OPTIMAL SELECTION OF SUBSYSTEMS FOR SYNCHRONOUS DIAGNOSIS

Lucas Nelson Ribeiro Reis

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

Orientador: Marcos Vicente de Brito Moreira

Rio de Janeiro
Março de 2022

OPTIMAL SELECTION OF SUBSYSTEMS FOR SYNCHRONOUS DIAGNOSIS

Lucas Nelson Ribeiro Reis

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

Orientador: Marcos Vicente de Brito Moreira

Aprovada por: Prof. Marcos Vicente de Brito Moreira
Prof. Lilian Kawakami Carvalho
Prof. Felipe Gomes de Oliveira Cabral
Prof. André Bittencourt Leal

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2022

Reis, Lucas Nelson Ribeiro

Optimal Selection of Subsystems for Synchronous Diagnosis/Lucas Nelson Ribeiro Reis. – Rio de Janeiro: UFRJ/COPPE, 2022.

XV, 79 p.: il.; 29, 7cm.

Orientador: Marcos Vicente de Brito Moreira

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

Referências Bibliográficas: p. 66 – 69.

1. Discrete Events Systems. 2. Fault Diagnosis. 3. Synchronous Diagnosis. I. Moreira, Marcos Vicente de Brito. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Agradecimentos

Agradeço primeiramente a Deus por toda força e saúde por me pertmitir chegar aqui.

Agradeço a Mariana Domingues e a meus filhos Emanuel Reis, Elisa Reis e Laura Reis por todo apoio, incentivo e compreensão pelos muitos momentos de ausência necessários para a conclusão desse trabalho.

Agradeço aos meus pais Mara Rocha e Luiz Reis por todo esforço, dedicação e incentivo na minha educação para que pudesse alçar vôos como esse.

Agradeço a toda a equipe do Centro de Projetos de Navios, em especial aos meus Diretores e Vice-Diretores pelo incentivo e apoio me permitindo estudar e me dedicar para a conclusão deste trabalho.

Agradeço ao meu orientador Marcos Moreira por toda orientação e ensinamentos passados ao longo do trabalho.

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.)

SELEÇÃO ÓTIMA DE SUBSISTEMAS PARA DIAGNOSE SÍNCRONA

Lucas Nelson Ribeiro Reis

Março/2022

Orientador: Marcos Vicente de Brito Moreira

Programa: Engenharia Elétrica

Diagnóstico de falhas em sistemas de automação é uma tarefa muito importante, visto que falhas podem alterar o comportamento esperado do sistema, danificando equipamentos e gerando riscos aos operadores. Normalmente, os sistemas são formados por diversos subsistemas ou módulos e, portanto, o modelo do sistema completo pode crescer exponencialmente com o número de componentes do sistema. Em função disso, pode ser necessário um espaço de memória elevado para implementar diagnosticadores calculados utilizando métodos tradicionais, uma vez que os mesmos são baseados no modelo da planta completa. Recentemente, foi proposto um novo método para diagnose de falhas, chamado diagnose síncrona. O diagnosticador calculado por esse método é baseado em estimadores de estados dos comportamentos livres de falha dos modelos dos componentes do sistema, evitando assim que seja necessário implementar o observador do sistema completo. Na estratégia de diagnose síncrona, todos os modelos livres de falha do sistema são utilizados para a detecção da ocorrência da falha. No entanto, na prática, alguns subsistemas podem não adicionar nenhuma informação útil para o diagnóstico da falha, ou ainda, a mesma informação pode ser obtida em outros módulos, mostrando que esses subsistemas não são necessários para a arquitetura de diagnose síncrona. Neste trabalho, é proposto um algoritmo para calcular todos os conjuntos minimais de módulos que garantem a diagnosticabilidade síncrona em um Sistema a Eventos Discreto. A performance do algoritmo é comparada com a performance do método de busca exaustiva, e é mostrado que usando o método proposto é possível uma redução significativa no custo computacional para encontrar todos os conjuntos de módulos minimais que garantem a diagnose síncrona.

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

OPTIMAL SELECTION OF SUBSYSTEMS FOR SYNCHRONOUS DIAGNOSIS

Lucas Nelson Ribeiro Reis

March/2022

Advisor: Marcos Vicente de Brito Moreira

Department: Electrical Engineering

Fault diagnosis of automated systems is a very important task, since faults can alter the expected behavior of systems, damaging equipment and bringing risk to operators. In general, systems are composed of several subsystems or modules, and therefore, the complete system model may grow exponentially with the number of system components. This fact shows that a large amount of memory space may be needed to implement diagnosers computed using traditional methods, since they are based on the complete system model. Recently, a new method for fault diagnosis, called synchronous diagnosis, has been proposed. The diagnoser computed using this method is based on the state estimators of the fault-free component models of the system, avoiding the implementation of the state observer of the composed system model. In the synchronous diagnosis strategy it is supposed that all fault-free subsystem models are used to detect the fault occurrence. However, in practice, some subsystems may not add useful information regarding the fault diagnosis, or the same information can be obtained from the other modules, which shows that these subsystems are not necessary in the synchronous diagnosis scheme. In this work, an algorithm for computing all minimal sets of modules that ensure the synchronous diagnosability of a Discrete Event System is proposed. The performance of the proposed algorithm is compared with the performance of the exhaustive search method, and we show that using the proposed method there is a significant reduction in the computational cost of finding all minimal sets of modules that ensure synchronous diagnosability.

Contents

List of Figures

List of Tables

List of Symbols

$Ac(G)$	Accessible part of G, p. 8
$CoAc(G)$	Coaccessible part of G, p. 8
D_k	Diagnoser for the k-th component of G, p. 20
G	Automaton, p. 6
G_N	Automaton that models the nonfailure behavior of the system G, p. 14
G_N^R	Automaton resultant from the parallel composition of automata $G_{N_k}^R$, p. 23
G_V	Diagnosability verifier automaton, p. 19
$G_V^{B\cup\{j\}}$	Synchronous diagnosis verifier automaton restricted to modules associated with $B \cup \{j\}$, p. 42
G_d	Diagnoser automaton, p. 16
G_k	Automaton model of the k-th component of G, p. 20
G_{N_k}	Nonfailure behavior moodel of k-th component of G, p. 20
$G_{N_k}^R$	Automaton G_{N_k} with unobservable events renamed, p. 23
$G_{V_p}^B$	Subautomaton formed from path p that violates synchronous diagnosability restricted to modules associated with B, p. 42
$G_{V_p}^{j,B}$	Automaton resultant from the parallel composition of the partial verifier $G_{V_p}^B$ and the verifier of the j-th module, p. 42
G_V^B	Synchronous diagnosis verifier automaton restricted to the modules associated with B, p. 34
G_V^{SD}	Synchronous diagnosis verifier automaton, p. 23

L	Generated language of automaton G, p. 8
L_F	Failure language, p. 14
L_N	Generated language of automaton G_N, p. 14
$Obs(G, \Sigma_o)$	Observer automaton of G in Σ_o, p. 11
P_o	Projection operation defined as $P_o : \Sigma^* \to \Sigma_o^*$, p. 11
P_s^l	Projection operation defined as $P_s^l : \Sigma_l^* \to \Sigma_s^*$, p. 5
Q	Set of states, p. 6
Q_m	Set of marked states, p. 6
$UR(q)$	Unobservable reach of state q, p. 11
$\bar{L}$	Prefix-closure operation on language L,, p. 5
$\Gamma_G(q)$	Feasible event function of automaton G, p. 7
Σ	Set of Events, p. 4
Σ_f	Set of failure events, p. 14
Σ_k	Set of events of automaton G_k, p. 20
Σ_o	Observable event set, p. 11
$\Sigma_{k,o}$	Observable event set of G_k in the synchronous diagnosis scheme, p. 20
$\Sigma_{k,uo}$	Unobservable event set of G_k in the synchronous diagnosis scheme, p. 20
Σ_{uo}	Unbservable event set, p. 12
$\mathbb{N}$	Natural number set, including the number zero, p. 15
$\mathcal{L}(G)$	Generated language of automaton G, p. 7
$\mathcal{L}_m(G)$	Marked language of automaton G, p. 7
ε	Empty trace, p. 4
f	Transition function, p. 6
q_0	Initial state, p. 6
r	Total number of components of a system G, p. 20

Chapter 1

Introduction

Discrete event systems (DES) are systems whose evolution is given by the occurrence of events [1, 2]. The act of pushing a button by an operator, starting or ending a task, or the changing of a sensor state are examples of events. This kind of systems can be seen in several applications, such as robotic systems, operational systems, manufacturing systems, and data management.

It is important to remark that, in DES, events are defined as instantaneous occurrences, that can change the system state. This avoids the adoption of mathematical formalism based on differential or difference equations to represent these systems. Thus, it is necessary to adopt a mathematical formalism capable of dealing with the characteristics of a DES. In order to model DES, the most common formalisms are Petri nets and automata [1–5]. Petri nets are bipartite graphs, or bigraphs, in the sense that it has two types of nodes (places and transitions), where nodes of the same type cannot be connected. Tokens are assigned to the places of the Petri net, such that the number of tokens of each place forms the marking of the Petri net, which also represents the system state modeled by the net. Automata are directed graphs, in which states and events are represented, respectively, by vertices and arcs. In this work, automata are used to model DES.

As any other system, DES are subject to the occurrence of faults, *i.e.*, events that can alter their expected behavior, reducing its reliability and performance, or, even in the worst case scenario, leading the system to a halt. Considering this, fault diagnosis of automated systems is a very important task, since faults can alter the expected behavior of systems, damaging equipment and bringing risk to operators. This problem is addressed in several works in the literature [6–22], with different objectives, such as: robust diagnosis, considering permanent loss of observation in TOMOLA *et al.* [14], CARVALHO *et al.* [15] and intermittent loss of observation in CARVALHO *et al.* [16, 17]; optimizing the sensors that ensure diagnosability in SANTORO *et al.* [23]; combining diagnosis and prognosis for safe controllability in WATANABE *et al.* [18, 19].

In the seminal work SAMPATH *et al.* [6], a diagnoser automaton is proposed to perform fault diagnosis and to verify the diagnosability of the system language, *i.e.*, to verify if the fault occurrence can be detected within a bounded number of event occurrences after the fault. The main problem with respect to the solution presented in SAMPATH *et al.* [6], is that the diagnoser is constructed based on an observer automaton, whose state space may grow exponentially with the number of system states.

In order to overcome the exponential complexity for verifying the diagnosability of the system language, in MOREIRA *et al.* [24] it is proposed a different strategy based on a verifier automaton that can be computed in polynomial time. However, the verifier cannot be straightforwardly used for online diagnosis.

In order to circumvent the problem of the size of the classical diagnoser, a new approach is presented in DEBOUK *et al.* [25] and CONTANT *et al.* [26] is the modular diagnosability, where the idea is to infer the occurrence of the fault event by observing only the local component where the fault is modeled. It is important to remark that in the modular diagnosis techniques, the following two assumptions are considered: (*i*) all common events between subsystems are observable; and (*ii*) the component where the fault event is modeled has persistent excitation, *i.e.*, the component where the fault is modeled must be able to perform events, otherwise, the observed component may stay in the same state and the complete system continuing to perform events and the diagnoser will not observe that. Note that, according to these assumptions, the system modules cannot be synchronized with unobservable events, which implies that the fault event cannot be modeled in more than one system module, restricting its applicability. In addition, it is necessary to guarantee the persistence of excitation property, which requires the previous knowledge of the system behavior, which is not shown in DEBOUK *et al.* [25] and CONTANT *et al.* [26].

In order to relax all assumptions considered in the modular diagnosis strategy and avoid the exponential growth with the number of components, a new diagnosis technique, called synchronous diagnosis, is proposed in CABRAL and MOREIRA [20]. The method relies on the computation of a diagnoser based on the state observers of the fault-free component models of the system. This approach avoids the implementation of the state observer of the composed system model, providing an online state estimate of each fault-free subsystem model, but deals with an augmented fault-free language. Another approach is the synchronous decentralized diagnosis [27, 28], which is based on local diagnosers, each one built considering one subsystem model with its own set of observable events, meaning that an event can be observable for a local diagnoser, but unobservable for other. It is important to remark that those diagnosers do not communicate with the other local diagnosers, and

the fault event is diagnosed when at least one local diagnoser identifies its occurrence and sends that information to a coordinator. Similar to the centralized synchronous diagnosis, the decentralized deals with an augmented fault-free language. Another recent scheme is the synchronous distributed diagnosis, where local diagnosers are allowed to exchange information regarding the observation of events and local state estimates through a network [22]. This allow the local diagnosers to refine the state estimate of the fault-free behavior of the system modules, reducing the augmented fault-free language.

In the synchronous centralized diagnosis strategy it is supposed that all fault-free subsystem models are used to detect the fault occurrence. However, in practice, some subsystems may not add useful information regarding the fault occurrence, or the same information can be obtained from the other modules, which implies that these modules are not necessary for the synchronous diagnosis scheme. It is important to remark that finding the minimum number of system modules needed for diagnosing the fault occurrence reduces the size of the diagnoser and the memory space required to store it on a computer. A method for the computation of the useful components or subsystems for synchronous fault diagnosis is not carried out in [20].

The simplest way to find all minimal subsets of modules that ensure language synchronous diagnosability is to perform an exhaustive search, computing the verifier for all $2^r - 1$ possible subsets of modules, where r denotes the number of system modules, and selecting those that have smaller cardinality and do not contain another subset of modules. This procedure has a high computational cost. Thus, in this work, we present a method to compute all minimal sets of modules that are necessary to guarantee the synchronous diagnosability of the system language, that, in general, does not require the computation of the verifier for all subsets of system modules. After that, the minimum cardinality sets can be obtained simply by choosing those that have smaller cardinality. In this work, we considered only the permanent faults.

This work is organized as follows. In Chapter 2, we present some preliminary concepts of DES modeled as automata. The notions of diagnosability, considering the classical approach and the synchronous approach are presented in Chapter 3. In addition, we present an example of a system that is synchronously diagnosable which motivates the possibility of reducing the number of modules. In Chapter 4, we present the method to compute the subsystems needed for ensuring synchronous diagnosability, examples to illustrate the implementation of the method, and discussions of the results. The conclusions are drawn in Chapter 5.

Chapter 2

Discrete Event Systems

In Section 2.1 we introduce the notion of languages of a system and some operations with languages, and in Section 2.2 we present the model of a deterministic and non-deterministic automaton, the language of these automata and some basic operations using them.

2.1 Languages

To introduce the concept of languages, it is first necessary to present some notations and definitions. The set formed of all possible events is the *"alphabet"* denoted as Σ. The concatenation of events forms sequences that can be interpreted as *"words"* of a language. The words can be called *strings* or *traces* as well, and the language of a system is the set of traces that the system can execute. The length of a sequence, denoted as $\|s\|$, is the number of events that form it, considering multiple occurrences of the same event. The empty sequence ε is a sequence with zero length.

Definition 2.1 (Language) *A language defined over an event set Σ is a set of finite length sequences formed with events in Σ.*

Example 2.1 *Let $\Sigma = \{a, b\}$. Then $L = \{\varepsilon, b, ab, ba, bab, baba\}$ is a language defined over Σ, where the length of sequence baba is $\|baba\| = 4$.*

Since languages are sets, it is important to remark that all set operations can be applied to languages, such as union, intersection, difference, and complement. Some other operations can be applied to languages, such as the ones presented in the sequel.

2.1.1 Language Operations

Concatenation is an important operation related to the construction of traces from a set of events Σ, traces and languages. For example, the trace *baba* is formed by the

concatenation of the trace ba with the trace ba. The trace ba itself is a concatenation of the event b with event a. It is important to remark that the empty trace ε is the identity element of concatenation operation, meaning that $\varepsilon e = e\varepsilon = \varepsilon e\varepsilon = e$.

Definition 2.2 (Concatenation) *Let $L_1, L_2 \subseteq \Sigma^*$, then the concatenation L_1L_2 is given by:*

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

A trace s is in L_1L_2 if it is formed by the concatenation of $s_1 \in L_1$ and $s_2 \in L_2$.

Let us denote by Σ^* the Kleene-closure of the set of events Σ, which consists of all sequences of finite length that can be formed using elements of Σ including the empty sequence ε. Thus, a language L defined over Σ is a subset of Σ^*. The Kleene-closure operation can also be applied to languages as presented in definition 2.3

Definition 2.3 (Kleene-clousure) *Let $L \subseteq \Sigma^*$. Then the Kleene-closure operation L^* is given by:*

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

Consider now the trace $s = tuv$, where $t, u, v \in \Sigma^*$, t is the prefix of s, u is the subtrace of s and v is the suffix of s. Considering that $t, u, v \in \Sigma^*$, then the traces ε and s are also prefixes, subtraces and sufixes of s. The Prefix-closure of a language L is defined as follows.

