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

DISTRIBUTED SYNCHRONOUS DIAGNOSIS OF DISCRETE-EVENT SYSTEMS

Maria Zeneide Mota Veras Neta

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.

Orientadores: Marcos Vicente de Brito Moreira
Felipe Gomes de Oliveira Cabral

Rio de Janeiro
Novembro de 2018

DISTRIBUTED SYNCHRONOUS DIAGNOSIS OF DISCRETE-EVENT SYSTEMS

Maria Zeneide Mota Veras Neta

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

Examinada por:

Prof. Marcos Vicente de Brito Moreira, D.Sc.

Prof. Felipe Gomes de Oliveira Cabral, D.Sc.

Prof. Antonio Eduardo Carrilho da Cunha, D.Eng.

Prof. José Eduardo Ribeiro Cury, Docteur d'Etat

RIO DE JANEIRO, RJ – BRASIL
NOVEMBRO DE 2018

Veras Neta, Maria Zeneide Mota

Distributed synchronous diagnosis of discrete-event systems/Maria Zeneide Mota Veras Neta. – Rio de Janeiro: UFRJ/COPPE, 2018.

X, 76 p.: il.; 29, 7cm.

Orientadores: Marcos Vicente de Brito Moreira
Felipe Gomes de Oliveira Cabral

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2018.

Referências Bibliográficas: p. 72 – 76.

1. Synchronous fault diagnosis. 2. Distributed diagnosis. 3. Discrete-event systems. I. Moreira, Marcos Vicente de Brito *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Agradecimentos

Agradeço primeiramente a Deus, pela vida e por me permitir chegar até aqui.

Agradeço também a meus pais, Maria Antoneide e Manoel Mariano, por todo carinho e apoio incondicional. Agradeço ao meu irmão Manoel Júnior, pela torcida e incentivo. E, em especial, agradeço a minha irmã Maria Albermária, por estar presente durante toda essa trajetória, e por sempre acreditar em mim.

Agradeço ao meu noivo Leandro de Sá, por todo apoio, paciência e companheirismo. Obrigada por estar sempre ao meu lado e, em especial, obrigada por me ajudar no que acabou aparecendo como uma "disciplina extra": a instabilidade emocional. Sem você o caminho teria sido, sem dúvidas, muito mais difícil.

Agradeço a minha sogra Rosimeri Nery, por toda preocupação e ajuda, principalmente nos finais de semana corridos, com muitas horas dedicadas ao estudo. Sua ajuda foi fundamental em muitos momentos.

Agradeço a toda equipe do Centro de Referência Tecnológica - Claro Brasil, pelo apoio e pela flexibilidade com relação a horário, me permitindo cursar todas as disciplinas e me ausentar nos momentos em que precisei, em especial ao José Silva, Laila, Carolina e Luiza.

Agradeço aos meus orientadores Felipe Cabral e Marcos Moreira, por toda a orientação e horas dedicadas a me ensinar, incentivar e aconselhar sempre que precisei.

Agradeço também à COPPE/UFRJ, seu corpo docente e administração, e a todos aqueles que, de alguma forma, contribuíram para que eu chegasse até aqui.

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

DIAGNÓSTICO SÍNCRONO DISTRIBUÍDO DE SISTEMAS A EVENTOS DISCRETOS

Maria Zeneide Mota Veras Neta

Novembro/2018

Orientadores: Marcos Vicente de Brito Moreira
Felipe Gomes de Oliveira Cabral

Programa: Engenharia Elétrica

Recentemente, o diagnóstico síncrono centralizado e descentralizado de sistemas a eventos discretos foi proposto na literatura. Neste trabalho, propomos uma estratégia de diagnóstico síncrono diferente, denominada diagnóstico síncrono distribuído. Neste esquema, diagnosticadores locais são construídos com base na observação do comportamento livre de falha dos componentes do sistema. Considera-se que esses diagnosticadores locais são agrupados em redes de comunicação e capazes de informar a ocorrência de eventos e sua estimativa de estado atual a outros diagnosticadores locais pertencentes à mesma rede. Os diagnosticadores são implementados considerando um protocolo de comunicação específico, o qual refina a estimativa de estado do comportamento livre de falha dos módulos do sistema, reduzindo, portanto, a linguagem aumentada livre de falha considerada no diagnóstico síncrono. Isso é feito com a adição de condições booleanas para a transposição de transições dos modelos livre de falha dos componentes do sistema, as quais verificam se a ocorrência de um evento observável é possível de acordo com a estimativa do estado atual dos outros diagnosticadores locais. Isso leva à noção de diagnosticabilidade síncrona distribuída. Um algoritmo para verificar a diagnosticabilidade síncrona distribuída com complexidade polinomial no espaço de estados dos modelos dos componentes do sistema é proposto.

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

DISTRIBUTED SYNCHRONOUS DIAGNOSIS OF DISCRETE-EVENT SYSTEMS

Maria Zeneide Mota Veras Neta

November/2018

Advisors: Marcos Vicente de Brito Moreira
Felipe Gomes de Oliveira Cabral

Department: Electrical Engineering

Recently, the centralized and decentralized synchronous diagnosis of discrete-event systems have been proposed in the literature. In this work, we propose a different synchronous diagnosis strategy called distributed synchronous diagnosis. In this scheme, local diagnosers are computed based on the observation of the fault-free behavior models of the system components. It is considered that these local diagnosers are separated into networks, and are capable of communicating the occurrence of events and their current state estimate to other local diagnosers that belong to the same network. The diagnosers are implemented considering an specific communication protocol that refines the state estimate of the fault-free behavior of the system modules, reducing, therefore, the augmented fault-free language considered for synchronous diagnosis. In order to do so, boolean conditions are added to the transitions of the fault-free component models, which check if the occurrence of an observable event is possible according to the current state estimate of other local diagnosers. This leads to the notion of distributed synchronous diagnosability. An algorithm to verify the distributed synchronous diagnosability with polynomial complexity in the state-space of the system component models is proposed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Industrial systems are becoming more complex, with several subsystems or local components operating concurrently, interacting and connected on local networks or through the internet. Such systems are known as cyber-physical systems (CPSs). CPS is a new generation of systems that integrate computing, communication and physical capabilities to control and monitor different processes [1–4].

Several CPSs can be considered as discrete-event systems (DESs), which are dynamic systems, whose evolution is governed by the occurrence of events, and have a discrete state-space [5, 6]. Events are directly associated to state changes in the system and are modeled as an instantaneous occurrence. Examples of events are the command of a controller, the realization of a task by a robot, or a change of position of an autonomous guided vehicle.

Due to the instantaneous occurrence of events and the discrete nature of the state-space of a DES, mathematical formalisms based on differential or difference equations are not suitable for representing these systems. Alternative mathematical formalisms are used in order to represent these characteristics properly. In the literature, there are several ways to describe DESs, and the most common are automata and Petri nets [5–9].

DESs are subject to the occurrence of faults, which are unexpected changes in the system behavior that can cause a reduction in the reliability and performance of the system. In CPSs, that typically are composed of several physical subsystems, the occurrence of a fault in one of these subsystems can alter the behavior of other integrated components, which impacts the whole system behavior. Thus, the detection and isolation of a fault can be a complex task to perform, leading to the need for efficient mechanisms to identify the occurrence of fault events. Moreover, it is also important to analyze the delay bound for diagnosis, which is the maximum number of events that the system can generate after the occurrence of the fault event until the fault is detected, in order to evaluate the efficiency of the diagnosis method.

There are several works in the literature that address the problem of fault diagnosis of DESs [10–26]. In the seminal work SAMPATH *et al.* [10, 11], a diagnosis scheme for systems modeled by finite state automata, is presented. The method based on the construction of a diagnoser that can be used to both detect and isolate fault event occurrences and to verify the diagnosability of a language, *i.e.*, verify if the fault event occurrence can be detected within a bounded delay. However, the implementation of the diagnoser presented in SAMPATH *et al.* [10, 11] is usually avoided since, in the worst-case, the state-space of the diagnoser grows exponentially with the size of the plant model state-space. This is due to the fact that the diagnoser proposed in SAMPATH *et al.* [10, 11] is based on the computation of an observer automaton. Moreover, the diagnoser is based on the global system model, which is in general obtained from the composition of the system components, and whose state-space can also grows exponentially with the number of components.

In the diagnosis scheme proposed in SAMPATH *et al.* [10, 11], it is considered that all information regarding the occurrence of events is available in a centralized way, which is not always the case for systems with a high degree of complexity and with a large number of components. In these cases, if the diagnosis information is physically distributed, diagnosis architectures such as the decentralized [12–14] and distributed [15, 16] are more suitable. In Protocol 3 of DEBOUK *et al.* [12], a decentralized diagnosis scheme where local diagnosers identify the occurrence of a fault event using only local observations of the global system model is presented. In this approach, each local diagnoser has a different set of observable events, and when at least one local diagnoser identifies the fault occurrence, it sends this information to a coordinator, that informs the operator of the system. The notion of decentralized diagnosability has been called codiagnosability [13]. The centralized diagnosis scheme [10, 11, 24] can be seen as a particular case of the decentralized architecture [12–14], and polynomial time algorithms for the verification of codiagnosability, that can also be used to verify diagnosability, have been proposed in the literature [13, 27, 28].

In the distributed diagnosis approach, local diagnosers are computed from the global system model and are based on local observations. In this scheme, differently from the decentralized diagnosis architecture, the local diagnosers exchange information with each other in order to improve the diagnosis decision. The information exchanged between local diagnosers can be associated, for example, with the observation of events and/or their current state estimates. In order to do so, different communication protocols for distributed diagnosis have been proposed in the literature [15–17].

In KEROGLOU and HADJICOSTIS [16], a protocol that allows the exchange of information regarding state estimates at predetermined synchronization points is presented. In KEROGLOU and HADJICOSTIS [16], the global system is modeled by a nondeterministic finite state automaton and local diagnosers are constructed based on different sets of observable events, resulting in different state estimates of the global system. The strategy considered in KEROGLOU and HADJICOSTIS [16] is such that when at least one local diagnoser observes a predetermined number of events, the state estimate of all local diagnosers is sent to a coordinator, that computes the intersection of the sets of state estimates and communicates this information to all local diagnosers. The information exchanged is used by the local diagnosers to refine their diagnosis decision in the next event observation. The fault is detected when at least one local diagnoser identifies its occurrence in the system.

It is important to notice that the exponential growth of the global model with the number of system components is not avoided in the architectures considered in [12–16], since the local diagnosers are computed from the global plant model. In SU and WONHAM [17], a different notion of distributed diagnosis is proposed, where local diagnosers are constructed from the component models of the system in order to avoid the construction of the composed system model. The idea is to infer if a fault event, that is modeled in a local component, has been executed by the composed system. The local diagnosers are computed based on the local behavior models, and exchange information with other diagnosers. Since both the faulty and fault-free behaviors of each local component is considered for the construction of the local diagnosers, a consistency analysis must be carried out. A communication protocol is defined in order to achieve global (resp. local) consistency, *i.e.*, for each local estimate, knowing all other local estimates does not help to further reduce redundant information in the local estimate (resp. knowing adjacent local estimates does not improve the local diagnosis).

Also taking advantage of the modularity of DESs, in DEBOUK *et al.* [29] and CONTANT *et al.* [30] different notions of modular diagnosability are proposed. In these works, it is considered that the fault event is modeled in a unique local component of the system, and the occurrence of the fault event is identified by observing only this local component. It is important to remark that in the modular diagnosis architecture, it is assumed that the component where the fault is modeled, has persistent excitation, *i.e.*, the system does not generate a faulty trace with arbitrarily long length formed only with events that do not belong to the component where the fault is modeled.

Recently, in CABRAL *et al.* [31], CABRAL and MOREIRA [32] and CABRAL [33] a new technique for fault diagnosis of DESs, called synchronous diagnosis, is proposed. In this approach, a synchronized diagnoser based on the fault-free behavior model of the system components is constructed, and the definition of synchronous diagnosability of the language of the system is introduced. The main advantage of this method is to use the modularity of a DES to avoid a diagnosis based on the composed system model, which can have exponential growth in the state-space with the number of system components. In CABRAL and MOREIRA [34] and CABRAL [33], the centralized synchronous diagnosis is extended to the decentralized architecture, and a notion of synchronous codiagnosability is defined. It is also shown in CABRAL and MOREIRA [34] and CABRAL [33], that the centralized synchronous diagnosis is a particular case of the decentralized synchronous diagnosis. Differently from the modular diagnosis scheme [29, 30], where the diagnoser is computed considering only the component where the fault is modeled, in the decentralized synchronous approach, local diagnosers are constructed for all system components. Thus, it is possible to detect the occurrence of the fault event in a local diagnoser based on a component where the fault is not modeled, which is not possible in the modular diagnosis scheme. Therefore, a system not modularly diagnosable can be synchronously diagnosable. It is important to remark that none of the assumptions made in CONTANT *et al.* [30] for modular diagnosis are considered in the synchronous diagnosis scheme [31–36].

The main drawback of the synchronous diagnosis technique is that the observed fault-free language considered for diagnosis can be a larger set than the observed fault-free language of the composed system. Thus, a diagnosable system, according to SAMPATH *et al.* [10], can be not synchronously diagnosable. In order to reduce the growth of the observed fault-free language for synchronous diagnosis, in CABRAL *et al.* [35], the addition of boolean conditions to the transitions of the local diagnosers in the centralized architecture is proposed. These conditions are computed from the fault-free behavior model of the composed system. By considering these conditions, the transitions of the diagnosers that are not associated with a transition in the fault-free behavior model of the composed system are disabled, avoiding some incorrect state estimates. By applying this modification, the augmented observed fault-free language considered in the synchronous centralized diagnosis is reduced, which improves the diagnosis decision. The notion of conditional synchronous diagnosability is introduced in CABRAL *et al.* [35], where it is shown that a system that is not synchronously diagnosable can be conditionally synchronously diagnosable. In CABRAL *et al.* [35], a method for the verification of the conditional synchronous diagnosability of a system is also presented.

In this work, we introduce the distributed synchronous diagnosis architecture. In this scheme, a communication protocol is developed and local diagnosers are constructed from the fault-free behavior model of the system components. For each component, we consider that there exists a corresponding local measurement site, which provides information of local event observations. In addition, we assume that local diagnosers can be connected through networks, and that they can exchange information regarding the observation of events and local state estimates. We also present the notion of distributed synchronous diagnosability, which takes into account the information that can be communicated between local diagnosers. The approach presented in this work, generalizes the conditional centralized synchronous diagnosis method proposed in CABRAL *et al.* [35] to the distributed case. The main advantage of the distributed synchronous diagnosis is the reduction of the fault-free language for synchronous diagnosis, in comparison with the synchronous decentralized diagnosis scheme proposed in CABRAL and MOREIRA [34] and CABRAL [33], leading to a less conservative fault diagnosis.

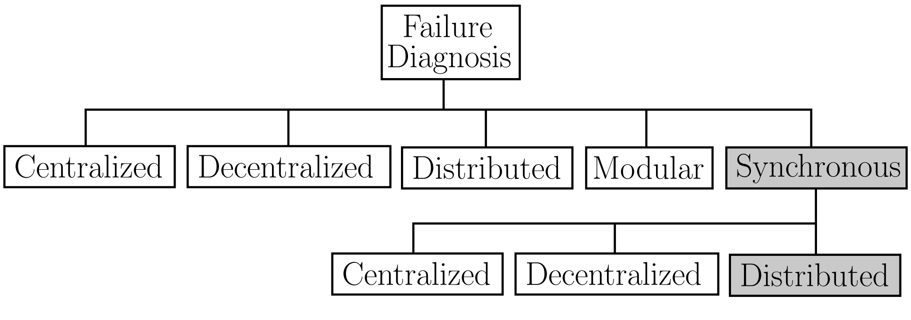


Figure 1.1: Different diagnosis schemes and the synchronous diagnosis approach.

It is important to remark that, in the synchronous distributed diagnosis approach, a fault event can be detected by a local diagnoser whose corresponding local component does not have the fault event modeled. Moreover, differently from other methods proposed in the literature, the same fault event can be modeled in more than one local component of the system. In Figure 1.1, it is presented the most common diagnosis architectures proposed in the literature. Since the synchronous diagnosis architecture cannot be classified as a centralized, decentralized, distributed nor modular architecture, it is highlighted in gray in Figure 1.1 as a new diagnosis framework. In summary, the synchronous diagnosis can be implemented in three different schemes: (*i*) centralized, where a single diagnoser is implemented, and all information regarding the observation of events is sent to the diagnoser by a centralized measurement [31–33, 35]; (*ii*) decentralized, where local diagnosers, based on the fault-free behavior component models are implemented locally, and the fault diagnosis decision is informed to a coordinator [33, 34]; and (*iii*) distributed, where diagnosers are implemented locally and can communicate their event observations and current state estimates in order to refine the diagnosis decision [36], which is the proposal of this work. In summary, in Table 1.1, the main characteristics of each diagnosis architecture depicted in Figure 1.1 is presented.

Table 1.1: Comparison between different diagnosis schemes.

Architecture	**Diagnoser computed from**	**Measurement sites**	**Diagnoser**
Centralized	Global plant model	Centralized measurement	Monolithic diagnoser
Decentralized	Global plant model	Distributed measurement	Local diagnosers
Distributed	Global plant model	Distributed measurement	Local diagnosers with communication
Modular	Faulty component model	Centralized measurement	Single diagnoser
Synchronous centralized	Fault-free component models	Centralized measurement	Single diagnoser
Synchronous decentralized	Fault-free component models	Distributed measurement	Local diagnosers
Synchronous distributed	Fault-free component models	Distributed measurement	Local diagnosers with communication

This work is organized as follows. In Chapter 2, we present some preliminary concepts about DESs. The notions of diagnosability and synchronous diagnosability are presented in Chapter 3. We introduce, in Chapter 4, the distributed synchronous diagnosis architecture, the communication protocol between local diagnosers, and the notion of distributed synchronous diagnosability. An example is used throughout the text to illustrate the results. The conclusions are drawn in Chapter 5.

Chapter 2

Fundamental Concepts of Discrete-Event Systems

A Discrete Event System (DES) is a system whose state-space is described by a discrete set and whose state transitions are driven by the occurrence of events. Due to the nature of a DES, differential or difference equations are not suitable to describe its behavior [5]. Therefore, it is necessary to introduce a different formalism to model and describe these types of systems. In this work, the automaton formalism is considered to model DESs.

In this chapter we present the theoretical background of DESs. In order to do so, we first introduce the notations and definitions regarding languages.

2.1 Languages

Before we introduce the concept of languages, we first present some notations. The set of events of a DES is represented by symbol Σ. The concatenation of events forms a trace, and the language of a system consists of the set of bounded length traces that can be executed by the system. A trace that does not contain any event is called the empty trace and is denoted by ε. The length of a trace s is represented by $\|s\|$ and, the length of the empty trace is equal to zero. In the sequel, we present the formal definition of a language [5].

Definition 2.1 (Language) *A language L defined over Σ, is a set of finite length traces formed with events of Σ.*

Example 2.1 *Consider a system with event set $\Sigma = \{a, b\}$. The language $L = \{\varepsilon, a, ab, aab, abb\}$ is composed of five traces, and the length of the traces of L are $\|\varepsilon\| = 0$, $\|a\| = 1$, $\|ab\| = 2$, $\|aab\| = 3$ and $\|abb\| = 3$.*

Since languages are sets, the usual operations of sets such as union, intersection, difference, and complement, can be applied to languages. Moreover, there are other important operations that can be applied to languages and are presented in the sequel.

2.1.1 Language operations

The Kleene-closure operation over the event set Σ is represented as $\Sigma^\star$, and consists of all finite length traces that are constructed with elements of Σ, including the empty trace ε. Therefore, a language L defined over Σ is a subset of $\Sigma^\star$. This operation can also be applied to languages and is defined as follows.

Definition 2.2 (Kleene-closure) *Let $L \subseteq \Sigma^\star$, the Kleene-closure operation $L^\star$ is given by:*

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

An important operation applied to traces and, consequently, to languages is the concatenation. A trace $s = abba$, for example, can be constructed by the concatenation of two traces ab and ba. Moreover, the empty trace ε is considered the identity element of the concatenation operation and, therefore, the trace ab is the concatenation of ε and ab, *i.e.*, $\varepsilon ab = ab\varepsilon = ab$. This operation can also be formally defined for languages.

Definition 2.3 (Concatenation) *Let $L_a, L_b \subseteq \Sigma^\star$. The concatenation operation L_aL_b is defined as:*

$$L_aL_b = \{s = s_as_b : (s_a \in L_a) \ \textit{and} \ (s_b \in L_b)\}.$$

The concatenation operation, when applied to languages L_a and L_b, generates a set containing the concatenation of each trace of set L_a with each trace of set L_b.

Consider a trace $s = tuv$, where $t, u, v \in \Sigma^\star$, t is a prefix of s, u is a subtrace of s and v if a suffix of s. Notice that, since $t, u, v \in \Sigma^\star$, then ε is always a prefix, a subtrace and a suffix of s. Now, the definition of prefix-closure of a language L can be stated.

Definition 2.4 (Prefix-closure) *Let $L \subseteq \Sigma^\star$, the prefix-closure operation $\overline{L}$ is given by:*

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

The prefix-closure of a language L is the set composed of all prefixes of all traces of L, thus $L \subseteq \overline{L}$. If $L = \overline{L}$, *i.e.*, if all prefixes of all traces of language L are also elements of L, this language is said to be prefix-closed.

Other important operations applied to traces and languages are the natural projection and the inverse projection, presented in the sequel.

