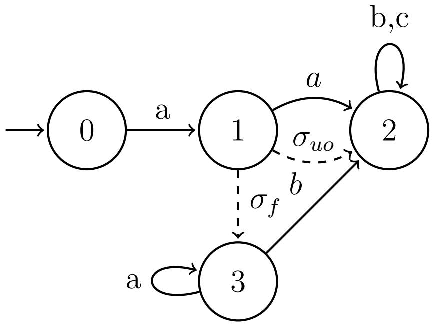


Figure 2.12: State transition diagram of G of example 2.9.

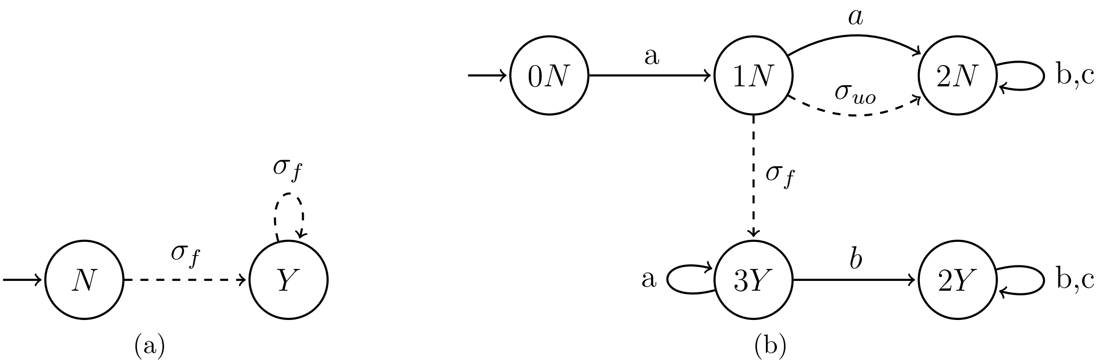


Figure 2.13: Automata A_ℓ (a) and G_ℓ (b) of Example 2.9.

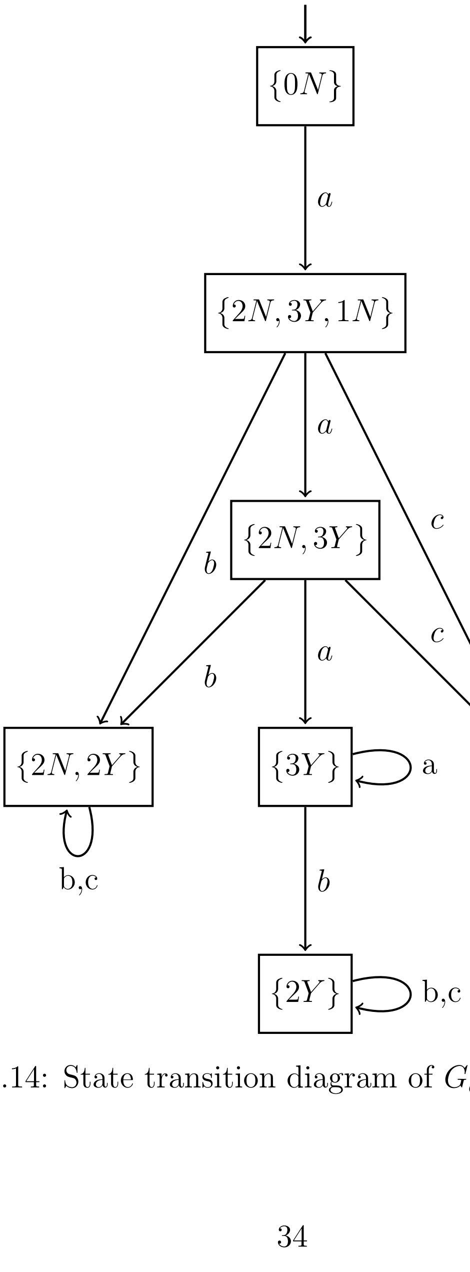


Figure 2.14: State transition diagram of G_d of example 2.9.

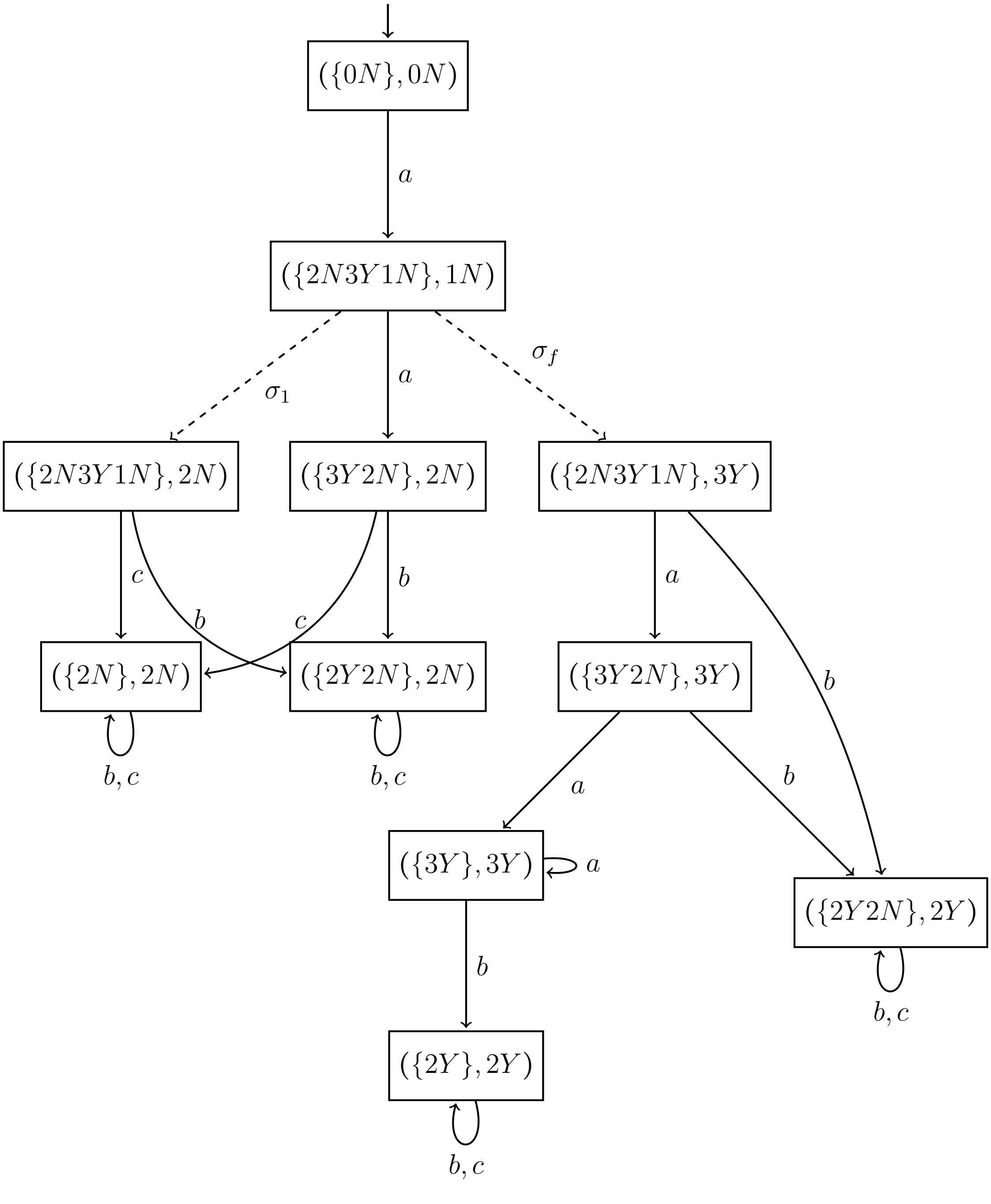


Figure 2.15: State transition diagram of $G_{scc} = G_d \parallel G_\ell$ of example 2.9.

Chapter 3

Methodology

3.1 Modular Discrete Event Systems

Most of discrete event fault diagnosis methodologies build "monolithic" models of system under consideration for diagnosability analysis and implementation; and almost all systems posses a modular structure, which can be composed of several modules, local components or even subsystems that could be formed by various smaller individual modules. If only one module is being analyzed, the modular diagnosability problem emerges and we define as:

Definition 19. *(Local Diagnosability) Let* $I_m := \{1, 2, \cdots, m\}$ *and take* $z \in I$. *The language* $L(G_z)$ *is locally diagnosable with respect to* $\Sigma_{f_z} = \{f\}$ *and* Σ_{z_o}*, if* $\forall s \in L(G_z)$ *that ends with* f*,* $\exists n \in \mathbb{N}$ *s.t.* $\forall t \in L(G_z)/s, |t| \geq n \Rightarrow \mathcal{D}_L(st) = 1$*, where*

$$\mathcal{D}_L(st) := \begin{cases} 1, & \text{if } \forall \omega \in P^{-1}_{\Sigma_z,\Sigma_{z_o}}[P_{\Sigma_z,\Sigma_{z_o}}(st)] \cap L(G_z) \Rightarrow f \in \omega \\ 0, & \text{otherwise} \end{cases}$$

In other words, a module is locally diagnosable, if every sequence that ends with the fault event is distinguishable in a finite number of local observable events after the occurrence of the fault. Notice that if the subscript z in Definition 19 is replaced with I_m, the definition of local diagnosability becomes identical to the monolithic diagnosability definition presented in CONTANT *et al.* (2006) for the monolithic model G_{I_m}. We will now review the definition of modular diagnosability, which extends the definition of local diagnosability by taking into account the possibility that a faulty sequence of is locally indistinguishable in one module to become distinguishable due to the concurrency with other modules.