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

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

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

Other important operation applied to traces and languages is the natural projection, defined as follows:

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

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

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

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

where $\backslash$ denotes set difference.

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

Another important operation applied to traces and languages is the inverse projection, defined as follows:

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

$$P_s^{l^{-1}}(t) = \{s \in \Sigma_l^* : P_s^l(s) = t\}. \tag{2.1}$$

For a given trace t, formed with events from Σ_s, $P_s^{l^{-1}}(t)$ produces a set formed with all the traces s that can be constructed with Σ_l whose projection $P_s^l(s)$ is equal to t.

Similarly to the projection operation, the inverse projection can be extended to languages by applying Equation 2.1 to all traces t that belong to the language.

The language of a DES is used to model the system behavior by representing all traces that the system is capable of executing. Nonetheless, the representation of the system behavior using only their languages is not simple to work with. Considering this, it is necessary to use another formalism to describe DES to make it easier to analyse and manipulate DES with more complex behavior.

2.2 Automata

An automaton is a device that is capable of representing a language according to well-defined rules, and is formally defined in the sequel [1, 2].

Definition 2.7 (Automaton) *An 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, $f : Q \times \Sigma \to \Sigma$ is the transition function, q_0 is the initial state, and Q_m is the set of marked states.

For the sake of simplicity, unless otherwise stated, the set of marked states Q_m will be omitted from the automata defined in this work.

$\Gamma_G(q)$ is the set of all events $\sigma \in \Sigma$ for which the transition function $f(q, \sigma)$ is defined.

Graphically, an automaton can be represented by an oriented graph called state transition diagram, which can reproduce all characteristics defined in G. The state transition diagram is formed of vertices and edges, represented by circles and arcs, respectively. The states of the system are represented by the vertices and the transition between states are represented by the edges. The events of Σ associated with the transitions appear as labels of the edges. The initial state is represented by an arc with no origin state, and a marked state is represented by two concentric circles. Example 2.2 shows an automaton and its state transition diagram.

Example 2.2 *Consider an automaton* $G = (Q, \Sigma, f, q_0, Q_m)$ *with state set* $Q = \{0, 1, 2, 3\}$*, event set* $\Sigma = \{b, e, h\}$*, transition function defined as* $f(0, b) = 1, f(1, h) = 2, f(2, e) = 3, f(3, h) = 1$ *and active event function given by* $\Gamma_G(0) = \{b\}$, $\Gamma_G(1) = \{h\}$, $\Gamma_G(2) = \{e\}$, $\Gamma_G(3) = \{h\}$*. The initial state* q_0 *is* 0*, and the set of marked states is* $Q_m = \{1, 3\}$*. The state transition diagram representing automaton* G *is depicted in Figure 2.1.*

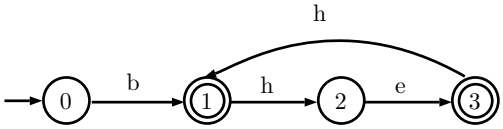


Figure 2.1: State transition diagram of Example 2.2.

Another important definition is of a path in automaton G, which is a sequence $(q_1, \sigma_1, q_2, \ldots, q_{n-1}, \sigma_{n-1}, q_n)$, where $\sigma_i \in \Sigma$ and $q_{i+1} = f(q_i, \sigma_i), i = 1, 2, \ldots, n-1$. A path $(q_1, \sigma_1, q_2, \ldots, q_{n-1}, \sigma_{n-1}, q_n)$ is said to be cyclic if $q_1 = q_n$ and the set of states of a cyclic path forms a cycle.

In the following we present the definition of generated and marked languages.

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

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

The marked language of G *is defined as:*

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

It is important to remark that, in Definition 2.8 the domain of the transition function is extended to $Q \times \Sigma^*$. Additionally, for any G such that $Q \neq \emptyset, \varepsilon \in \mathcal{L}(G)$

The language $\mathcal{L}(G)$ is composed of all traces that can be generated by following the transitions of the state transition diagram starting at the initial state. Consequently, knowing that a trace is only feasible if all its prefixes are also feasible, the generated language $\mathcal{L}(G)$ is, by definition, prefix-closed. Furthermore, if f is a total function over its domain, then $\mathcal{L}(G) = \Sigma^*$. In this work, for the sake of simplicity, the generated language of G, $\mathcal{L}(G)$, is also referred to as L.

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

The 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 next section we present some operations that can be applied to automata.

2.2.1 Operations on automata

There are several operations that can be applied to automata, and those can be separated into two groups: unary and composition operations.

Unary operations

Unary operations are applied to a single automata, altering its state transitions diagram, but keeping its event set the same. In the following we present two examples of unary operations: accessible part and coaccessible part.

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

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

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

In the operation of taking the accessible part of an automaton G, the states which are not reachable from the initial state q_0 and its related transitions are erased from G. Note that this operation does not modify the generated language of G.

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

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

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

In the operation of taking the coaccessible part of G, all states q such that a path from q to a marked state does not exist are deleted.

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

Composition operations

Composition operations applied to DES modeled by automata are those that allow us to combine automata, and the result of the operation is another automaton. In the sequel we present two important composition operations.

Definition 2.11 (Product composition) *Let* $G_1 = (Q_1, \Sigma_1, f_1, q_{0,1}, Q_{m1})$ *and* $G_2 = (Q_2, \Sigma_2, f_2, q_{0,2}, Q_{m2})$ *be two automata. The product composition of* G_1 *and* G_2 *results in automaton:*

$$G_1 \times G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f_{1\times 2}, (q_{0,1}, q_{0,2}), Q_{m1} \times Q_{m2}),$$

where

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

The product composition is known as completely synchronous composition due to the fact that an event can only occurs in the resulting automaton $G_1 \times G_2$ if it occurs simultaneously in G_1 and G_2.

Another important characteristic of the product composition, as consequence of the complete synchronization, is that the generated language of $G_1 \times G_2$ is the intersection of the languages of the automata used in the composition, *i.e.*, $\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$, and if $\Sigma_1 \cap \Sigma_2 = \emptyset$, then $\mathcal{L}(G_1 \times G_2) = \varepsilon$.

In general, complex systems are formed of several components, which work together to accomplish their tasks. The usual way to obtain the global model of the system from the model of its components is by making the parallel composition of the component models. Differently from the product composition, the parallel composition allows each component to maintain its private behavior and synchronize

only the common events of the components. In the following we present the formal definition of the parallel composition.

Definition 2.12 (Parallel composition) *Let* $G_1 = (Q_1, \Sigma_1, f_1, q_{0,1}, Q_{m1})$ *and* $G_2 = (Q_2, \Sigma_2, f_2, q_{0,2}, Q_{m2})$ *be two automata. The parallel composition of* G_1 *and* G_2 *results in automaton:*

$$G_1 \parallel G_2 = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f_{1\parallel 2}, (q_{0,1}, q_{0,2}), Q_{m1} \times Q_{m2}),$$

where

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

It is important to remark that in the parallel composition an event $\sigma \in \Sigma_1 \cup \Sigma_2$ can only occur in the composed automaton $G_1 \parallel G_2$ if it is enabled in G_1 and G_2 simultaneously, and, consequently, is performed in both at the same time. The private events, on the other hand, can be executed whenever possible in its automaton, *i.e.,* events in $\Sigma_1 \backslash \Sigma_2$ can occur in $G_1 \parallel G_2$ when possible in G_1 and events in $\Sigma_2 \backslash \Sigma_1$ can occur in $G_1 \parallel G_2$ when possible in G_2.

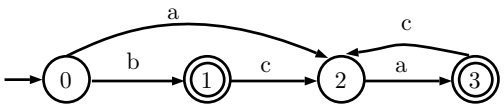
It is important to remark that in the cases where $\Sigma_1 = \Sigma_2$ the parallel composition is exactly the same as the product composition, namely $G_1 \parallel G_2 = G_1 \times G_2$.

The generated language of the parallel composition $G_1 \parallel G_2$ is obtained by using the natural projections $P_i = (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma_i^*$, for $i = 1, 2$. The generated language of the parallel composition is $\mathcal{L}(G_1 \parallel G_2) = P_1^{-1}(\mathcal{L}(G_1)) \cap P_2^{-1}(\mathcal{L}(G_2))$. In the sequel we present an example of product and parallel composition.

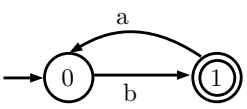
Example 2.3 *Let* $G_1 = (Q_1, \Sigma_1, f_1, q_{0,1})$ *and* $G_2 = (Q_2, \Sigma_2, f_2, q_{0,2})$ *be two automata, with event sets* $\Sigma_1 = \{a, b, c\}$ *and* $\Sigma_2 = \{a, b\}$*, whose state transition diagrams are presented in Figures 2.2a and 2.2b respectively. The automaton* G_{prod} *is obtained by making the product composition of* G_1 *and* G_2*,* $G_{prod} = G_1 \times G_2$*, and the result is shown in Figure 2.3a, and the parallel composition between* G_1 *and* G_2 *results in automaton* $G_{par} = G_1 \parallel G_2$*,, which is presented in Figure 2.3b.*

2.2.2 Automata with partially observed events

In real word systems, the observation of the occurrence of all events is a very difficult task. The number of sensors and the position where they must be placed in order

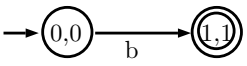


(a) G_1.

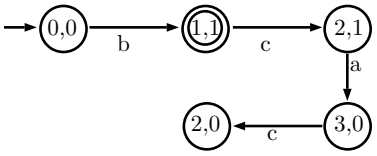


(b) G_2.

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.

to provide the information for the occurrence of events usually make it impossible to observe all events of interest. To represent that, we introduce the notion of unobservable events, which are those not associated with a sensor or fault events, that do not cause immediate change in sensor readings. One way to represent this is that the event set Σ can be partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where $\dot{\cup}$ represents the disjoint union, a union which the sets do not have any element in common.

One way to obtain the observable language of a system, is by applying the projection $P_o(L)$, where $P_o : \Sigma^* \to \Sigma_o^*$, and the unobservable reach of state $q \in Q$.

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

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

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

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

The unobservable reach of a state q_v is the set of states composed of all states reached from q_v by transitions and sequence of transitions with unobservable events. It is possible to build a deterministic automaton from G that generates the observed language of $G, P_o(L)$, using the unobservable reach. This automaton is called the observer automaton of G, denoted as $Obs(G, \Sigma_o)$, defined as follows.

Definition 2.14 (Observer automaton) *The observer of an automaton G with respect to a set of observable events Σ_o, denoted by $Obs(G, \Sigma_o)$, is defined as:*

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

where $q_{obs} \subseteq 2^Q$, f_{obs}, $q_{0,obs}$ and $Q_{m,obs}$ are obtained following the steps of algorithm.

Algorithm 2.1 *Observer automaton*

Input: $G = (Q, \Sigma, f, q_0, Q_m)$, and the observable event set Σ_o, where $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$
Output: Observer automaton $Obs(G, \Sigma_o) = (Q_{obs}, \Sigma_o, f_{obs}, q_{0,obs}, Q_{m,obs})$

1: Define $q_{0,obs} := UR(q_o), Q_{obs} := \{q_{0,obs}\}$ and $\tilde{Q}_{obs} := Q_{obs}$.

2: $\hat{Q}_{obs} := \tilde{Q}_{obs}$ and $\tilde{Q}_{obs} := \emptyset$.

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: $\tilde{Q}_{obs} := \tilde{Q}_{obs} \cup f_{obs}(B, \sigma)$.

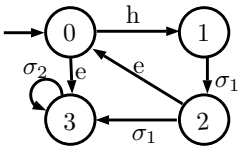
4: $Q_{obs} := Q_{obs} \cup \tilde{Q}_{obs}$.

5: Repeat steps 2 to 4 until all accessible part of $Obs(G, \Sigma_o)$ is constructed

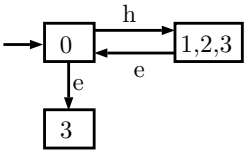
6: $Q_{m,obs} := \{B \in Q_{obs} : B \cap Q_m \neq \emptyset\}$.

In the sequel, we present an example of the construction of the observer $Obs(G, \Sigma_o)$ of G.

Example 2.4 *Consider automaton G shown in Figure 2.4a. The set of states is $Q = \{0, 1, 2, 3\}$ and the set of events is $\Sigma = \{e, h, \sigma_1, \sigma_2\}$, where $\Sigma_o = \{e, h\}$ and $\Sigma_{uo} = \{\sigma_1, \sigma_2\}$. The observer automaton of $G, Obs(G, \Sigma_o)$, computed using Algorithm 2.1, is shown in Figure 2.4b. Let us suppose that trace $s = h\sigma_1 eh$ has been executed. In this case, the observed trace is $P_o(s) = heh$, where $P_o : \Sigma^* \to \Sigma_o^*$. After observed the trace heh, it is impossible to be certain in which state the system is, but it is possible to estimate it, and in this case the estimated states are 1, 2 and 3. This can be seen in Figure 2.4b, where each state represent the state estimate of G after observing a trace $s \in P_o(L)$.*



(a) Automaton G.



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

Figure 2.4: Observer $Obs(G, \Sigma_o)$ of Example 2.4.

2.3 Final Comments

In this chapter, the background of DES is presented. This background includes the definition of languages and their operations, the automaton formalism to represent DES and automata with partially observed events. That background is important to study the diagnosability of a DES. In the next chapter we introduce the concepts of diagnosability and synchronous diagnosability of DES.

Chapter 3

Diagnosability and Synchronous Diagnosability of Discrete Event Systems

Systems are subject to fault events that may alter their expected normal behavior. If the fault event is observable it can be diagnosed trivially, thus, we focus on the diagnosis of unobservable fault events. In this chapter, some preliminary results considering the diagnosis of DES are first presented, and then we introduce the classical definition of diagnosability of DES (SAMPATH *et al.* [6]) in Section 3.1. Then, in Section 3.2, we introduce the definition of synchronous diagnosability.

3.1 Diagnosability of DES

Let G be the automaton model of the system, and let $\mathcal{L}(G) = L$ denote the language generated by G. The set of fault events is denoted by Σ_f, where $\Sigma_f \subseteq \Sigma_{uo}$ and, for the sake of simplicity, in this work, we assume that the set of fault events is composed of only one fault event $\Sigma_f = \{\sigma_f\}$. This simplification is not restrictive since, for systems with more than one fault type, each type of fault can be considered separately [29].

In order to present the definition of language diagnosability of DES, we will 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 fault-free language $L_N \subset L$ denotes the set of all fault-free traces of L, and the subautomaton of G that generates L_N is denoted by G_N. Thus, the set of all faulty traces is defined as $L_F = L \backslash L_N$.

After the definition of fault-free and faulty traces, prior to the definition of diagnosability, it is necessary to state two assumptions on the system under investigation:

A1) The language L generated by G is live. This means that there is a transition defined at each state $q \in Q$, *i.e.*, the system cannot reach a state at which no event is possible.

A2) There does not exist in G any cycle of unobservable events, *i.e.*,

$$\exists n_0 \in \mathbb{N} \text{ such that } \forall st \in L, s \in \Sigma_{uo}^* \Rightarrow \|s\| \leq n_0$$

where $\|s\|$ is the length of a trace s.

After the definition of fault-free and faulty traces, and assumptions $A1$ and $A2$, the definition of diagnosability of the system language can be stated [6].

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 $P_o : \Sigma^* \to \Sigma_o^*$ and Σ_f if:*

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

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

In [1, 6, 7] an automaton, called diagnoser, that can be used to verify the diagnosability of L and also for online fault diagnosis is presented. The procedure to construct the diagnoser automaton G_d is presented in Algorithm 3.1.

Algorithm 3.1 *Diagnoser automaton of the system [6].*

Input: $G = (Q, \Sigma, f, q_0)$, set of fault events Σ_f.
Output: Automaton G_d.

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

2: Compute $G_l = G \| A_l$.

3: Construct the diagnoser automaton $G_d = (Q_d, \Sigma_o, f_d, q_{0,d}) = Obs(G_l, \Sigma_o)$.

In Step 1 of Algorithm 3.1, automaton A_l is 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 presented in Figure 3.1. In Step 2, automaton G_l is computed from the plant model G, as $G_l = G \| A_l$. Such that, if a state of G is reached by a fault-free trace, then it is labeled with N, otherwise, if the state is reached by a trace that contains σ_f, it is labeled with F. In Step 3, the diagnoser automaton G_d is obtained by computing the observer of G_l with respect to its observable events, *i.e.*, $G_d = (Q_d, \Sigma_o, f_d, q_{0,d}) = Obs(G_l, \Sigma_o)$.