Definition 2.5 (Projection) *Consider Σ_s and Σ_l, such that $\Sigma_s \subset \Sigma_l$. The natural projection $P_s^l : \Sigma_l^\star \rightarrow \Sigma_s^\star$ is defined recursively as follows:*

$$P_s^l(\varepsilon) = \varepsilon,$$

$$P_s^l(\sigma) = \begin{cases} \sigma, & \textit{if } \sigma \in \Sigma_s, \\ \varepsilon, & \textit{if } \sigma \in \Sigma_l \setminus \Sigma_s, \end{cases}$$

$$P_s^l(s\sigma) = P_s^l(s)P_s^l(\sigma), \ \textit{for all } s \in \Sigma_l^\star, \sigma \in \Sigma_l,$$

where the operator $\setminus$ represents set difference.

The projection operation $P_s^l(s)$ erases all events $\sigma \in \Sigma_l \setminus \Sigma_s$ from the traces $s \in \Sigma_l^\star$. This operation can be extended to languages by applying the operation to all traces of the language.

The inverse projection operation is defined as follows.

Definition 2.6 (Inverse projection) *The inverse projection* $P_s^{l^{-1}} : \Sigma_s^\star \rightarrow 2^{\Sigma_l^\star}$ *is defined as:*

$$P_s^{l^{-1}}(t) = \{s \in \Sigma_l^\star : P_s^l(s) = t\}.$$

For a given trace $t \in \Sigma_s^\star$, the inverse projection operation $P_s^{l^{-1}}(t)$ generates a set formed of all traces s that can be constructed with the events of Σ_l whose projection P_s^l results in the trace t. This operation can also be extended to languages by applying the operation to all traces that belong to the language.

The language of a DES represents all traces that the system is capable of executing, *i.e.*, it can be used to represent the system behavior. However, mainly in large and complex systems, the representation of the behavior of systems using only their languages is not easy and viable to work with. Therefore, it is necessary to use another formalism to describe DESs to facilitate the manipulation and analysis of systems with complex behavior. In this work we use automata to represent DESs, which are detailed in the next section.

2.2 Automata

An automaton is a device that is capable of representing a language according to well-defined rules, and is formally defined as follows [5, 6].

Definition 2.7 (Automaton) *A deterministic automaton, denoted by G, is a five-tuple:*

$$G = (Q, \Sigma, f, q_0, Q_m),$$

where Q is the set of states, Σ is the set of events, $f : Q \times \Sigma \rightarrow Q$ is the transition function, q_0 is the initial state, and Q_m is the set of marked states.

For the sake of simplicity, when the set of marked states Q_m is the empty set, *i.e.*, $Q_m = \emptyset$, it will be omitted in the representation of the automaton.

We can also define $\Gamma_G : Q \rightarrow 2^{\Sigma}$ as the function of active events of a state of G, *i.e.*, $\Gamma_G(q)$ is the set of all events $\sigma \in \Sigma$ for which the transition function $f(q, \sigma)$ is defined.

Automata can be represented by state transition diagrams, which are oriented graphs capable of reproducing all characteristics defined in G. The state transition diagram is formed of vertices, represented by circles, and edges, represented by arcs. The vertices represent the states of the system, and the edges represent the transitions between these states, which are labeled with events of Σ in order to represent which event correspond to each state transition. The initial state of the automaton is represented by an arc without an origin state. Example 2.2 shows an automaton and its state transition diagram.

Example 2.2 *Consider automaton G with state set $Q = \{0, 1, 2\}$ and event set $\Sigma = \{a, g\}$. The transition function of G is defined as: $f(0, a) = 1$, $f(0, g) = 0$, $f(1, g) = 2$, $f(2, a) = 1$ and, therefore, the active event function is given by: $\Gamma_G(0) = \{a, g\}$, $\Gamma_G(1) = \{g\}$, $\Gamma_G(2) = \{a\}$. The initial state q_0 is 0 and the set of marked states is $Q_m = \{1\}$. The state transition diagram of automaton G is shown in Figure 2.1.*

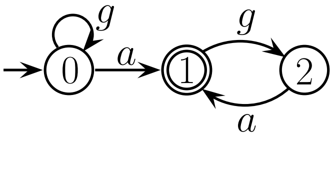


Figure 2.1: State transition diagram of automaton G of Example 2.2.

We also define a path in an automaton G as a sequence $(q_1, \sigma_1, q_2, \ldots, q_{n-1}, \sigma_{n-1}, q_n)$, where $\sigma_i \in \Sigma$, $q_{i+1} = f(q_i, \sigma_i)$, $i = 1, 2, \ldots, n-1$. A path $(q_1, \sigma_1, q_2, \ldots, q_{n-1}, \sigma_{n-1}, q_n)$ is said to be cyclic, if $q_1 = q_n$. The set of states of a cyclic path forms a cycle.

Another important definition is the generated and marked languages of an automaton, presented as follows.

Definition 2.8 (Generated and marked languages) *The generated language of an automaton $G = (Q, \Sigma, f, q_0, Q_m)$ is defined as*

$$\mathcal{L}(G) = \{s \in \Sigma^\star : f(q_0, s) \text{ is defined}\}.$$

The marked language of G is defined as

$$\mathcal{L}_m(G) = \{s \in \mathcal{L}(G) : f(q_0, s) \in Q_m\}.$$

Notice that, in Definition 2.8, the domain of the transition function is considered to be extended, *i.e.*, $f : Q \times \Sigma^\star \to Q$. In addition, notice that for any G such that $Q \neq \emptyset$, $\varepsilon \in \mathcal{L}(G)$.

In general, the language generated by G, $\mathcal{L}(G)$, is composed of all traces that, starting from the initial state, can be concatenated by following the transitions of the state transition diagram. Therefore, since a trace in G is only feasible if all its prefixes are also feasible, the generated language $\mathcal{L}(G)$ is prefix-closed by definition. Moreover, if f is a total function over its domain, then $\mathcal{L}(G) = \Sigma^\star$. In this work, the language generated by G, $\mathcal{L}(G)$, is also referred to as L.

The marked language of G, $\mathcal{L}_m(G)$, is a subset of L, which contains all traces s that reach a marked state, *i.e.*, all traces s such that $f(q_0, s) \in Q_m$. In this case, $\mathcal{L}_m(G)$ is not necessarily prefix-closed, since Q_m is not necessarily equal to Q.

The generated language of an automaton $G = (Q, \Sigma, f, q_0)$ is said to be live if $\Gamma_G(q) \neq \emptyset$ for all $q \in Q$.

In the following, we introduce some operations that can be applied to automata.

2.2.1 Operations on automata

There are several operations that can be used to modify the state transition diagram of a single automaton, or compose two or more automata. These operations are separated into two groups: unary and composition operations.

Unary operations

Unary operations are applied to a single automaton, in order to alter appropriately its state transition diagram, without change the automaton event set. In the sequel we present the definition of two unary operations.

Definition 2.9 (Accessible part) *Consider automaton $G = (Q, \Sigma, f, q_0, Q_m)$. The accessible part of G, Ac(G), is defined as:*

$$Ac(G) = (Q_{ac}, \Sigma, f_{ac}, q_0, Q_{ac,m}),$$

where $Q_{ac} = \{q \in Q : (\exists s \in \Sigma^\star)[f(q_0, s) = q]\}$, $Q_{ac,m} = Q_m \cap Q_{ac}$, and $f_{ac} : Q_{ac} \times \Sigma \rightarrow Q_{ac}$. The transition function f_{ac} corresponds to f restricted to the smaller domain of the accessible states Q_{ac}.

The operation of taking the accessible part of an automaton G erases the states that are not reachable from the initial state q_0 and its related transitions.

It is important to remark that the generated language of an automaton G is not modified with this operation.

Definition 2.10 (Coaccessible part) *Consider automaton $G = (Q, \Sigma, f, q_0, Q_m)$. The coaccessible part of G, CoAc(G), is defined as:*

$$CoAc(G) = (Q_{coac}, \Sigma, f_{coac}, q_{0,coac}, Q_m),$$

where $Q_{coac} = \{q \in Q : (\exists s \in \Sigma^\star)[f(q, s) \in Q_m]\}$, $q_{0,coac} = q_0$ if $q_0 \in Q_{coac}$ and $q_{0,coac}$ is not defined if $q_0 \notin Q_{coac}$, and $f_{coac} : Q_{coac} \times \Sigma \rightarrow Q_{coac}$.

The operation of taking the coaccessible part of automaton G deletes all states q such that a path from q to a marked state does not exist.

It is important to notice that the generated language of G can be reduced by applying the coaccessible part, *i.e.*, $\mathcal{L}(CoAc(G)) \subseteq \mathcal{L}(G)$, while the marked language is not modified.

Composition operations

Composition operations applied to DESs modeled by automata allow us to combine two or more automata, resulting in a single automaton. Moreover, using composition operations it is possible to construct the global system model from the models of its individual components. In the following, we present two important composition operations.

Definition 2.11 (Product composition) *Let* $G_1 = (Q_1, \Sigma_1, f_1, q_{0,1}, Q_{m_1})$ *and* $G_2 = (Q_2, \Sigma_2, f_2, q_{0,2}, Q_{m_2})$ *be two automata. The product of* G_1 *and* G_2 *results in the automaton*

$$G_1 \times G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f_{1\times 2}, (q_{0,1}, q_{0,2}), Q_{m_1} \times Q_{m_2}),$$

where

$$f_{1\times 2}((q_1, q_2), \sigma) = \begin{cases} (f_1(q_1, \sigma), f_2(q_2, \sigma)) & \textit{if } \sigma \in \Gamma_{G_1}(q_1) \cap \Gamma_{G_2}(q_2) \\ \textit{undefined}, & \textit{otherwise.} \end{cases}$$

In the product composition, an event can only occurs in the resulting automaton $G_1 \times G_2$ if it occurs simultaneously in G_1 and G_2. For this reason, the product operation is also known as completely synchronous composition.

Due to the complete synchronization of the product operation, the generated language of $G_1 \times G_2$ is the intersection of the generated languages of the automata used in the composition, *i.e.*, $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$. If $\Sigma_1 \cap \Sigma_2 = \emptyset$, then

$\mathcal{L}(G_1 \times G_2) = \{\varepsilon\}$.

In general, systems are formed by several components that work together and whose event sets have private events, representing the internal behavior of each component, and common events, representing the coupling behavior between the components. The common way to obtain the global model of a system from the models of its components is applying the parallel composition. With this operation, it is possible to maintain the private behavior of each component and capture the synchronism between the components. The formal definition of parallel composition is presented in the sequel.

Definition 2.12 (Parallel composition) *Let* $G_1 = (Q_1, \Sigma_1, f_1, q_{0,1}, Q_{m_1})$ *and* $G_2 = (Q_2, \Sigma_2, f_2, q_{0,2}, Q_{m_2})$ *be two automata. The parallel composition of* G_1 *and* G_2 *results in automaton*

$$G_1 \| G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f_{1\|2}, (q_{0,1}, q_{0,2}), Q_{m_1} \times Q_{m_2}),$$

where

$$f_{1\|2}((q_1, q_2), \sigma) = \begin{cases} (f_1(q_1, \sigma), f_2(q_2, \sigma)) & \textit{if } \sigma \in \Gamma_{G_1}(q_1) \cap \Gamma_{G_2}(q_2); \\ (f_1(q_1, \sigma), q_2) & \textit{if } \sigma \in \Gamma_{G_1}(q_1) \setminus \Sigma_2; \\ (q_1, f_2(q_2, \sigma)) & \textit{if } \sigma \in \Gamma_{G_2}(q_2) \setminus \Sigma_1; \\ \textit{undefined,} & \textit{otherwise.} \end{cases}$$

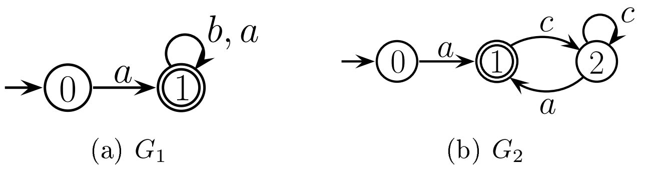
The parallel composition synchronizes the common events of components, *i.e.*, an event $\sigma \in \Sigma_1 \cap \Sigma_2$ can only occur in the resulting automaton $G_1 \| G_2$ if it occurs in G_1 and G_2 simultaneously. On the other hand, private events of each automaton, *i.e.*, the events in $(\Sigma_1 \setminus \Sigma_2) \cup (\Sigma_2 \setminus \Sigma_1)$, can be executed whenever possible in G_1 and G_2.

It is important to notice that if $\Sigma_1 = \Sigma_2$, then $G_1 \| G_2 = G_1 \times G_2$, since all transitions can only occur synchronously.

In order to correctly define the language generated by $G_1\|G_2$, it is necessary to consider the natural projections $P_i = (\Sigma_1 \cup \Sigma_2)^\star \rightarrow \Sigma_i^\star$, for $i = 1, 2$. Based on these projections, the generated language of $G_1\|G_2$ is equal to $\mathcal{L}(G_1\|G_2) = P_1^{-1}(\mathcal{L}(G_1)) \cap P_2^{-1}(\mathcal{L}(G_2))$.

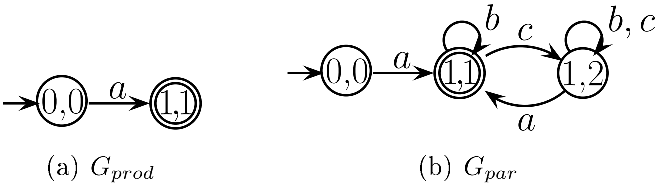
An example of the product and parallel composition operations is presented in the sequel.

Example 2.3 *Consider automata G_1 and G_2 presented in Figure 2.2(a) and 2.2(b), respectively. The event set of G_1 and G_2 are, respectively, $\Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{a, c\}$. Computing the product and parallel compositions of automata G_1 and G_2, we obtain automata $G_{prod} = G_1 \times G_2$ and $G_{par} = G_1\|G_2$, respectively, presented in Figure 2.3. Notice that since the only common event of G_1 and G_2 is event a, i.e., $\Sigma_1 \cap \Sigma_2 = \{a\}$, automaton G_{prod} has only transitions labeled with event a, while in automaton G_{par} it is possible to observe the concurrent behavior of G_1 and G_2, represented by transitions labeled with events b and c.*



(a) G_1 (b) G_2

Figure 2.2: Automata G_1 and G_2 of Example 2.3.



(a) G_{prod} (b) G_{par}

Figure 2.3: Automata G_{prod} and G_{par} of Example 2.3.

In the following, we present an important characteristic that must be taken into account when we use automata for modeling real systems.

2.2.2 Automata with partially observed events

In real systems it is not always possible to detect the occurrence of all events, due to limitations of the sensors used in the system. Events that do not have an associated sensor, such as fault events that do not cause immediate change in sensors readings, are called unobservable events. With the view to representing this, the event set Σ can be partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o is the set of observable events, Σ_{uo} is the set of unobservable events, and $\dot{\cup}$ represents union of disjoint sets. The observed language of an automaton G can be defined as $P_o(\mathcal{L}(G))$, where $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$ is the natural projection.

In order to analyze a system with unobservable events, it is important to define the concept of unobservable reach of a state q, denoted as $UR(q)$. The unobservable reach of a given state $q \in Q$ represents the set of states that can be reached from q after the occurrence of a trace formed with only unobservable events, and it is formally defined as follows.

Definition 2.13 (Unobservable reach) *The unobservable reach of a state $q \in Q$, represented by $UR(q)$, is defined as:*

$$UR(q) = \{y \in Q : (\exists t \in \Sigma_{uo}^\star)[f(q,t) = y]\}. \tag{2.1}$$

The unobservable reach can also be defined for a set of states $B \in 2^Q$ as:

$$UR(B) = \bigcup_{q \in B} UR(q). \tag{2.2}$$

From the definitions of observed language and unobservable reach, it is possible to compute a deterministic automaton that generates the observed language of G with respect to Σ_o, $P_o(\mathcal{L}(G))$. This automaton is called observer of G and is denoted by $Obs(G, \Sigma_o)$.

Definition 2.14 (Observer automaton) *The observer of automaton G with respect to the set of observable events Σ_o, $Obs(G, \Sigma_o)$, is given by:*

$$Obs(G, \Sigma_o) = (Q_{obs}, \Sigma_o, f_{obs}, q_{0,obs}, Q_{m,obs}),$$

where $q_{obs} \subseteq 2^Q$. f_{obs}, $q_{0,obs}$ and $Q_{m,obs}$ are obtained by following the steps of Algorithm 2.1 [5, 37].

Algorithm 2.1 *Observer automaton*

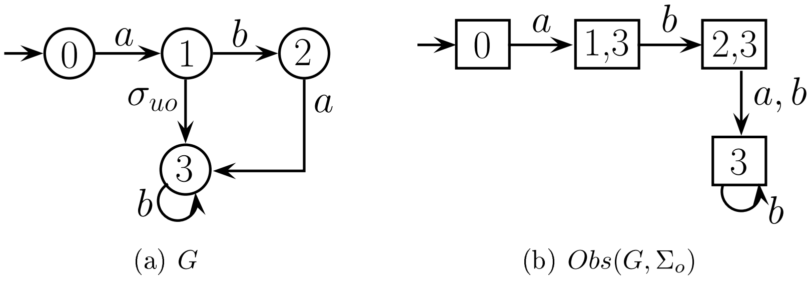
Input: *$G = (Q, \Sigma, f, q_0, Q_m)$, and the set of observable events Σ_o, where $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$.*

Output: *$Obs(G, \Sigma_o) = (Q_{obs}, \Sigma_o, f_{obs}, q_{0,obs}, Q_{m,obs})$.*

```
1: Define $q_{0,obs} := UR(q_0)$, $Q_{obs} := \{q_{0,obs}\}$ and $\widetilde{Q}_{obs} := Q_{obs}$.
2: $\hat{Q}_{obs} := \widetilde{Q}_{obs}$ and $\widetilde{Q}_{obs} := \emptyset$.
3: For each $B \in \hat{Q}_{obs}$:
    3.1: $\Gamma_{obs}(B) := \left(\bigcup_{q \in B} \Gamma_G(q)\right) \cap \Sigma_o$.
    3.2: For each $\sigma \in \Gamma_{obs}(B)$,
             $f_{obs}(B, \sigma) := UR(\{q \in Q : (\exists y \in B)[q = f(y, \sigma)]\})$.
    3.3: $\widetilde{Q}_{obs} := \widetilde{Q}_{obs} \cup f_{obs}(B, \sigma)$.
4: $Q_{obs} := Q_{obs} \cup \widetilde{Q}_{obs}$.
5: Repeat steps 2 to 4 until all accessible part of $Obs(G, \Sigma_o)$ is constructed.
6: $Q_{m,obs} := \{B \in Q_{obs} : B \cap Q_m \neq \emptyset\}$.
```

We present now an example with the observer $Obs(G, \Sigma_o)$ of a system modeled by automaton G.

Example 2.4 *Consider automaton G presented in Figure 2.4(a). The set of events is given by $\Sigma = \{a, b, \sigma_{uo}\}$, where $\Sigma_o = \{a, b\}$ and $\Sigma_{uo} = \{\sigma_{uo}\}$, and the set of states of G is $Q = \{0, 1, 2, 3\}$. The observer of G, $Obs(G, \Sigma_o)$, is shown in Figure 2.4(b). Let us assume that the system has executed trace $s = a\sigma_{uo}b$, then the observed trace* *is $P_o(s) = ab$, where $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$. Notice that the state reached in $Obs(G, \Sigma_o)$ after the observation of trace ab is $\{2, 3\}$, which is the state estimate of G after observation of trace s. As it can be seen in Figure 2.4(b), each state of the observer $Obs(G, \Sigma_o)$ is the state estimate of G after the observation of a trace.*



(a) G

(b) $Obs(G, \Sigma_o)$

Figure 2.4: Automaton G of Example 2.4 (a), and observer automaton of G, $Obs(G, \Sigma_o)$ (b).

2.3 Final comments

In this chapter, the background of DESs, such as the definition of language, operations and the automaton formalism used to represent DESs is presented. Automata with partially observed events, which models systems where not all events are possible to be detected, is also presented.

An example of unobservable event in real systems is the occurrence of a fault and, methods with the aim to detect and isolate its occurrence are needed. In the next chapter, we present the theoretical background of diagnosis for DESs modeled as automata.

Chapter 3

Diagnosability of DESs

Systems are subject to faults that can alter their expected behavior. Thus, it is necessary to define mechanisms that are capable of diagnosing the occurrence of fault events. In this work, a fault event is an unobservable event, since observable events are trivially diagnosed. In this chapter we present some preliminary results regarding diagnosis for DESs. In order to do so, we first introduce the seminal definition of diagnosability of DESs presented in SAMPATH *et al.* [10].

Consider a system modeled by automaton G and consider the language generated by G as $\mathcal{L}(G) = L$. The set of fault events is denoted by Σ_f, where $\Sigma_f \subseteq \Sigma_{uo}$ and, for the sake of simplicity, assume that the set of fault events is composed of only one fault event type, *i.e.*, $\Sigma_f = \{\sigma_f\}$. It is important to remark that in systems with more than one fault event type, each fault event can be considered separately [38] and, therefore, there is no loss of generality in the results presented in this work by making this assumption.

Before presenting the definition of language diagnosability of DESs, we first introduce the notion of faulty and fault-free traces as follows.

Definition 3.1 (Faulty and fault-free traces) *A trace $s \in L$ is a faulty trace if σ_f is one of the events that form s, otherwise, the trace is said to be a fault-free trace.*

The set of all fault-free traces that can be generated by the system is the fault-free language, denoted as L_N, where $L_N \subset L$, and the subautomaton of G that generates L_N is denoted by G_N. Thus, the set of all faulty traces is $L_F = L \setminus L_N$, called faulty language. Now, the definition of language diagnosability, presented in SAMPATH *et al.* [10], can be stated.