Definition 20 (*Modular Diagnosability*)**.** *Given a subset $I \subseteq I_M$ and $z \in I$, language $L(G_I)$ is modularly diagnosable w.r.t. $P_{\Sigma_I,\Sigma_{zo}} : \Sigma_I^* \to \Sigma_{zo}^*$, and Σ_f, if*

$$(\exists n \in \mathbb{N})(\forall s \in L_N(G_I))(\forall st \in L(G_I) \setminus L_N(G_I))$$
$$(|P_{\Sigma_I,\Sigma_z}(t)| \geq n \Rightarrow M_D)$$

where the modular diagnosability condition M_D is as follows:

$$M_D : \forall \omega \in P^{-1}_{\Sigma_I,\Sigma_{zo}}(P_{\Sigma_I,\Sigma_{zo}}(st)) \cap L(G_I),$$
$$\omega \in L(G_I) \setminus L_N(G_I)).$$

Notice that the above definition of modular diagnosability differs from that presented in CONTANT *et al.* (2006) in the sense that we use $|P_{\Sigma_I,\Sigma_z}(t)| \geq n$ rather than $|P_{\Sigma_I,\Sigma_{zo}}(t)| \geq n$ since we have not posed any restriction regarding the non-existance of cycle of states connected by unobservable events only, as assumed in SAMPATH *et al.* (1995) and CONTANT *et al.* (2006). Definition 20 also differs from Contant's since in M_D, we take $P^{-1}_{\Sigma_I,\Sigma_{zo}}(P_{\Sigma_I,\Sigma_{zo}}(st)) \cap L(G_I)$ whereas CONTANT *et al.* (2006) takes $P^{-1}_{\Sigma_I,\Sigma_{I_o}}(P_{\Sigma_I,\Sigma_{I_o}}(st)) \cap L(G_I)$ since the former provides a more intuitive idea of the "persistent excitation condition", which requires the occurrence of some event of local module z after every sequence in the language continuation after the fault occurrence. In addition, by considering $P^{-1}_{\Sigma_I,\Sigma_{zo}}(P_{\Sigma_I,\Sigma_{zo}}(st)) \cap L(G_I)$, we are able to identify those existing ambiguities in local module z that carry over the system formed with all modules in I. Finally, it is worth remarking that Definition 20 generalizes Definition 14, since when $I = \{z\}$, then $P_{\Sigma_I,\Sigma_z}(t) = t$.

As in CONTANT *et al.* (2006), the following result can be stated.

Theorem 3.1. *Let $z \in I$ be the index of the module containing the fault event. If language $L(G_z)$ is monolithically diagnosable with respect to $P_{\Sigma_z,\Sigma_{z_o}} : \Sigma_z^* \to \Sigma_{z_o}^*$ and Σ_f, then $L(G_I)$ is modularly diagnosable with respect to $P_{\Sigma_I,\Sigma_{z_o}} : \Sigma_I^* \to \Sigma_{z_o}^*$, and Σ_f.*

Proof. Let $I = \{i_1, i_2, \ldots, i_p\}$, where $p \leq m$, and assume, without loss of generality, that $z = i_1$. Therefore, $G_I = G_z \| G_{i_2} \| \ldots \| G_{i_p}$, $\Sigma_I = \Sigma_z \cup \Sigma_{i_2} \cup \ldots \cup \Sigma_{i_p}$ and

$$\begin{aligned} L(G_I) &= P^{-1}_{\Sigma_I,\Sigma_z}(L(G_z)) \cap P^{-1}_{\Sigma_I,\Sigma_{i_2}}(L(G_z)) \cap \ldots \\ &\quad \cap P^{-1}_{\Sigma_I,\Sigma_{i_p}}(L(G_z)) \end{aligned} \tag{3.1}$$

If $L(G_I)$ is not modularly diagnosable with respect to $P_{\Sigma_I,\Sigma_{z_o}}$ and Σ_f, there must exist two sequences $s_Y, s_N \in L(G_I)$, where $s_Y \in L(G_I) \setminus L_N(G_I)$, $P_{\Sigma_I,\Sigma_z}(s_Y)$ of unbounded length, and $s_N \in L_N(G_I)$ such that $P_{\Sigma_I,\Sigma_{z_o}}(s_Y) = P_{\Sigma_I,\Sigma_{z_o}}(s_N)$. Therefore, from Eq. (3.1), $s_Y, s_N \in P^{-1}_{\Sigma_I,\Sigma_z}(L(G_z))$, which implies that there exist $s_{z_Y}, s_{z_N} \in L(G_z)$ such that $s_{z_Y} = P_{\Sigma_I,\Sigma_z}(s_Y)$ and $s_{z_N} = P_{\Sigma_I,\Sigma_z}(s_N)$, respectively.

Notice that

$$P_{\Sigma_z,\Sigma_{z_o}}(s_{z_Y}) = P_{\Sigma_z,\Sigma_{z_o}}(P_{\Sigma_I,\Sigma_z}(s_Y)) = P_{\Sigma_I,\Sigma_{z_o}}(s_Y)$$

and

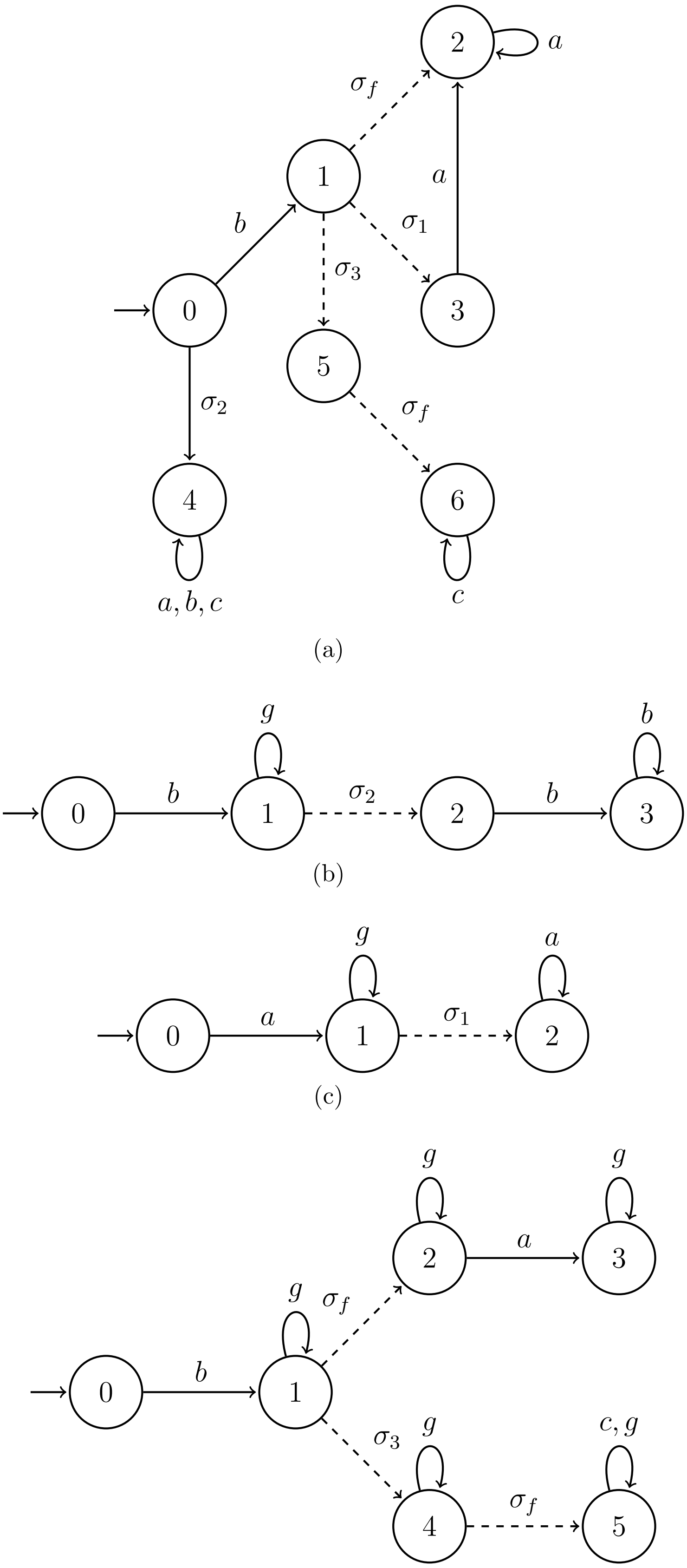
$$P_{\Sigma_z,\Sigma_{z_o}}(s_{z_N}) = P_{\Sigma_z,\Sigma_{z_o}}(P_{\Sigma_I,\Sigma_z}(s_N)) = P_{\Sigma_I,\Sigma_{z_o}}(s_N)$$

which implies that $P_{\Sigma_z,\Sigma_{z_o}}(s_{z_Y}) = P_{\Sigma_z,\Sigma_{z_o}}(s_{z_N})$. In addition, since $P_{\Sigma_I,\Sigma_z}(s_Y)$ is of unbounded length, so is s_{z_Y}. Therefore, G_z is not monolithically diagnosable. ■

Example 3.1. *Let us consider a DES composed of three modules G_1, G_2 and G_3, shown in Fig. 3.1(a)–(c), respectively, whose event sets are $\Sigma_1 = \{a, b, c, \sigma_1, \sigma_2, \sigma_3, \sigma_f\}$, $\Sigma_2 = \{b, g, \sigma_2\}$ and $\Sigma_3 = \{a, \sigma_1\}$. The unobservable events are σ_1, σ_2, σ_3 and the fault event σ_f.*

Notice that $L(G_1)$ is not diagnosable with respect to $P_{\Sigma_1,\Sigma_{o_1}}$ and σ_f. This is so because of the existence of the sequences $s_Y = b\sigma_f a^n$ and $s_N = b\sigma_1 a a^{n-1}$, $n \in \mathbb{Z}_+^*$ (the former trace of unbounded length that has the fault event σ_f, while the latter does not *have the fault event) with the same projection, i.e.,* $P_{\Sigma_1,\Sigma_{o_1}}(s_Y) = P_{\Sigma_1,\Sigma_{o_1}}(s_N) = ba^n$*. In addition, after the fault occurrence, the module evolution persists with the occurrence of event* a*. This result can also be obtained by inspecting Fig 3.4, which shows the test automaton* G_{scc_1} *computed for automaton* G_1 *in accordance with Eq.* (2.2)*. Notice that* G_{scc_1} *has two nontrivial SCCs* $\{\{6Y4N, 6Y\}\}$ *and* $\{\{2N4N2Y, 2Y\}\}$ *whose second (resp. first) state component is a* Y *(resp. uncertain) state; thus showing that* $L(G_1)$ *is not diagnosable with respect to* $P_{\Sigma_1,\Sigma_{o_1}}$ *and* σ_f*.*

Fig. 3.1(d) shows the monolithic system obtained by the synchronization of the three modules ($G = G_1 \| G_2 \| G_3$*). From the state diagram of Fig. 3.1, it can be concluded that* $L(G)$ *is not diagnosable since there are at least two sequences,* $s_Y = bg^p\sigma_f g^n$ *and* $s_N = bg^p\sigma_3 g^n$*,* $p, n \in \mathbb{Z}_+$*,* s_Y *with unbounded length and containing the fault event* σ_f *with the same projection* $P_{\Sigma,\Sigma_o}(s_Y) = P_{\Sigma,\Sigma_o}(s_N) = bg^{p+n}$*, where* $\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 = \{a, b, c, g, \sigma_1, \sigma_2, \sigma_3, \sigma_f\}$ *and* $\Sigma_o = \Sigma_{o1} \cup \Sigma_{o_2} \cup \Sigma_{o_3} = \{a, b, c, g\}$*. The same conclusion can be drawn by inspecting the test automaton* G_{scc} *constructed for the monolithic system* G *since it has a non trivial SCC* $= \{\{4N5Y1N2Y, 5Y\}\}$ *whose second (resp. first) state component is a* Y *(resp. uncertain) state.*



(a)

(b)

(c)

(d)

Figure 3.1: Automaton models G_1 (a), G_2 (b), G_3 (c), and $G = G_1 \| G_2 \| G_3$ (d).