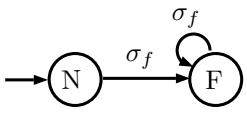


Figure 3.1: Automaton A_l.

It is important to notice that the generated language of G_d is the natural projection of L, *i.e.*, $\mathcal{L}(G_d) = P_o(L)$. Furthermore, it is important to notice that the states of G_d are the state estimates of G_l after the observation of a trace. Consequently, if G_d reaches a state where all labels are F, the fault has occurred and is diagnosed. On the other hand, if G_d reaches a state where all labels are N, the fault has not occurred.

In the cases that there are states labeled with F and N in a state estimate $q_d \in Q_d$, then q_d is called an uncertain state, since after the observation of a trace, it is uncertain if this trace is a fault-free trace or a faulty one. A cycle formed of uncertain states is called an uncertain cycle, and in the cases that an uncertain cycle can be associated with two cycles in G_l, one with states labeled with F and another one with states labeled with N, it is called an indetermined cycle. In order to verify the diagnosability of L it is necessary to search for indeterminate cycles in G_d. If G_d has an indetermined cycle, L is not diagnosable. On the other hand, if G_d does not have indeterminate cycles then L is diagnosable [1, 6, 7].

In the sequel we present an example showing the construction of the diagnoser automaton G_d.

Example 3.1 *Consider the system G depicted in Figure 3.2a. The state set is $Q = \{0, 1, 2, 3, 4\}$, the event set is $\Sigma = \{e, h, \sigma_1, \sigma_2, \sigma_f\}$, where the observable event set is $\Sigma_o = \{e, h\}$, the unobservable event set is $\Sigma_{uo} = \{\sigma_1, \sigma_2, \sigma_f\}$, and the fault event set is $\Sigma_f = \{\sigma_f\}$. In Figure 3.2b automaton $G_l = G \| A_l$ is presented and the diagnoser automaton, depicted in Figure 3.2c, is obtained by computing the observer of G_l with respect to its observable event set Σ_o.*

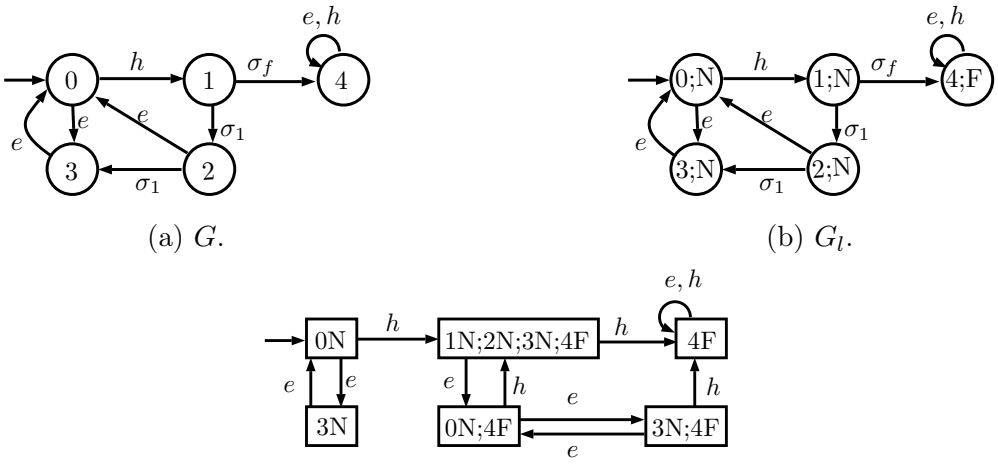


Figure 3.2: System automaton G (a), label automaton G_l (b), and diagnoser automaton G_d (c) from Example 3.1.

Considering that the first observed event is e, we are certain that the fault has not occurred, since state $\{3N\}$ *is reached. However, if the first observed event is h, automaton* G_d *reaches an uncertain state* $\{1N; 2N; 3N; 4F\}$*. After that, if event h is observed again, we are certain that the fault has occurred since state* $\{4F\}$ *is reached. It is important to remark that, in this example, there is an uncertain cycle* $\{\{1N, 2N, 3N, 4F\}, e, \{0N, 4F\}, h, \{1N, 2N, 3N, 4F\}\}$ *which can be associated with two cycles in* G_l*, where one is reached after the occurrence of* σ_f $(h\sigma_f\{eh\}^*)$ *and the other without the occurrence of* σ_f $(h\{\sigma_1 eh\}^*)$*. Thus, L is not diagnosable with respect to* $P_o : \Sigma^* \to \Sigma_o^*$ *and* Σ_f*.*

Even though the diagnoser automaton G_d can be used for the verification of the diagnosability of L, the set of states of G_d may grow exponentially with the cardinality of the system states. To avoid this problem, in MOREIRA *et al.* [24, 30], an algorithm for the construction of a verifier automaton whose cardinality of the set of states grows polynomially with the set of states of the system is presented. It is important to remark that the verifier does not require assumptions on the liveness of the language generated by the system or the nonexistence of cycles of unobservable events, assumptions A_1 and A_2.

In order to use the method proposed in MOREIRA *et al.* [24], we first need to present Algorithm 3.2 to obtain the fault-free and faulty automata [24].

Algorithm 3.2 *Fault-free and faulty model of the system [24].*

Input: $G = (Q, \Sigma, f, q_0)$*, set of fault events* Σ_f*.*
Output: Automata G_N *and* G_F*.*

1: Define $\Sigma_N := \Sigma \backslash \Sigma_f$*.*

2: Define $A_N := (Q_N, \Sigma_N, f_N, q_{0,N})$ *where* $Q_N = \{N\}, q_{0,N} = \{N\}, f_N(N, \sigma) = N$ *for all* $\sigma \in \Sigma_N$.

3: Construct the fault-free automaton $G_N = G \times A_N = (Q_N, \Sigma, f_N, q_{0,N})$.

4: Redefine the event set of G_N *as* Σ_N*, i.e.,* $G_N = (Q_N, \Sigma_N, f_N, q_{0,N})$.

5: Compute automaton G_F*, whose marked language corresponds to the fault behavior of the system, as follows:*

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

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

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

With Algorithm 3.2, it is possible to compute automata G_N and G_F, and with those automata, it is possible to use the method proposed in MOREIRA *et al.* [24]. This method is used to compute a verifier, and the verifier is used to verify if the language is diagnosable with respect to P_o and Σ_f, by following Algorithm 3.3.

Algorithm 3.3 *Diagnosability verification [24].*

Input: $G = (Q, \Sigma, f, q_0)$*, set of fault events* Σ_f*, and* $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$.
Output: Diagnosability decision.

1: Compute G_N *and* G_F *according to Algorithm 3.2.*

2: Define function $R = \Sigma_N \to \Sigma_R$ *as:*

$$R(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_o \\ \sigma_R, & \text{if } \sigma \in \Sigma_{uo} \backslash \Sigma_f \end{cases}.$$

Construct automaton $\tilde{G}_N = (Q_N, \Sigma_R, \tilde{f}_N, q_{0,N})$*, with* $\tilde{f}_N(q_N, R(\sigma)) := f_N(q_N, \sigma)$ *for all* $\sigma \in \Sigma_N$.

3: Compute the verifier automaton $G_V = \tilde{G}_N \| G_F = (Q_V, \Sigma_R \cup \Sigma, f_V, q_{0,V})$.

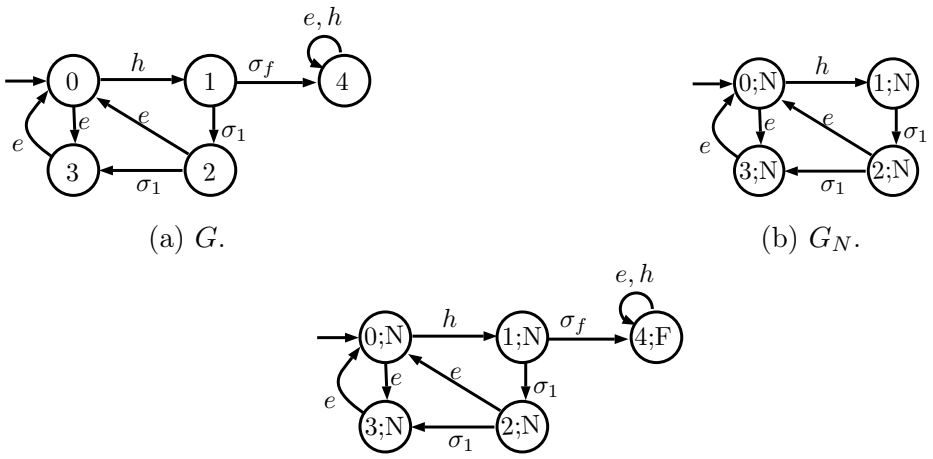
4: Verify the existence of a cyclic path $cl = (q_V^\delta, \sigma_\delta, q_V^{\delta+1}, \ldots, q_V^\gamma, \sigma_\gamma, q_V^\delta)$*, where* $\gamma \geq \delta > 0$*, in* G_V *satisfying the following conditions:*

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

If the answer is yes, then L is not diagnosable with respect to P_o and Σ_f. Otherwise, L is diagnosable.

A state of G_V is given by $q_V = (q_N, q_F)$, where q_N and q_F are the states of $\tilde{G}_N$ and G_F, respectively, and $q_F = (q, q_l)$, where q and q_l are states of G and A_l, respectively.

Example 3.2 *Consider, again, the system G of Example 3.1 depicted in Figure 3.3a. In Figure 3.3b automaton G_N is presented and automaton G_F is shown in Figure 3.3c. In Figure 3.4 automaton $\tilde{G}_N$ is presented and the verifier automaton, depicted in Figure 3.5, is obtained by computing the parallel composition of $\tilde{G}_N$ with G_F, i.e., $G_V = \tilde{G}_N \parallel G_F$.*



(a) G. (b) G_N.

(c) G_F.

Figure 3.3: System automaton G (a), automaton G_N (b), and automaton G_F (c) from Example 3.2.

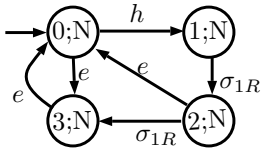


Figure 3.4: Automaton $\tilde{G}_N$ from Example 3.2.

Considering Algorithm 3.3, it is possible to notice that there are two cyclic paths with events that were not renamed, violating the diagnosability. The first one is $\{(0N, 4F)$, e, $(3N, 4F)$, e, $(0N, 4F)\}$; and the second one is $\{(1N, 4F)$, σ_{1R}, $(2N, 4F)$, e, $(0N, 4F)$, h, $(1N, 4F)\}$. Thus, the language is not diagnosable.

Example 3.2 shows the implementation of the verifier presented in MOREIRA *et al.* [24]. This verifier advantage is to present a diagnosability decision of the

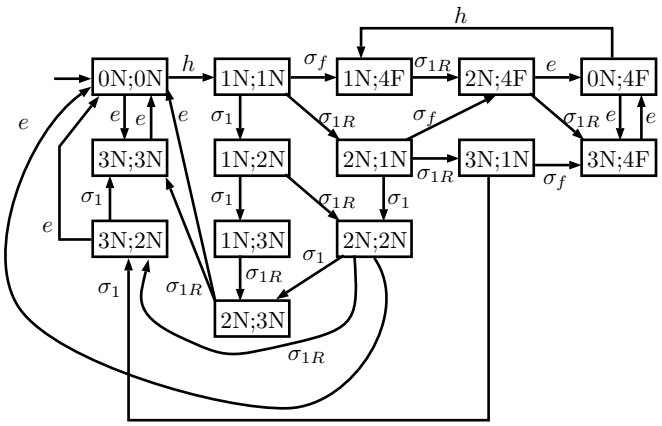


Figure 3.5: Automaton G_V from Example 3.2.

language in polynomial time in the number of states and events of the system. However, in the cases where the system is complex, composed of several subsystems, the plant model may grow exponentially with the number of subsystems, making the implementation cost of the monolithic diagnoser based on G very high. In these cases, other schemes have been proposed in the literature, such as the synchronous diagnosis. In the following we present this architecture.

3.2 Centralized Synchronous diagnosability of DES

In [20], the definition of centralized synchronous diagnosability of a DES is presented. To do so, it is assumed that the system is composed of r modules $G_k = (Q_k, \Sigma_k, f_k, q_{0,k})$, $k = 1, \ldots, r$, *i.e.*, the composed plant is given by $G = \|_{k=1}^{r} G_k$. It is also assumed that the event set of each module G_k can be partitioned as $\Sigma_k = \Sigma_{k,o} \dot{\cup} \Sigma_{k,uo}$, where $\Sigma_{k,o}$ and $\Sigma_{k,uo}$ denote the sets of observable and unobservable events of G_k, respectively. In this scheme, if an event is observable for one module G_i, and is defined for another module G_j, then it is observable for G_j. In addition, each component has its fault-free behavior modeled by automaton $G_{N_k} = (Q_{N_k}, \Sigma_k \setminus \Sigma_f, f_{N_k}, q_{0,k})$, and it is important to remark that in this approach, assumptions A_1 and A_2 are not required.

Figure 3.6 presents the synchronous diagnosis architecture. In this strategy, a state observer D_k is constructed for each module, performing the online state estimation of each fault-free model G_{N_k}. If an event $\sigma \in \Sigma_{k,o}$ generated by the plant is observed by D_k, and σ is feasible in at least one state of the current state estimate of G_{N_k}, then the state estimate is updated. Otherwise, if σ is not feasible for all states of the current state estimate of G_{N_k}, then D_k indicates that the fault has occurred. This leads to the following definition of synchronous diagnosability [20].

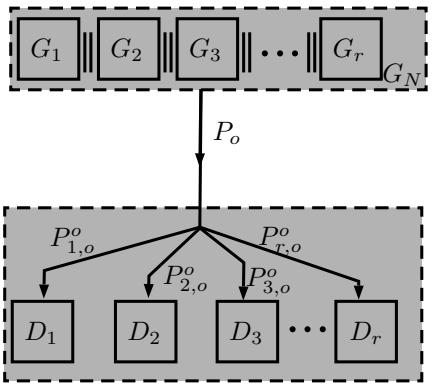


Figure 3.6: Synchronous diagnosis architecture.

Definition 3.3 (Synchronous diagnosability) *Let $G_N = \|_{k=1}^{r} G_{N_k}$, and let L_{N_k} denotes the language generated by G_{N_k}, for $k = 1, \ldots, r$. Let $P_o : \Sigma^* \to \Sigma_o^*$, with $\Sigma_o = \cup_{k=1}^{r} \Sigma_{k,o}$. Then, L is synchronously diagnosable with respect to P_o, L_{N_k}, $k = 1, \ldots, r$, and Σ_f if*

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

where $L_{N_a} = \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$, such that $P^o_{k,o} : \Sigma_o^* \to \Sigma_{k,o}^*$, and $P_{k,o} : \Sigma^* \to \Sigma_{k,o}^*$ are projections. □

Remark 3.1 *Since $P_o(L_N) \subseteq \cap_{k=1}^{r} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k})) = L_{N_a}$ [28], then a language can be diagnosable but not synchronously diagnosable.* □

According to Definition 3.3, the system language L is synchronously diagnosable if any occurrence of the fault event σ_f can be detected after a number $z \in \mathbb{N}$ of event occurrences after the fault, by at least one local diagnoser D_k constructed based on module G_{N_k}. In order to verify the the centralized synchronous diagnosability of a language, two algorithms were proposed in CABRAL and MOREIRA [20]. Algorithm 3.4 is one approach to compute the fault-free behavior models G_{N_k} from the system modules G_k; and Algorithm 3.5 is used to verify the synchronous diagnosability of the language of a composed system.

Algorithm 3.4 *Fault-free behavior models of the system components [20].*

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 G_N according to Algorithm 3.2 [24].*

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 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 set $\Sigma_{N_k} := \Sigma_k \backslash \Sigma_f$ for $k = 1, \ldots, r$.

In the sequel, we present an example to illustrate that, in some cases, even in modules where the fault event is not modeled, the fault-free behavior may be restricted, due to the interactions with other modules.