Definition 3.2 (Language diagnosability) *Let L and $L_N \subset L$ be the live and prefix-closed languages generated by G and G_N, respectively. L is said to be diagnosable with respect to projection $P_o : \Sigma^\star \to \Sigma_o^\star$ and Σ_f if*

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

From Definition 3.2, it can be seen that L is diagnosable with respect to P_o and Σ_f if, and only if, for all faulty traces st with arbitrarily long length after the occurrence of a fault event, there does not exist a fault-free trace $s_N \in L_N$, such that $P_o(st) = P_o(s_N)$. Thus, if L is diagnosable, then it is always possible to identify the occurrence of a fault event after a bounded number of event occurrences.

In order to verify the diagnosability of L and for implementation of a fault diagnosis scheme, a diagnoser automaton, denoted by G_d can be computed [5, 10, 11]. In order to construct the diagnoser automaton G_d, it is necessary to present the labeler automaton automaton A_l, defined as $A_l = (Q_l, \Sigma_f, f_l, q_{0,l})$, where $Q_l = \{N, F\}$, $f_l(N, \sigma_f) = F$, $f_l(F, \sigma_f) = F$, $q_{0,l} = N$. The state transition diagram of A_l is shown in Figure 3.1.

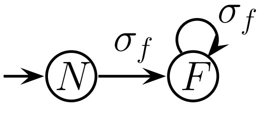


Figure 3.1: Automaton A_l.

Now, consider a system modeled by automaton $G = (Q, \Sigma, f, q_0)$. By computing the parallel composition between automata G and A_l, we obtain automaton $G_l = G \| A_l$. A state $q_l \in G_l$ is labeled with N if it is reached by a fault-free trace, and it is labeled with F if it is reached by a faulty trace. The language generated by G_l is $\mathcal{L}(G_l) = L$. After the construction of automaton G_l, the diagnoser automaton G_d is computed by making the observer of G_l with respect to its observable events, *i.e.*, $G_d = Obs(G_l, \Sigma_o)$. In the following, the diagnoser automaton G_d is formally defined.

Definition 3.3 (Diagnoser automaton) *The diagnoser automaton G_d of the system G, with respect to the faulty event set Σ_f and observable events set Σ_o, is defined as:*

$$G_d = Obs(G_l, \Sigma_o).$$

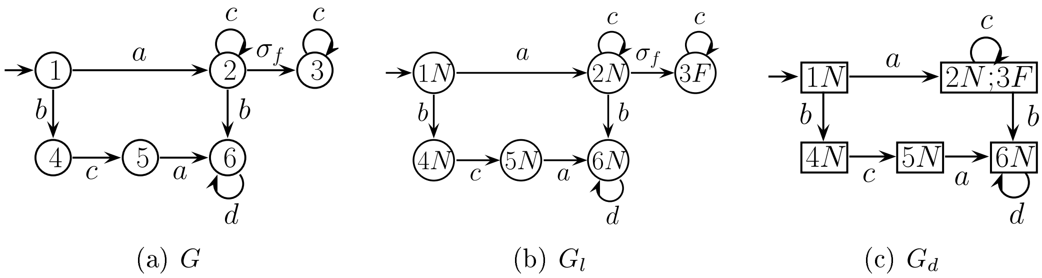
Notice that the generated language of G_d is the natural projection of L, *i.e.*, $\mathcal{L}(G_d) = P_o(L)$. Moreover, we can also notice that the states of G_d are the state estimates of G_l after the observation of a trace. Thus, if G_d reaches a state that has only labels N, it can be affirmed that the fault did not occur, however if G_d reaches a state where all labels are F, the fault certainly occurred and is diagnosed.

The states of G_d that have both labels, N and F, are called uncertain states, since it indicates that the diagnoser is not certain about the fault occurrence status. A cycle formed by uncertain states is called an uncertain cycle. When an uncertain cycle can be associated with at least two cycles in G_l, one with states labeled with N and one with states labeled with F, this cycle is called indeterminate. Thus, the verification of diagnosability of L can be done by searching for indeterminate cycles in G_d, such that if G_d has an indeterminate cycle, then L is not diagnosable, otherwise, L is diagnosable [10, 11, 39].

The example in the sequel is presented in order to illustrate the construction of the diagnoser automaton G_d.

Example 3.1 *Consider the system G depicted in Figure 3.2(a). The set of events is given by $\Sigma = \{a, b, c, d, \sigma_f\}$, where the set of observable events is $\Sigma_o = \{a, b, c, d\}$ and the set of uonbservable event is $\Sigma_{uo} = \{\sigma_f\}$. The fault event set is $\Sigma_f = \{\sigma_f\}$. Automaton $G_l = G \| A_l$ is shown in Figure 3.2(b), and the diagnoser automaton G_d,* *obtained by computing the observer of G_l with respect to its observable event set Σ_o, is shown in Figure 3.2(c).*

Notice that, if the first observed event is b, the fault event has not occurred. However, if the first observed event is a, G_d reaches the uncertain state $\{2N; 3F\}$. If, in the sequel, event b is observed, G_d reaches a fault-free state, confirming the non occurrence of the fault event. However if only event c is observed, G_d remains in the uncertain state $\{2N; 3F\}$. Notice that the uncertain cycle formed by the self-loop labeled with event c in state $\{2N, 3F\}$ is also an indeterminate cycle, since there are two cycles in G, a faulty and a fault-free, associated with the uncertain cycle of G_d, namely, the traces $ac^\star$ and $a\sigma_f c^\star$. Since there exists an indeterminate cycle in G_d, the generated language of G, L, is not diagnosable with respect to $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$ and Σ_f.



(a) G (b) G_l (c) G_d

Figure 3.2: Automaton G (a), automaton G_l (b), and diagnoser automaton G_d (c) of Example 3.1.

Although the diagnoser automaton G_d can be used for the verification of diagnosability of L, its computation is, in general, avoided due to the exponential computational growth of the state-space of G_d with the cardinality of the state-space of the system Q. In order to circumvent this problem, in MOREIRA *et al.* [25, 27] an algorithm for the construction of a verifier automaton is presented, and it is shown that the cardinality of the set of states of the verifier grows polynomially with the set of states of the system.

Besides the monolithic diagnosis architecture presented in SAMPATH *et al.* [10], with the computation of the diagnoser automaton G_d, there exists several diagnosis architectures, such as decentralized, distributed and modular diagnosis in the literature. Recently, a new architecture, called synchronous diagnosis, which takes advantage of the modularity of DESs modeled by automata, has been proposed. This architecture is presented in the sequel.

3.1 Synchronous centralized diagnosability of DESs

In general, systems are composed of several subsystems, modules or components, such that the global plant model G is obtained from the parallel composition of these components, *i.e.*, $G = \|_{k=1}^{r} G_k$, where r is the total number of components, and $G_k = (Q_k, \Sigma_k, f_k, q_{0,k})$, $k = 1, \ldots, r$, are the automaton models of the system components. Let $\Sigma_k = \Sigma_{k,o} \dot{\cup} \Sigma_{k,uo}$ be the set of events of G_k, where $\Sigma_{k,o}$ and $\Sigma_{k,uo}$ are the set of observable and unobservable events of G_k, respectively.

In the most common diagnosis architectures presented in the literature, for example the monolithic, decentralized and distributed architectures, the diagnosis is based on the global model of the system, G, which may result in a large number of states, since the computation of G is obtained from the parallel composition of the system component models, G_k. In order to avoid the computation of the global plant model for diagnosis, in [31–33] a method that uses the modularity of DESs modeled by automata, is proposed.

The diagnosis method presented in [31–33] is called synchronous diagnosis, and is based on the observation of the fault-free behavior of the system components, G_{N_k}, for $k = 1, \ldots, r$, which provides a superset of the state estimate of the fault-free behavior model G_N after the occurrence of an observable event. In this method, local observers that return the online state estimate of G_{N_k}, are constructed. The diagnosis of a fault event is given by using a fault detection logic, which detects the fault event when, in at least one local state observer, the state estimate is equal to the empty set, *i.e.*, when an observable event $\sigma_o \in \Sigma_{k,o}$ that is not feasible in the current state estimate of G_{N_k} is executed.

In Figure 3.3 the architecture of the synchronous diagnosis method is presented. In this approach, there is a unique communication channel and, therefore, an observable event $\sigma_o \in \Sigma_o$ is observable for all system components for which σ_o is defined, *i.e.*, $\Sigma_{i,o} \cap \Sigma_j \subseteq \Sigma_{j,o}$, for any $i, j \in \{1, 2, \ldots, r\}$. The diagnoser consists of the fault-free component model observers implemented concurrently, in addition to the fault detection logic that detects the fault event occurrence.

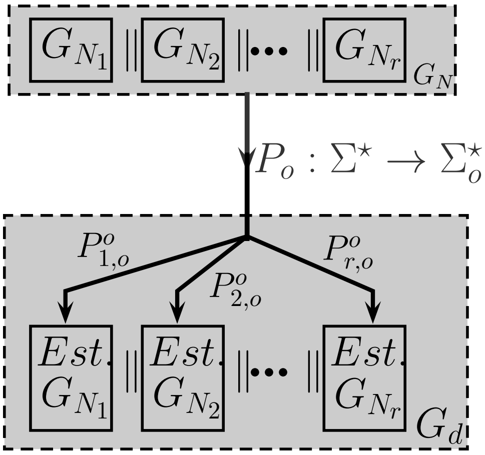


Figure 3.3: Synchronous centralized diagnosis architecture.

In the synchronous diagnosis scheme, the modular structure of the system is taken into account. Thus, in order to provide the current state estimate of the fault-free behavior model of the system, the diagnoser provides the online state estimate of each component model, which are synchronized by the occurrence of the observable events. The resulting language is given by $\mathcal{L}(\|_{k=1}^{r} Obs(G_{N_k}, \Sigma_{k,o})) = \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$, where $P_{k,o}^o : \Sigma_o^\star \rightarrow \Sigma_{k,o}^\star$, $P_{k,o} : \Sigma^\star \rightarrow \Sigma_{k,o}^\star$, and $\Sigma_o = \cup_{k=1}^{r} \Sigma_{k,o}$.

Let L_{N_a} denotes the augmented fault-free language obtained by applying the synchronous diagnosis scheme, *i.e.*, $L_{N_a} = \mathcal{L}(\|_{k=1}^{r} Obs(G_{N_k}, \Sigma_{k,o}))$. Then, we have that $P_o(L_N) \subseteq L_{N_a}$, which indicates that a diagnoser that uses the information provided by the parallel composition of the observers of the system components may represent more observable traces than the system is capable of generating. Thus, the diagnosis based on the observation of the system modules is equivalent to the diagnosis of an augmented system G_a whose generated language is $L_a = L_{N_a} \cup \overline{L}_F$, where L_F is the faulty language of the system [33]. The direct consequence of that, is that a diagnosable system can be not synchronously diagnosable. It occurs when the observation of a fault-free trace in $L_{N_a} \setminus L_N$ is equal to the observation of a faulty trace in L_F. In this case, L_a is not synchronously diagnosable, even if L is diagnosable. In the following we present the formal definition of synchronous diagnosability.

Definition 3.4 (Synchronous diagnosability) *Let L and $L_N \subset L$ be the languages generated by automata G and G_N, respectively, and let $L_F = L \setminus L_N$. Consider a system composed of r modules, such that $G_N = \|_{k=1}^{r} G_{N_k}$, where G_{N_k} is the automaton that models the fault-free behavior of G_k, and let L_{N_k} denote the language generated by G_{N_k}, for $k = 1, \ldots, r$. Then, L is said to be synchronously diagnosable with respect to L_{N_k}, $P_{k,o}^{o} : \Sigma_o^{\star} \to \Sigma_{k,o}^{\star}$, $P_{k,o} : \Sigma^{\star} \to \Sigma_{k,o}^{\star}$, for $k = 1, \ldots, r$, $P_o : \Sigma^{\star} \to \Sigma_o^{\star}$, and Σ_f if*

$$(\exists z \in \mathbb{N})(\forall s \in L_F)(\forall st \in L_F, \|t\| \geq z) \Rightarrow$$
$$(P_o(st) \notin \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))).$$

It is important to remark that if there is no unobservable events in common between the system components, *i.e.*, if $\Sigma_{i,uo} \cap \Sigma_{j,uo} = \emptyset$ for all $i \neq j \in \{1, 2, \ldots, r\}$, the augmented fault-free language L_{N_a} is equal to the observation of the fault-free language of the system $P_o(L_N)$. Thus, if there is no synchronization between unobservable events, the synchronous diagnosability condition is the same as the diagnosability condition presented in SAMPATH *et al.* [10].

The verification of the synchronous diagnosability of the language of a composed system can be carried out by using Algorithm 3.2. Before presenting this algorithm, we show the algorithm used to compute the fault-free behavior models G_{N_k} from the system component models G_k.

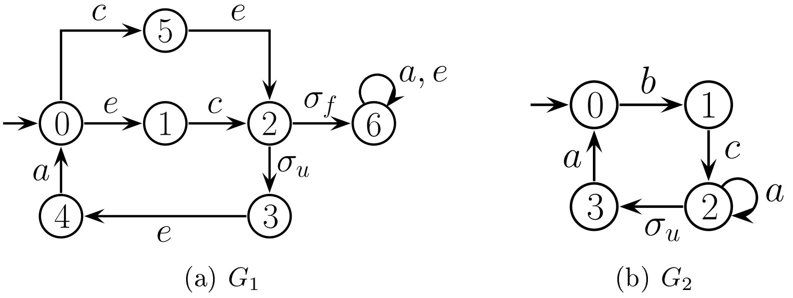
Algorithm 3.1 *Fault-free behavior models of the system components.*

```
Input: $G_k = (Q_k, \Sigma_k, f_k, q_{0,k})$, for $k = 1, \ldots, r$, and $G = (Q, \Sigma, f, q_0)$.
Output: $G_{N_k} = (Q_{N_k}, \Sigma_{N_k}, f_{N_k}, q_{0,N_k})$, for $k = 1, \ldots, r$.

1: Compute automaton $G_N = (Q_N, \Sigma_N, f_N, q_0)$ as follows:
    1.1: Define $\Sigma_N := \Sigma \setminus \Sigma_f$.
    1.2: Construct automaton $A_N$ composed of a single state $N$, that is also its initial state, with a self-loop labeled with all events in $\Sigma_N$.
    1.3: Compute the fault-free automaton $G_N = G \times A_N = (Q_N, \Sigma, f_N, q_{0,N})$.
    1.4: Redefine the event set of $G_N$ as $\Sigma_N$, i.e., $G_N = (Q_N, \Sigma_N, f_N, q_{0,N})$.
2: For all transitions $f_N(q_N, \sigma) = q'_N$ in $G_N$, flag the transitions $f_k(q_k, \sigma) = q'_k$ in $G_k$, for $k = 1, \ldots, r$, where $q_k$ and $q'_k$ are the $k$-th elements of $q_N$ and $q'_N$, respectively.
3: Obtain automata $G'_k$ by erasing from $G_k$ all transitions that are not flagged.
4: Compute automata $G_{N_k} = Ac(G'_k) = (Q_{N_k}, \Sigma_{N_k}, f_{N_k}, q_{0,N_k})$, for $k = 1, \ldots, r$.
5: Redefine the event sets $\Sigma_{N_k} := \Sigma_k \setminus \Sigma_f$, for $k = 1, \ldots, r$.
```

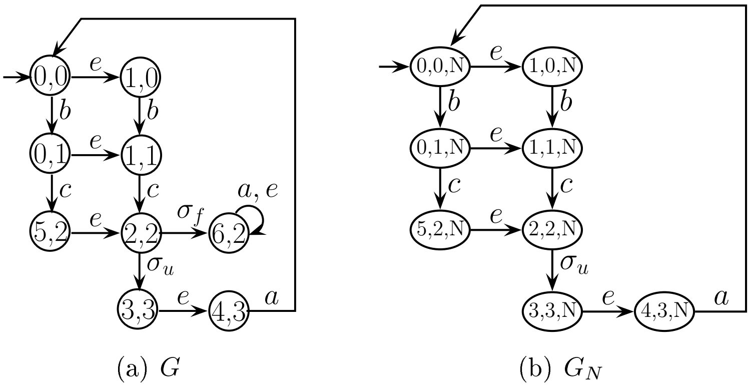
Algorithm 3.1 is necessary since the post-faulty behavior of a component model G_i can interact with another component model G_j, $i \neq j$ where the fault event is not modeled. Therefore, the behavior of G_j after the occurrence of the fault event can be different from its behavior without the occurrence of the fault event, resulting in an automaton G_{N_j} different from G_j, even if the fault event is not modeled in G_j. This problem is illustrated in the following example.

Example 3.2 *Consider the system G composed of two components G_1 and G_2, i.e., $G = G_1 \| G_2$, where G_1 and G_2 are shown in Figures 3.4(a) and 3.4(b), respectively. The event sets of G_1 and G_2 are $\Sigma_1 = \Sigma_{1,o} \cup \Sigma_{1,uo} = \{a, c, e, \sigma_u, \sigma_f\}$, and $\Sigma_2 = \Sigma_{2,o} \cup \Sigma_{2,uo} = \{a, b, c, \sigma_u\}$, respectively, where $\Sigma_{1,o} = \{a, c, e\}$, $\Sigma_{1,uo} = \{\sigma_u, \sigma_f\}$, $\Sigma_{2,o} = \{a, b, c\}$, and $\Sigma_{2,uo} = \{\sigma_u\}$. Automaton G is depicted in Figure 3.5(a), where the event set is given by $\Sigma = \{a, b, c, e, \sigma_u, \sigma_f\}$. Following Step 1 of Algorithm 3.1 we obtain automaton G_N, shown in Figure 3.5(b), which is the automaton that models the fault-free behavior of G. According to G_N it is possible to notice that transition $(2, a, 2)$ of automaton G_2 only can occurs after the occurrence of the fault event σ_f and, therefore, although the fault event is not modeled in automaton G_2,* *the transition* $(2, a, 2)$ *of* G_2 *does not belong to its fault-free behavior. Automata* G_{N_1} *and* G_{N_2}*, obtained by following Step 4 of Algorithm 3.1, are presented in Figure 3.6.*



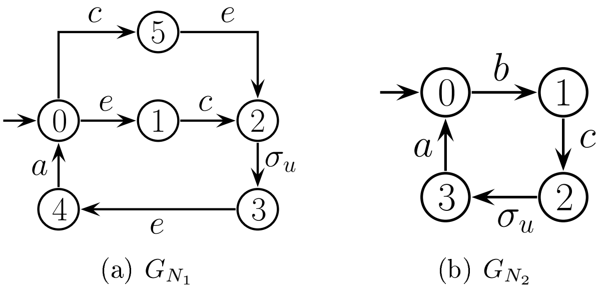
(a) G_1 (b) G_2

Figure 3.4: Automata G_1 and G_2 of Example 3.2.



(a) G (b) G_N

Figure 3.5: Automata G and G_N of Example 3.2.



(a) G_{N_1} (b) G_{N_2}

Figure 3.6: Automata G_{N_1} and G_{N_2} of Example 3.2.

Now we can state the algorithm used to verify the synchronous diagnosability of the language of a system [32, 33].

Algorithm 3.2 *Synchronous Diagnosability Verification*

Input: *System modules G_k, for $k = 1, \ldots, r$, and $G = \|_{k=1}^{r} G_k$.*

Output: *Synchronous diagnosability decision.*

```
1: Compute automaton $G_F$ that models the faulty behavior of $G$, whose marked language is $L_F = L \setminus L_N$, as follows:
    1.1: Set $A_l = (Q_l, \Sigma_f, f_l, q_{0,l})$, where $Q_l = \{N, F\}$, $q_{0,l} = \{N\}$, $f_l(N, \sigma_f) = F$ and $f_l(F, \sigma_f) = F$, for all $\sigma_f \in \Sigma_f$.
    1.2: Compute $G_l = G \| A_l$ and mark all states of $G_l$ whose second coordinate is equal to $F$.
    1.3: Compute the faulty automaton $G_F = CoAc(G_l)$.
2: Compute automata $G_{N_k}$ by following the steps of Algorithm 3.1.
3: Compute automaton $G_N^R = (Q_N^R, \Sigma^R, f_N^R, q_0)$ as follows:
    3.1: Define function $R_k : \Sigma_{N_k} \to \Sigma_{N_k}^R$, as:
         $$R_k(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_{k,o}, \\ \sigma^{R_k}, & \text{if } \sigma \in \Sigma_{k,uo}. \end{cases} \qquad (3.1)$$
    3.2: Construct automata $G_{N_k}^R = (Q_{N_k}, \Sigma_{N_k}^R, f_{N_k}^R, q_{0,N_k})$, $k = 1, \ldots, r$, with $f_{N_k}^R(q_{N_k}, R_k(\sigma)) = f_{N_k}(q_{N_k}, \sigma)$, $\forall q_{N_k} \in Q_{N_k}$ and $\forall \sigma \in \Sigma_{N_k}$.
    3.3: Compute $G_N^R = \|_{k=1}^{r} G_{N_k}^R$.
4: Compute the verifier automaton $G_V^{SD} = (Q_V, \Sigma_V, f_V, q_{0,V}) = G_F \| G_N^R$. Notice that a state of $G_V^{SD}$ is given by $q_V = (q_F, q_N^R)$, where $q_F$ and $q_N^R$ are states of $G_F$ and $G_N^R$, respectively, and $q_F = (q, q_l)$, where $q \in Q$ and $q_l \in \{N, F\}$.
5: Verify the existence of a cyclic path $cl = (q_V^\delta, \sigma_\delta, q_V^{\delta+1}, \ldots, q_V^\gamma, \sigma_\gamma, q_V^\delta)$, where
```

$\gamma \geq \delta > 0$, in G_V^{SD} such that:

$$\exists j \in \{\delta, \delta+1, \ldots, \gamma\} \text{ such that for some } q_V^j,$$
$$(q_l^j = F) \wedge (\sigma_j \in \Sigma).$$

If the answer is yes, then L is not synchronously diagnosable with respect to L_{N_k}, $P_{k,o}^o : \Sigma_o^\star \rightarrow \Sigma_{k,o}^\star$, $P_{k,o} : \Sigma^\star \rightarrow \Sigma_{k,o}^\star$, for $k = 1, \ldots, r$, $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$, and Σ_f. Otherwise, L is synchronously diagnosable.