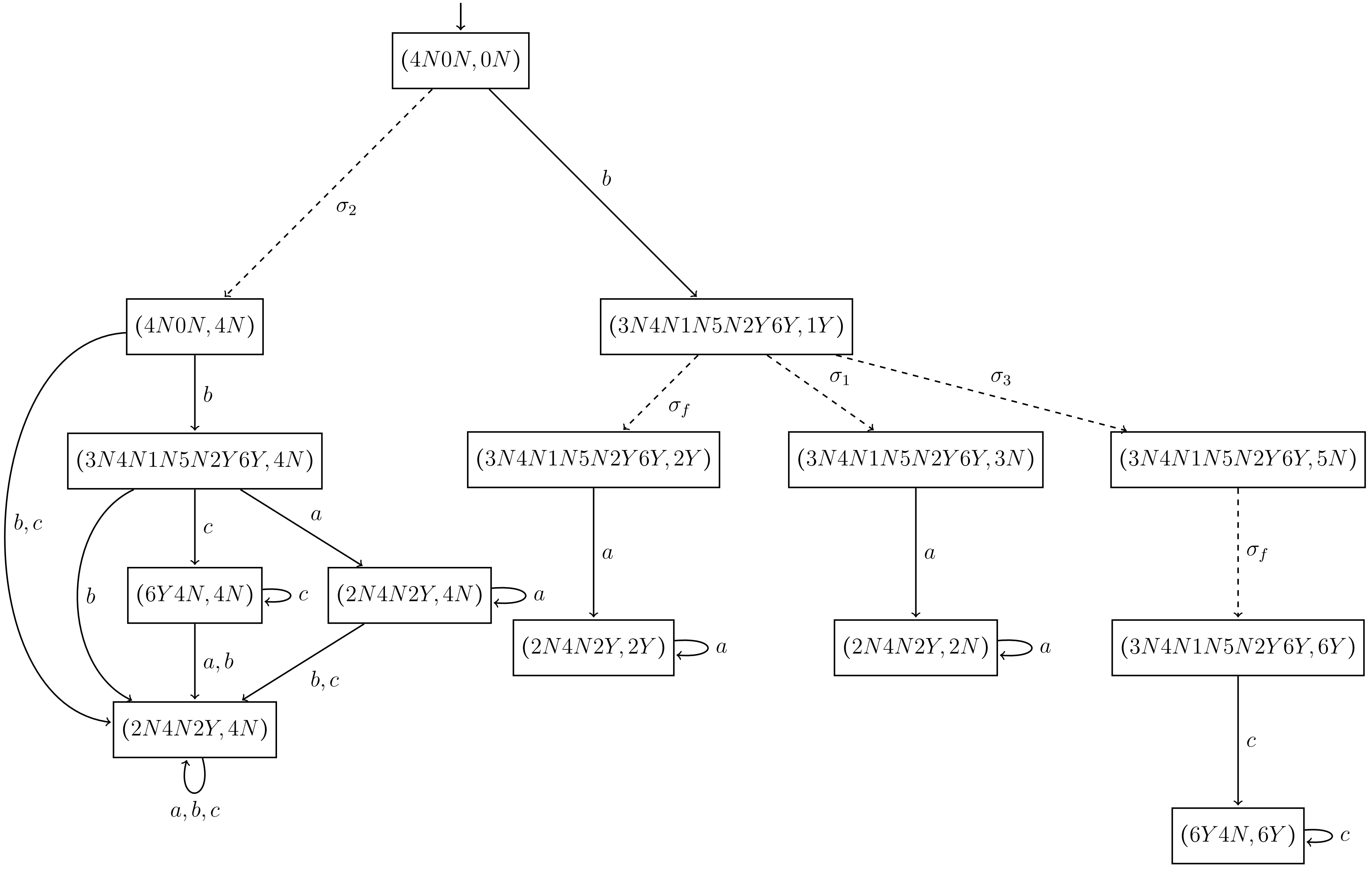


Figure 3.2: Test automata G_{scc_1} for faulty module G_1.

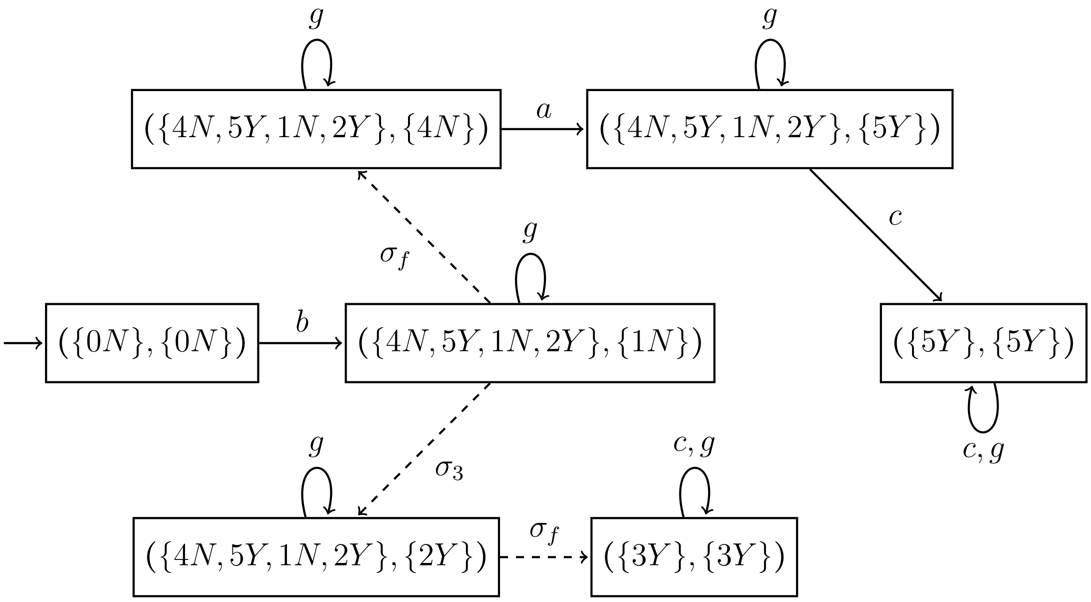

g
g
$(\{4N, 5Y, 1N, 2Y\}, \{4N\})$
a
$(\{4N, 5Y, 1N, 2Y\}, \{5Y\})$
c
σ_f
g
$(\{0N\}, \{0N\})$
b
$(\{4N, 5Y, 1N, 2Y\}, \{1N\})$
$(\{5Y\}, \{5Y\})$
c, g
g
σ_3
c, g
$(\{4N, 5Y, 1N, 2Y\}, \{2Y\})$
σ_f
$(\{3Y\}, \{3Y\})$

Figure 3.3: Test automata G_{scc} for the monolithic system G.

Remark 3.1. *(Differences and similarities between monolithic and modular diagnosabilities) From the definitions of monolithic and modular diagnosabilities (Definitions 14 and 20, respectively)), it is clear that both definitions are concerned with language diagnosability of the overall system, namely, the system G formed with all modules. As Example 3.1 shows, if the diagnosability decision is made based on the overall composition of the modules (monolithic diagnosability), one may conclude that the language generated by the whole system is not diagnosable due to the existence of ambiguous sequences*[1] $s_Y = bg^p\sigma_f g^n$ *and* $s_N = bg^p\sigma_3 g^n$*, where* $p, n = \mathbb{Z}_+$*. However, as will be clear later on in the thesis, if you take into account the structure of the system, and analyze how the behavior of the module where the fault occurs synchronizes with the behavior of the other modules, you will conclude that the ambiguous sequences that occur in the overall system do not exist since they do not represent sequences that can actually occur in the faulty module. In this case, building a diagnoser for the overall system does not work; instead, it suffices to build a diagnoser based on the relevant events to the diagnosability of the language generate by the fault module.*

Remark 3.2. *(particularities on CONTANT* et al. *(2006))*

[1] Two sequences $s_Y, s_n \in L(G)$, $\sigma \in s_Y$ but $\sigma \notin s_N$, if $P_{\Sigma,\Sigma_o}(s_Y) = P_{\Sigma,\Sigma_o}(s_N)$.

The definition of monolithic diagnosability 21 provided by (CONTANT et al., *2006) differs from the diagnosability definition introduced in (SAMPATH* et al., *1995), since:*

- *The equation $P_{\Sigma,\Sigma_o}(|t|) > n$ instead of $|t| > n$. This modification implies that cycles of unobservable events are not taken into account when verifying the diagnosability properties of a system.*
- *The order of the quantifiers allows one natural number n for each trace s that ends with a fault event, instead of requiring one natural number for each fault event σ_f, i.e., for all traces s ending with σ_f.*

Definition 21 (Monolithic Diagnosability). *Let L be the live and prefix-closed language generated by the system, and $L_N \in L$ be the fault-free language of L. $P_{\Sigma,\Sigma_o} = \Sigma^* \to \Sigma_o^*$ be a projection operation. Then, L is said to be diagnosable with respect to projection P_o and Σ_f, if*

$$((\exists n \in \mathbb{N})(\forall s \in L \setminus L_N)(\forall st \in L \setminus L_N, P_{\Sigma,\Sigma_o}(|t|) \geq n) \Rightarrow D)$$

where the diagnosability condition D is as follows:

$$(\forall \omega \in P^{-1}_{\Sigma,\Sigma_o}(P_{\Sigma,\Sigma_o}(st)) \cap L, \omega \in L \setminus L_N).$$

Another remarkable particularity presented is a detailed statement of our Modular Diagnosability Algorithm (MDA), in three parts. Part one is the core of MDA; it calls the part two to perform preliminary steps involving indeterminate cycles that could lead to a violation of modular diagnosability. The goal of the part two is to identify all the indeterminate cycles that are present in the modules and yield a list of sequences of states and events that is used in the other algorithms. After the second part, part one also the part three where the incremental analysis of each indeterminate cycle is performed CONTANT et al. *(2006).*

3.2 An automaton-based algorithm for modular diagnosability verification

In order to present a new necessary and sufficient condition for modular diagnosability based on an extension of the test automaton given in Eq. (2.2), the following result, which is a direct consequence of the diagnosability condition proposed in VIANA e BASILIO (2019) and reviewed in Section 2 is required.