Example 3.3 *Consider a system G shown in Figure 3.7, with event set $\Sigma = \{a, b, c, d, \sigma_1, \sigma_f\}$, composed of two subsystems G_1 and G_2, shown respectively in Figures 3.8a and 3.8b. Automaton G is obtained by the parallel composition of G_1 and G_2, i.e., $G = G_1 \| G_2$. The event sets of G_1 and G_2 are, respectively, $\Sigma_1 = \{a, b, c, \sigma_1, \sigma_f\}$ and $\Sigma_2 = \{b, d, \sigma_1\}$, where $\Sigma_{1,o} = \{a, b, c\}$, $\Sigma_{1,uo} = \{\sigma_1, \sigma_f\}$, $\Sigma_{2,o} = \{b, d\}$, and $\Sigma_{2,uo} = \{\sigma_1\}$. Following step 1 of Algorithm 3.4, G_N, showed in Figure 3.9, is obtained, which is the automaton that models the fault-free behavior of G. Notice that according to G_N the transition $(0, \sigma_1, 2)$ of automaton G_2 is only allowed after the occurrence of σ_f, and, consequently, even when the fault event is not modeled in G_2, the transition $(0, \sigma_1, 2)$ does not belong to the fault-free behavior of G_2. Automata G_{N_1} and G_{N_2}, obtained by following step 4 of Algorithm 3.4, are presented in Figure 3.10a and 3.10b, respectively.*

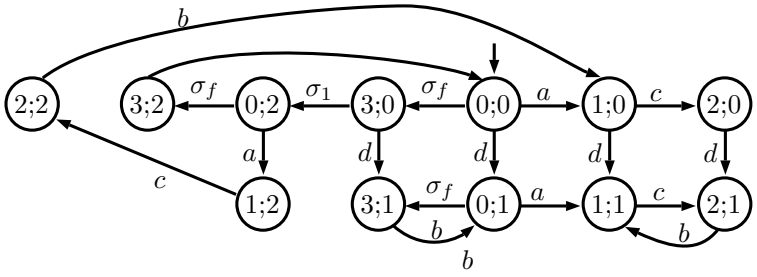
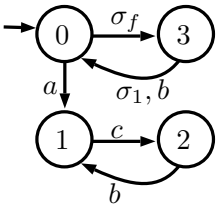
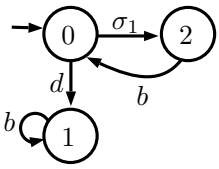


Figure 3.7: Automaton G of Example 3.3.



(a) Automaton G_1.



(b) Automaton G_2.

Figure 3.8: Automata G_1 and G_2 of Example 3.3.

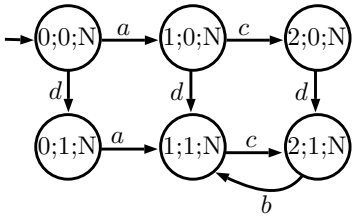
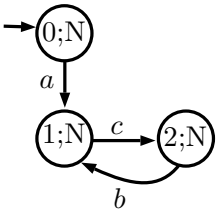
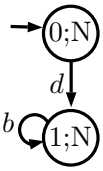


Figure 3.9: Automaton G_N of Example 3.3.



(a) Automaton G_{N_1}.



(b) Automaton G_{N_2}.

Figure 3.10: Automata that represents the fault-free behavior for each local diagnoser of Example 3.3.

After computing the fault-free behavior of each module, we can present the algorithm for the verification of the synchronous diagnosability of the language of a system proposed by CABRAL and MOREIRA [20].

Algorithm 3.5 *Synchronous Diagnosability Verification [20].*

Input: System modules G_k, for $k = 1, \ldots, r$.
Output: Synchronous diagnosability decision.

1: Compute automaton G_F according to Algorithm 3.2.

2: Compute automaton G_{N_k} by following the steps of Algorithm 3.4.

3: Compute automaton $G_N^R = (Q_N^R, \Sigma^R, f_N^R, q_0)$:

3.1: Define function $R_k = \Sigma_{N_k} \to \Sigma_{N_k}^R$ as:

$$R_k(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \in \Sigma_{k,o}, \\ \sigma_{R_k}, & \text{if } \sigma \in \Sigma_{k,uo}. \end{cases}.$$

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 $q_l \in \{N, F\}$.

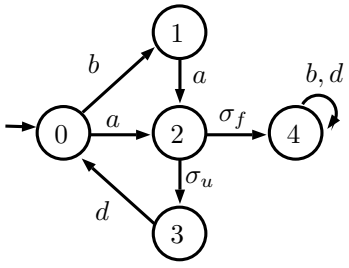
5: Verify the existence of a cyclic path $cl = (q_V^{\delta}, \sigma_{\delta}, q_V^{\delta+1}, \ldots, q_V^{\gamma}, \sigma_{\gamma}, q_V^{\delta})$*, where* $\gamma \geq \delta > 0$*, in* G_V^{SD} *such that:*

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

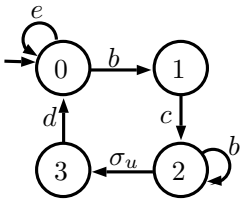
If the answer is yes, then L *is not synchronously diagnosable with respect to* $L_{N_k}, P_{k,o}^o : \Sigma_o^* \to \Sigma_{k,o}^*, P_{k,o} : \Sigma^* \to \Sigma_{k,o}^*$*, for* $k = 1, \ldots, r, P_o : \Sigma^* \to \Sigma_o^*$ *and* Σ_f*. Otherwise,* L *is synchronously diagnosable.*

In the following, to show the augmented fault-free language generated in the process, we present an example of the verification of synchronous diagnosis using Algorithm 3.5.

Example 3.4 *Consider a system* G*, composed of two components* G_1 *and* G_2*, presented, respectively, in Figures 3.11a and 3.11b. The automaton that models the global system* $G = G_1 \| G_2$ *is shown in Figure 3.12, the set of events is* $\Sigma = \{a, b, c, d, e, \sigma_u, \sigma_f\}$ *where* $\Sigma_o = \{a, b, c, d, e\}$*,* $\Sigma_{uo} = \{\sigma_u, \sigma_f\}$*,* $\Sigma_f = \{\sigma_f\}$*,* $\Sigma_1 = \{a, b, d, \sigma_u, \sigma_f\}$*,* $\Sigma_2 = \{b, c, d, e, \sigma_u\}$*,* $\Sigma_{1,o} = \{a, b, d\}$ *and* $\Sigma_{2,o} = \{b, c, d, e\}$*. Following Algorithm 3.5 in Step 1 automaton* G_F *is constructed, depicted in Figure 3.13. Continuing to Step 2, automata* G_{N_1} *and* G_{N_2} *are computed and presented in Figures 3.14a and 3.14b, respectively. In the following, in Step 3, by applying function* $R_k(\sigma)$*, automata* $G_{N_1}^R$ *and* $G_{N_2}^R$ *are obtained and shown in Figure 3.15. Automaton* G_N^R *is constructed as the parallel composition of* $G_{N_1}^R$ *and* $G_{N_2}^R$*, i.e.,* $G_N^R = G_{N_1}^R \| G_{N_2}^R$*, presented in Figure 3.16. The process to obtain* G_N^R *generates an augmented generated language, and in order to represent that, the states that represents the growth of faulty-free language, which means, the states that does not exist in* G_N *are indicated in gray. Finally, in Step 4 we obtain the synchronous verifier automaton* G_V^{SD}*, shown in Figure 3.17.*



(a) Automaton G_1.



(b) Automaton G_2.

Figure 3.11: Automata G_1 and G_2 of Example 3.4.

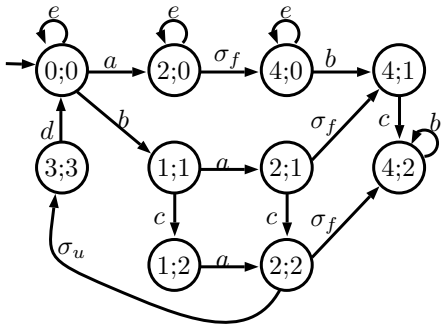


Figure 3.12: Automaton G of Example 3.4.

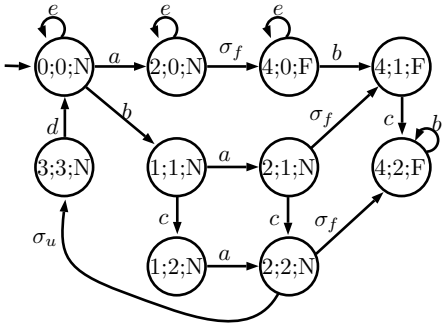
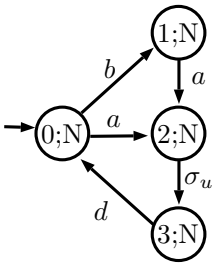
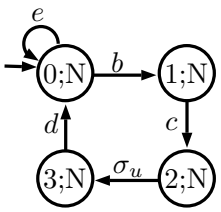


Figure 3.13: Automaton G_F of Example 3.4.

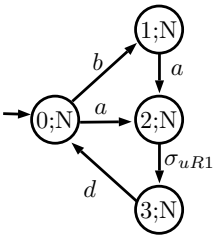


(a) Automaton G_{N_1}.

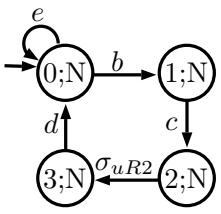


(b) Automaton G_{N_2}.

Figure 3.14: Automata G_{N_1} and G_{N_2}, representing the fault-free languages of each module of Example 3.4.



(a) Automaton $G^R_{N_1}$.



(b) Automaton $G^R_{N_2}$.

Figure 3.15: Automata $G^R_{N_1}$ and $G^R_{N_2}$, representing the fault-free languages of each module, with the unobserved events renamed in order to make them particular events of Example 3.4.

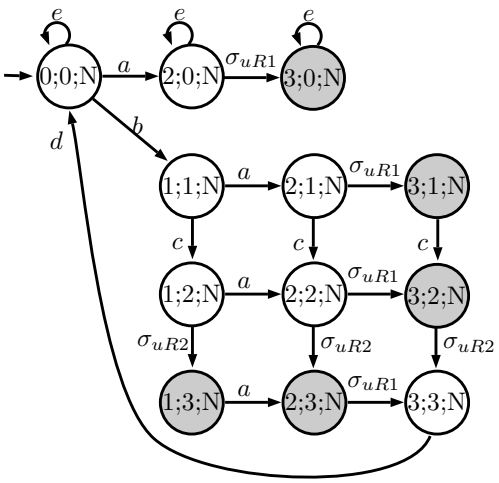


Figure 3.16: Automaton G_N^R, representing the augmented fault-free language of of the system considering the renamed unobserved events as particular events of Example 3.4.

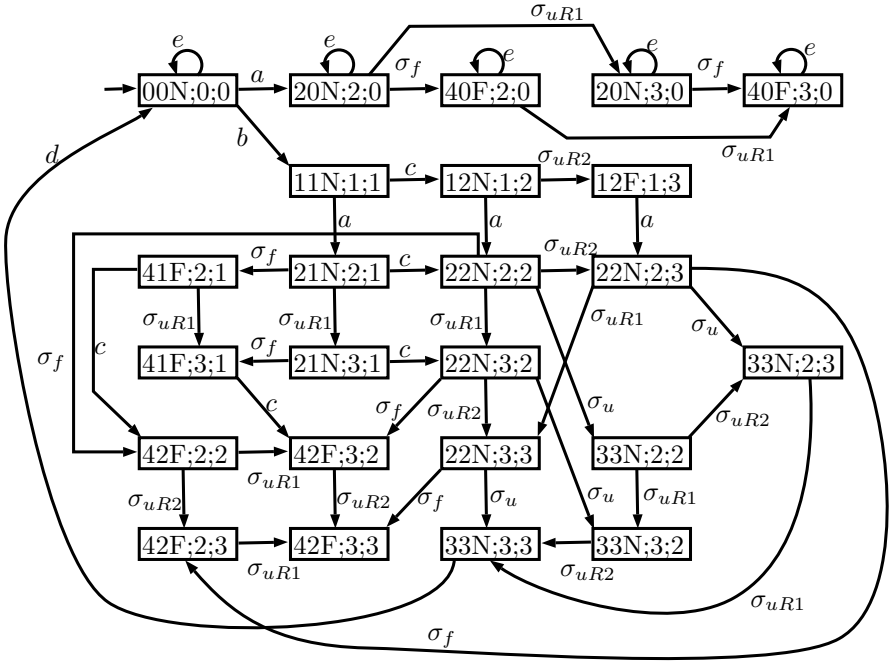


Figure 3.17: Automaton G_V^{SD}, representing the synchronous diagnosis verifier for the system of Example 3.4.

In this example, it is possible to notice that there exists a cyclic path in $G_V^{SD}, (\{4,0,F;2,0\}, e, \{4,0,F;2,0\})$ *labeled with* F *such that at least one transition is labeled with a non-renamed event, event* e*, thus, we conclude that* L *is not synchronously diagnosable with respect to* $L_{N_1}, L_{N_2}, P_{1,o}^o : \Sigma_o^* \to \Sigma_{1,o}^*$*,* $P_{2,o}^o : \Sigma_o^* \to \Sigma_{2,o}^*$*,* $P_{1,o} : \Sigma^* \to \Sigma_{1,o}^*$*,* $P_{2,o} : \Sigma^* \to \Sigma_{2,o}^*$*,* $P_o : \Sigma^* \to \Sigma_o^*$ *and* Σ_f*.*

In example 3.4, we presented the implementation of Algorithms 3.4 and 3.5, that results in the diagnosability decision for the system. In the example, the system

is not synchronously diagnosable with respect to $L_{N_1}, L_{N_2}, P^o_{1,o} : \Sigma^*_o \to \Sigma^*_{1,o}$, $P^o_{2,o} : \Sigma^*_o \to \Sigma^*_{2,o}$, $P_{1,o} : \Sigma^* \to \Sigma^*_{1,o}$, $P_{2,o} : \Sigma^* \to \Sigma^*_{2,o}$, $P_o : \Sigma^* \to \Sigma^*_o$ and Σ_f. In the following, we present a practical example, presented in [20], composed of two modules that is synchronously diagnosable using only the second module.

The system is the cube assembly mechatronic system of the manufecturer Christiani [31], installed at the Laboratory of Control and Automation of the Federal University of Rio de Janeiro. Figure 3.18 presents the schematic of the system, and this mechatronic system is composed of two modules: *(i)* a conveyor belt with a sensor testing unit that can be fed with plastic or metallic cube halves; and *(ii)* a handling unit composed of a robotic arm, which has a pneumatic mechanism that activates a suction cup in order to pick up, transport and deliver pieces to a press used to assemble a cube.

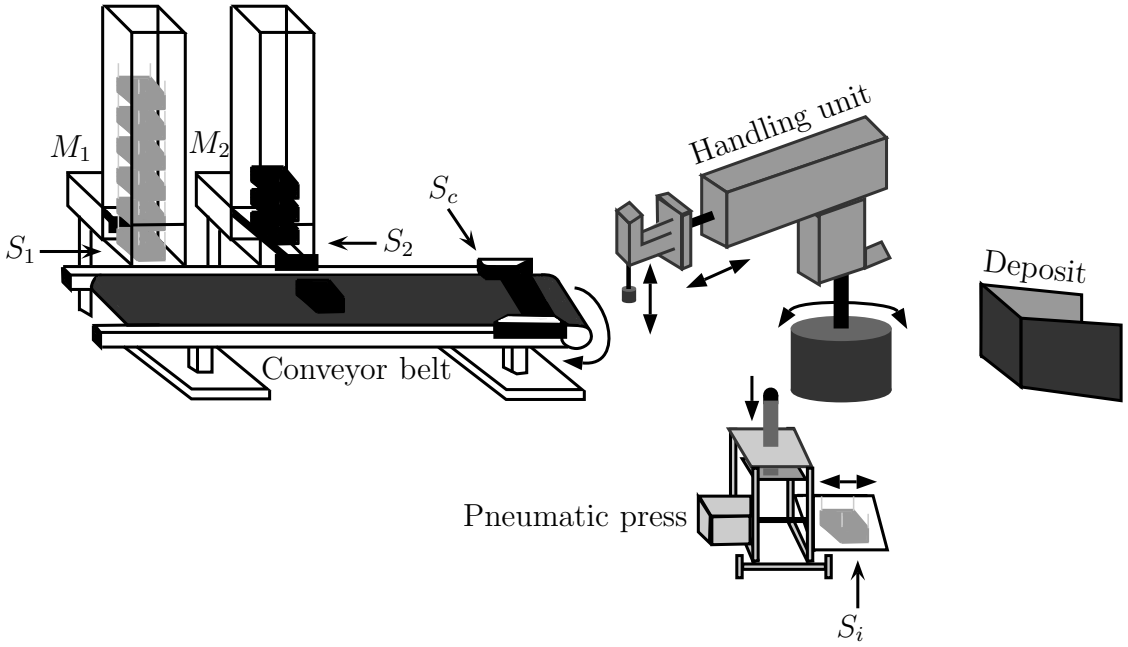
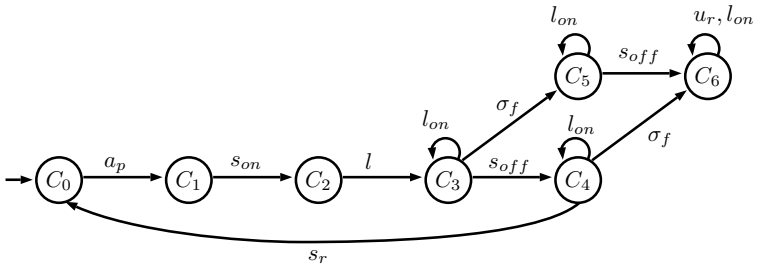
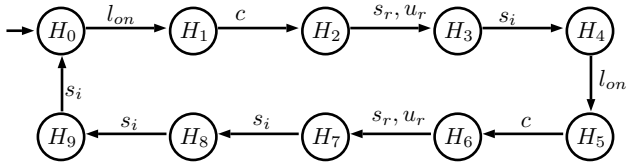


Figure 3.18: Schematic of the mechatronic system installed in Laboratory of Control and Automation of the Federal University of Rio de Janeiro.