Notice that the method used to verify the synchronous diagnosability is based on the comparison between the projections of the languages generated by G_F and G_N^R, where G_F models the faulty behavior of the system G and G_N^R is the automaton that models the augmented fault-free behavior considered in the synchronous diagnosis scheme. Thus, the projection in Σ_o of the generated language of G_N^R is equal to the fault-free language observed by the synchronous diagnoser, *i.e.*, $P_o^R(\mathcal{L}(G_N^R)) = L_{N_a}$, where $P_o^R : \Sigma^{R^\star} \rightarrow \Sigma_o^\star$ [32, 33].

In the sequel we present an example that illustrates the application of Algorithm 3.2 for the verification of synchronous diagnosability.

Example 3.3 *Consider automata G_1 and G_2 depicted in Figure 3.4, and automaton $G = G_1 \| G_2$ shown in Figure 3.5(a), where $\Sigma = \{a, b, c, e, \sigma_u, \sigma_f\}$, $\Sigma_o = \{a, b, c, e\}$, $\Sigma_{uo} = \{\sigma_u, \sigma_f\}$, $\Sigma_f = \{\sigma_f\}$, $\Sigma_1 = \{a, c, e, \sigma_u, \sigma_f\}$, $\Sigma_{1,o} = \{a, c, e\}$, $\Sigma_2 = \{a, b, c, \sigma_u\}$, $\Sigma_{2,o} = \{a, b, c\}$. Following the first step of Algorithm 3.2, automaton G_F, shown in Figure 3.7, is constructed, which models the faulty behavior of the system. Applying the Step 2, we compute automata G_{N_1} and G_{N_2}, shown in Figure 3.6 and, in Step 3, automaton G_N^R is constructed by making the parallel composition of $G_{N_1}^R$ and $G_{N_2}^R$. In Figure 3.8 we present automata $G_{N_1}^R$ and $G_{N_2}^R$, while automaton G_N^R is depicted in Figure 3.9. Notice that the gray states of G_N^R and their corresponding transitions labeled with observable events do not belong to G_N, which indicate the growth of the fault-free language considered in the synchronous diagnosis scheme. Finally, applying Step 4 of Algorithm 3.2, we obtain the synchronous verifier automaton*

G_V^{SD}, depicted in Figure 3.10. Since there are no cyclic path in G_V^{SD} labeled with F such that at least one transition is labeled with a non-renamed event, we conclude that L is synchronously diagnosable with respect to L_{N_1}, L_{N_2}, $P_{1,o}^o : \Sigma_o^\star \to \Sigma_{1,o}^\star$, $P_{2,o}^o : \Sigma_o^\star \to \Sigma_{2,o}^\star$, $P_{1,o} : \Sigma^\star \to \Sigma_{1,o}^\star$, $P_{2,o} : \Sigma^\star \to \Sigma_{2,o}^\star$, $P_o : \Sigma^\star \to \Sigma_o^\star$, and Σ_f.

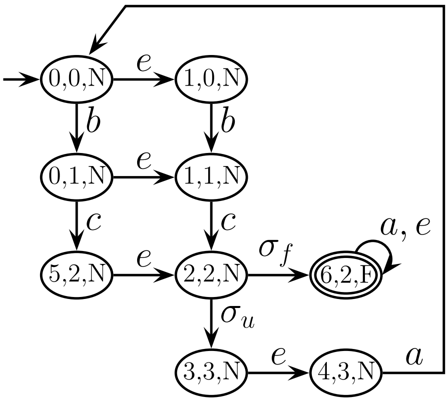
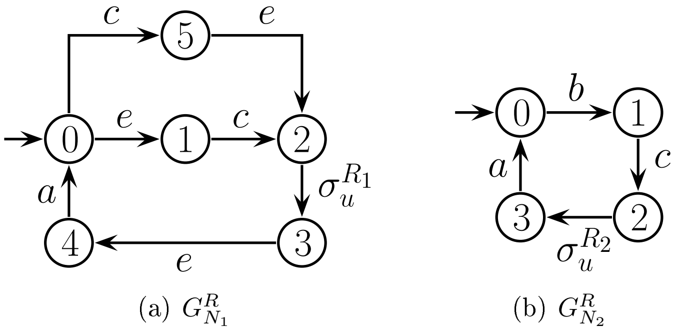


Figure 3.7: Automaton G_F of Example 3.3.



(a) $G_{N_1}^R$

(b) $G_{N_2}^R$

Figure 3.8: Automata $G_{N_1}^R$ and $G_{N_2}^R$ of Example 3.3.

3.1.1 Delay bound for synchronous diagnosis

In CABRAL and MOREIRA [32] and CABRAL [33], a method for the computation of the delay bound for synchronous diagnosis is proposed. The delay bound is the maximum number of events that the system can generate after the occurrence of the fault event until the fault is detected by the diagnoser, and can be used to evaluate the efficiency of the diagnosis method.

Since the fault-free language observed by the synchronous diagnoser can be a larger set that the natural projection of the fault-free language of the system, *i.e.*,

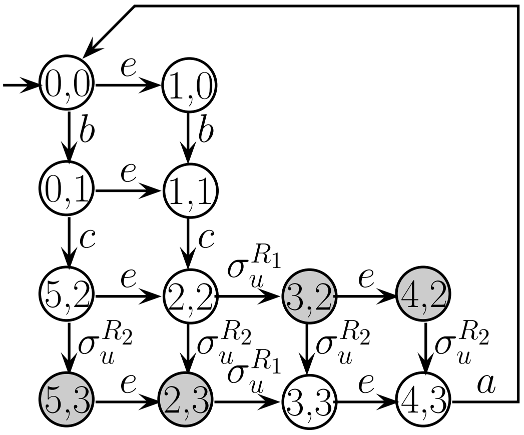


Figure 3.9: Automaton G_N^R of Example 3.3.

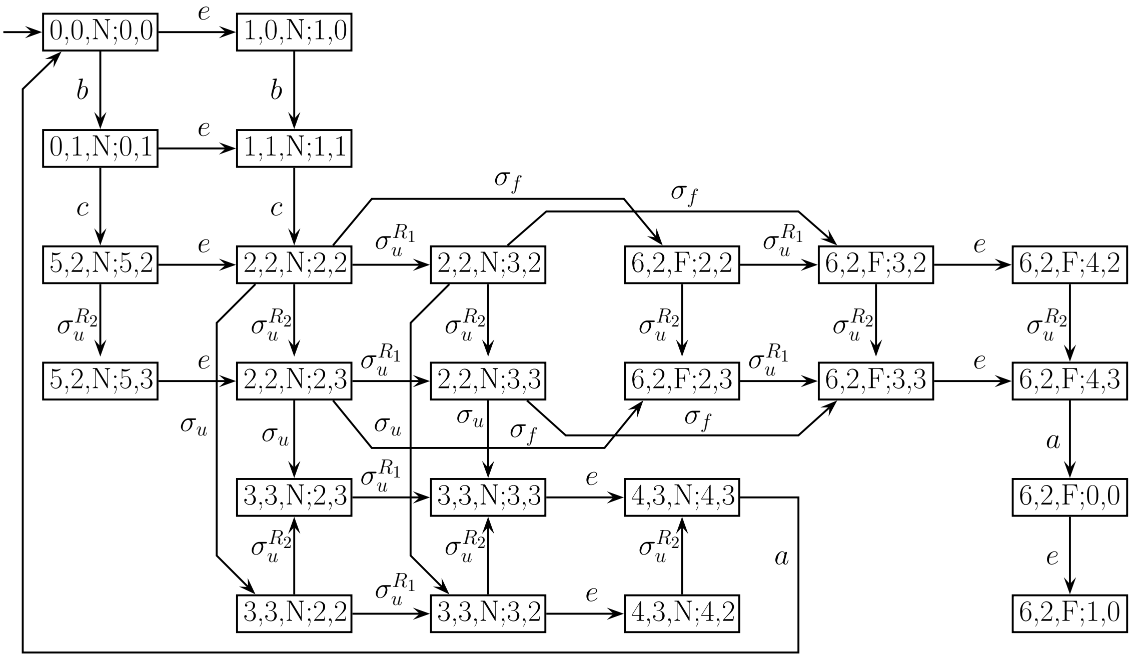


Figure 3.10: Automaton G_V^{SD} of Example 3.3.

$P_o(L_N) \subseteq L_{N_a}$, then, the delay bound for synchronous diagnosis can be larger than the delay bound for the monolithic diagnosis. This fact can cause a decrease in the diagnosis performance and, for this reason, it is important to compute the delay bound $z^\star$ for synchronous diagnosis, in order to evaluate if it can be implemented in a real system.

The method proposed in [32, 33] for the computation of $z^\star$ is a polynomial time algorithm in the size of the composed plant model, adapted from the method presented in TOMOLA *et al.* [40] for the computation of the length of the longest path in a directed acyclic graph (DAG). Before introducing the algorithm, it is necessary to present how to compute the maximum number of events that the system can execute after the occurrence of the fault event σ_f, namely d, for which there exists a faulty trace st and a fault-free trace ω with the same observation, such that $P_o(\omega) \in L_{N_a}$ [32, 33]:

$$
\begin{aligned}
d = \max\{\|t\| : (s \in L_F)(st \in L_F)(P_o(st) = P_o(\omega), \\
P_o(\omega) \in \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k})))\}.
\end{aligned}
$$

It is important to notice that, for the computation of d, we need to search for traces $st \in L_F$ and $P_o(\omega)$, such that $P_o(st) = P_o(\omega)$, and t has maximum length. Since automaton G_V^{SD} represents the faulty traces st and fault-free traces $P_o(\omega) \in L_{N_a}$ with the same projection P_o, then, d can be computed by searching in G_V^{SD} for a path associated with a trace in $\Sigma^\star$ with the largest length after the occurrence of the fault event σ_f.

Now the following algorithm for the computation of d can be stated [32, 33].

Algorithm 3.3 *Computation of d.*

Input: G_V^{SD}.

Output: d.

1: *Compute the graph* $\overline{G}_V^{SD}$ *by eliminating all states that have label* N *and their related transitions from* G_V^{SD} .

2: *Find all strongly connected components of* $\overline{G}_V^{SD}$.

3: *Obtain the acyclic graph* $G_{dag} = (Q_{dag}, \Sigma_{dag}, f_{dag}, q_{0,dag})$, *where* $\Sigma_{dag} = \cup_{k=1}^{r} \Sigma_{N_k}^{R} \cup \Sigma$, *from* $\overline{G}_V^{SD}$ *by shrinking each strongly connected component to a single state [41].*

```
4: $(v_1, v_2, \ldots, v_\eta) \leftarrow$ Topological Sort($G_{dag}$), where $v_j \in Q_{dag}$, for $j = 1, \ldots, \eta$,
   and $\eta = |Q_{dag}|$.
5: Define the weight function $\rho : Q_{dag} \times Q_{dag} \rightarrow \{0, 1\}$, where
   $\rho(v_i, v_j) := \begin{cases} 1, & \textit{if } \exists \sigma \in \Sigma \textit{ such that } f_{dag}(v_i, \sigma) = v_j, \\ 0, & \textit{otherwise.} \end{cases}$
6: For $j = 1, \ldots, \eta$:
   $l(v_j) := \begin{cases} max\{l(v_i) + \rho(v_i, v_j) : (\exists \sigma \in \Sigma_{dag})(f_{dag}(v_i, \sigma) = v_j)\}, \\ 0, \textit{if } \nexists(v_i, \sigma) \in Q_{dag} \times \Sigma_{dag} \textit{ such that } (f_{dag}(v_i, \sigma) = v_j). \end{cases}$
7: $d := max_{j \in \{1, \ldots, \eta\}} l(v_j)$.
```

In Step 1 of Algorithm 3.3, it is computed the graph $\overline{G}_V^{SD}$, from automaton G_V^{SD}, in order to obtain only the states of G_V^{SD} reached after the occurrence of the fault event σ_f. It is important to remark that, for the computation of the delay bound, the system must be synchronously diagnosable according to Definition 3.4. Thus, the automaton verifier G_V^{SD} can have cyclic paths composed of transitions labeled with renamed events. By applying Steps 2 and 3 of Algorithm 3.3, these cyclic paths of $\overline{G}_V^{SD}$ are eliminated by shrinking all its strongly connected components and obtaining the directed acyclic graph G_{dag}.

In Step 4 of Algorithm 3.3, the Topological Sort of G_{dag} is performed, which returns the linked list of vertices of a DAG G, such that if G has an edge (u, v), then, u appears before v in the ordering [42, 43]. In the sequel, in Step 5, a weight function ρ is applied in order to assign weight zero to transitions of G_{dag} labeled with renamed events, and weight one to transitions labeled with events of Σ. In Steps 6 and 7, the number of transitions labeled with events of Σ of the longest path in G_{dag}, d, is computed.

Finally, in order to obtain the delay bound z^*, it is necessary to add to d the occurrence of the event that leads to the detection of the fault event. Therefore, the delay bound for synchronous diagnosis can be computed as

$$z^* = d + 1. \tag{3.2}$$

In the sequel we present an example using Algorithm 3.3 to compute the delay bound for synchronous diagnosis.

Example 3.4 *Consider again automata* G_1 *and* G_2 *depicted in Figure 3.4, and automaton* $G = G_1 \| G_2$ *depicted in Figure 3.5(a). As shown in Example 3.3, the language of the system,* L*, is synchronously diagnosable with respect to* L_{N_1}*,* L_{N_2}*,* $P^o_{1,o} : \Sigma^\star_o \to \Sigma^\star_{1,o}$*,* $P^o_{2,o} : \Sigma^\star_o \to \Sigma^\star_{2,o}$*,* $P_{1,o} : \Sigma^\star \to \Sigma^\star_{1,o}$*,* $P_{2,o} : \Sigma^\star \to \Sigma^\star_{2,o}$*,* $P_o : \Sigma^\star \to \Sigma^\star_o$*, and* Σ_f*. Therefore, the maximum number of transitions that can be executed by the system after occurrence of the fault event* σ_f*, such that exist a faulty trace* st *and fault-free trace* $P_o(\omega)$ *with the same projection, can be computed by applying Algorithm 3.3. From Example 3.3, we obtain automaton* G^{SD}_V *depicted in Figure 3.10. Using* G^{SD}_V *as input of Algorithm 3.3 and following Steps 1, 2 and 3, we obtain automata* $\overline{G}^{SD}_V$ *and* G_{dag}*. In this example,* $\overline{G}^{SD}_V = G_{dag}$*, as shown in Figure 3.11, since there is no strongly connected component to be shrunk. By following Step 4, the Topological Sort of* G_{dag} *is computed, resulting in the graph depicted in Figure 3.12. Applying Steps 5 and 6, the weighting functions* ρ *and* l *are computed, as presented in Figure 3.13. Finally, from Step 7,* $d = 3$ *and, the delay for synchronous diagnosis of the system* G *is* $z^* = 4$*.*

It is important to remark that the delay bound of the classical monolithic diagnoser [10] is also $z^* = 4$*. Thus, although the delay bound can be larger in the synchronous diagnosis method than in the monolithic diagnosis approach, there are systems where the fault event can be diagnosed with the same delay bound in both approaches, even with the growth of the observed fault-free language for synchronous diagnosis.*

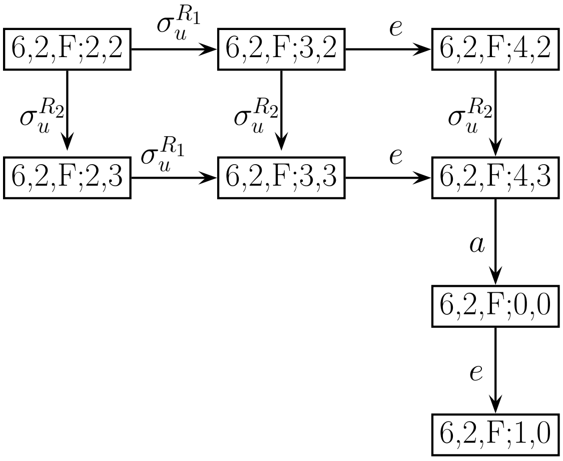


Figure 3.11: Graph $\overline{G}_V^{SD} = G_{dag}$ of Example 3.4.

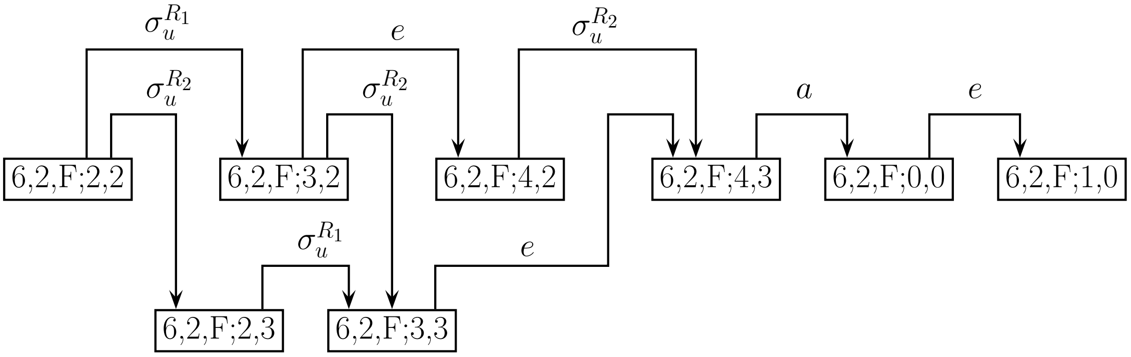


Figure 3.12: Topological Sort of graph G_{dag} of Example 3.4.

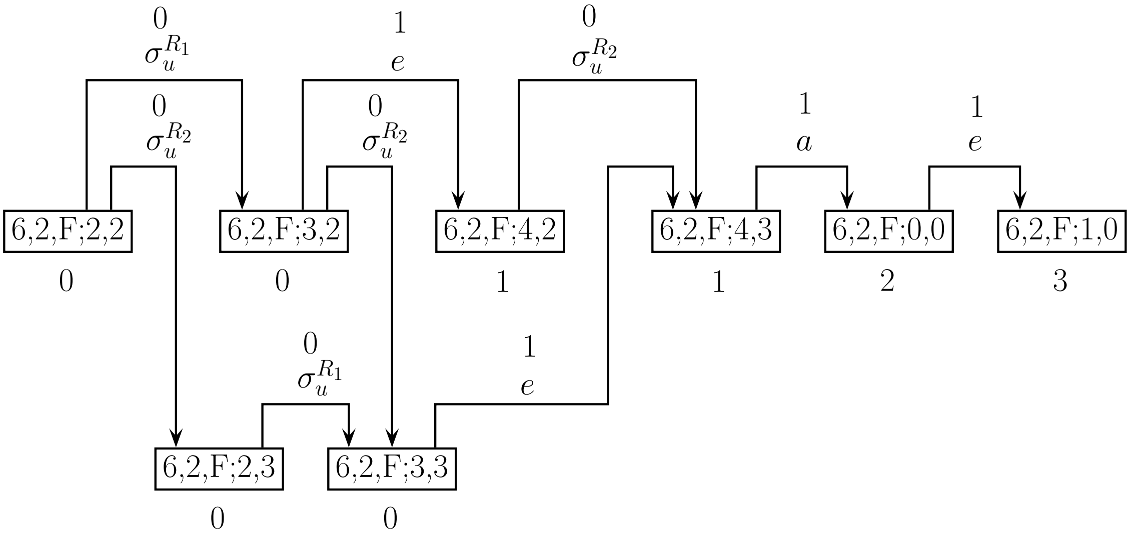


Figure 3.13: Topological Sort of graph G_{dag} of Example 3.4, with values of weighting functions $\rho(v_i, v_j)$ (above the edges) and $l(v_j)$ (below the vertices).

In the next section, the notion of synchronous diagnosis is generalized to the decentralized diagnosis scheme. In this approach, we take into account that all information regarding the occurrence of events is not available in a centralized way, which is usually the case for systems with a large number of components and high degree of complexity.

3.2 Synchronous codiagnosability of DESs

The synchronous decentralized diagnosis scheme, presented in CABRAL and MOREIRA [34] and CABRAL [33], consists in r local diagnosers, where each local diagnoser, constructed based on one component model of the system, has its own set of observable events, and does not communicate with the others local diagnosers. The set of events can, in this case, be partitioned as $\Sigma_i = \hat{\Sigma}_{i,o} \dot{\cup} \hat{\Sigma}_{i,uo}$, for $i = 1, \ldots, r$, where $\hat{\Sigma}_{i,o}$ and $\hat{\Sigma}_{i,uo}$ are, respectively, the set of observable and unobservable events of the local component modeled by automaton G_i. According to this architecture, a fault event is diagnosed when at least one local diagnoser identifies its occurrence and send this information to a coordinator.