Lemma 3.1. *If the language $L(G)$ generated by the automaton G is diagnosable with respect to projection P_{Σ,Σ_o} and $\Sigma_f = \{\sigma_f\}$, then, each nontrivial strongly connected component $SCC_Y = \{(X_{YN_1}, X_{Y_1}), (X_{YN_2}, X_{Y_2}), ..., (X_{YN_p}, X_{Y_p})\}$, has no corresponding strongly connected component $SCC_N = \{(X'_{YN_1}, X_{N_1}), (X'_{YN_2}, X_{N_2}), ..., (X'_{YN_q}, X_{N_q})\}$, either trivial or nontrivial, such that $X_{YN} = \cup_{i=1}^{p}\{\{X_{YN_i}\}\}$, and $X'_{YN} = \cup_{i=1}^{p}\{\{X'_{YN_i}\}\}$ satisfying $X_{YN} = X'_{YN}$.*

Proof. $(\Rightarrow)$ Supose that exists a $SCC_Y = \{(X_{YN_1}, X_{Y_1}), (X_{YN_2}, X_{Y_2}), ..., (X_{YN_p}, X_{Y_p})\}$, and exists a corresponding $SCC_N = \{(X'_{YN_1}, X_{N_1}), (X'_{YN_2}, X_{N_2}), ..., (X'_{YN_q}, X_{N_q})\}$, since $L(G_{test}) = L(G_\ell) = L(G) = L$, then $\exists st \in L : s \in \Sigma_f, |t| > n$. Two possibilities exists:

- $X_{YN_1} = X_{YN_2} = X_{YN_2} = \cdots = X_{YN_p}$. It means that states $(X_{YN_1}, X_{Y_1}), (X_{YN_2}, X_{Y_2}), ..., (X_{YN_p}, X_{Y_p})$ are connected by events σ, such that $\sigma \notin \Sigma_o$, since these events are strictly from $_\ell$. Also, since $X_{Y_i} \in X_{YN_i}, i = 1, 2, \ldots, n$, then exists an indeterminate cycle in X_{YN_i}, which leads to the existence of an SCC_N.
- Exists $\{i_1, i_2, \ldots, i_n\} \subseteq \{1, 2, \ldots, p\}$ such that $X_{YN_{i_k}} \neq X_{YN_{i_l}}$, $k \neq l$, $k, l \in \{1, 2, \ldots, m\}$. Since $X_{Y_i}, i = 1, 2, \ldots, p$ are not indeterminate states, and $L(G_) = L$, then exists a indefinitely long sequence $s_Y = st \in L$, and the s ends with a faulty event, and $|t| \geq n, \forall n \in \mathbb{N}$. Also, since $X_{YN_{i_k}}, i = 1, 2, \ldots, m$, are uncertain states, there exists $s_N \in L$, such that $P_{\Sigma,\Sigma_o}(s_Y) = P_{\Sigma,\Sigma_o}(s_N)$, then, the sequence s_N leads to an SCC_N

($\Leftarrow$) Consider that L is not diagnosable with respect to projection P_{Σ,Σ_o} and $\Sigma_f = \{\sigma_f\}$. Thus, there must exist two indefinitely long sequences $s_Y ands_N$, such that $P_{\Sigma,\Sigma_o}(s_Y) = P_{\Sigma,\Sigma_o}(s_N)$, such that $\sigma_f \notin s_N$ $s_Y = st \in L$, and the s ends with the σ_f event, and $|t| \geq n, \forall n \in \mathbb{N}$. Considering $|X_{YN}||X_Y| = q$, and $n > q$, then if $\varepsilon_{G_{test}}(x_0, s_Y) = X = (x_d, x_\ell) = (X_{YN}, X_Y)$, a strongly connected component is formed in G_{test}. If we consider the first component of the state x_d is composed only by certain components, once x_d is in an cycle, it can never become uncertain again, then any sequence $s \in L$, such that $P_{\Sigma,\Sigma_o}(s) = P_{\Sigma,\Sigma_o}(s_N)$ is going to lead to an SCC_Y, which contradicts the hypothesis that exists an $s_N, \sigma_f \notin s_N$, such that $P_{\Sigma,\Sigma_o}(s_Y) = P_{\Sigma,\Sigma_o}(s_N)$. Thus, $\varepsilon_{G_{test}}(x_0, s_N) = (x_d, x_\ell) = (X_{YN}, X_N) = SCC_N$.

■

Lemma 3.2. *If the language $L(G)$ generated by the automaton G is not diagnosable with respect to projection P_{Σ,Σ_o} and $\Sigma_f = \{\sigma_f\}$, then, each nontrivial strongly connected component $SCC_Y = \{(X_{YN_1}, X_{Y_1}), (X_{YN_2}, X_{Y_2}), ..., (X_{YN_p}, X_{Y_p})\}$, has at least one corresponding strongly connected component $SCC_N = \{(X'_{YN_1}, X_{N_1}), (X'_{YN_2}, X_{N_2}), ..., (X'_{YN_q}, X_{N_q})\}$, either trivial or nontrivial, such that $X_{YN} = \cup_{i=1}^{p}\{\{X_{YN_i}\}\}$, and $X'_{YN} = \cup_{i=1}^{p}\{\{X'_{YN_i}\}\}$ satisfying $X_{YN} = X'_{YN}$.*

Proof. The proof is straightforward and comes from the fact that for a language to be nondiagnosable it is necessary to exist an unbounded length faulty sequence s_Y and a not necessary unbounded normal sequence s_N with the same projection, i.e., $P_{\Sigma,\Sigma_o}(s_Y) = P_{\Sigma,\Sigma_o}(s_N)$. ■

The importance of this result is that if there exist observable or unobservable events that prevent ambiguous sequences to reach strongly connected components SCC_Y or SCC_N, therefore removing either of them, then the language of the composed module becomes diagnosable.

In order to further develop this idea, let us assume, without loss of generality, that $z = 1$, i.e., the first module is the faulty module, and denote the composite module built with M modules as

$$G_M = \|_{i=1}^{M} G_i. \tag{3.2}$$

Let

$$\begin{aligned} G_{scc_1} &= G_{1_d} \| G_{1_\ell}, & (3.3)\\ G_{scc_M} &= G_{M_d} \| G_{M_\ell}, & (3.4)\end{aligned}$$

be Viana and Basilio's test automata for the faulty and for the composite module G_M, respectively, as reviewed in Eq. (2.2), and define the following test automaton:

$$G_{test_M} = G_{scc_1} \|_{i=2}^{M} G_{i_\ell} \tag{3.5}$$

The following result can be stated.

Lemma 3.3. *The language $L(G_M)$ generated by the composite module G_M is modularly diagnosable with respect to P_{Σ_M,Σ_z} and Σ_f if, and only if, the language $L(G_{test_M})$ generated by the test automaton G_{test_M} is modularly diagnosable with respect to P_{Σ_M,Σ_z} and Σ_f.*

Proof. In order to prove this result, it is enough to prove that the languages generated by G_{test_M} and G_M are equal, i.e., $L(G_{test_M}) = L(G_M)$. Notice that

$$G_{test_M} = G_{scc_1} \|_{i=2}^{M} G_{i_\ell} = G_{1_d} \| G_{1_\ell} \|_{i=2}^{M} G_{i_\ell} = G_{scc_1} \|_{i=1}^{M} G_{i_\ell},$$

and thus:

$$L(G_{test_M}) = P_{\Sigma_M,\Sigma_1}^{-1} L(G_{scc_1}) \cap_{i=1}^{M} P_{\Sigma_M,\Sigma_i}^{-1}(L(G_{i_\ell})).$$

According to VIANA e BASILIO (2019, Fact 1), $L(G_{scc_1}) = L(G_1) = L(G_{1_\ell})$. Therefore

$$\begin{aligned} L(G_{test_M}) &= \cap_{i=1}^{M} P^{-1}_{\Sigma_M,\Sigma_i}(L(G_{i_\ell})) \\ &= \cap_{i=1}^{M} P^{-1}_{\Sigma_M,\Sigma_i}(L(G_i)) = L(G_M). \end{aligned}$$

■

According to Lemma 3.3, in order to ascertain whether a centralized system whose modular structure is known is modularly diagnosable diagnosability, it is not necessary to build the composed module each time modular diagnosability needs to be verified; instead, all that has to be done is to check if the language generated by the automaton obtained by performing the parallel composition between G_{scc_1} and the other plant modules. Based on this rational, we propose an algorithm (Algorithm 3) for modular diagnosability verification. In Algorithm 3, it is assumed that the language generated by the faulty module is nondiagnosable, and therefore after computing G_{scc_1} in accordance with Eq. (2.2), two sets of SCCs, $\mathcal{S} = \{(SCC_{Y_1}, SCC_{N_1}), (SCC_{Y_2}, SCC_{N_2}), \ldots, (SCC_{Y_p}, SCC_{N_p})\}$, where p is the number of pairs of SCC that satisfy the condition imposed by Lemma 3.1 have been computed; it is worth remarking that it is possible that besides pair (SCC_{Y_k}, SCC_{N_k}), other pairs (SCC_{Y_l}, SCC_{N_l}), $k \neq l$, can be formed.

The basic idea behind Algorithm 3 is, at each step, to add each module G_i to $G_{test_{i-1}}$ so as to verify if all pairs of SCCs responsible for the non-diagnosability of the faulty module survive or not. If for some value of i set $\mathcal{S}$ becomes empty, then all existing ambiguities have been removed, and thus the language is diagnosable. On the other hand, if all modules are added to G_{test} and some SCC does survive, then it is clear that the language is not diagnosable. The algorithm correctness is ensured by the following result.