The automated system was designed to deliver two cube halves to the press, and then discard this two halves without assembling them. It starts when the conveyor belt is fed with a cube half that is delivered to the handling unit. Then, the robotic arm allocates the cube half in the press and wait for the second half. In the sequel, a plastic half is delivered to the conveyor belt and is transported to the handling unit. After that, the second cube half is delivered to the press by the robotic arm, and then, both halves are discarded by the robotic arm, once at a time. The automata that models the conveyor belt and the handling unit are, respectively G_1, presented in Figure 3.19a, and G_2, presented in 3.19b. In order to understand the automaton models, the states and events are described, respectively, in Tables 3.1 and 3.2.



(a) G_1.



(b) G_2.

Figure 3.19: System components from Example 3.5.

The robotic arm model uses a high speed counter and a inductive sensor. The high speed counter is triggered when the arm starts to turn, and when the high speed counter reaches a specific value, representing an angular position, the robotic arm stops. As a routine to avoid positioning errors, after delivering a piece to the press or discarding a piece, the robotic arm is rotated to a position where an inductive sensor is activated and the high speed counter is reseted. This action is modeled by event s_i, meaning that when s_i occurs, the process of removing or delivering a cube half to the press is completed, and then the robotic arm is ready to remove a piece from the conveyor belt or from the press.

The malfunctioning of the suction cup of the robotic arm is modeled as the fault event σ_f. If the fault event occurs, the robotic arm will not be able to pick up the cube halves. In that case, the pieces will not be removed from the conveyor belt and event s_r does not happen. On the other hand, the robotic arm still tries to pick up the pieces, and the unsuccessful attempt is modeled as u_r.

As there are no sensors in the robotic arm or in the press in order to indicate the presence of the pieces, the behavior of the G_2 is the same, represented by the parallel transitions between H_2 and H_3, as much as between the states H_6 and H_7, labeled with s_r and u_r. Regarding the automaton that models the conveyor belt, G_1, the fault event changes the behavior of the system, once the conveyor belt cannot be switched on if the cube is not removed from it by the robotic arm.

In order to verify the synchronous diagnosability of the system, it is necessary to compute the fault free behavior of each component. Notice that for the automaton that models the fault-free behavior of the handling unit, G_{N_2}, depicted in Figure 3.20b, the only difference is that event u_r is removed. For the automaton that models the fault-free behavior of the conveyor belt, G_{N_1}, presented in Figure 3.20a,

Table 3.1: States of G.

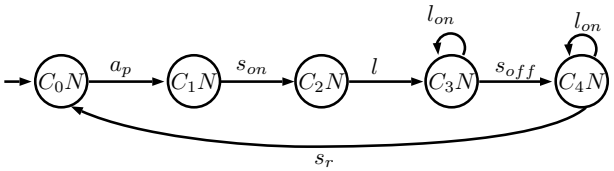
State	Meaning
C_0	Conveyor belt switched off and no cube halves on it
C_1	Conveyor belt switched off and one cube halves on it
C_2	Conveyor belt switched on and one cube halves on it
C_3	Conveyor belt switched on and no cube halves at its end
C_4	Conveyor belt switched off and no cube halves at its end
C_5	Conveyor belt switched on and no cube halves after fault
C_6	Conveyor belt switched off and no cube halves after fault
H_0	Robotic arm ready to remove the first cube half
H_1	Waiting the command to remove the first cube half
H_2	Removing the first cube half from the conveyor belt
H_3	Delivering the first cube half to the press
H_4	Robotic arm ready to remove the second cube half
H_5	Waiting the command to remove the first cube half
H_6	Removing the second cube half from the conveyor belt
H_7	Delivering the second cube half to the press
H_8	Robotic arm discarding the first cube half from the press
H_9	Robotic arm discarding the second cube half from the press

Table 3.2: Events of G.

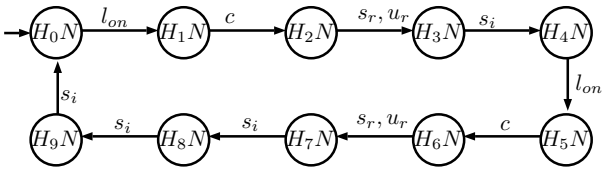
Event	Meaning
a_p	A cube half arrives in the conveyor belt
s_{on}	The conveyor belt is switched on
l	A cube half reaches the end of the conveyor belt
s_{off}	The conveyor belt is switched off
s_r	A cube half is successfully removed from the conveyor belt
l_{on}	Cube half at the end of the conveyor belt
u_r	Unsuccessful attempt to remove a piece from the conveyor belt
σ_f	The robotic suction cup fails
c	Command to remove a piece from the conveyor belt
s_i	Inductive sensor is activated

the events σ_f and u_r are removed, and the accessible part is a version of a fault-free behavior of the system. In the sequel we present Example 3.5, showing that it is possible for a system to be synchronously diagnosable using a subset of modules, in this case, only the second module. Starting now, in order to let the figures more comprehensible and avoid the excess of information, for the verifiers automata, we will omit the label *"N"* in the components of $G^R_{N_i}$, keeping the labels *"N"* and *"F"* for the components of G_F in the states.

Example 3.5 *Consider the mechatronic system G described previously. The set of events of G_1 and G_2 are $\Sigma_1 = \{a_p, s_{on}, l, s_{off}, s_r, l_{on}, u_r, \sigma_f\}$ and $\Sigma_2 = \{c, s_i, s_r, l_{on}, u_r\}$, respectively, where the set of observable events are $\Sigma_{1,o} = \{a_p, s_{on}, l, s_{off}, s_r, l_{on}\}$ and $\Sigma_{2,o} = \{c, s_i, s_r, l_{on}\}$, and the set of unobservable events are $\Sigma_{1,uo} = \{u_r, \sigma_f\}$ and $\Sigma_{2,uo} = \{u_r\}$. The objective is to show that the language is*



(a) G_{N_1}.



(b) G_{N_2}.

Figure 3.20: Fault-free modules automata from Example 3.5.

synchronously diagnosable with a subset of modules, composed only of one module.

The automata that models the fault-free languages and faulty language, G_F, are presented in Figures 3.20 and 3.21, respectively. In this case, the automata that models the fault-free languages of each module with renamed events are the same as the automata that models the fault-free languages. The verifiers $G_V^{\{1\}}$ and $G_V^{\{2\}}$ are computed as the parallel composition of the renamed fault-free automaton and the automaton that models the faulty behavior, i.e., $G_V^{\{1\}} = G_F \parallel G_{N_1}N^R$ and $G_V^{\{2\}} = G_F \parallel G_{N_2}N^R$, presented in 3.23 and 3.24, respectively. Notice that in verifier $G_V^{\{2\}}$ there does not exist any cyclic path labeled with F in which at least one transition is performed by a non-renamed event. Thus, we conclude that L is synchronously diagnosable with respect to $L_{N_2}, P_{2,o}^o : \Sigma_o^* \to \Sigma_{2,o}^*, P_o : \Sigma^* \to \Sigma_o^*$, and Σ_f.

3.3 Final comments

In this chapter, two notions of fault diagnosability of DES modeled by automata are introduced, starting with the classical definition of diagnosability presented in the seminal work of SAMPATH *et al.* [6]. Due to the difficulty associated with the growth of the diagnoser with the number of modules and states, a new architecture was proposed by [20], called centralized synchronous diagnosis, an architecture that relies on the modularity of the system.

Even with the synchronous diagnosis approach, we notice that in cases with great number of system components, it is possible that the diagnosability is assured without computing a verifier with all system components. In the next chapter, an approach to perform a synchronous diagnosis avoiding using all system components is proposed.

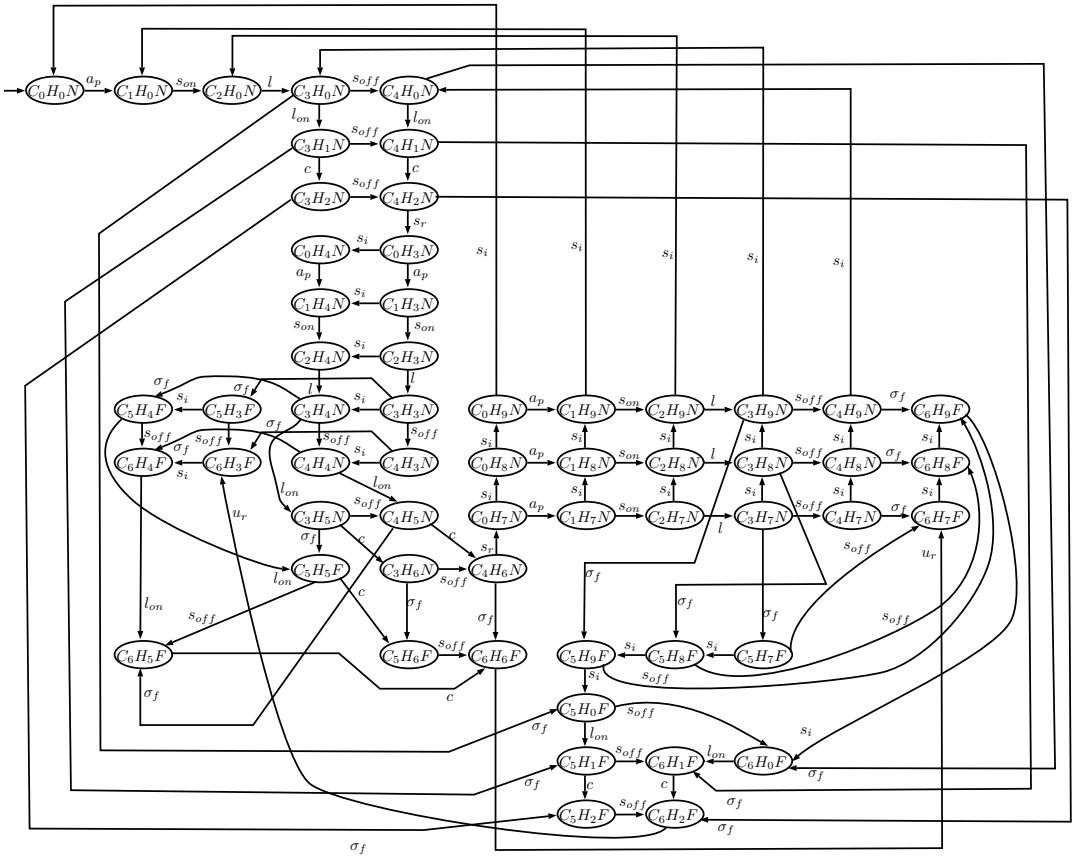


Figure 3.21: Automaton G_F from Example 3.5.

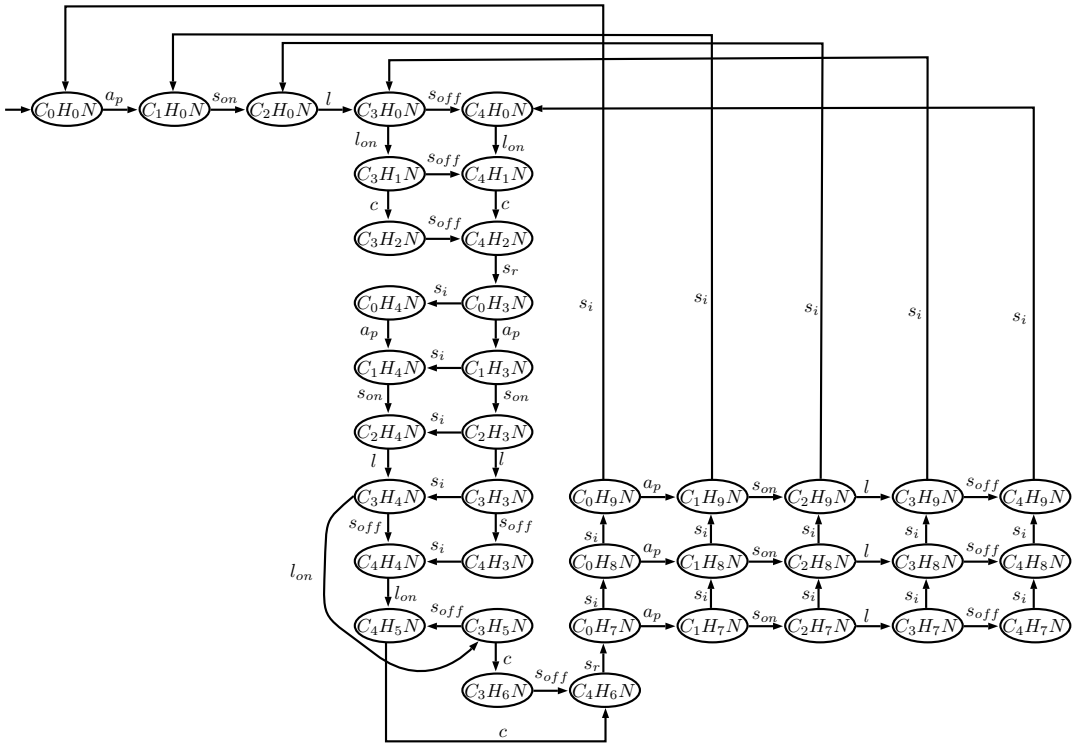


Figure 3.22: Automaton G_N from Example 3.5.

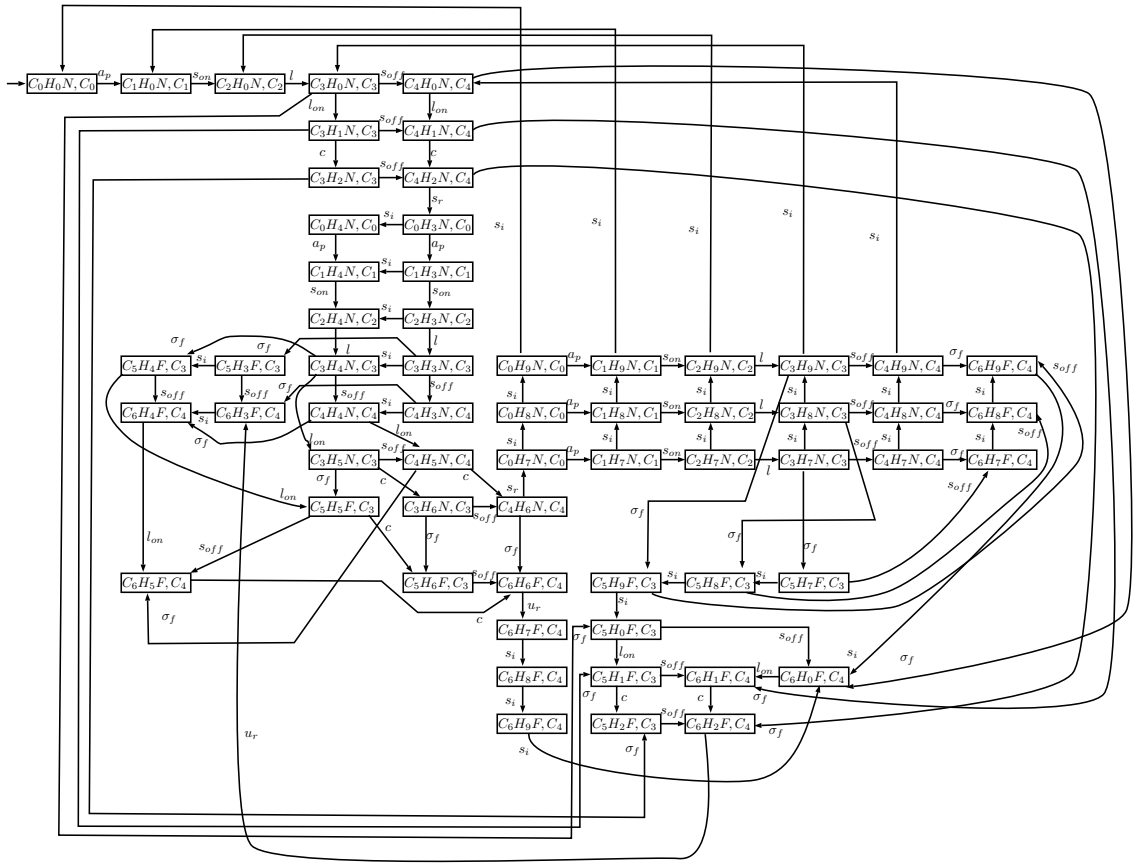


Figure 3.23: Verifier of module 1 from Example 3.5. $G_V^{\{1\}} = G_F \| G_{N_1}^R$.