The synchronous decentralized diagnosis scheme is based on Protocol 3 of DEBOUK *et al.* [12], where it is assumed that two different sets of observable events can have events in common, *i.e.*, $\hat{\Sigma}_{i,o} \cap \hat{\Sigma}_{j,o}$ is not necessarily equal to the empty set, for $i \neq j$, $i, j \in \{1, \ldots, r\}$. However, it is also assumed in DEBOUK *et al.* [12], that local diagnosers are constructed based on the global model of the system, G, and therefore, may grow exponentially with the number of system components. Differently from DEBOUK *et al.* [12], in CABRAL and MOREIRA [34] and CABRAL [33] local diagnosers are constructed based on the fault-free behavior model of the system components, avoiding the exponential growth with the number of system components.

It is important to notice that one difference between the synchronous centralized scheme presented in section 3.1 and the synchronous decentralized approach, is that in the synchronous decentralized approach an event can be observable to a local diagnoser and unobservable to another local diagnoser, and therefore, $\hat{\Sigma}_{i,o} \subseteq \Sigma_{i,o}$.

In Figure 3.14 we present the architecture of the synchronous decentralized diagnosis scheme. Local diagnosers $\mathcal{D}_k$ are constructed based on the fault-free behavior models of the system components, G_{N_k}, for $k = 1, \ldots, r$. The occurrence of a fault event is identified based on the observation of each component separately, *i.e.*, when an event that is not feasible in the current state estimate of the fault-free behavior of one component is observed, and they send the diagnosis decision to a coordinator.

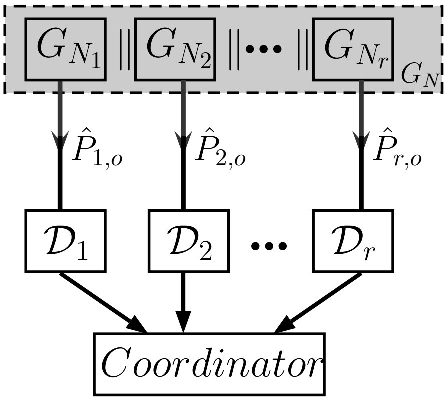


Figure 3.14: Synchronous decentralized diagnosis architecture.

Based on the synchronous decentralized diagnosis scheme, the following definition of synchronous codiagnosability can be stated [33, 34].

Definition 3.5 (Synchronous codiagnosability) *Let $G_N = \|_{k=1}^{r} G_{N_k}$, where G_{N_k} is the automaton that models the fault-free behavior of G_k, and let L_{N_k} denote the language generated by G_{N_k}, for $k = 1, \ldots, r$, where r is the number of system components. Assume that there are r local sites with projections $\hat{P}_{k,o} : \Sigma^\star \rightarrow \hat{\Sigma}_{k,o}^\star$, $k = 1, \ldots, r$. Then, L is said to be synchronously codiagnosable with respect to L_{N_k}, $\hat{P}_{k,o}$, and Σ_f if*

$$(\exists z \in \mathbb{N})(\forall s \in L_F)(\forall st \in L_F, \|t\| \geq z) \Rightarrow$$
$$(\exists k \in \{1, 2, \ldots, r\})(\hat{P}_{k,o}(st) \notin \hat{P}_{k,o}(L_{N_k})).$$

Let $\hat{L}_{N_a}$ denotes the augmented fault-free language obtained by applying this synchronous approach. Then, the augmented fault-free language for synchronous decentralized diagnosis is given by $\hat{L}_{N_a} = \cap_{k=1}^{r} \hat{P}_{k,o}^{o^{-1}}(\hat{P}_{k,o}(L_{N_k}))$. It was also shown in [33, 34] that $P_o(L_N) \subseteq L_{N_a} \subseteq \hat{L}_{N_a}$. Therefore, the synchronous codiagnosability implies in synchronous diagnosability, which ultimately implies in the diagnosability of L. However, the converse is not always true, *i.e.*, L can be synchronously diagnosable and not synchronously codiagnosable. But, there is a condition which ensures that if L is synchronously diagnosable, then L is also synchronously codiagnosable. This condition is presented in the following corollary:

Corollary 3.1 *Let $\hat{\Sigma}_{i,uo} \cap \hat{\Sigma}_{j,o} = \emptyset$ for all $i, j \in \{1, \ldots, r\}$. Then, L is synchronously codiagnosable with respect to L_{N_k}, $\hat{P}_{k,o} : \Sigma^\star \to \hat{\Sigma}_{k,o}^\star$, and Σ_f, if, and only if, L is synchronously diagnosable with respect to L_{N_k}, $P_{k,o}^o : \Sigma_o^\star \to \Sigma_{k,o}^\star$, $P_{k,o} : \Sigma^\star \to \Sigma_{k,o}^\star$, for $k = 1, 2$, $P_o : \Sigma^\star \to \Sigma_o^\star$, and Σ_f.*

Proof. See [33].

The verification of synchronous codiagnosability of the language L can be done by applying Algorithm 3.2, replacing the renaming function R_k (Equation (3.1)), shown in Step 3, by the new renaming function $\hat{R}_k : \Sigma_{N_k} \to \hat{\Sigma}_{N_k}^R$ defined as follows:

$$\hat{R}_k(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \hat{\Sigma}_{k,o} \\ \sigma^{R_k}, & \text{if } \sigma \in \hat{\Sigma}_{k,uo} \end{cases}. \tag{3.3}$$

After replacing function R_k (Equation (3.1)) with function $\hat{R}_k$ (Equation (3.3)) in Algorithm 3.2, the synchronous codiagnosability verifier automaton G_V^{SC} is computed. The synchronous codiagnosability is verified by searching for cyclic paths in G_V^{SC} formed by states with the label F and non-renamed events.

In the following example we illustrate the synchronous codiagnosability verification of the language of a DES.

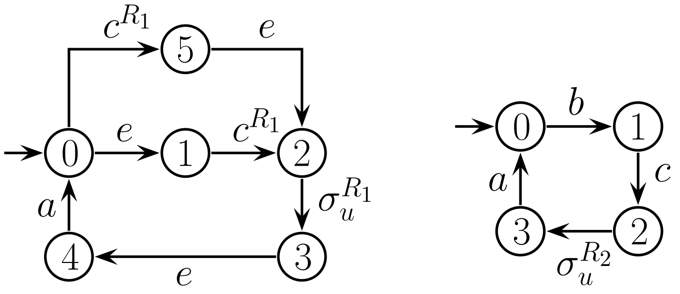
Example 3.5 *Consider again the system G composed of two components, G_1 and G_2, such that $G = G_1 \| G_2$. Automata G_1 and G_2 are depicted, respectively, in*

Figures 3.4(a) and 3.4(b), and automata G and G_N are shown in Figures 3.5(a) and 3.5(b), where G_N is the automaton that models the fault-free behavior of G. Differently from Example 3.3, in this example we consider that event c is unobservable to local diagnoser 1, such that, $\Sigma_1 = \hat{\Sigma}_{1,o} \cup \hat{\Sigma}_{1,uo} = \{a, b, e, \sigma_u, \sigma_f\}$, where $\hat{\Sigma}_{1,o} = \{a, e\}$ and $\hat{\Sigma}_{1,uo} = \{c, \sigma_u, \sigma_f\}$. The set of fault events is composed of only one event, $\Sigma_f = \{\sigma_f\}$, and the event set of automaton G_2 is $\Sigma_2 = \hat{\Sigma}_{2,o} \cup \hat{\Sigma}_{2,uo} = \{a, b, c, \sigma_u\}$, where $\hat{\Sigma}_{2,o} = \{a, b, c\}$, and $\hat{\Sigma}_{2,uo} = \{\sigma_u\}$.

Following Steps 1 and 2 of Algorithm 3.2, automata G_F, G_{N_1}, and G_{N_2} are computed and can be seen in Figures 3.7, 3.6(a) and 3.6(b), respectively. In the sequel, it is necessary to rename the unobservable events of G_{N_1} and G_{N_2} according to Equation (3.3), resulting in automata $\hat{G}^R_{N_1}$ and $\hat{G}^R_{N_2}$, shown in Figures 3.15(a) and 3.15(b), respectively. In order to model the fault-free language considered in the synchronous decentralized diagnosis approach, we compute automaton $\hat{G}^R_N$ by making the parallel composition between $\hat{G}^R_{N_1}$ and $\hat{G}^R_{N_2}$ in Step 3 of Algorithm 3.2. Automaton $\hat{G}^R_N$ is depicted in Figure 3.16. Since event c is unobservable to local diagnoser D_1, then language $\hat{L}_{N_a}$ is a larger set than language L_{N_a} of Example 3.3 where the synchronous centralized verification is presented. Indeed, it can be seen by comparing automaton G^R_N, in Figure 3.9, with automaton $\hat{G}^R_N$, in Figure 3.16. Notice that the growth of the fault-free language considered in the synchronous decentralized scheme is represented by gray states, that are states that do not exist in G_N, and their related transitions labeled with observable events.

The verifier automaton G^{SC}_V, depicted in Figure 3.17, is constructed by following Step 4 of Algorithm 3.2. Since there are no cyclic paths in G^{SC}_V labeled with F such that at least one transition is labeled with a non-renamed event, then L is synchronously codiagnosable with respect to L_{N_1}, L_{N_2}, $\hat{P}_{1,o}$, $\hat{P}_{2,o}$, and Σ_f.

Since the fault-free language $\hat{L}_{N_a}$ considered for synchronous decentralized diagnosis can be a larger set than the language considered for synchronous centralized diagnosis L_{N_a}, then, it is also important to compute the delay bound for synchronous decentralized diagnosis. It can be computed by following the steps



(a) $\hat{G}_{N_1}^R$ (b) $\hat{G}_{N_2}^R$

Figure 3.15: Automata $\hat{G}_{N_1}^R$ and $\hat{G}_{N_2}^R$ of Example 3.5.

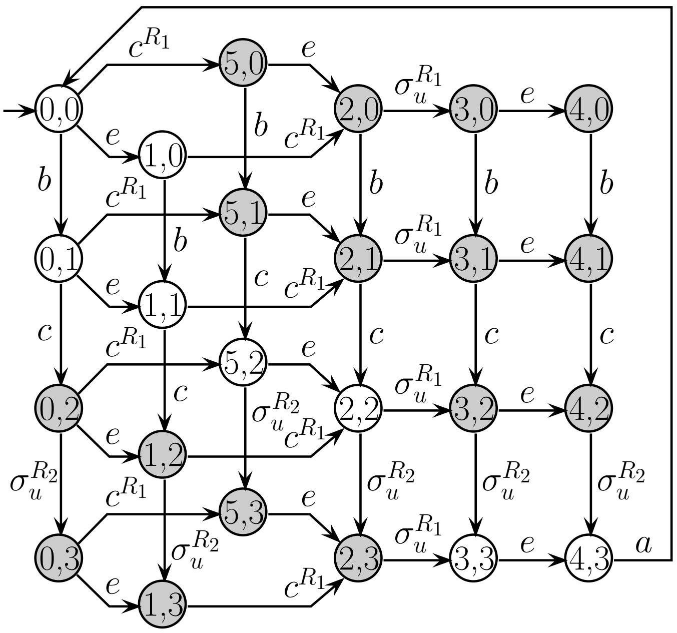


Figure 3.16: Automaton $\hat{G}_N^R$ of Example 3.5.

of Algorithm 3.3, replacing the input G_V^{SD} by automaton G_V^{SC}. Notice that, due to $L_{N_a} \subseteq \hat{L}_{N_a}$, the delay bound for synchronous decentralized diagnosis can be larger than the delay bound for synchronous centralized diagnosis.

In the following example, the delay bound for synchronous decentralized diagnosis for the system G of Example 3.5 is computed.

Example 3.6 *Let us consider again the system $G = G_1 \| G_2$ presented in Example 3.5. The maximum number of events that can be executed by the system, d, after occurrence of σ_f, such that exist a faulty trace st and fault-free trace $P_o(\omega)$ with the same observation, can be computed by using the verifier automaton G_V^{SC}, depicted*

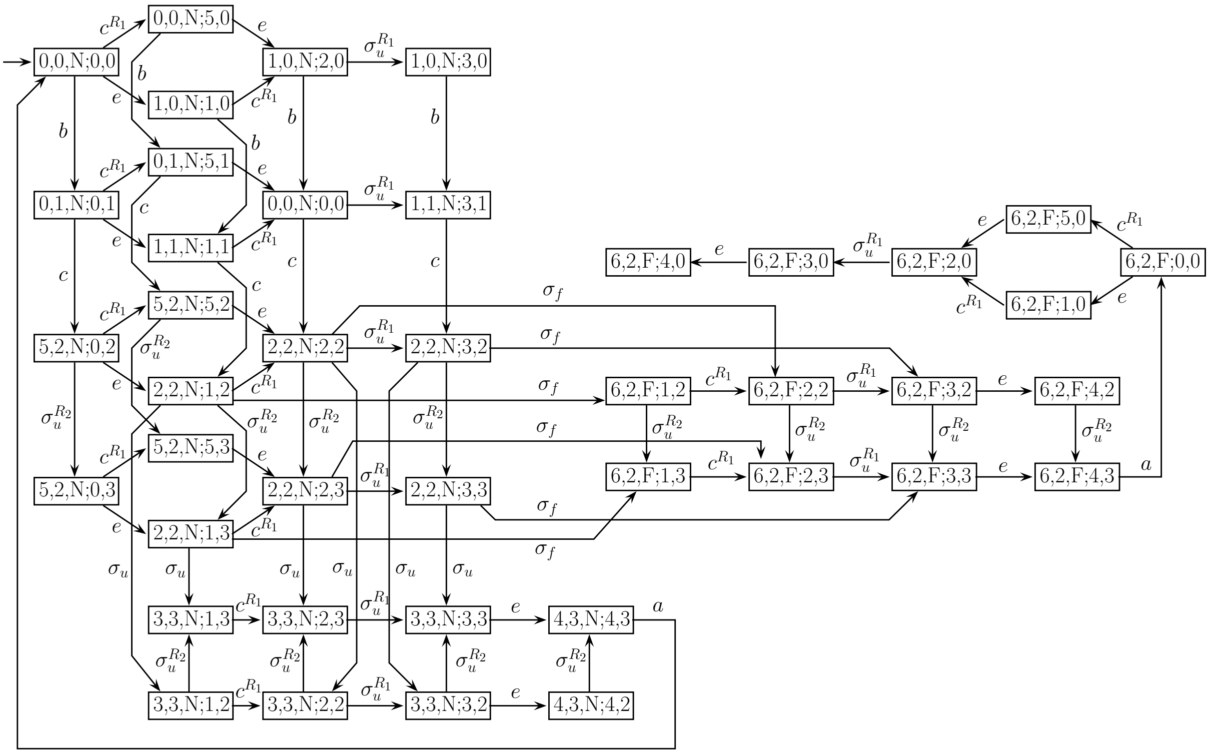


Figure 3.17: Automaton G_V^{SC} of Example 3.5.

in Figure 3.17, as input of Algorithm 3.3, Following Steps 1, 2 and 3 of Algorithm 3.3, we can see that $\overline{G}_V^{SD} = G_{dag}$, which is shown in Figure 3.18. By Step 4, the Topological Sort of G_{dag} is computed, which is depicted in Figure 3.19. Applying Steps 5 and 6, we obtain the weighting functions ρ and l, presented in Figure 3.20. Then, with Step 7, d is computed, resulting in $d = 4$ and, finally, with Equation (3.2), the delay bound for synchronous decentralized diagnosis is $z^* = 5$.

Comparing automaton G_N^R of Example 3.3 with automaton $\hat{G}_N^R$ of Example 3.5, we can see the the fault-free language considered for synchronous decentralized scheme is larger than the fault-free language for the synchronous centralized scheme. For this reason, the resulting delay bound for synchronous decentralized diagnosis can also be larger than the delay bound for synchronous centralized diagnosis, which indeed occurs in the system G considered.

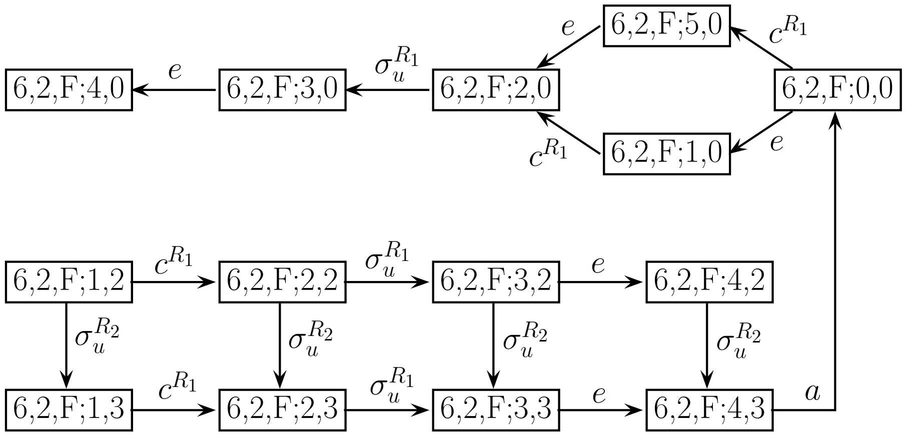


Figure 3.18: Graph $\overline{G}_V^{SD} = G_{dag}$ of Example 3.6.

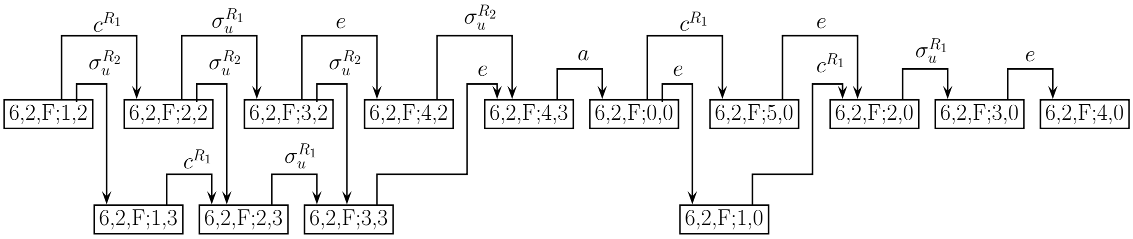


Figure 3.19: Topological Sort of graph G_{dag} of Example 3.6.

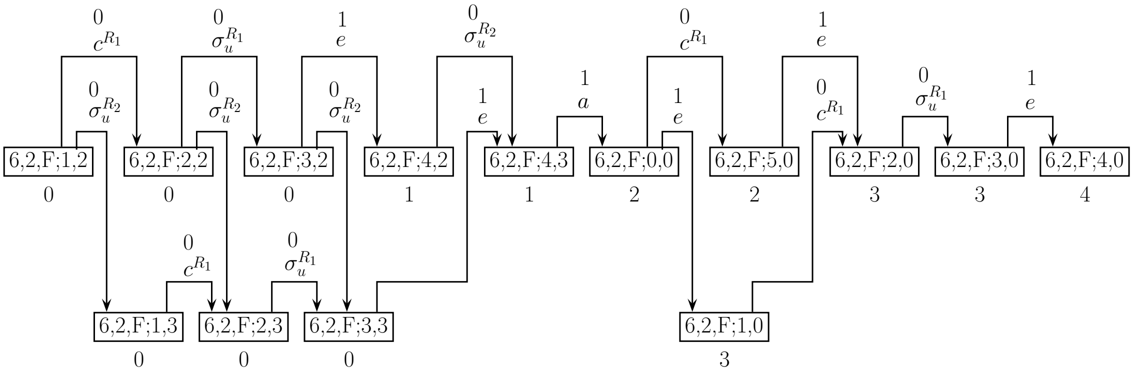


Figure 3.20: Topological Sort of graph G_{dag} of Example 3.4, with values of weighting functions $\rho(v_i, v_j)$ (above the edges) and $l(v_j)$ (below the vertices).

3.3 Final comments

In this chapter, the problem of fault diagnosis for DES modeled by automata is introduced, presenting the classical definition of diagnosability of SAMPATH *et al.* [10]. A new architecture that takes advantage of the modularity of DESs, called centralized synchronous diagnosis, is also presented. This scheme is generalized to the decentralized case, called synchronous decentralized diagnosis scheme. Both synchronous diagnosis approaches lead to different notions of diagnosability, namely synchronous centralized diagnosability and synchronous codiagnosability.

In the next chapter, a new diagnosis method, called distributed synchronous diagnosis is proposed. In this scheme, local diagnosers can exchange information with each other in order to reduce the size of the augmented language considered for diagnosis.