Theorem 3.2. *Let $\mathcal{S} = \{(SCC_{Y_1}, SCC_{N_1}), (SCC_{Y_2}, SCC_{N_2}), \ldots, (SCC_{Y_p}, SCC_{N_p})\}$ be the set of SCC of G_{scc_1} computed according to Lemma 3.1 and Lemma 3.2, and let $\mathcal{S}_{1,i_1,\ldots,i_k}$ $(\{i_1, i_2, \ldots, i_k\} \in \{2, 3, \ldots, m\})$ be the set of SCC pairs of $\mathcal{S}$ that remains*

Algorithm 3: Modular Diagnosability Verification

```
Input: $G_{scc_1}$, $\mathcal{G} = \{G_2, \ldots, G_M\}$, $\Sigma_f = \{\sigma_f\}$,
$\mathcal{S} = \{(SCC_{Y_1}, SCC_{N_1}), (SCC_{Y_2}, SCC_{N_2}), \ldots, (SCC_{Y_p}, SCC_{N_p})\}$: set of SCC of
$G_{scc_1}$ computed according to Lemma 3.1.
Output: $(MD, J)$
$MD \in \{Yes, No\}$: modular diagnosability decision
$J \subseteq (I_M \cup \{1\})$ (set of plant module indexes $G_i$ to ensure modular
diagnosability); $J = \varnothing$, if $\cup_{i=1}^{M}$ is not modularly diagnosable
$G_{test_{old}} \leftarrow G_{scc_1}$; $flag \leftarrow T$
$J \leftarrow \{1\}$; $I \leftarrow \{2, 3, \ldots, M\}$, while $flag$ do
    for $k \in I$ do
        $CE_k \leftarrow |\Sigma_k \cap \Sigma_{test_{i-1}}|$
    $CE_{max} \leftarrow \max_{k \in I} CE_k$
    $j \leftarrow \min k \in I, CE_k = CE_{max}$
    if $CE_{max} \neq 0$ then
        $G_{test} \leftarrow G_{test_{old}} \| G_j$
        $J \leftarrow J \cup \{j\}$
        $I \leftarrow I \setminus \{j\}$
        Compute all SCCs of $G_{test}$ and remove from set $\mathcal{S}$ all pairs
          $(SCC_Y, SCC_N)$ that appear in $G_{test_{old}}$ but not in $G_{test}$
        if $\mathcal{S} == \varnothing$ then
            $flag \leftarrow F$; $MD \leftarrow Yes$;
        else
            if $I == \varnothing$ then
                $flag \leftarrow F$; $MD \leftarrow No$; $J \leftarrow \varnothing$;
    else
        $flag \leftarrow F$; $MD \leftarrow No$; $J \leftarrow \varnothing$;
    $G_{test_{old}} \leftarrow G_{test}$
return $MD, J$
```

in $G_{test_{1,i_1,\dots,i_k}}$. Then $G_{1,i_1,\dots,i_k} = G_1 \| (\|_{k \in \{1,i_1,\dots,i_k\}} G_i)$ will be modularly diagnosable with respect to $P_{\Sigma_{1,i_1,\dots,i_k},\Sigma_{o_1}}$ and $\Sigma_f = \{\sigma_f\}$ if, and only if, $\mathcal{S}_{1,i_1,\dots,i_k} = \varnothing$.

Proof. Consider $\mathcal{S} = \{(SCC_{Y_1}, SCC_{N_1}), (SCC_{Y_2}, SCC_{N_2}), \dots, (SCC_{Y_p}, SCC_{N_p})\}$, and according to Lemma 3.2, for each nontrivial strongly connected component $SCC_Y = \{(X_{YN_1}, X_{Y_1}), (X_{YN_2}, X_{Y_2}), ..., (X_{YN_p}, X_{Y_p})\}$, has at least one corresponding strongly connected component $SCC_N = \{(X'_{YN_1}, X_{N_1}), (X'_{YN_2}, X_{N_2}), ..., (X'_{YN_q}, X_{N_q})\}$, either trivial or nontrivial, such that $X_{YN} = \cup_{i=1}^{p}\{\{X_{YN_i}\}\}$, and $X'_{YN} = \cup_{i=1}^{p}\{\{X'_{YN_i}\}\}$ satisfying $X_{YN} = X'_{YN}$. Since Algorithm 3 builds automaton G_{test} with all possible modules that are able to build an modular diagnosable composition, if the search for new modules to compose G_{test} is null, then there are not enough information to clarify if the system is in a faulty state or normal state, then the algorithm is going to return a negative diagnosability result. Otherwise, using the criteria, according to Lemma 3.1 and Lemma 3.2, if there are a number of modules p, such that $p \in \{2, \dots, M\}$, where M is the number of modules in the system, that are able to empty the set S, then Algorithm 3 is going to return a set of modules, J, that reaches an empty set S. And also, since in the worst case scenario, all modules are going to be added to the composition, hence $J = \{1, 2, \dots, M\}$, it assures that if there is a set of modules H that return $S = \{\}$, $H \subseteq J$.

∎

The following example illustrates the verification algorithm proposed in the thesis.

Example 3.2. *Let us consider, again, the same system whose model has the modular structure given in Fig. 3.1(a)–(c). As shown in Example 3.1, G_1 is not locally diagnosable, i.e., $L(G_1)$ is not diagnosable with respect to $P_{\Sigma_1,\Sigma_{o_1}}$ and σ_f. Therefore, in accordance with Lemma 3.1, to each SCC_Y strongly connected component there must exist a SCC_N strongly connected component whose first state components have the same states as the corresponding SCC_Y. Indeed, from Fig. 3.4, the following set* *of pairs* (SCC_Y, SCC_N) *can be formed:*

$$\begin{aligned}\mathcal{S} \;=\; & \Big\{\Big(\{\{6Y4N, 6Y\}\}, \{\{6Y4N, 4N\}\}\Big),\\ & \Big(\{\{2N4N2Y, 2Y\}\}, \{\{2N4N2Y, 4N\}\}\Big),\\ & \Big(\{\{2N4N2Y, 2Y\}\}, \{\{2N4N2Y, 2N\}\}\Big)\Big\} \qquad (3.6)\end{aligned}$$

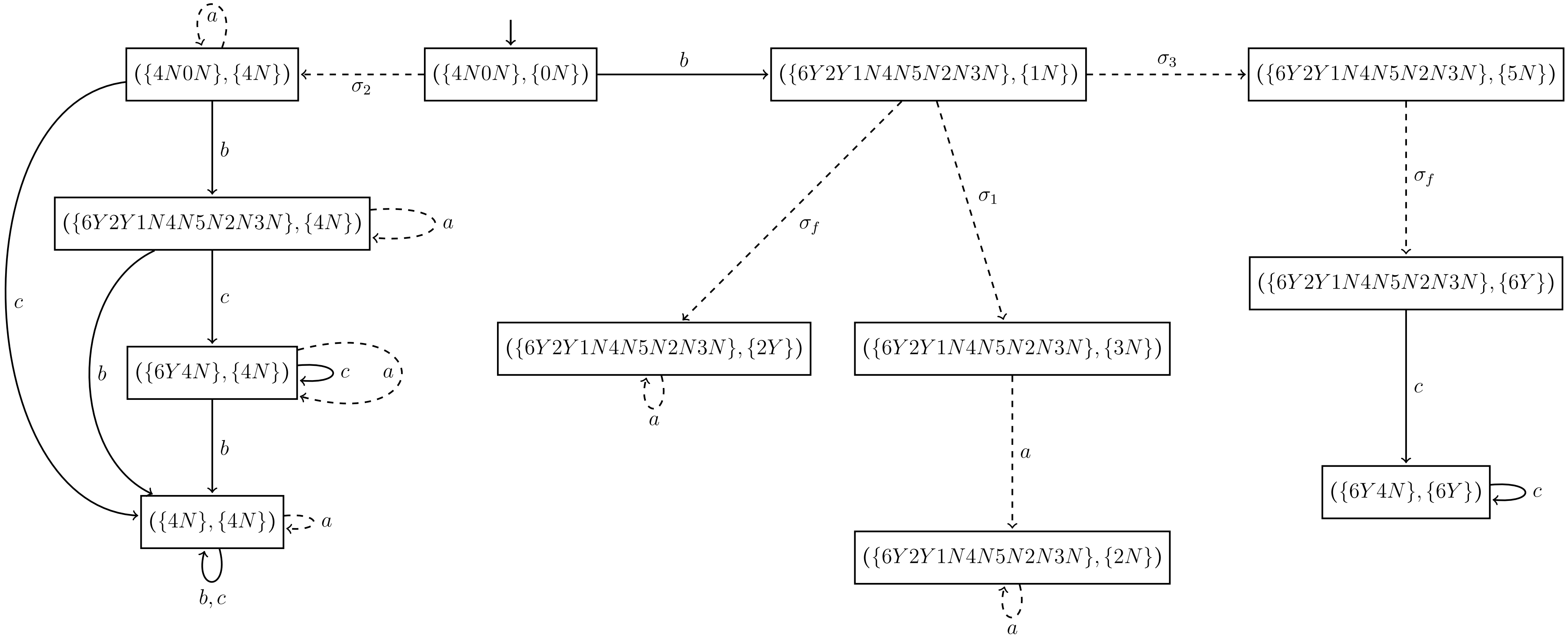

$(\{4N0N\},\{0N\})$
$(\{4N0N\},\{4N\})$
$(\{6Y2Y1N4N5N2N3N\},\{1N\})$
$(\{6Y2Y1N4N5N2N3N\},\{5N\})$
$(\{6Y2Y1N4N5N2N3N\},\{4N\})$
$(\{6Y4N\},\{4N\})$
$(\{4N\},\{4N\})$
$(\{6Y2Y1N4N5N2N3N\},\{2Y\})$
$(\{6Y2Y1N4N5N2N3N\},\{3N\})$
$(\{6Y2Y1N4N5N2N3N\},\{2N\})$
$(\{6Y2Y1N4N5N2N3N\},\{6Y\})$
$(\{6Y4N\},\{6Y\})$
σ_2
σ_3
σ_1
σ_f
a
b
c
b, c

Figure 3.4: Test automata G_{scc_1} for faulty module G_1.