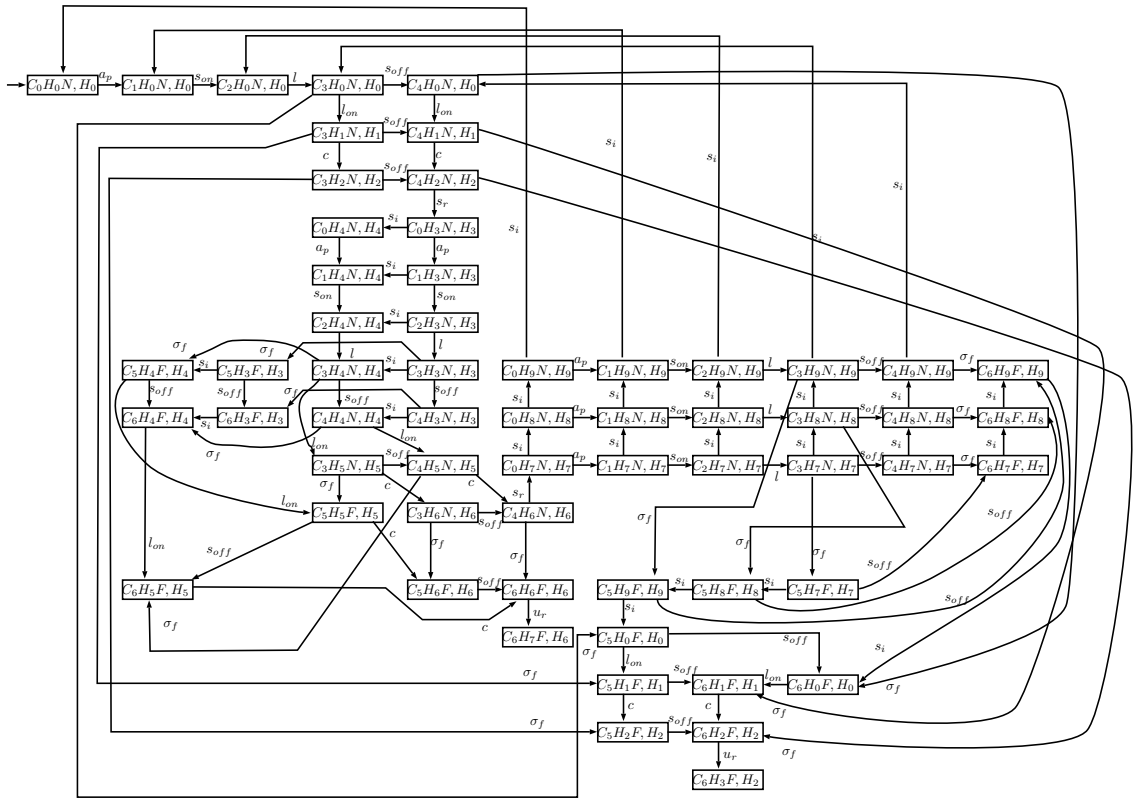


Figure 3.24: Verifier of module 2 from Example 3.5. $G_V^{\{2\}} = G_F \| G_{N_2}^R$.

Chapter 4

Optimal Selection of Subsystems for Ensuring Synchronous Diagnosability

The main advantage of the synchronous diagnosis approach is that the size of the diagnoser grows polynomially with the number of system modules, which reduces the memory space required to store the diagnoser in comparison with traditional techniques. This reduction can be, in some cases, even greater since, depending on the fault and on the system model, the modules needed for diagnosis can be a subset of the complete set of system modules. In Section 4.1, we present a method to compute all minimal subsets that ensure the synchronous diagnosability of the system language. Then, in Section 4.2 we present the results and discussion.

4.1 Method for the Computation of all Minimal Subsets that Ensure Synchronous Diagnosticability

Let $I_r = \{1, 2, \ldots, r\}$ denote the set of indices of all system components. Thus, our objective is to find all minimal subsets $B \in 2^{I_r}$, such that L is synchronously diagnosable with respect to P_o, L_{N_k}, for $k \in B$, and Σ_f, *i.e.,* we want to find the minimal sets B such that

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

where $\dot{L}_{N_a} = \cap_{k \in B} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$. It is important to remark, as it can be straightforwardly deduced from Definition 3.3, that if the system language L is synchronously diagnosable with respect to B, then it is synchronously diagnosable with respect to any subset $B' \in 2^{I_r}$ such that $B \subset B'$, *i.e.*, the monotonicity property is valid, as stated in Theorem 4.1.

Theorem 4.1 *If the system language L is synchronously diagnosable with respect to B, then, the monotonicity property is valid, and the system language is synchronously diagnosable with respect to any subset $B' \in 2^{I_r}$ such that $B \subset B'$*

Proof: Considering the synchronous diagnosability with respect to a subset of modules B, if a language is synchronously diagnosable with respect to P_o, L_{N_k}, for $k \in B$, and Σ_f, it means that

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

where $\dot{L}_{N_a} = \cap_{k \in B} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k}))$.

Suppose now that $\exists s \in L_F$ and $st \in L_F$ such that $(P_o(st) \in P_{j,o}^{o^{-1}}(P_{j,o}(L_{N_j}))$. In this case,

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

$$(P_o(st) \notin \cap_{k \in B} P_{k,o}^{o^{-1}}(P_{k,o}(L_{N_k})) \cap P_{j,o}^{o^{-1}}(P_{j,o}(L_{N_j})).$$

Thus, the language of the system is synchronously diagnosable with respect to any B' such that $B \subseteq B'$.

□

Thus, if we obtain all minimal subsets $B \in I_r$ that ensure synchronous diagnosability, then we are able to provide all possible subsets of modules that ensure synchronous diagnosability. In this work, those subsets are called Synchronous Diagnosis Modular bases as the Definition 4.1.

Definition 4.1 *The subset of indices $B \in 2^{I_r}$ such that the associated modules ensure the synchronous diagnosability of the system language with respect to P_o, L_{N_k}, for $k \in B$, and Σ_f, is called a Synchronous Diagnosis Modular Basis (SDMB).*

□

In Example 3.5, we presented a system in which it is possible to verify the synchronous diagnosability of a system with a subset of modules, in that case, with one module. In a more complex system, it is possible to reach a subset B that forms an SDMB starting with one module and adding modules to it.

A method to verify if B forms an SDMB can be obtained by constructing the verifier restricted to the modules associated with B, $G_V^B = G_F \| (\|_{k \in B} G_{N_k}^R)$, and then searching for the existence of a cyclic path in G_V^B that violates the synchronous diagnosability condition. Thus, if an exhaustive search method is used to find all minimal SDMB, then it is necessary to compute $2^r - 1$ verifiers G_V^B, for each non-empty subset $S \in 2^{I_r}$, and therefore the minimum SDMB will be those with smallest cardinality. In the exhaustive search method, to compute a new verifier, adding a

module to a subset, it is necessary to compute the parallel composition of the verifier of that subset with the verifier of the module to be added, *i.e.*, $G_V^{B'} = G_V^B \| G_V^j$, where j is the module to be added.

In order to mitigate the exponential complexity problem, we present a test to avoid the computation of a verifier. The test consists in selecting a path that violates the synchronous diagnosability in the verifier of subset B, compute an automaton whose generated language is the prefix-closure of the sequence associated with the selected path $G_{V_p}^B$. Instead of computing the parallel composition of the verifier of subset B with the module to be added, the test is to compute the parallel composition of the path automaton $G_{V_p}^B$ with the verifier of the module to be added G_V^j. If exists a sequence that violates the synchronous diagnosability in the parallel composition $G_{V_p}^B \| G_V^j$, the verifier composed by the subset $B \cup j$ does not need to be computed because the language is not synchronously diagnosable with respect to P_o, L_{N_k}, for $k \in B \cup j$, and Σ_f, as stated in Theorem 4.2.

Theorem 4.2 *If there exists a sequence $s \in L$ that violates the synchronous diagnosability with respect to P_o, L_{N_k}, for $\forall k \in B$, and Σ_f, and s violates the synchronous diagnosability with respect to P_o, L_{N_j}, and Σ_f then s violates the synchronous diagnosability with respect to P_o, L_{N_k}, for $k \in B \cup \{j\}$, where $j \notin B$, and Σ_f.*

Proof: The proof can be straightforwardly deduced from Definition 3.3. The language is not synchronously diagnosable with respect to P_o, L_{N_k}, for $\forall k \in B$, and Σ_f, and the language is not synchronously diagnosable with respect to P_o, L_{N_j}, where $j \notin B$, and Σ_f, the language is not synchronously diagnosable with respect to P_o, L_{N_k}, for $k \in B \cup \{j\}$, and Σ_f. □

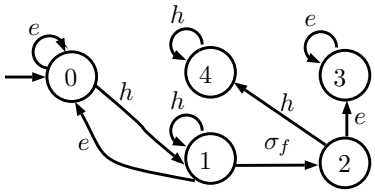
This avoids computational costs due to the fact that it is possible to avoid the computation of parallel composition of automata with elevated number of states and transitions, computing only the parallel composition of a reduced automaton and the verifier of the module to be added. In the following, we present an example to illustrate that situation.

Example 4.1 *Consider a system G, composed of four modules, G_1, G_2, G_3 and G_4 showed in Figure 4.1, where their event sets are $\Sigma_1 = \{a, c, e, g, \sigma_1\}, \Sigma_2 = \{e, h, \sigma_1, \sigma_2, \sigma_f\}, \Sigma_3 = \{b, d, h, \sigma_f\}, \Sigma_4 = \{e, h, \sigma_f\}$, with observable event set $\Sigma_o = \{a, b, c, d, e, h, g\}$, unobservable event set $\Sigma_{uo} = \{\sigma_1, \sigma_2, \sigma_f\}$, and fault event set $\Sigma_f = \{\sigma_f\}$. In order to compute the verifiers of each module, it is necessary to compute automaton G_F, which is presented in Figure 4.2.*

In Figure 4.3 automata $G_{N_1}, G_{N_2}, G_{N_3}$ and G_{N_4}, that represent the fault-free behavior of each subsystem are depicted. The next step consists in computing automata $G_{N_k}^R$, the automata of the fault-free behavior of the system modules after applying the renaming function, i.e., $G_{N_1}^R$, $G_{N_2}^R$, $G_{N_3}^R$, and $G_{N_4}^R$, presented in Figure 4.4.

(a) G_1. (b) G_2.

(c) G_3.



(d) G_4.

Figure 4.1: Automata that models the system components G_1, G_2, G_3 and G_4 from Example 4.1.

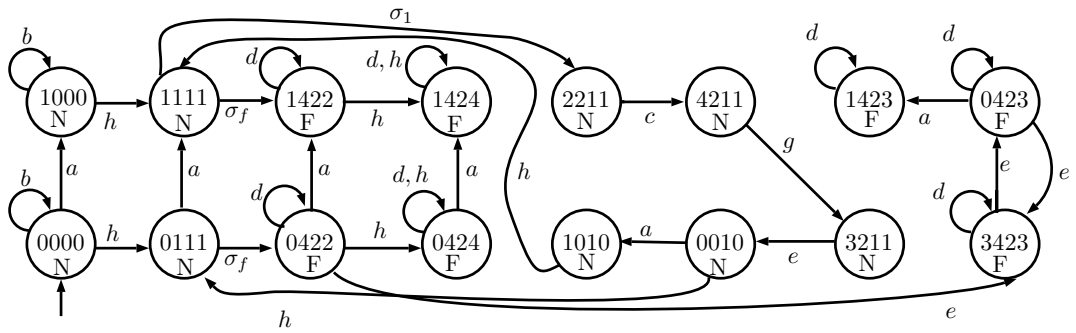
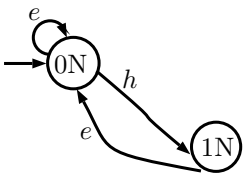


Figure 4.2: Automaton that models the faulty language of the system G_F from Example 4.1.

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

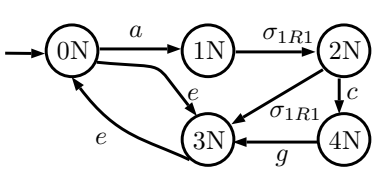
(c) G_{N_3}.



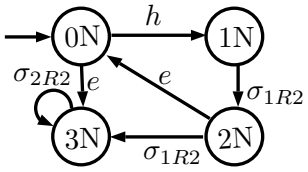
(d) G_{N_4}.

Figure 4.3: Automata that models the fault-free modules language of the modules of the system from Example 4.1.

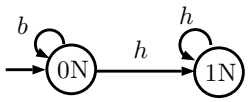
Computing the verifier of module 3 *it is possible to notice in Figure 4.5 that the path* $\{(0000N;0)$, h, $(0111N;1)$, σ_f, $(0422F;1)$, e, $(3423F;1)$, e, $(0423F;1)$, e, $(3423F;1)\}$ *is associated with a sequence that violates synchronous diagnosability*



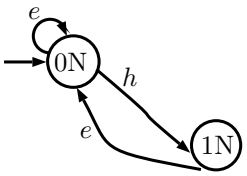
(a) $G^R_{N_1}$.



(b) $G^R_{N_2}$.



(c) $G^R_{N_3}$.



(d) $G^R_{N_4}$.

Figure 4.4: Automata that represents the fault-free modules with unobservable events renamed from Example 4.1.

with respect to $L_{N_3}, P^o_{3,o} : \Sigma^*_o \to \Sigma^*_{3,o}, P_o : \Sigma^* \to \Sigma^*_o$, *and* Σ_f.

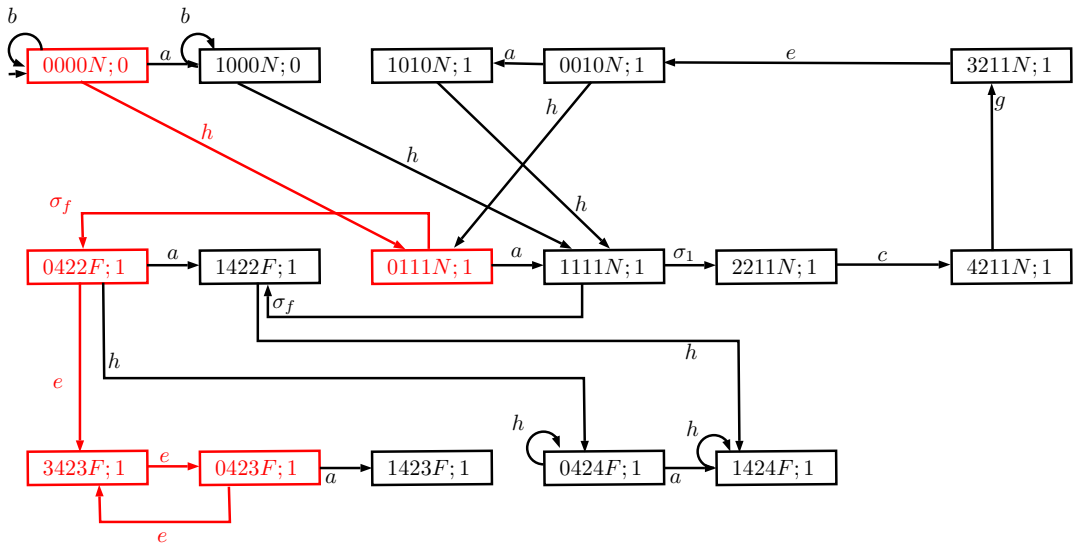


Figure 4.5: Verifier of module 3 from Example 4.1, highlighting a sequence that violates the synchronous diagnosability.

In Figure 4.6 it is possible to notice that the sequence that violates the synchronous diagnosability in module 3 *exists in the verifier computed for module* 4, *consequently, the language is not synchronously diagnosable with respect to* $L_{N_4}, P^o_{4,o} : \Sigma^*_o \to \Sigma^*_{4,o}, P_o : \Sigma^* \to \Sigma^*_o$, *and* Σ_f.