Chapter 4

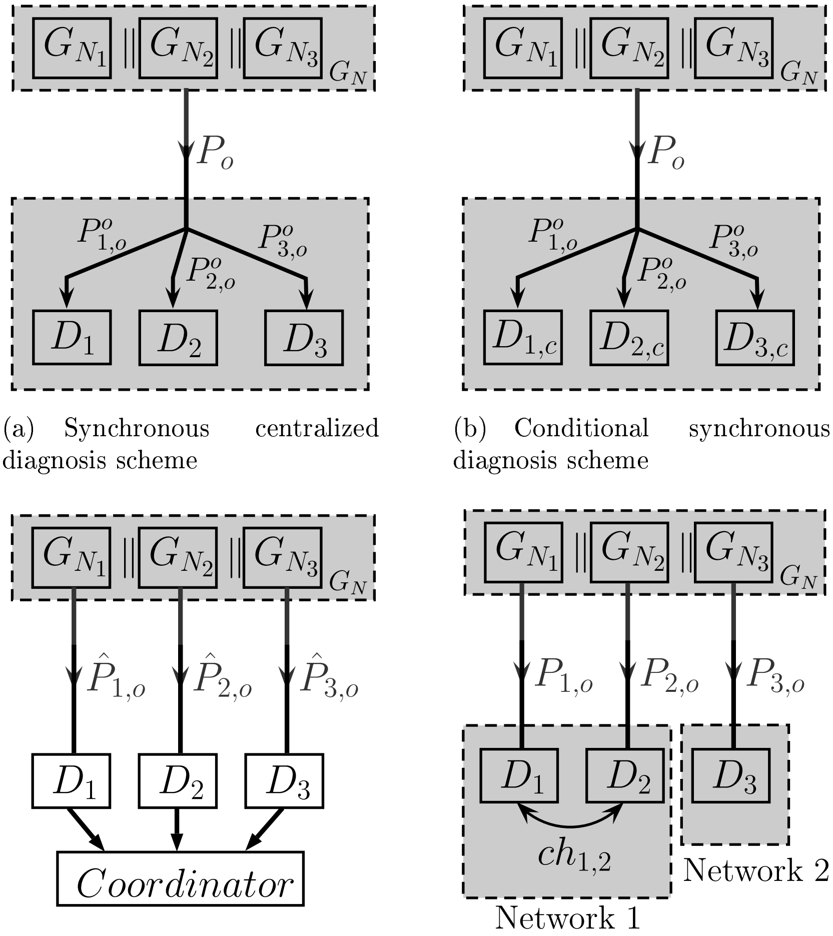
Distributed Synchronous Diagnosability of DESs

In [31–33], a method for fault diagnosis of DES based on the observation of the fault-free behavior of the system components is presented, called synchronous centralized diagnosis method. In this scheme, an event is observable for all system components for which it is defined. The diagnoser consists of local state estimators of the fault-free component models, providing the online state estimate of each fault-free component model, which are naturally synchronized by the observable events executed by the system, as presented in Section 3.1.

In CABRAL *et al.* [35], a modification in the synchronous centralized diagnosis method with the view to refining the diagnosis decision, is proposed. This modification is done by adding boolean conditions to the local diagnosers transitions, based on the fault-free model of the global plant. These conditions are implemented to prevent fault-free traces that cannot occur in the system to be considered as belonging to the estimated fault-free observed behavior. With this refinement, the augmented fault-free language considered in the synchronous centralized diagnosis method can be reduced, improving the synchronous diagnosis.

However, in CABRAL *et al.* [35] it is considered that all information associated with event observations and state estimates is available in a centralized way, which is not always true in systems with a high degree of complexity and with large number of local components. In these cases, architectures such as the decentralized and distributed are more suitable. Thus, in order to improve the synchronous diagnosis for systems where the information is not available in a centralized way, we propose in this work a distributed synchronous diagnosis approach. As in Section 3.2, we consider that the global plant model is composed of r modules, *i.e.*, $G = \|_{k=1}^{r} G_k$, and, associated with each module G_k, for $k \in \{1, \ldots, r\}$, there is a local diagnoser D_k constructed from the fault-free behavior model G_{N_k}. The main difference between the synchronous decentralized and distributed schemes is that in the decentralized approach, local diagnosers are based only on the local observations of the system components, while in the synchronous distributed method, local diagnosers can exchange information regarding the observation of events and local state estimates [36]. This information can be used to refine the diagnosis decision based on the strategy proposed in CABRAL *et al.* [35], by adding conditions to the fault-free component models and reducing the augmented fault-free language considered for synchronous diagnosis.

In Figure 4.1 we show the synchronous diagnosis schemes applied to a system composed of three local components: (*i*) the synchronous centralized scheme, where the diagnoser consists in observers of the fault-free component models implemented concurrently; (*ii*) the conditional synchronous scheme, where conditions associated to state estimate of the global system model are included in the local observers of the synchronous centralized scheme; (*iii*) the synchronous decentralized scheme, where local diagnosers are constructed based on the fault-free component models, each one with its own set of observable events, and a coordinator indicates the fault occurrence; (*iv*) the distributed synchronous scheme, where local diagnosers are separated into networks, allowing the exchange of information between them in order to improves the synchronous diagnosis, which is the proposal of this work.



(a) Synchronous centralized diagnosis scheme

(b) Conditional synchronous diagnosis scheme

(c) Synchronous decentralized diagnosis scheme

(d) Distributed synchronous diagnosis scheme

Figure 4.1: Comparison between the synchronous diagnosis architectures: (a) the synchronous centralized scheme; (b) the conditional synchronous scheme; (c) the synchronous decentralized scheme; (d) the distributed synchronous scheme.

In this chapter, we introduce the distributed synchronous diagnosis scheme for DESs, first presenting its architecture with more details. Then, we introduce the distributed synchronous diagnosis method and explain how it can improve the fault diagnosis. In the sequel we present a communication protocol that allows the exchange of information between local diagnosers. Finally, the notion of distributed synchronous diagnosability is presented, and an algorithm for the verification of distributed synchronous diagnosability, that has polynomial complexity in the size of the system components, is proposed. An example is used throughout the text to illustrate the results.

4.1 Architecture

In the synchronous distributed diagnosis approach it is considered that local diagnosers can commmunicate with each other through a network and, therefore, the construction of each local diagnoser takes into account the communication between diagnosers that belong to the same network.

Figure 4.2 depicts the distributed synchronous diagnosis scheme for a system composed of five modules and two networks. In this setting, there are two networks of local diagnosers: (*i*) a network composed of diagnosers D_1, D_2, and D_3, with communication channels $ch_{1,2}$, $ch_{1,3}$, and $ch_{2,3}$; and (*ii*) a network composed of diagnosers D_4 and D_5, and communication channel $ch_{4,5}$. It is considered that each component G_k has a local measurement site, denoted as LM_k, that communicates the observation of events directly to diagnoser D_k. In this configuration, a local diagnoser connected in a network works as a node in the net, being capable of sending and receiving information from all local diagnosers in this network, regarding the observation of events and state estimates. Therefore, observable events associated with diagnoser D_k, of a given module G_k, is formed by the events that are directly observed by the local measurement site LM_k, and the events whose observation are communicated to D_k from the other local diagnosers in the same network. It is important to remark that, in this work, it is considered that each diagnoser belongs to a unique network.

The event set of each module G_k can be partitioned as $\Sigma_k = \Sigma_{k,o} \dot{\cup} \Sigma_{k,uo}$, where $\Sigma_{k,uo} = \Sigma_k \setminus \Sigma_{k,o}$ is the set of unobservable events for local diagnoser D_k, *i.e.*, is the set of events whose occurrence cannot be detected locally by LM_k, or communicated to D_k by any other local diagnoser D_i, $i \neq k$. Thus, the set of observable events of G_k in the distributed synchronous diagnosis scheme can be defined as $\Sigma_{k,o} = (\cup_{i=1}^{r} \Sigma_o^{i,k}) \cap \Sigma_k$, where $\Sigma_o^{i,k}$, $i \neq k$, denotes the set of observable events that can be communicated from local diagnoser D_i to local diagnoser D_k, and $\Sigma_o^{k,k}$, is the set of events whose observations are directly sent to local diagnoser D_k from local measurement site LM_k, as shown in Figure 4.2. Notice that if two local diagnosers

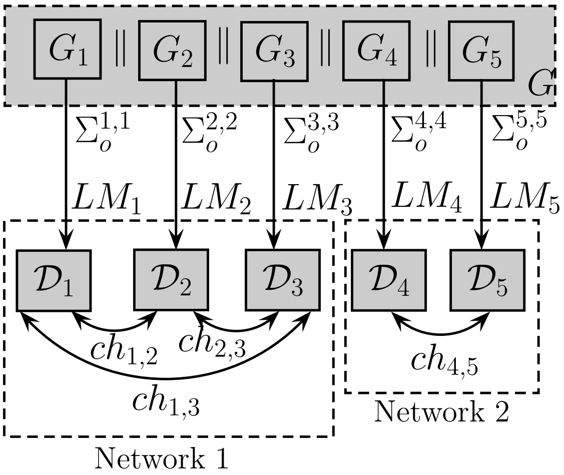


Figure 4.2: The distributed synchronous diagnosis scheme for a system composed of five modules and two networks.

D_i and D_k are in different networks, then $\Sigma_o^{i,k} = \Sigma_o^{k,i} = \emptyset$.

Before introducing the synchronous diagnosis method, we make one last assumption: the communication between local diagnosers is supposed to be ideal, *i.e.*, there is no communication delays and/or package losses.

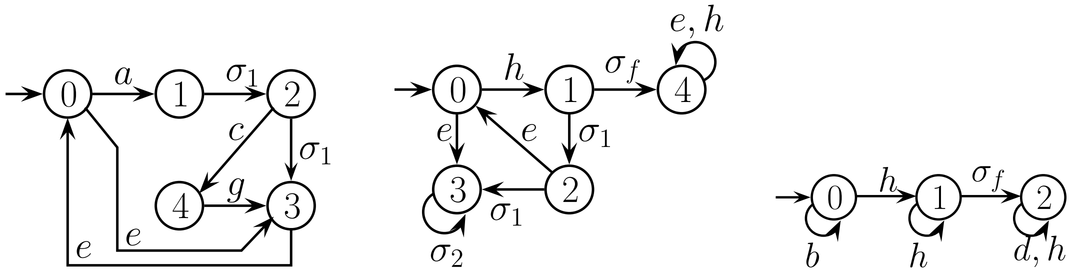
4.2 Distributed synchronous diagnosis method

When local diagnosers connected in a communication network exchange only the information regarding event occurrences, the distributed synchronous diagnosis scheme becomes equivalent to a decentralized synchronous diagnosis architecture, as proposed in [33, 34] and presented in Section 3.2, where $\hat{\Sigma}_{k,o} = \Sigma_{k,o}$. The main drawback of this strategy is the growth of the fault-free language considered for diagnosis, which is represented in the augmented automaton G_N^R. The following example illustrate this problem.

Example 4.1 *Let the system be composed of three modules* G_1*,* G_2 *and* G_3*, presented in Figure 4.3. The event sets of each module are, respectively,* $\Sigma_1 = \Sigma_{1,o} \dot{\cup} \Sigma_{1,uo} = \{a, c, e, g, \sigma_1\}$*,* $\Sigma_2 = \Sigma_{2,o} \dot{\cup} \Sigma_{2,uo} = \{e, h, \sigma_1, \sigma_2, \sigma_f\}$*, and* $\Sigma_3 = \Sigma_{3,o} \dot{\cup} \Sigma_{3,uo} = \{b, d, h, \sigma_f\}$*, where* $\Sigma_{1,o} = \{a, c, e, g\}$*,* $\Sigma_{1,uo} = \{\sigma_1\}$*,* $\Sigma_{2,o} = \{e, h\}$*,* $\Sigma_{2,uo} = \{\sigma_1, \sigma_2, \sigma_f\}$*,* $\Sigma_{3,o} = \{b, d\}$*, and* $\Sigma_{3,uo} = \{h, \sigma_f\}$*. The set of fault events is*

$\Sigma_f = \{\sigma_f\}$. *The composed plant model,* $G = G_1 \| G_2 \| G_3$, *and the fault-free behavior model,* G_N *are shown in Figures 4.4 and 4.5, respectively. The fault-free behavior model of the components* G_1, G_2 *and* G_3, *denoted by* G_{N_1}, G_{N_2} *and* G_{N_3} *respectively, are represented in Figure 4.6. Following the method presented in [33, 34], the unobservable events of* G_{N_1}, G_{N_2} *and* G_{N_3} *are renamed and* G_N^R *is obtained from the parallel composition of the resulting automata,* $G_{N_1}^R$, $G_{N_2}^R$ *and* $G_{N_3}^R$, *depicted in Figure 4.7. Automaton* $G_N^R = G_{N_1}^R \| G_{N_2}^R \| G_{N_3}^R$ *is shown in Figure 4.8.*

The gray states of G_N^R *and the associated transitions do not exist in the fault-free behavior model of the system,* G_N. *These states, and their associated transitions labeled with observable events, represent the growth of the fault-free observed language for synchronous decentralized diagnosis compared to the classical diagnosis method of SAMPATH* et al. *[10]. Moreover, the faulty trace* $h\sigma_f(eh)^z$, *for* $z \in \mathbb{N}$, *has the same observation in* $\Sigma_o = \Sigma_{1,o} \cup \Sigma_{2,o} \cup \Sigma_{3,o} = \{a, b, c, d, e, g, h\}$ *than the fault-free trace* $h\sigma_1^{R_2}(eh\sigma_1^{R_2})^z$ *generated by automaton* G_N^R, *which shows that the composed system,* G, *is not synchronously codiagnosable.*



(a) G_1 (b) G_2 (c) G_3

Figure 4.3: Automata G_1, G_2, and G_3 of Example 4.1.

When we consider the communication of the occurrence of observable events and state estimates between local diagnosers, it is possible to reduce the fault-free language for synchronous diagnosis. This can be done by checking if the occurrence of an observable event is possible according to the state estimate of the local diagnosers. The following example illustrate how this communication can be used to improve the synchronous diagnosis decision.

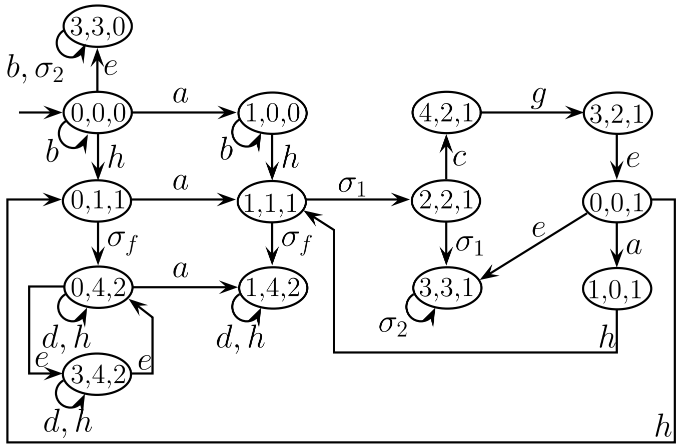


Figure 4.4: Automaton G of Example 4.1.

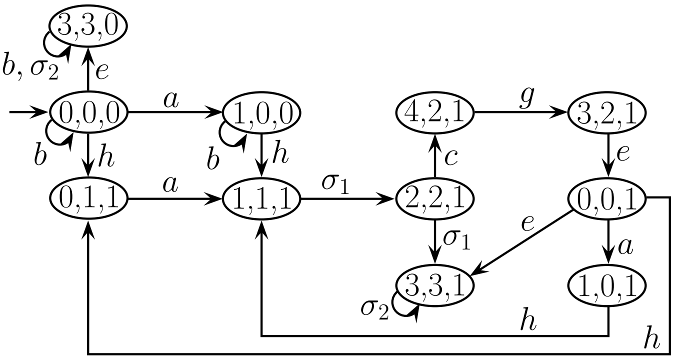
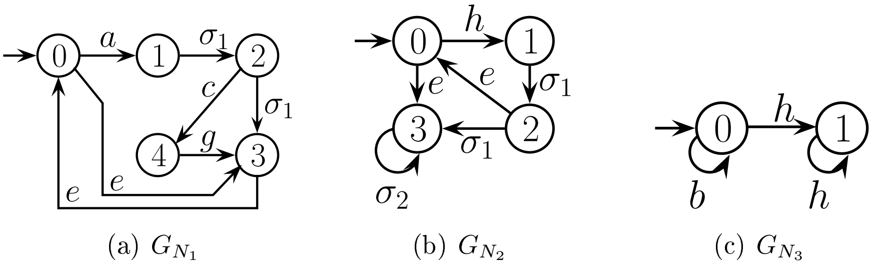


Figure 4.5: Automaton G_N of Example 4.1.

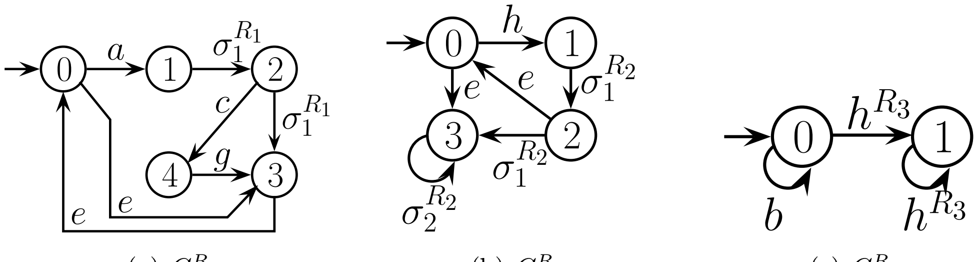


(a) G_{N_1}

(b) G_{N_2}

(c) G_{N_3}

Figure 4.6: Automata G_{N_1}, G_{N_2} and G_{N_3} of Example 4.1.



(a) $G_{N_1}^R$

(b) $G_{N_2}^R$

(c) $G_{N_3}^R$

Figure 4.7: Automata $G_{N_1}^R$, $G_{N_2}^R$ and $G_{N_3}^R$ of Example 4.1.

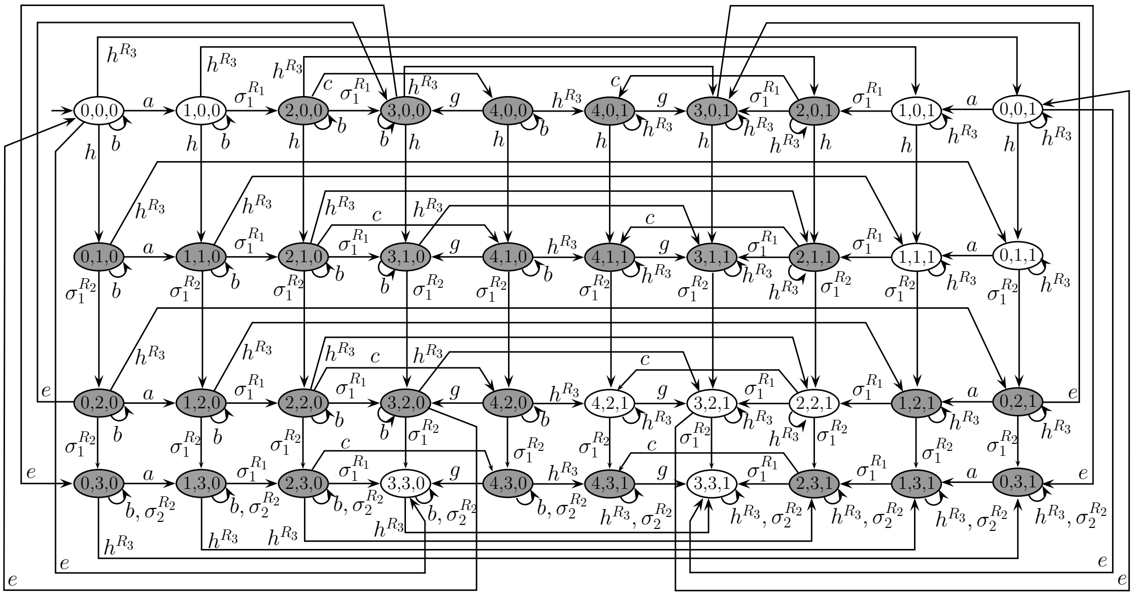


Figure 4.8: Automaton G_N^R of Example 4.1.

Example 4.2 *Consider again automaton G_N of Example 4.1, shown in Figure 4.5. It can be seen that a transition labeled with event e in G_N is only possible in states $(0,0,0)$, $(0,0,1)$, and $(3,2,1)$, i.e., a transition labeled with event e in G_N can only occur if the first module G_{N_1} is in state 0 and the second module G_{N_2} is in state 0, or the first module G_{N_1} is in state 3 and the second module G_{N_2} is in state 2. Now, let us consider that diagnosers D_1 and D_2 are in the same network, as depicted in Figure 4.9 and, therefore, can exchange information regarding the state estimate of G_{N_1} and G_{N_2}, and observable event occurrences.*

Let us assume now that we add to transition $(0,e,3)$ of G_{N_1} a condition associated with state 0 of G_{N_2}, such that transition $(0,e,3)$ of G_{N_1} can only be transposed if the current state estimate of G_{N_2} has state 0 and event e is observed. Considering the same faulty trace $h\sigma_f(eh)^z$, for $z \in \mathbb{N}$, of Example 4.1, we can see that when event e is observed, the current state estimate of automaton G_{N_1}, depicted in Figure 4.6(a), is $\{0\}$, while the current state estimate of G_{N_2}, depicted in Figure 4.6(a), does not have state 0. Thus, since we have added to transition $(0,e,3)$ of G_{N_1} a condition associated with state 0 of G_{N_2}, event e is not feasible in state 0 of D_1 anymore and, thus, diagnoser D_1 is capable of diagnosing the fault event, after the occurrence of trace $h\sigma_f e$.

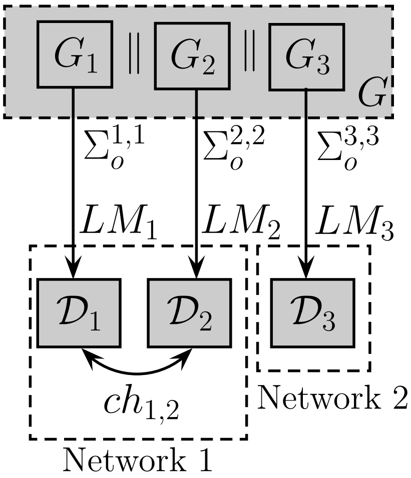


Figure 4.9: Distributed synchronous diagnosis architecture for the system of Example 4.2.

The idea of this work is to use the knowledge of the fault-free behavior model of the system, G_N, to add conditions to the fault-free component models G_{N_k} for the transposition of transitions. These conditions are associated with the states of the other components of the system, whose corresponding local diagnosers are in the same network. If an event $\sigma_o \in \Sigma_{k,o}$ that is enabled in the current state estimate of G_{N_k} is observed, all conditions of the enabled transitions labeled with σ_o must be satisfied, otherwise, the fault event is identified by the local diagnoser D_k. In order to do so, we define in the sequel the extended automaton with conditions G_φ.