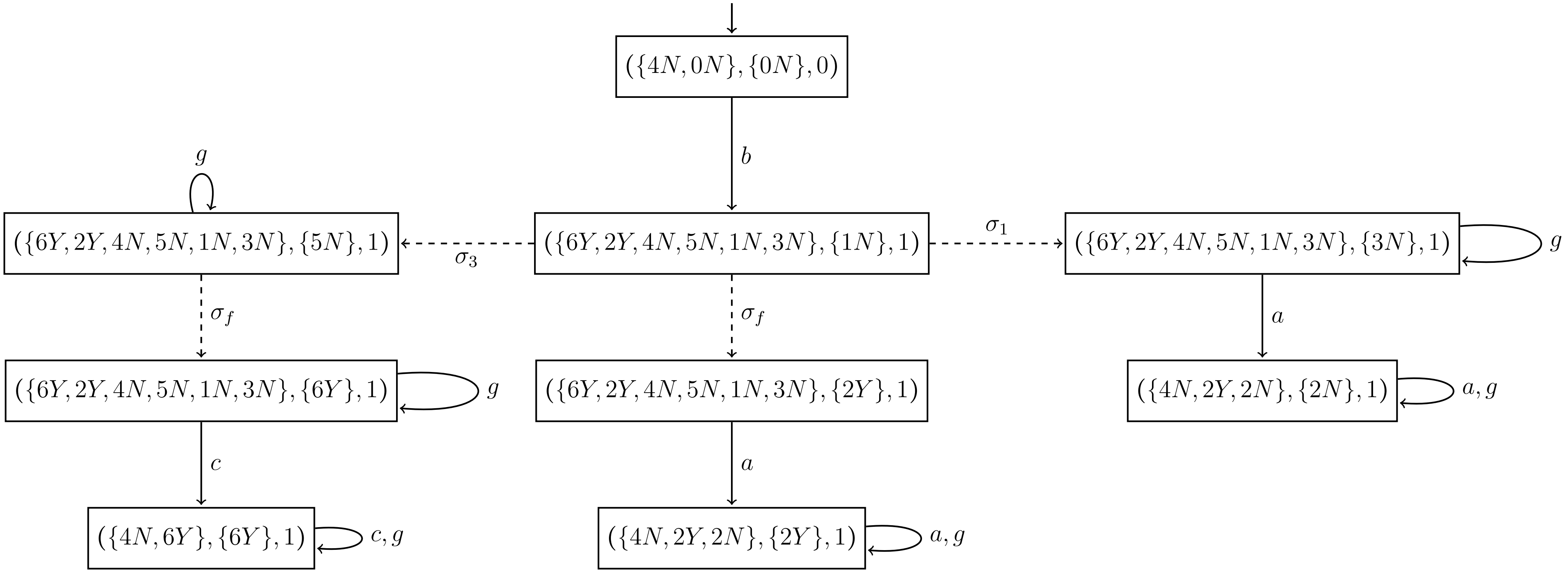
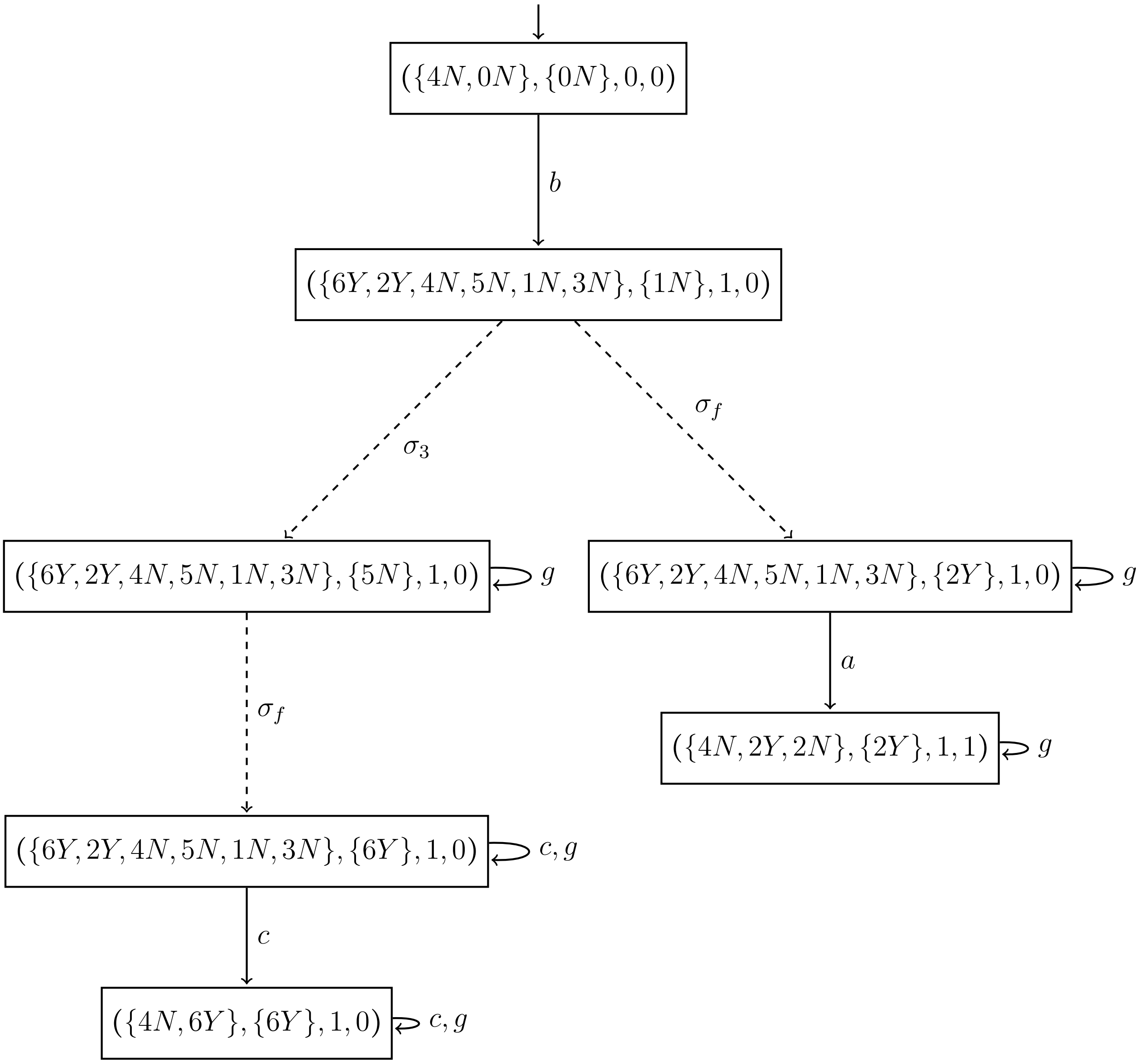

({4N, 0N}, {0N}, 0)
b
({6Y, 2Y, 4N, 5N, 1N, 3N}, {1N}, 1)
σ3
σ1
g
({6Y, 2Y, 4N, 5N, 1N, 3N}, {5N}, 1)
({6Y, 2Y, 4N, 5N, 1N, 3N}, {3N}, 1)
g
σf
σf
a
({6Y, 2Y, 4N, 5N, 1N, 3N}, {6Y}, 1)
g
({6Y, 2Y, 4N, 5N, 1N, 3N}, {2Y}, 1)
({4N, 2Y, 2N}, {2N}, 1)
a, g
c
a
({4N, 6Y}, {6Y}, 1)
c, g
({4N, 2Y, 2N}, {2Y}, 1)
a, g

Figure 3.5: Test automata $G_{test_{12}}$ of Example 3.2.



$(\{4N,0N\},\{0N\},0,0)$
b
$(\{6Y,2Y,4N,5N,1N,3N\},\{1N\},1,0)$
σ_3
σ_f
$(\{6Y,2Y,4N,5N,1N,3N\},\{5N\},1,0)$
g
$(\{6Y,2Y,4N,5N,1N,3N\},\{2Y\},1,0)$
g
a
σ_f
$(\{4N,2Y,2N\},\{2Y\},1,1)$
g
$(\{6Y,2Y,4N,5N,1N,3N\},\{6Y\},1,0)$
c,g
c
$(\{4N,6Y\},\{6Y\},1,0)$
c,g

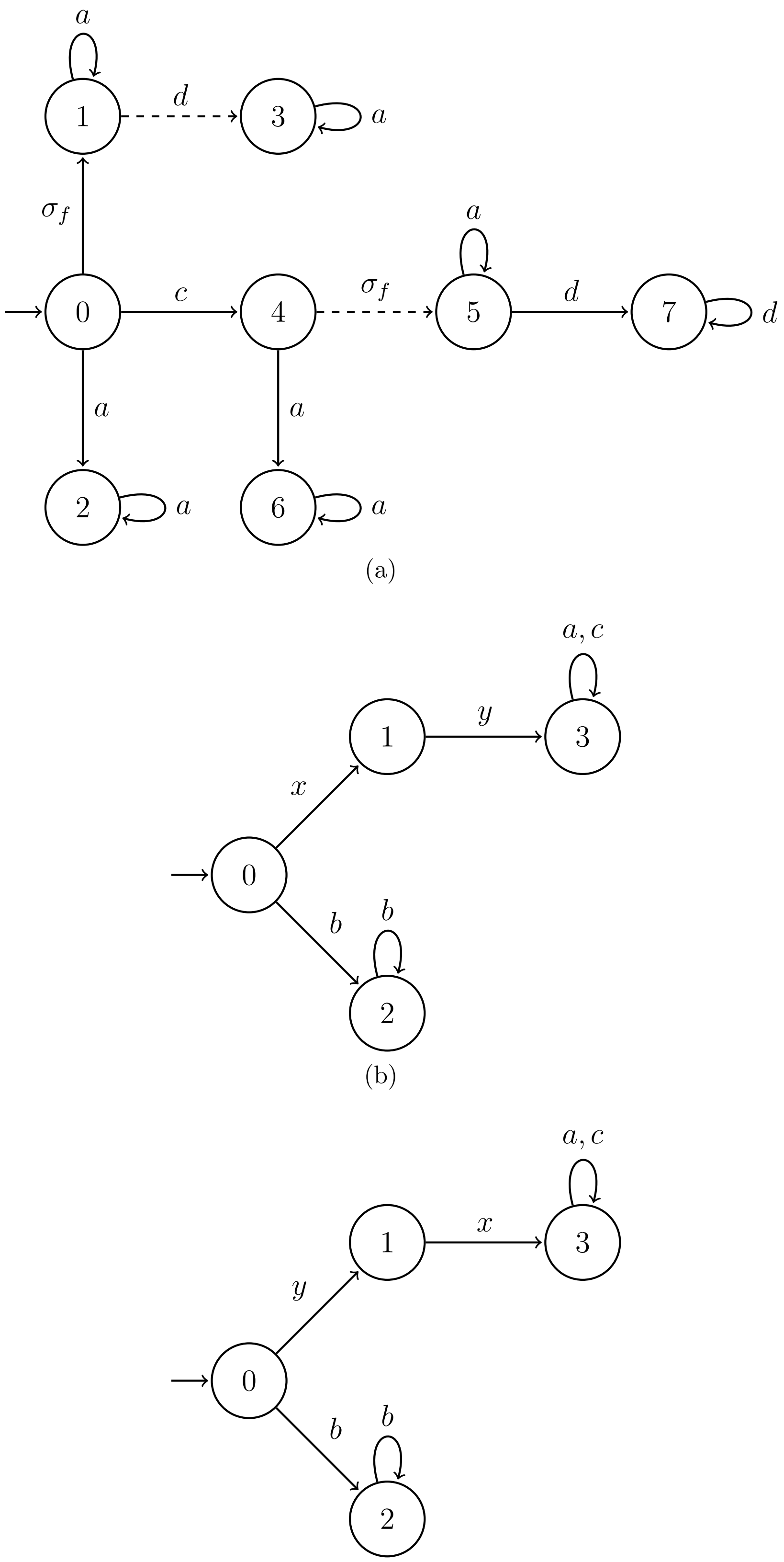
Figure 3.6: Test automata $G_{test_{123}}$ of Example 3.2.