Finally, as stated in Theorem 4.2, it is possible to notice the sequence that violates the syncrhonous diagnosability in modules 3 *and* 4 *in the verifier computed with the subset of modules* $\{3, 4\}$. *Thus, the language is not synchronously diagnosable with respect to* $L_{N_3}, P^o_{3,o} : \Sigma^*_o \to \Sigma^*_{3,o}, L_{N_4}, P^o_{4,o} : \Sigma^*_o \to \Sigma^*_{4,o}, P_o : \Sigma^* \to \Sigma^*_o$, *and* Σ_f.

Example 4.1 illustrates the cases where the computation of the verifier can be

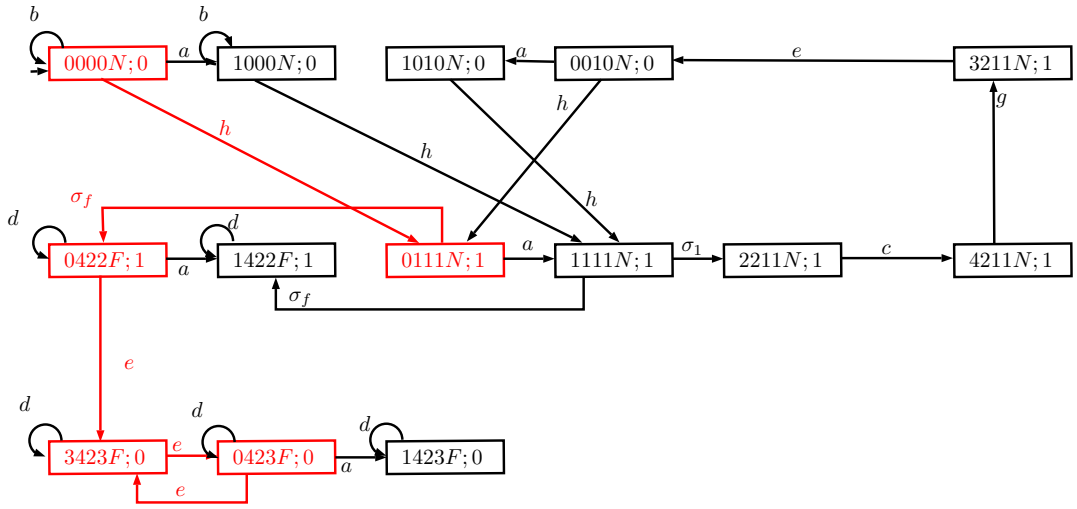


Figure 4.6: Verifier of module 4 from Example 4.1, highlighting a sequence that violates the synchronous diagnosability.

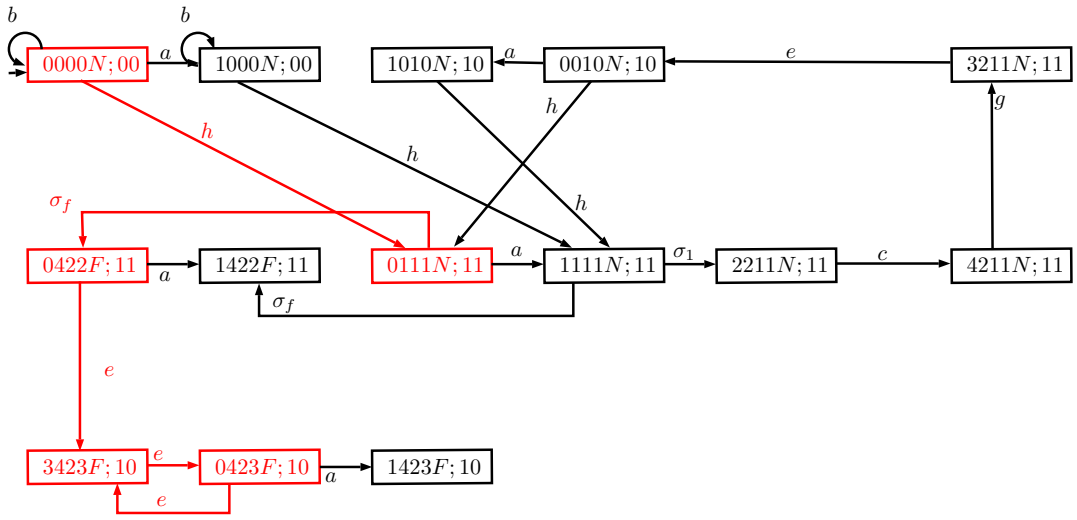


Figure 4.7: Verifier of modules $\{3, 4\}$ from Example 4.1, highlighting a sequence that violates the synchronous diagnosability.

avoided. On the other hand, in cases where the sequence that violates the synchronous diagnosability $G^B_{V_p}$ is not possible in the parallel composition with the verifier to be added ,$G^B_{V_p} \| G^j_V$, it is necessary to compute the new verifier. The new verifier is computed as the parallel composition $G^B_V \| G^j_V$. The necessity of computing this verifier lies in the fact that at least one sequence that violates the synchronous diagnosability in the subset B is eliminated adding the module j. Thus, with the new verifier is possible to evaluate if the other sequences are eliminated and consequently if the language is synchronously diagnosable with respect to L_{N_K}, $k \in B \cup j$, P_o and Σ_f.

In the following, in Example 4.2, we present the two cases, one that the sequence that violates the synchronous diagnosability is eliminated and the language

is synchronously diagnosable with respect to the new subset. And another that the sequence is eliminated but the language is not synchronously diagnosable with respect to the new subset.

Example 4.2 *Consider the same system G of Example 4.1, G_1, G_2, G_3 and G_4 are shown in Figure 4.1 and automaton G_F is presented in Figure 4.2. Automata $G_{N_1}, G_{N_2}, G_{N_3}$ and G_{N_4}, that represent the fault-free behavior of each subsystem are depicted in Figure 4.3, and automata $G^R_{N_1}$, $G^R_{N_2}$, $G^R_{N_3}$, and $G^R_{N_4}$ are presented in Figure 4.4.*

For this example, computing the verifier of module 3 *it is possible to notice in Figure 4.8 another path* $\{(0000N;0)$, h, $(0111N;1)$, σ_f, $(0422F;1)$, h, $(0424F;1)$, h, $(0424F;1)\}$ *that is associated with a sequence that violates synchronous diagnosability with respect to* $L_{N_3}, P^o_{3,o} : \Sigma^*_o \to \Sigma^*_{3,o}, P_o : \Sigma^* \to \Sigma^*_o$, *and* Σ_f.

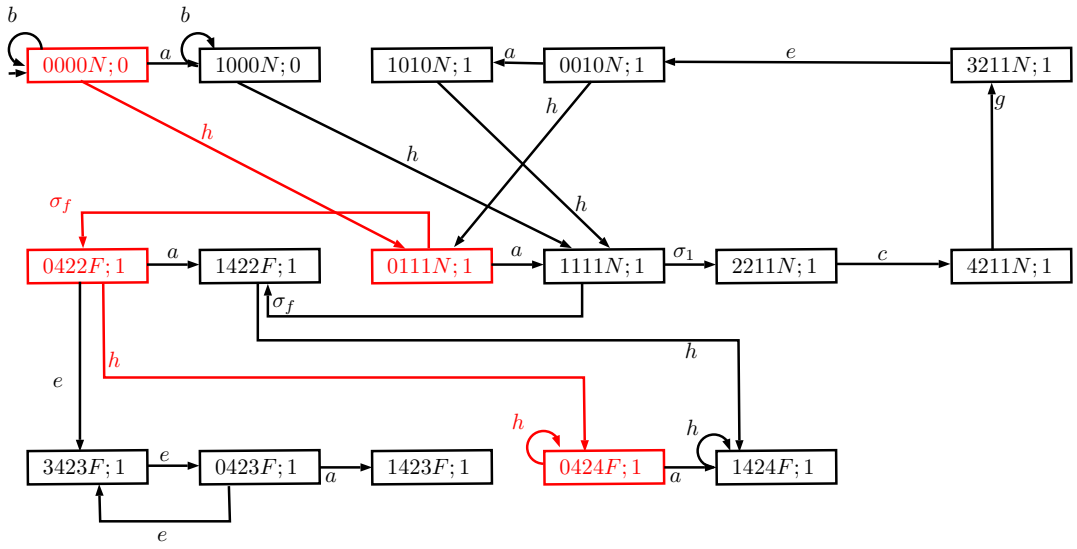


Figure 4.8: Verifier of module 3 from Example 4.2, highlighting a sequence that violates the synchronous diagnosability.

In Figure 4.9 it is possible to notice that the sequence that violates the synchronous diagnosability in module 3 *does not exist in the verifier computed for module* 2, *consequently, it is necessary to compute the verifier considering the subset* $\{2,3\}$, *presented in Figure 4.10.*

In Figure 4.10 it is possible to notice that there is no sequence that violates the synchronous diagnosability in the verifier computed for subset $\{2,3\}$, *consequently, the language is synchronously diagnosable with respect to* $L_{N_2}, P^o_{2,o} : \Sigma^*_o \to \Sigma^*_{2,o}, L_{N_3}, P^o_{3,o} : \Sigma^*_o \to \Sigma^*_{3,o}, P_o : \Sigma^* \to \Sigma^*_o$, *and* Σ_f.

In Figure 4.6 it is possible to notice that the sequence that violates the synchronous diagnosability in module 3 *does not exist in the verifier computed for module* 4, *consequently, it is necessary to compute the verifier considering the subset* $\{3,4\}$, *presented in Figure 4.12.*

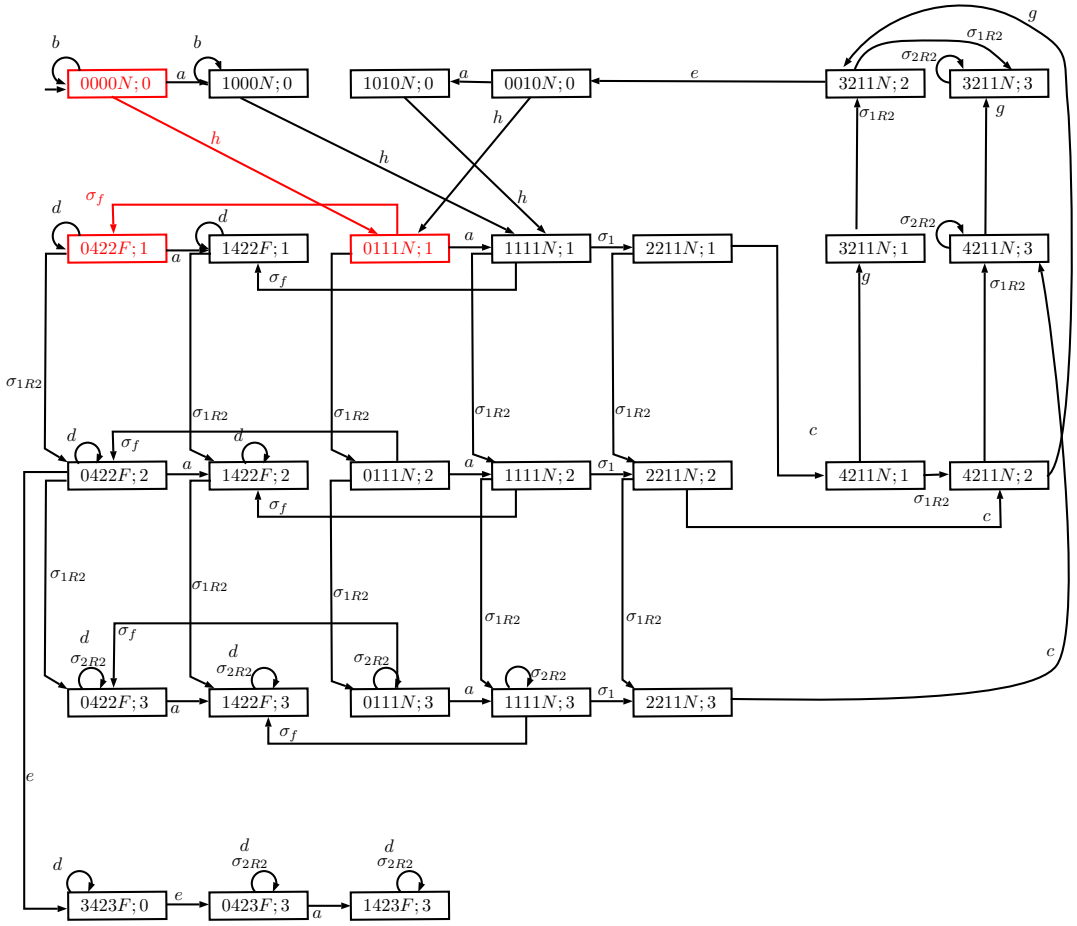


Figure 4.9: Verifier of module 2 from Example 4.2.

It is possible to notice that there exist a sequence that violates the syncrhonous diagnosability in subset $\{3, 4\}$*. Thus, the language is not synchronously diagnosable with respect to* $L_{N_3}, P^o_{3,o} : \Sigma^*_o \to \Sigma^*_{3,o}, L_{N_4}, P^o_{4,o} : \Sigma^*_o \to \Sigma^*_{4,o}, P_o : \Sigma^* \to \Sigma^*_o$*, and* Σ_f*.*

Considering this, in REIS and MOREIRA [32], a method to compute all minimal SDMB for the system language was presented. This method is based on the depth-first search and implemented by Algorithm 4.1. The sequence that the modules are added follows the tree architecture, and, for a system composed of 4 modules, the tree architecture is presented in Figure 4.13.

In Step 1 of Algorithm 4.1, M and T, representing the set of all minimal SDMB and the set of module indices k such that $\{k\}$ is not a minimal SDMB, respectively, are defined as the empty set. In Step 2, for each module k, the verifier $G_V^{\{k\}}$ is computed as the parallel composition of the renamed fault-free behavior model $G^R_{N_k}$ and the fault automaton G_F. Then, the indices of the verifiers that do not have cyclic paths that violate the synchronous diagnosability condition are added to M as minimal SDMB, and those that have this kind of cyclic path are added to N. In Step 3, the set B_{G_V} of verifiers $G_V^{\{k\}}$, such that $k \in T$, is created. This step is important due to the fact that the indices $k \in M$ already form minimal SDMB with

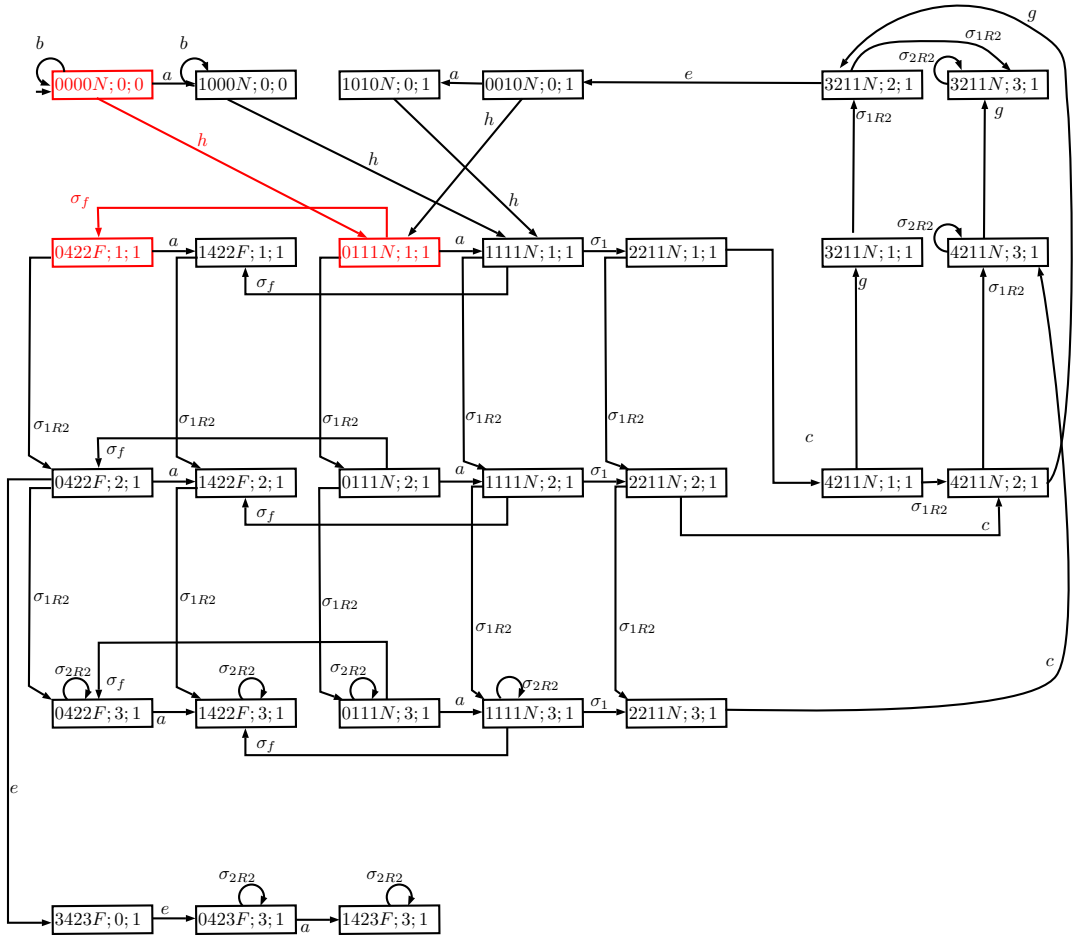


Figure 4.10: Verifier of subset $\{2, 3\}$ from Example 4.2.