Definition 4.1 *An extended automaton with conditions is the five-tuple* $G_\varphi = (Q, \Sigma, \Phi, f_\varphi, q_0)$*, where* Q *is the set of states,* Σ *is the set of events,* Φ *is a set of boolean conditions,* $f_\varphi : Q \times \Sigma \times \Phi \to Q$ *is the conditional transition function, and* q_0 *is the initial state.*

In the extended automaton with conditions G_φ, a transition $q' = f_\varphi(q, \sigma, \varphi)$, where $\sigma \in \Sigma$ and $\varphi \in \Phi$, can only be transposed if the associated event σ occurs, and condition φ is true.

In order to model the conditions for the transposition of transitions in automaton G_{N_k}, associated with the state estimates of the fault-free component models G_{N_j} whose local diagnosers are in the same network, as shown in Example 4.2, it is necessary to extend automaton G_{N_k}, as presented in Definition 4.1, obtaining the fault-free extended automaton with conditions $G_{N_{k,\varphi}}$. In order to do so, let us consider, without loss of generality, that local diagnosers D_k, for $k = 1, \ldots, m$, where $m \leq r$, are in the same network. We first define, for each state q_{N_k} of G_{N_k}, the following set of states of G_N:

$$B_{N_k} = \{q_N \in Q_N : q_{N_k} \text{ is the } k\text{-}th \text{ coordinate of } q_N\}. \tag{4.1}$$

Then, the following set of states formed with all *j-th* coordinates of the states of B_{N_k} can be defined as:

$$Q_{k,j} = \{q_{N_j} \in Q_{N_j} : \exists q_N = (q_{N_1}, \ldots, q_{N_j}, \ldots, q_{N_r}) \in B_{N_k}\}. \tag{4.2}$$

Let us define the projection operation $P_{j,o} : \Sigma^\star \to \Sigma_{j,o}^\star$, and let $Reach_j(s)$ denote the state estimate of automaton G_{N_j} after the occurrence of a trace $s \in L$. The procedure to compute $G_{N_{k,\varphi}}$ is shown in Algorithm 4.1.

Algorithm 4.1 *Computation of the fault-free extended automaton with conditions* $G_{N_{k,\varphi}}$.

Input: *Automata* G_N, *and* G_{N_k}, *for* $k \in \{1, \ldots, m\}$.

Output: *Automaton* $G_{N_{k,\varphi}} = (Q_{N_k}, \Sigma_k \setminus \Sigma_f, \Phi_k, f_{N_{k,\varphi}}, q_{0,k})$.

```
1: For each state $q_{N_k} \in Q_{N_k}$ of $G_{N_k}$ do:
    1.1: Form sets $Q_{k,j}$, $j = 1, \ldots, m$, $j \neq k$, as presented in Equation (4.2).
    1.2: For all $\sigma \in \Gamma_{G_{N_k}}(q_{N_k})$ do:
        1.2.1: If $\sigma \in \Sigma_{k,uo}$, set $\varphi = true$.
        1.2.2: If $\sigma \in \Sigma_{k,o}$, set
               $$\varphi = \bigwedge_{j=1, j\neq k}^{m} [Reach_j(s) \cap Q_{k,j} \neq \emptyset].$$
    1.3: Define $f_{N_{k,\varphi}}(q_{N_k}, \sigma, \varphi) = q'_{N_k}$, where $q'_{N_k} = f_{N_k}(q_{N_k}, \sigma)$.
2: Form set $\Phi_k$ with all conditions created in Step 1.2.
```

Notice that, in Step 1.2 of Algorithm 4.1, a condition associated to the state estimate of the fault-free component models G_{N_j}, for $j \neq k$ and $j = 1, \ldots, m$, is added to each transition of G_{N_k} labeled with an observable event. With that, it is possible to reduce the size of the fault-free language considered for diagnosis, when the communication of the state estimates $Reach_j(s)$ is assumed between local diagnosers in the same network. It is important to remark that the complexity of adding the conditions to automata G_{N_k} according to Algorithm 4.1 is polynomial with the number of system components.

In the following example, the construction of the fault-free component models with conditions, $G_{N_{k,\varphi}}$, of a composed system is presented.

Example 4.3 *Let us consider again the system $G = G_1 \| G_2 \| G_3$ presented in Figure 4.4, and let us assume that diagnosers D_1 and D_2 are in the same network and, therefore, can exchange information regarding state estimates. The fault-free behavior model of the composed system, G_N, is depicted in Figure 4.5, and the fault-free behavior model of automata G_1, G_2 and G_3, denoted as G_{N_1}, G_{N_2} and G_{N_3}, respectively, are depicted in Figure 4.6. In order to extend automata G_{N_1}, G_{N_2} and G_{N_3} according to Definition 4.1, we apply Algorithm 4.1, resulting respectively in the fault-free extended automata with conditions $G_{N_{1,\varphi}}$, $G_{N_{2,\varphi}}$ and $G_{N_{3,\varphi}}$, shown in Figure 4.10.*

Since we have the knowledge of the fault-free behavior model of the system, G_N, it can be seen that when G_{N_1} is in state 0, the transition labeled with event e can only be transposed if state 0 of G_{N_2} belongs to its current state estimate. Applying Step 1.3 of Algorithm 4.1 to state $q_{N_1} = 0$ of G_{N_1} and $\sigma = e$, then transition $f_\varphi(q_{N_1}, \sigma, \varphi) = q'_{N_1}$ of $G_{N_{1,\varphi}}$ becomes $f_\varphi(0, e, [q_{N_2} = 0]) = 3$. This procedure is repeated to all transitions of G_{N_k}, for $k = 1, 2, 3$. Notice that, since diagnoser D_3 is not connected to D_1 and D_2, the condition φ associated with the transitions labeled with observable events is always $\varphi = true$. It is important to remark that, in Figure 4.10, we do not represent the conditions $\varphi = true$.

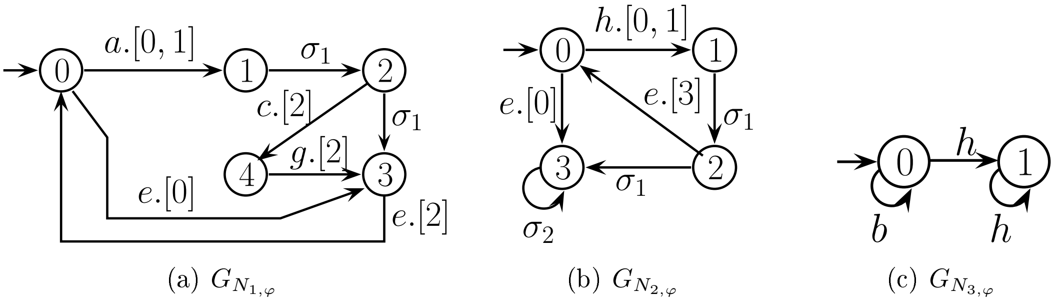


Figure 4.10: Automata $G_{N_{1,\varphi}}$, $G_{N_{2,\varphi}}$ and $G_{N_{3,\varphi}}$ of Example 4.3.

In order to consider the communication between local diagnosers through a network, it is necessary to define a communication protocol. The communication protocol proposed in this work is described for a network composed of an arbitrary number of local diagnosers, and the same procedure is considered for all networks of the system. The communication protocol can be divided into two steps: (i) when an event $\sigma_o \in \Sigma_o^{i,i}$ is directly observed by the local measurement site LM_i of local diagnoser D_i, $i \in \{1, \dots, r\}$, it sends the information of the occurrence of σ_o to all other local diagnosers in the same network; (ii) then, all local diagnosers D_j send the state estimate of its corresponding module $G_{N_{j,\varphi}}$ to the other diagnosers in the same network. After the end of communication of the state estimates in the network, the conditions for the transposition of the transitions labeled with σ_o, in the fault-free component models $G_{N_{j,\varphi}}$, for which $\sigma_o \in \Sigma_j$, are verified. If there is at least one feasible transition in $G_{N_{j,\varphi}}$, then D_j updates its state estimate. Otherwise, the fault is identified and its occurrence can be communicated to the operator of the system. It is important to remark that, in this work, it is assumed that while steps (i) and (ii) of the communication protocol are being performed, no other observable event defined in a local diagnoser belonging to the same network occurs.

In the next section the distributed synchronous diagnosability of the language of a system is defined.

4.3 Distributed synchronous diagnosability

In Algorithm 4.1 we add conditions for the transposition of transitions in the fault-free component models of the system, G_{N_k}, $k \in \{1, \ldots, r\}$, in order to reduce the size of the augmented fault-free language for synchronous diagnosis. Notice that, if we assume that there is no communication of state estimates between diagnosers, the augmented fault-free language can be modeled by using automaton G_N^R and, therefore, the distributed synchronous diagnosis can be seen as a decentralized synchronous diagnosis problem. When we consider the effect of the addition of conditions and the communication between diagnosers of the same network, we need to define an automaton that models this effect in the fault-free language considered for the distributed synchronous diagnosis. In Algorithm 4.2, we compute automaton $G_{N,\varphi}^R$ that models the fault-free language for distributed synchronous diagnosis.

Algorithm 4.2 *Fault-free model for distributed synchronous diagnosis $G_{N,\varphi}^R$.*

Input: *Automata G_N^R and G_N, and set $N = \{(i,j) \in \{1,\ldots,r\} \times \{1,\ldots,r\}$: D_i and D_j belong to the same network$\}$.*

Output: *Automaton $G_{N,\varphi}^R$.*

```
1: For each pair $(i,j) \in N$, flag all transitions $(q_N^R, \sigma, \tilde{q}_N^R)$ of $G_N^R$ such that $\sigma \in \Sigma_{i,o} \cup \Sigma_{j,o}$, and the combination of the i-th and j-th coordinates of $q_N^R$ does not exist in any state of $G_N$.
2: Delete all flagged transitions of $G_N^R$, obtaining automaton $G_N^{R'}$.
3: Compute automaton $G_{N,\varphi}^R = Ac(G_N^{R'})$.
```

Consider that $L_{N_{a,d}}$ denotes the augmented observed fault-free language obtained by using the synchronous distributed method proposed in this work. The following theorem shows that automaton $G_{N,\varphi}^R$, computed by applying Algorithm 4.2, can be used to model the fault-free behavior considered in the distributed synchronous diagnosis scheme.

Theorem 4.1 *$L_{N_{a,d}} = P_o^R(\mathcal{L}(G_{N,\varphi}^R))$, where $P_o^R : \Sigma_R^\star \rightarrow \Sigma_o^\star$ and $\Sigma_R = \cup_{k=1}^r \Sigma_{N_k}^R$, where $\Sigma_{N_k}^R$ is the event set of $G_{N_k}^R$ obtained after renaming all unobservable events of G_{N_k}.*

Proof. If no conditions are added to G_{N_k}, the observed augmented language is $\hat{L}_{N_a} = P_o^R(\mathcal{L}(G_N^R))$. Thus, the addition of conditions for the transposition of transitions in automata G_{N_k}, erases transitions of G_N^R in order to obtain automaton $G_{N,\varphi}^R$. According to Algorithm 4.1, transitions labeled with unobservable events, *i.e.*, $\sigma \in \Sigma_{k,uo}$, can be transposed whenever possible, since condition φ is true when σ is unobservable to G_k. This fact is considered in Algorithm 4.2, since transitions labeled with an unobservable event are not erased from G_N^R in the construction of $G_{N,\varphi}^R$.

Now, without loss of generality, let us suppose that diagnoser D_k computed from automaton $G_{N_{k,\varphi}}$ belongs to a network composed of diagnosers D_j, $j = 1, \ldots, m$, where $j \neq k$, and $m \leq r$. According to Algorithm 4.1, transitions $(q_{N_k}, \sigma, q'_{N_k})$ of $G_{N_{k,\varphi}}$, where $\sigma \in \Sigma_{k,o}$, can be transposed only if $G_{N_{k,\varphi}}$ is in state q_{N_k}, event σ occurs, and condition φ is true. Notice that, according to Algorithm 4.1, condition φ is true, only if $q_{N_j} \in Q_{k,j}$, for $j = 1, \ldots, m$, and $j \neq k$, where q_{N_j} is the j-th coordinate of the states of G_N. Thus, any transition labeled with σ leaving a state q of G_N^R, such that q_{N_k} is the k-th coordinate of q, and $q_{N_j} \notin Q_{k,j}$ is the j-th coordinate of q, must be erased from G_N^R. This elimination of transitions is performed in Algorithm 4.2 in order to obtain $G_{N,\varphi}^R$. Since only these transitions are eliminated in Algorithm 4.2, then $L_{N_{a,d}} = P_o^R(\mathcal{L}(G_{N,\varphi}^R))$. ■

In the following example the construction of $G_{N,\varphi}^R$ according to Algorithm 4.2 is illustrated.

Example 4.4 *Let us consider again automata G_{N_1}, G_{N_2} and G_{N_3} depicted in Figure 4.6, and presented in Example 4.1. Consider again that local diagnosers D_1 and D_2 are connected, forming a network. According to Step 1 of Algorithm 4.2, all transitions labeled with observable events associated to states where the combination of states of G_{N_1} and G_{N_2} do not exist in automaton G_N must be flagged. In*

Figure 4.11, we show automaton G_N^R with dashed transitions representing the flagging operation executed in Step 1. In Steps 2 and 3 of Algorithm 4.2, these transitions are erased, and when the accessible part of the resulting automaton is computed, the hatched states depicted in Figure 4.11 are eliminated, resulting in automaton $G_{N,\varphi}^R$ shown in Figure 4.12.

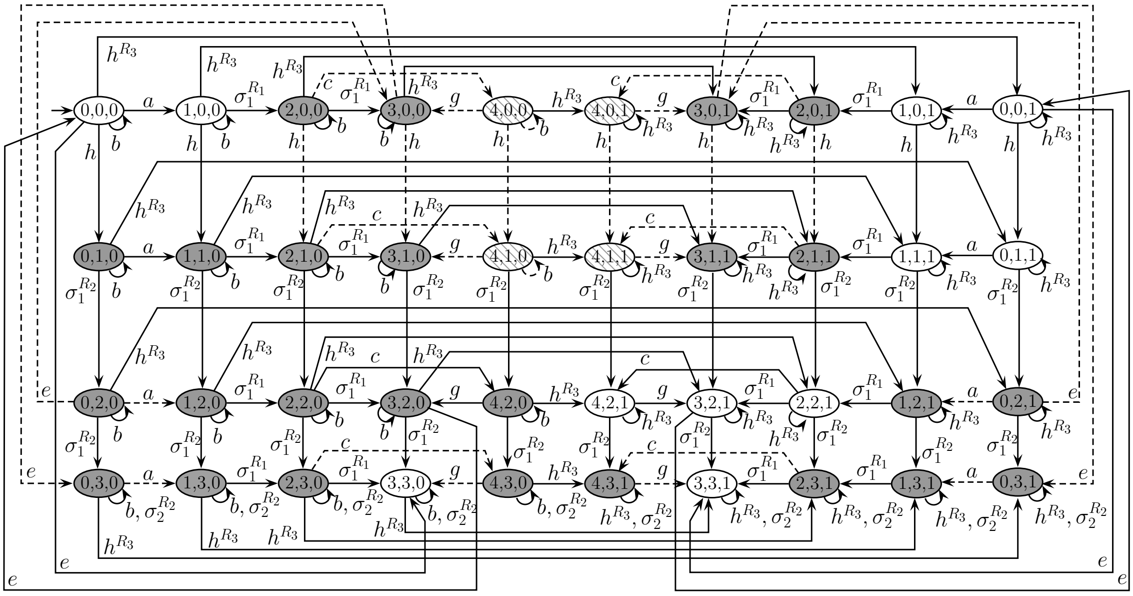


Figure 4.11: Automaton G_N^R. The white states represent the states of G_N. The hatched states and the dashed transitions are the states and transitions of G_N^R that are eliminated by applying Algorithm 4.2 in Example 4.4.

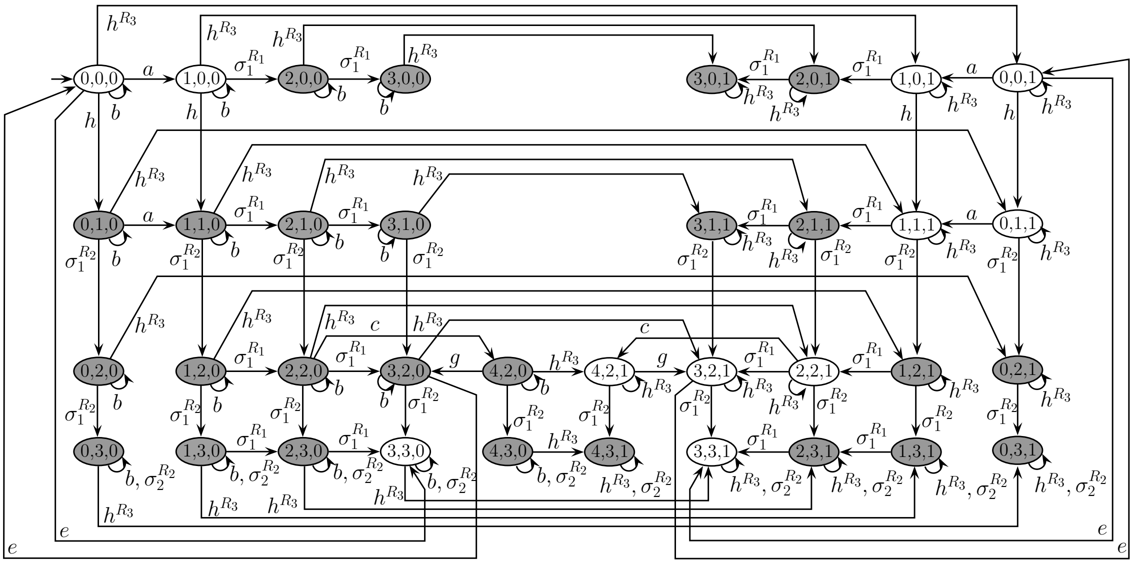


Figure 4.12: Automaton $G_{N,\varphi}^R$ of Example 4.4.

Since the augmented observed fault-free language $L_{N_{a,d}}$ can be a smaller set than language $\hat{L}_{N_a}$, obtained without considering the communication of state estimates, it is necessary to introduce the notion of distributed synchronous diagnosability.

Definition 4.2 (Distributed synchronous diagnosability)
Consider a system composed of r modules, such that $G_N = \|_{k=1}^{r} G_{N_k}$, where G_{N_k} is the automaton that models the fault-free behavior of G_k, and let L_{N_k} denote the language generated by G_{N_k}, for $k = 1, \ldots, r$. Then, L is said to be distributed synchronously diagnosable with respect to $L_{N_{a,d}}$, P_o, and Σ_f if

$$(\exists z \in \mathbb{N})(\forall s \in L_F)(\forall st \in L_F, \|t\| \geq z) \Rightarrow P_o(st) \notin L_{N_{a,d}}.$$

The observed language of $G^R_{N,\varphi}$ with respect to Σ_o, denoted by $L_{N_{a,d}}$, can be a larger set than the observation of the fault-free language of the composed system $P_o(L_N)$, *i.e.*, $P_o(L_N) \subseteq L_{N_{a,d}}$. Thus, it is necessary to verify the distributed synchronous diagnosability in order to implement the distributed synchronous diagnosis scheme.

It is important do remark that the definition of distributed synchronous diagnosability is equivalent to the definition of synchronous diagnosability of a system with fault-free language given by $L_{N_{a,d}}$ and faulty language given by L_F. Thus, the verification of distributed synchronous diagnosability of language L can be performed by using the same strategy presented in Algorithm 3.2 for the verification of synchronous diagnosability, replacing automaton G^R_N with automaton $G^R_{N,\varphi}$. In the following we present an algorithm that can be used to verify the distributed synchronous diagnosability of the language generated by a system.

Algorithm 4.3 *Distributed synchronous diagnosability verification*

Input: *System modules G_k, for $k = 1, \ldots, r$, and $G = \|_{k=1}^{r} G_k$.*

Output: *Distributed synchronous diagnosability decision.*

1: Compute automaton G_F *that models the faulty behavior of* G*, whose marked language is* $L_F = L \setminus L_N$*, as follows:*

1.1: Set $A_l = (Q_l, \Sigma_f, f_l, q_{0,l})$*, where* $Q_l = \{N, F\}$*,* $q_{0,l} = \{N\}$*,* $f_l(N, \sigma_f) = F$ *and* $f_l(F, \sigma_f) = F$*, for all* $\sigma_f \in \Sigma_f$*.*

1.2: Compute $G_l = G \| A_l$ *and mark all states of* G_l *whose second coordinate is equal to* F*.*

1.3: Compute the faulty automaton $G_F = CoAc(G_l)$*.*

2: Compute automata G_{N_k}*,* $k = 1, \ldots, r$*, by following the steps of Algorithm 3.1.*

3: Compute automaton $G^R_{N,\phi}$ *following the steps of Algorithm 4.2.*

4: Compute the verifier automaton $G^{DD}_V = (Q_V, \Sigma_V, f_V,\ q_{0,V}) = G_F \| G^R_{N,\phi}$*. Notice that a state of* G^{DD}_V *is given by* $q_V = (q_F, q^R_{N,\phi})$*, where* q_F *and* $q^R_{N,\phi}$ *are states of* G_F *and* $G^R_{N,\phi}$*, respectively, and* $q_F = (q, q_l)$*, where* $q \in Q$ *and* $q_l \in \{N, F\}$*.*

5: Verify the existence of a cyclic path $cl = (q^\delta_V, \sigma_i, q^{\delta+1}_V,\ \ldots, q^\gamma_V, \sigma_\gamma, q^\delta_V)$*, where* $0 < \delta \leq \gamma$*, in* G_V *such that:*

$$\begin{aligned} &\exists j \in \{\delta, \delta+1, \ldots, \gamma\} \text{ s.t. for some } q^j_V, \\ &(q^j_l = F) \wedge (\sigma_j \in \Sigma). \end{aligned} \tag{4.3}$$

If the answer is yes, then L *is not distributed synchronously diagnosable with respect to* $L_{N_{a,d}}$*,* $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$*, and* Σ_f*. Otherwise,* L *is distributed synchronously diagnosable with respect to* $L_{N_{a,d}}$*,* $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$*, and* Σ_f*.*

Notice that any verification method could be applied by using the automata that generate languages $\overline{L_F}$ and $L_{N_{a,d}}$, or any language whose projection in Σ_o corresponds to $L_{N_{a,d}}$. The method presented in Algorithm 4.3 has polynomial complexity with respect to the number of states of the system components, since we do not use observers to obtain $L_{N_{a,d}}$, and exponential complexity with the number of components r. The verification of the distributed synchronous diagnosability is performed by searching for cycles of states in G_V^{DD} with label F such that at least one transition of the cycle is labeled with a non renamed event.