Notice that since $\Sigma_1 \cap \Sigma_2 = \{b,\sigma_2\}$ and $\Sigma_1 \cap \Sigma_3 = \{a,\sigma_1\}$ have the same cardinality, any one of the modules can be chosen to synchronize with G_1. When module G_1 synchronizes with module G_2, the unnobservable common event σ_2 can only occur after the occurrence of observable event b, which is also common to the first two modules. Therefore, event σ_2 will never occur in $G_{12} = G_1\|G_2$. As a consequence, $G_{test_{12}} = G_{test_1}\|G_2$. It removes in G_{test_1} model all event sequences starting with σ_2 as shown in Fig. 3.5. The only pair of $\mathcal{S}$ that survives is $\big(\{\{2N4N2Y,2Y\}\},\{\{2N4N2Y,2N\}\}\big)$. Finally, by performing the synchronization of $G_{test_{12}}$ with G_3, test automaton $G_{test_{123}}$ shown in Fig. 3.6 is obtained, from which we can see that $\mathcal{S} = \{\}$, meaning that G is modularly diagnosable with respect to $P_{\Sigma,\Sigma_{o_1}}$ and $\Sigma_f = \{\sigma_f\}$.

Example 3.3. *Let us consider, a similar system from the one proposed in CONTANT* et al. *(2006),whose model has the modular structure given in Fig. 3.7(a)–(c). It can be seen that, G_1 is not locally diagnosable, i.e., $L(G_1)$ is not diagnosable with respect to $P_{\Sigma_1,\Sigma_{o_1}}$ and σ_f, since $s_Y = \sigma_f a^*$, and $s_N = a^*$ where $P_{\Sigma_1,\Sigma_{o_1}}(s_Y) = P_{\Sigma_1,\Sigma_{o_1}}(s_N)$. Therefore, in accordance with Lemma 3.1, to each SCC_Y strongly connected component there must exist a SCC_N strongly connected component whose first state components have the same states as the corresponding SCC_Y. Indeed, from Fig. 3.8, the following set of pairs (SCC_Y, SCC_N) can be formed:*

$$\begin{aligned}\mathcal{S} \;=\; & \Big\{\Big(\{\{1Y2N, 2N\}\}, \{\{1Y2N, 1Y\}\}\Big), \\ & \Big(\{\{5Y6N, 5Y\}\}, \{\{5Y6N, 6N\}\}\Big)\Big\} \end{aligned} \tag{3.7}$$

Notice that since $\Sigma_1 \cap \Sigma_2 = \{a, c\}$ and $\Sigma_1 \cap \Sigma_3 = \{a, c\}$ have the same cardinality, any one of the modules can be chosen to synchronize with G_1. By performing the synchronization of G_{scc_1} with G_2 and with G_3, test automaton $G_{test_{123}}$ shown in Fig. 3.9 is obtained, can be seen that the pair $\Big(\{\{5Y6N, 5Y\}\}, \{\{5Y6N, 6N\}\}\Big)$ is no longer appearing in automaton $G_{test_{123}}$, and although the pair of states $\Big\{\Big(\{\{1Y2N, 2N\}\}, \{\{1Y2N, 1Y\}\}\Big)$ is still represented in automaton $G_{test_{123}}$, it no longer represents a diagnosability violation, since it only contains events σ such that $\sigma \notin \Sigma_1$. Thus, since $\mathcal{S} = \{\}$, then $G = G_1 \| G_2 \| G_3$, depicted in 3.10 is modularly diagnosable with respect to $P_{\Sigma,\Sigma_{o_1}}$ and $\Sigma_f = \{\sigma_f\}$.



(a)

(b)

(c)

Figure 3.7: Automaton models G_1 (a), G_2 (b), and G_3 (c) of Example 3.3

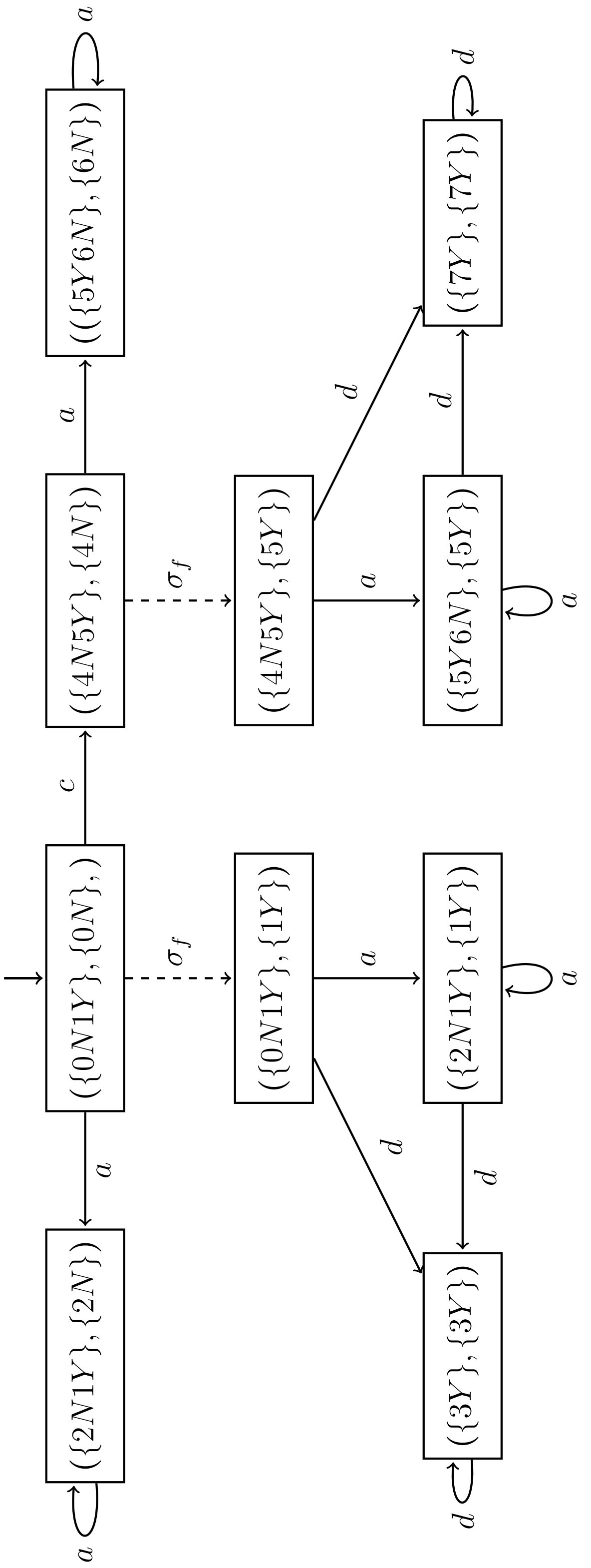
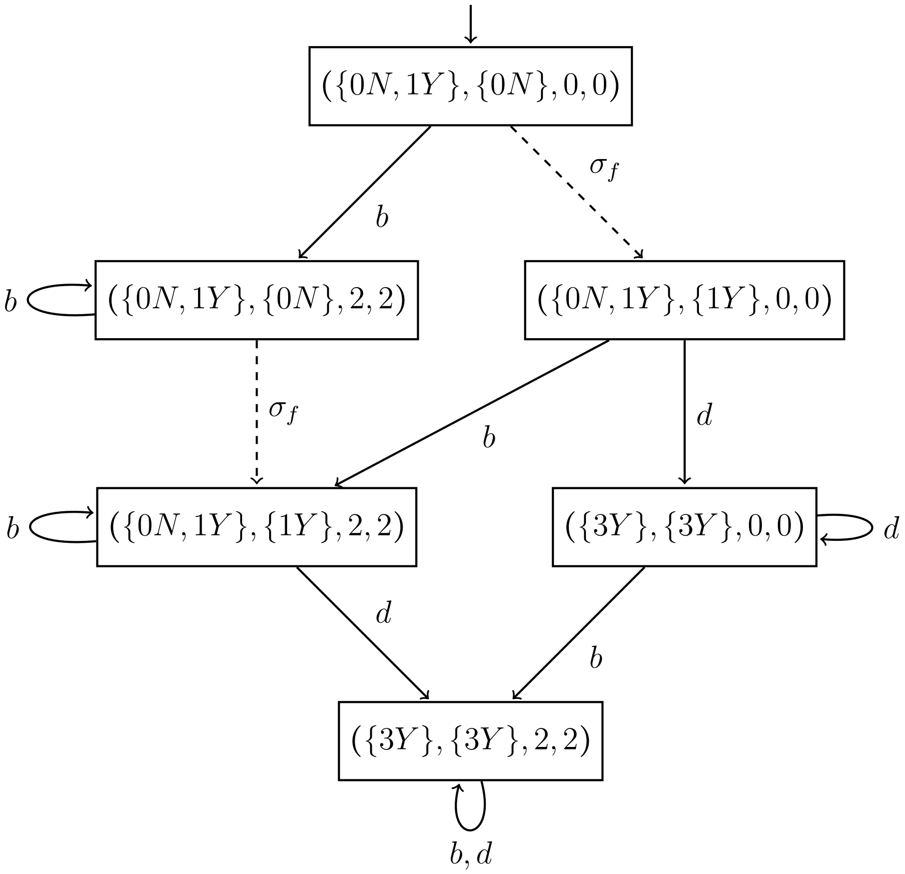