The following theorem proves the correctness of Algorithm 4.3 for the verification of distributed synchronous diagnosability.

Theorem 4.2 *Let L_{N_k} denotes the language generated by G_{N_k}, for $k = 1, \ldots, r$, and consider $G_V^{DD} = G_F \| G_{N,\varphi}^R$, where $G_{N,\varphi}^R$ is computed by following Algorithm 4.2. A state of G_V^{DD} is given by $q_V = (q_F, q_N^R)$, where q_F and q_N^R are states of G_F and $G_{N,\varphi}^R$, respectively, and $q_F = (q, q_l)$, where $q \in Q$ and $q_l \in \{N, F\}$. Then, L is not distributed synchronously diagnosable, according to Definition 4.2, with respect to L_{N_k}, $P_{k,o}$, and Σ_f if, and only if, there exists a cyclic path $cl = (q_V^\delta, \sigma_\delta, q_V^{\delta+1}, \ldots, q_V^\gamma, \sigma_\gamma, q_V^\delta)$ in $G_V^{DD} = G_F \| G_{N,\varphi}^R$, where $\gamma \geq \delta > 0$, such that:*

$$\begin{aligned}&\exists j \in \{\delta, \delta+1, \ldots, \gamma\} \text{ such that for some } q_V^j,\\ &(q_l^j = F) \wedge (\sigma_j \in \Sigma). \end{aligned} \tag{4.4}$$

Proof. According to Definition 4.2, language L of the composed system $G = \|_{k=1}^r G_k$ is distributed synchronously diagnosable if there does not exist an arbitrarily long length faulty trace st such that $P_o(st) \in L_{N_{a,d}}$. Theorem 4.1 shows that $L_{N_{a,d}} = P_o^R(\mathcal{L}(G_{N,\varphi}^R))$, where $P_o^R : \Sigma_R^\star \rightarrow \Sigma_o^\star$. Thus, in order to verify the distributed synchronous diagnosability of language L, it is necessary to check if there exists a faulty trace st with the same observation of a fault-free trace $\omega \in \mathcal{L}(G_{N,\varphi}^R)$, where $P_o^R(\omega) \in L_{N_{a,d}}$. Notice that the unobservable events of $G_{N,\varphi}^R$ are renamed, and thus, are private events of $G_{N,\varphi}^R$. Therefore, it can be seen that the verifier automaton G_V^{DD} proposed in this work is equal to the verifier automaton G_V proposed in MOREIRA *et al.* [27] applied to a system where the faulty behavior automaton marks L_F and whose observable fault-free behavior automaton generates $L_{N_{a,d}}$. Besides that, using the verification method proposed in [27], the same necessary and sufficient condition (4.4) would be obtained, which concludes the proof. ■

In the following example, we illustrate the method for the verification of distributed synchronous diagnosability.

Example 4.5 *Considering again the system composed of three modules, such that $G = G_1\|G_2\|G_3$, where G_1, G_2, and G_3 are shown in Figure 4.3. Let us also assume that local diagnosers D_1 and D_2 are connected in a network, and diagnoser D_3 is not connected to D_1 and D_2, as depicted in Figure 4.9. Then, applying Step 1 of Algorithm 4.3, we compute automaton G_F, depicted in Figure 4.13. According to Step 2, we obtain automata G_{N_1}, G_{N_2} and G_{N_3}, depicted in Figure 4.6. Following Step 3 of Algorithm 4.3, which compute automaton $G_{N,\varphi}^R$ by using Algorithm 4.2, we obtain the automaton presented in Figure 4.12. Finally, the verifier automaton G_V^{DD} is computed by making the parallel composition of automaton G_F and automaton $G_{N,\varphi}^R$. Only part of G_V^{DD} is shown in Figure 4.14 due to the lack of space. Since there are no cycles in G_V^{DD} that satisfy condition (4.4) of Theorem 4.2, then, L is distributed synchronously diagnosable with respect to $L_{N_{a,d}}$, $P_o : \Sigma^\star \to \Sigma_o^\star$, and Σ_f.*

Now, let us consider the faulty trace $h\sigma_f(eh)^z$ of G. In Example 4.1, it was shown that L is not synchronously codiagnosable, since this faulty trace has the same observation as the fault-free trace $h\sigma_1^{R_2}(eh\sigma_1^{R_2})^z$ of automaton G_N^R. Notice that there is no trace in $G_{N,\varphi}^R$ whose observation with respect to Σ_o is equal to $h(eh)^z$. This shows, as expected, that a system can be distributed synchronously diagnosable, and not synchronously codiagnosable.

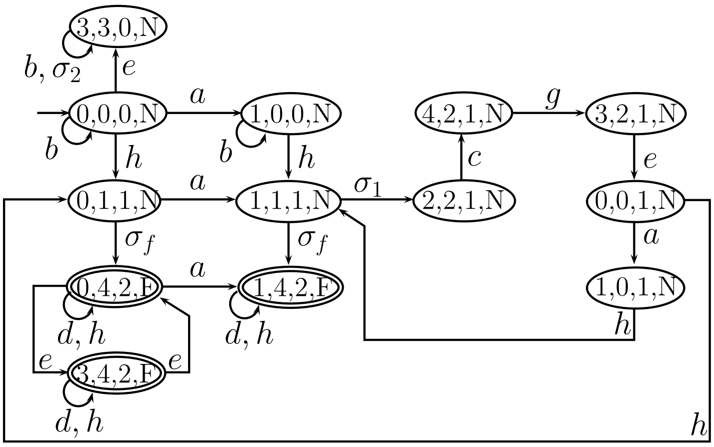


Figure 4.13: Automaton G_F of Example 4.5.

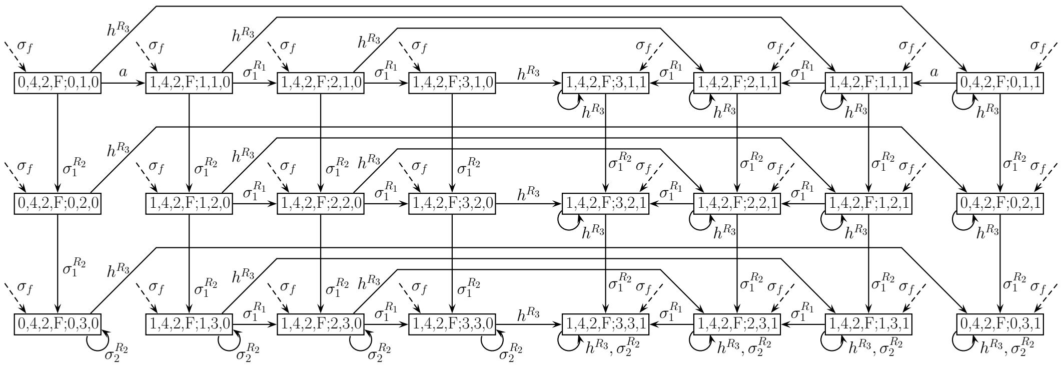


Figure 4.14: Part of automaton G_V^{DD} with states labeled with F of Example 4.5.

Although in the worst case scenario the distributed synchronous diagnosability verification method has exponential complexity in the number of system components, in the distributed synchronous diagnosis architecture proposed in this work, each local diagnoser has polynomial growth with the number of states of its corresponding component model. Therefore, the use of the global plant model is avoided for the distributed synchronous diagnosis.

Besides the need of verification of distributed synchronous diagnosability due to the fault-free language considered in this approach be a larger set than the fault-free language of the composed system, we may also compute the delay bound $z^\star$ for distributed synchronous diagnosis. It can be computed by using Algorithm 3.3 and Equation (3.2), replacing the input automaton by G_V^{DD}. In the next example we compute the delay bound for distributed synchronous diagnosis for the system of Example 4.5.

Example 4.6 *Let us consider again the system $G = G_1 \| G_2 \| G_3$ presented in Example 4.5. Since L is distributed synchronously diagnosable with respect to $L_{N_{a,d}}$, $P_o : \Sigma^\star \to \Sigma_o^\star$, and Σ_f, we can compute the delay bound $z^\star$ for distributed synchronous diagnosis. Using the verifier automaton G_V^{DD}, whose states labeled with F and their correspondent transitions is shown in Figure 4.14, as input of Algorithm 3.3 and applying the result in Equation (3.2), we obtain $z^\star = 2$. Computing the delay bound for the monolithic diagnosis, we obtain the same result $z^\star = 2$. This shows that, for this system, using the distributed synchronous diagnosis approach, we take* *advantage of the modularity of the system, takes into consideration that information is not available in a centralized way and, besides that, the resulting delay bound is the same as in the centralized monolithic architecture.*

It is important to notice that the notion of synchronous codiagnosability presented in Section 3.2 is a particular case of the notion of distributed synchronous diagnosability presented in this work. The distributed synchronous diagnosis is equal to the decentralized synchronous diagnosis when there is no network formed with local diagnosers and, consequently, no exchange of information between local diagnosis.

In CABRAL [33], a comparison between the notion of modular diagnosability, proposed by CONTANT *et al.* [30], and the notion of synchronous codiagnosability (Definition 3.5) is presented. The approach presented in [30] shows a different notion of modular diagnosability, where necessary and sufficient conditions that ensure the modular diagnosability of a DES are proposed. The assumptions assumed by [30] are: (*i*) the language of the system is considered live, and there are no cycles of unobservable events in the system component models; (*ii*) common events between two or more components are observable, which implies that the fault event belongs only to one local component model of the system; (*iii*) the model that exhibits the faulty behavior has persistent excitation. In [30], only the observation of the local component where the fault event is modeled is taken into account to diagnose a global fault occurrence.

In order to compare the notions of modular diagnosability and synchronous codiagnosability, in CABRAL [33], the assumptions proposed by [30] are applied to the synchronous decentralized diagnosis scheme. The effect of considering these assumptions is that the definition of synchronous codiagnosability becomes equal to the definition of modular diagnosability, which implies that modular diagnosis can be seen as a particular case of synchronous decentralized diagnosis.

Therefore, we can conclude that modular diagnosability can also be seen as a particular case of distributed synchronous diagnosability. Thus, if the language of a system is modularly diagnosable according to [30], only the local diagnoser associated with the fault-free component model can be used for fault diagnosis.

4.4 Final comments

In this chapter, we propose a new synchronous diagnosis method, which consider that local diagnosers are separated into networks. Each local diagnoser works as node in the net, and can exchange information regarding observation of events and state estimates. This information is used to refine the diagnosis decision, by adding boolean conditions for the transposition of transitions of the fault-free component models of the system. These conditions are associated to the state estimates of local diagnosers that belong to the same network. For the implementation of the distributed synchronous diagnosis method, the local diagnosers considering these boolean conditions can be constructed following the method presented in CABRAL *et al.* [35]. The notion of distributed synchronous diagnosability is introduced, and a method to verify the distributed synchronous diagnosability based on the method for the verification of synchronous diagnosability presented in Section 3.1, is also presented.

In Table 4.1, the notations of each synchronous diagnosis architecture is presented, in order to summarize and compare the preliminary results presented in Chapter 3 and the distributed synchronous diagnosis proposed in this chapter.

Table 4.1: Summary of notations regarding the synchronous diagnosis architectures.

Architecture	Augmented observed fault-free language	Augmented fault-free automaton	Verifier automaton	Diagnoser
Synchronous centralized diagnosis	L_{N_a}	G_N^R	G_V^{SD}	Single diagnoser
Synchronous decentralized diagnosis	$\hat{L}_{N_a}$	$\hat{G}_N^R$	G_V^{SC}	Local diagnosers
Distributed synchronous diagnosis	$L_{N_{a,d}}$	$G_{N,\phi}^R$	G_V^{DD}	Local diagnosers with communication

Chapter 5

Conclusions and future work

In this work, we propose the distributed synchronous diagnosis scheme for modular discrete-event systems. In this scheme, local diagnosers are computed based on the fault-free behavior models of the system components, and are capable of communicating the observation of events and state estimate to other local diagnosers in the same network. The communication between diagnosers is used to improve the fault diagnosis in comparison with other synchronous diagnosis strategies, leading to the notion of distributed synchronous diagnosability.

In order to implement the distributed synchronous diagnosis scheme, a communication protocol is proposed. The addition of boolean conditions for the transposition of transitions of the fault-free component models are presented. These conditions are associated with the state estimate of other local components whose corresponding local diagnosers are in the same network, which result in the definition of an extended automaton with conditions. The fault detection logic considered in this work is that, when an event is observed by a local diagnoser, all conditions of the enabled transitions labeled with the same event should by satisfied, otherwise, the fault event is identified.

In summary, the main contributions of this work are as follows.

- A fault diagnosis scheme with distributed architecture for modular discrete-event systems modeled by automata, called distributed synchronous diagnosis, is proposed. In this scheme, local diagnosers are constructed based on the observation of the fault-free behavior model of the system components.

- A communication protocol is introduced in order to allow the exchange of information between local diagnosers that belong to the same network.
- An extended automaton with conditions is introduced in order to alter its transition function according to the boolean conditions added to the transitions of the fault-free component models.
- The notion of distributed synchronous diagnosability is presented.
- A method for the verification of distributed synchronous diagnosability of DESs with polynomial computational complexity in the state-spate of the system components is proposed.

Future works

In order to avoid a diagnosis technique based on the composed system model, the synchronous diagnosis has been proposed in the literature. In this scheme, although the composed plant model is not used for diagnosis, all system components are considered in order to construct the synchronous diagnoser. However, in several cases, the language of the system could be diagnosed using a subset of its components. Therefore, an idea of future work is to obtain a method of computing minimal subsets of local components that ensure synchronous diagnosability of the language of a composed discrete-event system. This idea is similar to the problem of finding minimal diagnosis bases of events for diagnosability of DESs [22, 26], with the difference that the objective is to provide a method for the computation of a minimal synchronous diagnosis base of automata. It is important to notice that if the minimal number of components necessary for synchronous diagnosis is used, then the computational cost of the synchronous diagnoser is also decreased, which is particularly interesting for systems with a high degree of concurrency.

For the implementation of the distributed synchronous diagnosis method, it is considered that the network topology is known, *i.e.*, the information of which diagnosers can exchange information between then is previously known. Thus, another idea of future work is to obtain a mechanism that returns an optimal network topology in order to obtain the lowest delay bound for the distributed synchronous diagnosis. In addition, exploring other communication protocols present in the literature applied to this architecture, in order to increase the efficiency of the method, may also be interesting.

Bibliography

[1] SHI, J., WAN, J., YAN, H., et al. "A survey of cyber-physical systems". In: *International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 9–11, Nanjing, China, 2011.

[2] BAHETI, R., GILL, H. "Cyber-physical systems". In: *The impact of control technology*, pp. 161–166, 2011.

[3] LEE, J., BAGHERI, B., KAO, H. "A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems", *Manufacturing Letters*, v. 3, pp. 18–23, 2015.

[4] LIMA, P. M., ALVES, M. V. S., CARVALHO, L., et al. "Security Against Communication Network Attacks of Cyber-Physical Systems", *Journal of Control, Automation and Electrical Systems*, pp. 1–11, 2018.

[5] CASSANDRAS, C., LAFORTUNE, S. *Introduction to Discrete Event System*. Secaucus, NJ, Springer-Verlag New York, Inc., 2008.

[6] HOPCROFT, J. E., MOTWANI, R., ULLMAN, J. D. *Introduction to automata theory, languages, and computation*. Boston, Addison Wesley, 2006.

[7] MIYAGI, P. E. *Controle programável: fundamentos do controle de sistemas a eventos discretos*. Edgard Blücher, 1996.

[8] LAWSON, M. V. *Finite automata*. Florida, CRC Press, 2003.

[9] DAVID, R., ALLA, H. *Discrete, Continuous and Hybrid Petri Nets*. Springer, 2005.

[10] 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.

[11] 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.

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

[13] 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.

[14] WANG, Y., YOO, T.-S., LAFORTUNE, S. "Diagnosis of discrete event systems using decentralized architectures", *Discrete Event Dynamic Systems: Theory And Applications*, v. 17, pp. 233–263, 2007.

[15] QIU, W., KUMAR, R. "Distributed diagnosis under bounded-delay communication of immediately forwarded local observations", *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, v. 38, n. 3, pp. 628–643, 2008.

[16] KEROGLOU, C., HADJICOSTIS, C. N. "Distributed Fault Diagnosis in Discrete Event Systems via Set Intersection Refinements", *IEEE Transactions on Automatic Control*, v. 63, n. 10, pp. 3601 – 3607, 2018.

[17] SU, R., WONHAM, W. M. "Global and local consistencies in distributed fault diagnosis for discrete-event systems", *IEEE Transactions on Automatic Control*, v. 50, n. 12, pp. 1923–1935, 2005.

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

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

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

[21] CABASINO, M., GIUA, A., LAFORTUNE, S., et al. "A New Approach for Diagnosability Analysis of Petri Nets using Verifiers Nets", *IEEE Transactions on Automatic Control*, v. 57, n. 12, pp. 3104–3117, 2012.

[22] BASILIO, J. C., LIMA, S. T. S., LAFORTUNE, S., et al. “Computation of minimal event bases that ensure diagnosability”, *Discrete Event Dynamic Systems: Theory And Applications*, v. 22, pp. 249–292, 2012.

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

[24] 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.

[25] MOREIRA, M. V., BASILIO, J. C., CABRAL, F. G. ““Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems” Versus “Decentralized Failure Diagnosis of Discrete Event Systems”: A Critical Appraisal”, *IEEE Transactions on Automatic Control*, v. 61, n. 1, pp. 178–181, 2016.

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

[27] 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.

[28] CASSEZ, F. “A note on fault diagnosis algorithms”. In: *Proceedings of the 48th IEEE Conference on Decision and Control held jointly with the 28th Chinese Control Conference, CDC/CCC.*, pp. 6941–6946, Shanghai, China, 2009.

[29] DEBOUK, R., MALIK, R., BRANDIN, B. “A modular architecture for diagnosis of discrete event systems”. In: *41st IEEE Conference on Decision and Control*, pp. 417–422, Las Vegas, Nevada USA, 2002.

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

[31] CABRAL, F. G., MOREIRA, M. V., DIENE, O. “Online fault diagnosis of modular discrete-event systems”. In: *IEEE 54th Annual Conference on Decision and Control (CDC)*, pp. 4450–4455, Osaka, Japan, 2015.

[32] CABRAL, F. G., MOREIRA, M. V. "Synchronous Diagnosis of Discrete-Event Systems", *Transactions on Automation Science and Engineering*, 2018. Submitted for publication.

[33] CABRAL, F. G. *Synchronous Failure Diagnosis of Discrete-Event Systems.* Tese de Doutorado, Programa de Pós-Graduação em Engenharia Elétrica - COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2017.

[34] CABRAL, F. G., MOREIRA, M. V. "Synchronous Decentralized Diagnosis of Discrete-Event Systems". In: *20th World Congress of the International Federation of Automatic Control*, pp. 7025–7030, Toulouse, France, 2017.

[35] CABRAL, F. G., VERAS, M. Z. M., MOREIRA, M. V. "Conditional Synchronized Diagnoser for Modular Discrete-Event Systems". In: *14th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, v. 2, pp. 88–97, Madrid, Spain, 2017.

[36] VERAS, M. Z. M., CABRAL, F. G., MOREIRA, M. V. "Distributed Synchronous Diagnosability of Discrete-Event Systems". In: *Discrete Event Systems (WODES), 2018 14th International Workshop on*, pp. 88–93, 2018.

[37] BASILIO, J. C., CARVALHO, L. K., MOREIRA, M. V. "Diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos", *Revista Controle & Automação*, v. 21, n. 5, pp. 510–533, 2010.

[38] 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.

[39] BASILIO, J. C., LAFORTUNE, S. "Robust codiagnosability of discrete event systems". In: *American Control Conference (ACC)*, pp. 2202–2209, St. Louis, MO, USA, 2009.

[40] TOMOLA, J. H. A., CABRAL, F. G., CARVALHO, L. K., et al. "Robust Disjunctive-Codiagnosability of Discrete-Event Systems Against Permanent Loss of Observations", *IEEE Transactions on Automatic Control*, v. 62, n. 11, pp. 5808–5815, 2017.

[41] YOO, T.-S., GARCIA, H. "Computation of fault detection delay in discrete-event systems". In: *Proceedings of the 14th International Workshop on Principles of Diagnosis, DX'03*, pp. 207–212, Washington, USA, 2003.

[42] DASGUPTA, S., PAPADIMITRIOU, C., VAZIRANI, U. *Algorithms*. McGraw-Hill, 2008.

[43] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., et al. *Introduction to algorithms*. Massachusetts, MIT Press, 2007.