Figure 3.8: Test automata $_{scc_1}$ of Example 3.3



$(\{0N, 1Y\}, \{0N\}, 0, 0)$
b
σ_f
$(\{0N, 1Y\}, \{0N\}, 2, 2)$
$(\{0N, 1Y\}, \{1Y\}, 0, 0)$
σ_f
b
d
$(\{0N, 1Y\}, \{1Y\}, 2, 2)$
$(\{3Y\}, \{3Y\}, 0, 0)$
d
b
$(\{3Y\}, \{3Y\}, 2, 2)$
b, d

Figure 3.9: Test automata $G_{test_{123}}$ of Example 3.3

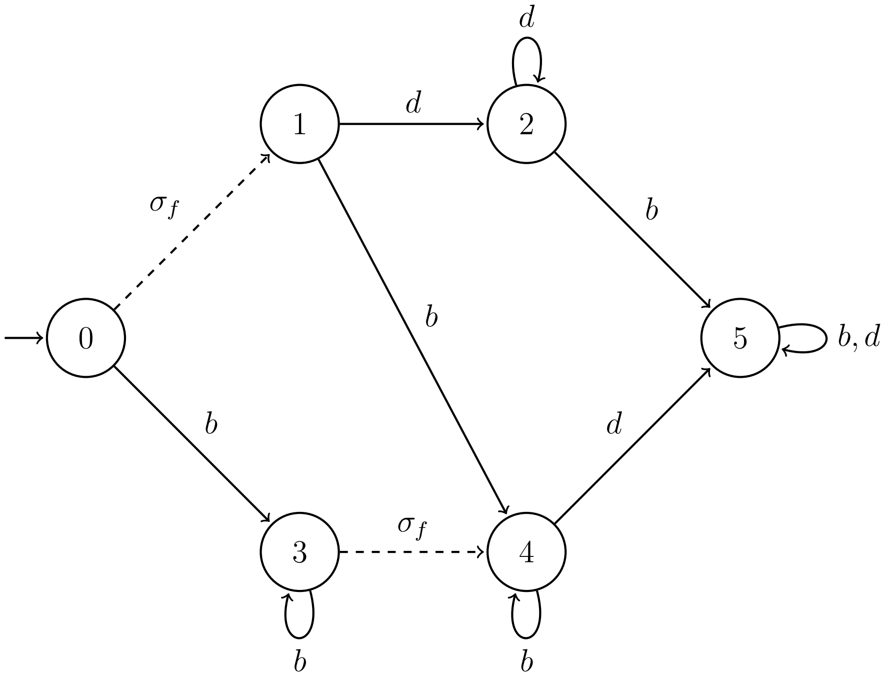


Figure 3.10: $G = G_1 \| G_2 \| G_3$ of Example 3.3

Remark 3.3 (Diagnoser automaton). *Notice that if we compute obs*$(G_{test_{123}}, \Sigma_{o_1})$ *with respect to the second state component we obtain the automaton shown in Fig. 3.11. This automaton can be regarded as the modular diagnoser for the system. Notice that after the first occurrence of events a or c, the diagnoser is sure that the fault has occurred. Therefore, as this picture suggests, online fault diagnosis of DES whose component modules are known can be performed solely by following the event occurrences in the faulty module.*

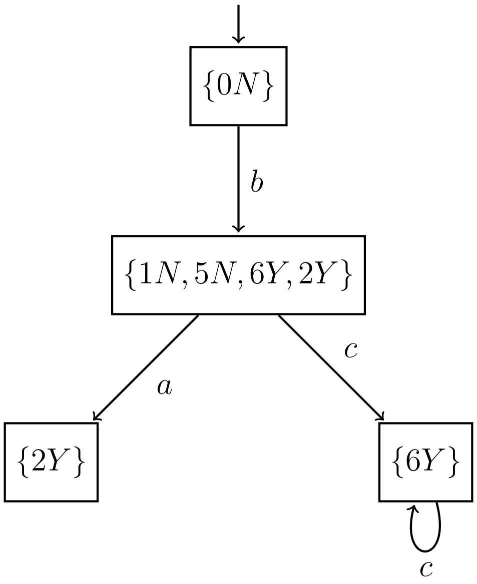


Figure 3.11: Modular diagnoser of Example 3.2.

In CONTANT *et al.* (2006), the local diagnosability of each module is analyzed by using the diagnoser approach, and, for each module $i \in H$ that is not locally diagnosable with respect to its own set of fault events Σ_{f_k} , and observable events, there must exists an F-indeterminate cycle (CONTANT *et al.*, 2006) in the local diagnoser G_{d_k}. The idea of CONTANT *et al.* (2006) is to perform parallel compositions of diagnoser G_{d_k} and the local diagnosers of other modules in order to verify if the F-indeterminate cycle survives. If there exists a set of local diagnosers such that, after building the parallel composition of G_{d_I} and those diagnosers, the F-indeterminate cycle does not survive, the system is modularly diagnosable. Let us now analyze the computational complexity of the Modular Diagnosability Algorithm

(MDA) proposed in CONTANT *et al.* (2006). Assuming that $|X_z|$ is the maximal number of states of one module and $|\Pi_{f_z}$ is the maximal number of fault classes in one module, in worst case, the complexity of constructing the local diagnoser is $\mathcal{O}(2^{|X_z| \times 2^{|\Pi_{f_z}}})$. In the worst case, the parallel composition of all the local diagnosers are built, and thus, the computational complexity of the approach in CONTANT *et al.* (2006) is $\mathcal{O}((2^{|X_z| \times 2^{|\Pi_{f_z}}})^m)$, or, equivalently, $\mathcal{O}(m(2^{|X_z| \times 2^{|\Pi_{f_z}}}))$ Thus, the worst case computational complexity is exponential, which justifies the search for a new algorithm that has polynomial computational complexity.

Remark 3.4 (Computational complexity). *Notice that, in the worst case, it will be necessary to synchronize all non faulty modules with G_{test_1}, i.e., $G_{test_{1,\ldots,m}} = G_{test_1} \|_{i=2}^{m} G_i$. Assuming that the $|X|$ is the largest cardinality among all module state sets, then, in the worst case, the cardinality of the state set of $G_{test_{1,\ldots,m}}$ will be $2^{|X|} \times 2|X| \times |X|^{m-1}$. Thus, the computational complexity to perform the verification proposed here will be $O(2^{|X|} \times |X|^m)$. It is worth noticing that all searches involved in the verification algorithm can be performed in polynomial time.*

Chapter 4

Conclusion and future works

We have presented in this work a new approach for the modular diagnosability problem that has no constraints regarding the observability of the common events, therefore, providing a general formulation for the modular diagnosability problem. In this regard, we have presented a necessary and sufficient condition for modular diagnosability based on a test automaton.

The main contributions of this work as as follows: *(i)* we remove the assumption regarding the need for the events that are common to two or more modules to be observable; *(ii)* we present necessary and sufficient condition for modular diagnosability of regular languages; *(iii)* we propose an automaton-based algorithm for modular diagnosability verification, and; *(iv)* a diagnoser that relies on the observation of the events of the faulty module.

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

(i) Further studies involving the modular diagnoser, the development of lemmas and theorems in order to provide a complete overview of the area;

(ii) Polynomial-time algorithm for the verification of modular diagnosability using G_v (MOREIRA *et al.*, 2011), (in development).

(iii) Search of a minimal module basis, i.e., a set with smallest cardinality formed by the system modules required to ensure modular diagnosability.

Bibliography

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

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

ZAYTOON, J., LAFORTUNE, S. "Overview of fault diagnosis methods for discrete event systems", *Annual Reviews in Control*, v. 37, n. 2, pp. 308–320, 2013.

GENC, S., LAFORTUNE, S. "Distributed diagnosis of place-bordered Petri nets", *IEEE Transactions on Automation Science and Engineering*, v. 4, n. 2, pp. 206–219, 2007.

RAMIREZ-TREVINO, A., RUIZ-BELTRAN, E., ARAMBURO-LIZARRAGA, J., et al. "Structural diagnosability of DES and design of reduced Petri net diagnosers", *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, v. 42, n. 2, pp. 416–429, 2011.

CABRAL, F. G., MOREIRA, M. V., DIENE, O., et al. "A Petri net diagnoser for discrete event systems modeled by finite state automata", *IEEE Transactions on Automatic Control*, v. 60, n. 1, pp. 59–71, 2015.

YOO, T.-S., LAFORTUNE, S. "Polynomial-time verification of diagnosability of partially observed discrete-event systems", *IEEE Transactions on automatic control*, v. 47, n. 9, pp. 1491–1495, 2002.

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

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

CLAVIJO, L. B., BASILIO, J. C. “Empirical studies in the size of diagnosers and verifiers for diagnosability analysis”, *Discrete Event Dynamic Systems*, v. 27, n. 4, pp. 701–739, 2017.

VIANA, G. S., BASILIO, J. C. “Codiagnosability of discrete event systems revisited: A new necessary and sufficient condition and its applications”, *Automatica*, v. 101, pp. 354–364, 2019.

SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., et al. “Failure diagnosis using discrete-event models”, *IEEE transactions on control systems technology*, v. 4, n. 2, pp. 105–124, 1996.

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

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

SU, R., WONHAM, W., KURIEN, J., et al. “Distributed diagnosis for qualitative systems”. In: *Sixth International Workshop on Discrete Event Systems, 2002. Proceedings.*, pp. 169–174. IEEE, 2002.

MYADZELETS, D., PAOLI, A. “Virtual Modules in Discrete-Event Systems: Achieving Modular Diagnosability”. In: *arXiv preprint arXiv:1311.2850*, 2013. Disponível em: <http://arxiv.org/abs/1311.2850>.

SCHMIDT, K. “Verification of Modular Diagnosability with Local Specifications for Discrete-Event Systems”, *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans*, v. 43, n. 5, pp. 1130–1140, 2013.

LI, B., BASILIO, J. C., KHLIF-BOUASSIDA, M., et al. “Polynomial time verification of modular diagnosability of discrete event systems”, *IFAC-PapersOnLine*, v. 50, n. 1, pp. 13618–13623, 2017.

CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to discrete event systems.* New York, NY, Springer Science & Business Media, 2009.

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al. *Introduction to algorithms.* Cambridge, MA, MIT press, 2009.

JIANG, S., HUANG, Z.AND CHANDRA, V., KUMAR, R. “Codiagnosability of discrete event systems revisited: A new necessary and sufficient condition and its applications", *IEEE Transactions on Automatic Control*, v. 46, pp. 1318–1321, 2001.

CHO, H., MARCUS, S. I. "Supremal and maximal sublanguages arising in supervisor synthesis problems with partial observations", *Mathematical Systems Theory*, v. 22, n. 1, pp. 177–211, 1989.