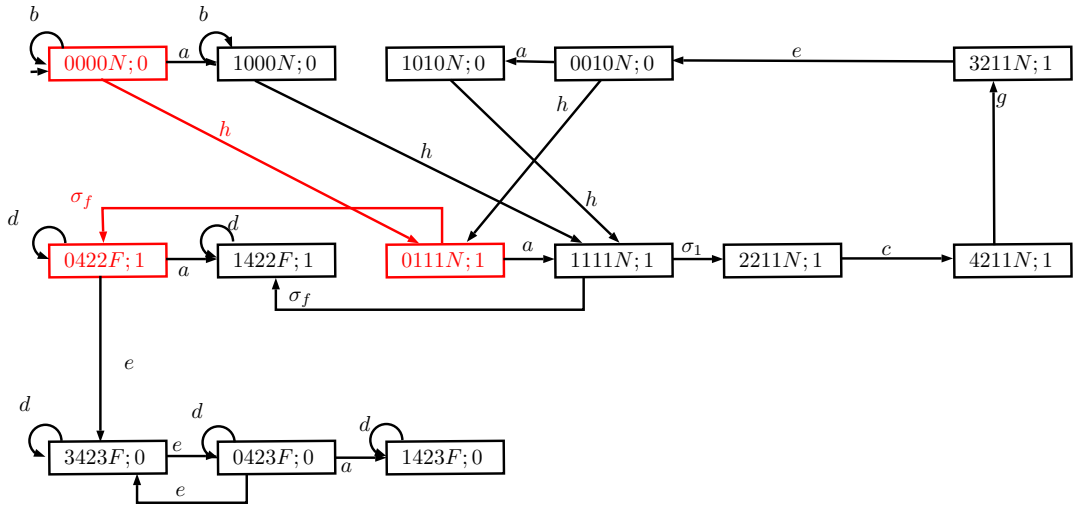


Figure 4.11: Verifier of module 4 from Example 4.1.

cardinality one, and therefore, are not added to other subsets when searching for minimal SDMB with cardinality greater than one. In Step 4, the recursive inclusion of modules using Algorithm 4.2 is performed. Finally, in Step 5, the SDMB that are not minimal are removed from M.

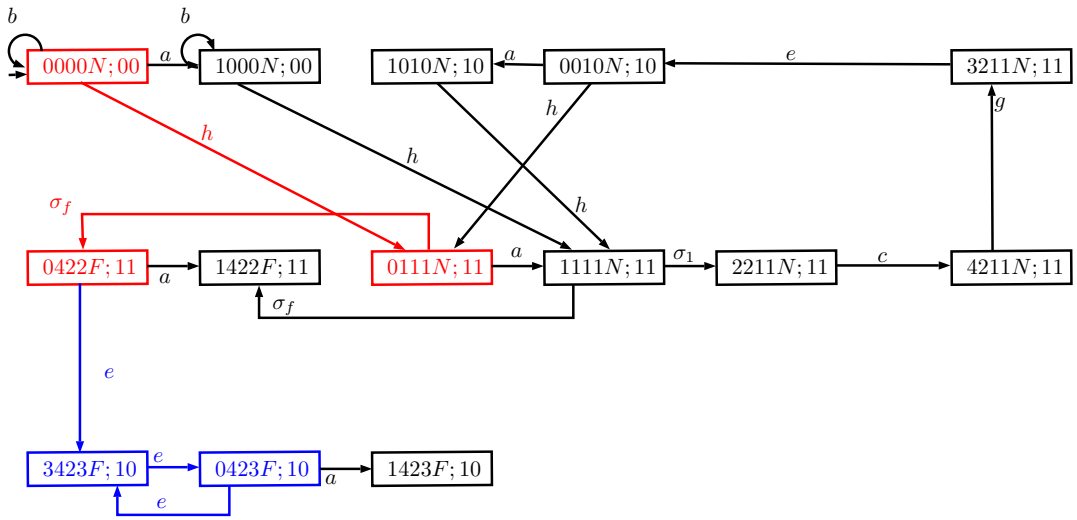


Figure 4.12: Verifier of subset $\{3, 4\}$ from Example 4.2, highlighting a sequence that violates the synchronous diagnosability.

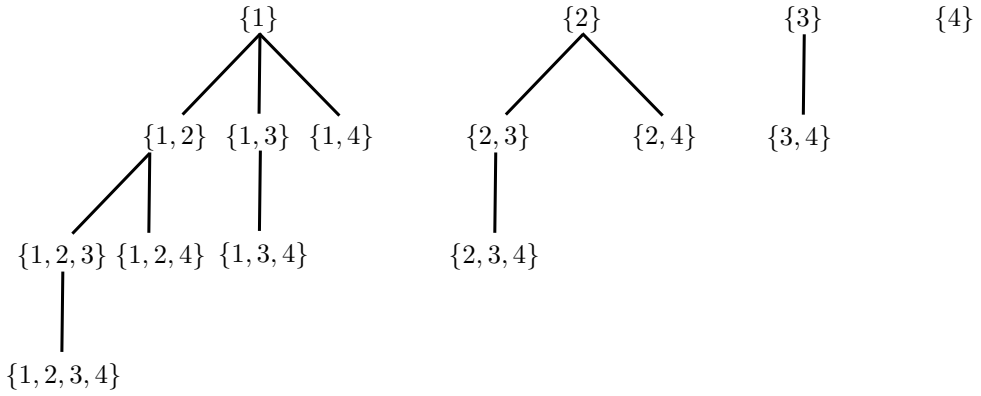


Figure 4.13: Trees architecture that defines the order of adding components.

Algorithm 4.2 is responsible for the recursive module inclusions to form the SDMBs. In Step 1, a path p of verifier G_V^B, associated with a fault sequence that violates the synchronous diagnosability, is obtained, and in Step 2, the subautomaton $G_{V_p}^B$ formed from path p is computed. In Step 3 the loop to add modules is started, and the addition of modules to B is carried out each module at a time, as described in the following steps. Firstly, in Step 3.1, if there does not exist an element of M, E, such that $E \subseteq B \cup \{j\}$, the automaton $G_{V_p}^{j,B}$ is computed as the parallel composition of the partial verifier, $G_{V_p}^B$, and the verifier of the j-th module, that is being checked if it can be added to B, $G_V^{\{j\}}$. If there exists a cyclic path that violates the synchronous diagnosability in $G_{V_p}^{j,B}$, it means that module j is not capable of eliminating the violating cyclic path associated with p, and consequently, it will not be added to B. Otherwise, if no violating cyclic path remains in the test performed in Step 3.1.1, j is added to B in Step 3.1.2 and verifier $G_V^{B\cup\{j\}} = G_V^B \| G_V^{\{j\}}$ is computed. In Step 3.1.2.1, it is verified if $B \cup \{j\}$ forms an SDMB. If $B \cup \{j\}$ forms an SDMB,

then it is added to M, else, a new module is searched to be added to $B \cup \{j\}$ by running function ADD_MODULE_MINIMAL($G_V^{B\cup\{j\}}$,B_{G_V}).

Algorithm 4.1 *Computation of all minimal SDMB*

Input: *$G_{N_k}^R$, for $k \in I_r$, and G_F*
Output: *Set of all minimal SDMB, M.*

1: $M := \emptyset$, $T := \emptyset$

2: For $k \in I_r$

2.1: Compute $G_V^{\{k\}} = G_F \| G_{N_k}^R$

2.2: If L is synchronously diagnosable with respect to P_o, L_{N_k}, and Σ_f, then $M := M \cup \{\{k\}\}$, *else* $T := T \cup \{k\}$

3: $B_{G_V} = \{G_V^{\{k\}} : k \in T\}$

4: For $k \in T$

4.1: ADD_MODULE_MINIMAL($G_V^{\{k\}}$, B_{G_V})

5: Find all elements of M, E, such that there exists another element E', where $E \subset E'$, and eliminate E' from M

Algorithm 4.2 *ADD_MODULE_MINIMAL(G_V^B,B_{G_V})*

1: Find a path p of G_V^B that departs from its initial state with an embedded cyclic path cl that violates the synchronous diagnosability

2: Compute the subautomaton of G_V^B from path p, denoted as $G_{V_p}^B$

3: For $j \in T \setminus B$

3.1: If there does not exist $E \in M$ such that $E \subseteq B \cup \{j\}$ then

3.1.1: Compute $G_{V_p}^{j,B} = G_{V_p}^B \| G_V^{\{j\}}$

3.1.2: If there does not exist a cyclic path in $G_{V_p}^{j,B}$ associated with the cyclic path cl of G_V^B then compute $G_V^{B\cup\{j\}} = G_V^B \| G_V^{\{j\}}$

3.1.2.1: If there does not exist a cyclic path violating the synchronous diagnosability in $G_V^{S\cup\{j\}}$, *then* $M := M \cup \{S \cup \{j\}\}$, *else : ADD_MODULE_MINIMAL($G_V^{S\cup\{j\}}$,B_{G_V})*

Theorem 4.3 *Algorithm 4.1 computes all minimal synchronous diagnosis module basis considering fault-free behavior models of the system modules given by* G_{N_k}, $k \in I_r$, *and fault behavior model given by* G_F.

Proof: Note that in Algorithm 4.1, each element of M is computed recursively by adding to it only the modules that eliminate a cyclic path violating the synchronous diagnosability, until there does not exist any violating cyclic path. Thus, all elements of M are SDMB. In Step 5 of Algorithm 4.1, all elements of M that contains another element of M are removed, and consequently, all redundant SDMB are eliminated from M. In addition, note that Algorithm 4.1 computes all possible subsets of I_r by adding to B, incrementally, each module of I_r that eliminates a cyclic path that violates the synchronous diagnosability. Thus, the elements of M form all minimal SDMB of the system. □

Remark 4.1 *It is important to remark that an algorithm for the computation of all minimal sets of observable events that guarantee the system diagnosability is proposed in [23]. The main difference in comparison with the method proposed in this paper is that we are searching for sets of modules, and not sets of events, that ensure the synchronous diagnosability of the system language.* □

It is important to remark that the minimum SDMB can be obtained by searching in the set of all minimal SDMB, M, those that have the smallest cardinality.

In order to find all minimal SDMB and all minimum SDMB, applying Algorithm 4.1, we implemented a python program which takes as input only the system modules. The program calculates the automata required by the algorithms and returns the minimal and minimum SDMB, in a way that no further knowledge of the system is required. The python program is available in [33]. In the following, we present an example of the implementation of Algorithm 4.1. For the sake of comprehension, some verifiers automata are presented to illustrate Example 4.3, but all the verifiers automata and auxiliary automata computed are presented in Appendix 5.1.

Example 4.3 *Consider the system G presented in Example 4.1, and consider the problem of computing all the minimal and minimum SDMB. In order to do so, in this example we will use Algorithm 4.1.*

The automata that model the system components, their fault-free behavior, and faulty behavior remain the same as in Example 4.1, and are presented in Figures 4.1, 4.3, 4.4, and 4.2, respectively. In Step 1 *of Algorithm 4.1, the sets* M *and* T *are defined as* $\emptyset$. *In Step* 2 *the verifiers* $G_V^{\{1\}}, G_V^{\{2\}}, G_V^{\{3\}}, G_V^{\{4\}}$ *are computed as the parallel*

composition of the automaton that models the faulty behavior and the renamed fault-free automaton, presented in Figures 4.14, 4.15, 4.16 and 4.17, respectively. In this case, no module can be used alone to diagnose the system language, and the cycles with events of Σ are highlighted in each of the verifiers. Thus, $M = \emptyset$, $T = \{1, 2, 3, 4\}$, and $B_{G_V} = \{G_V^{\{1\}}, G_V^{\{2\}}, G_V^{\{3\}}, G_V^{\{4\}}\}$. Then, Algorithm 4.1 starts a recursive search for the subsets with Algorithm 4.2.

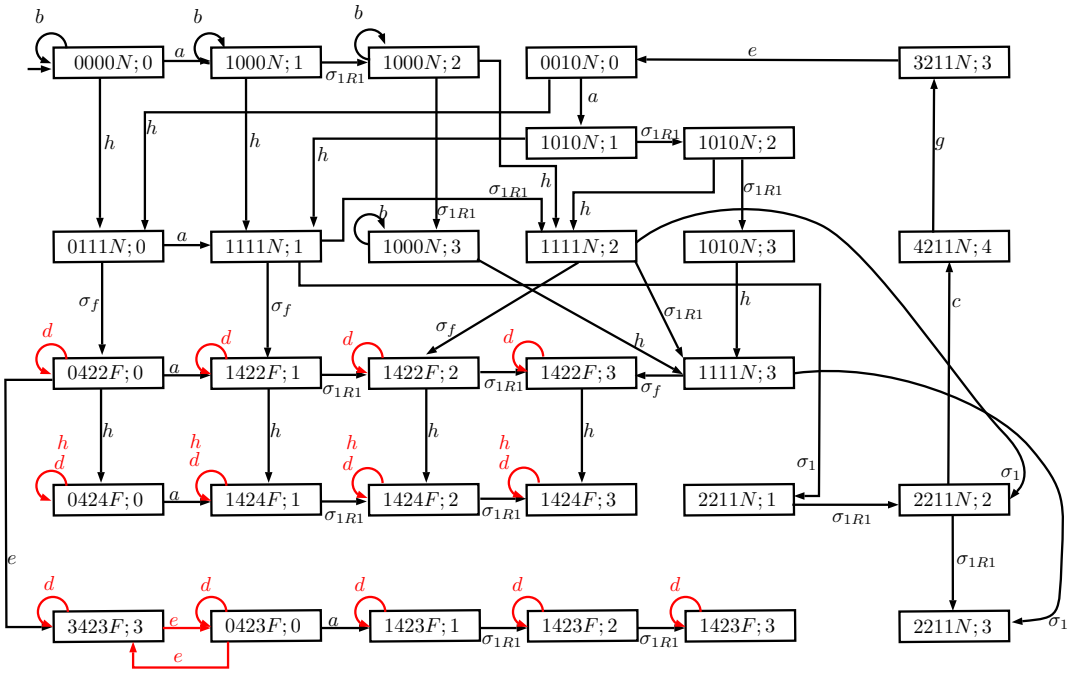


Figure 4.14: Verifier of module 1 of Example 4.3. $G_V^{\{1\}} = G_F \| G_{N_1}^R$.

Starting with the subset $\{1\}$, attempting to add module 2, Algorithm 4.2 selected the path $\{(0000N;0),\ h,\ (0111N;0),\ \sigma_f,\ (0422F;0),\ d,\ (0422F;0)\}$. Automaton $G_{V_p}^{\{1\}}$, whose generated language is the prefix-closure of the sequence associated with the selected path, is presented in Figure 4.18. To verify if the addition of module 2 eliminates the path $\{(0000N;0),\ h,\ (0111N;0),\ \sigma_f,\ (0422F;0),\ d,\ (0422F;0)\}$, the parallel composition of $G_{V_p}^{\{1\}}$ with $G_V^{\{2\}}$ is computed, presented in Figure 4.19. Notice that the sequence that violates the diagnosability leads to a violating cyclic path in $G_{V_p}^{2,\{1\}}$, and, consequently in $G_V^{\{1,2\}}$. Thus, the language of the system is not synchronously diagnosable using only modules 1 and 2.

In the sequel, Algorithm 4.2 attempts to add module 3 to verifier $G_V^{\{1,2\}}$. As $G_V^{\{1,2\}}$ is not computed, it needs to be computed now. The path $\{(0000N;00),\ h,\ (0111N;01),\ \sigma_f,\ (0422F;01),\ d,\ (0422F;01)\}$ is selected. Automaton $G_{V_p}^{\{1,2\}}$ is obtained, and the parallel composition of $G_{V_p}^{\{1,2\}}$ with $G_V^{\{3\}}$, depicted in Figure 4.20, is computed. Notice that the sequence that violates the diagnosability does not lead to a violating cyclic path in $G_{V_p}^{3,\{1,2\}}$. Considering this, the verifier $G_V^{\{1,2,3\}}$ is computed, and it can be verified that there does not exist any cyclic path labeled with F in which

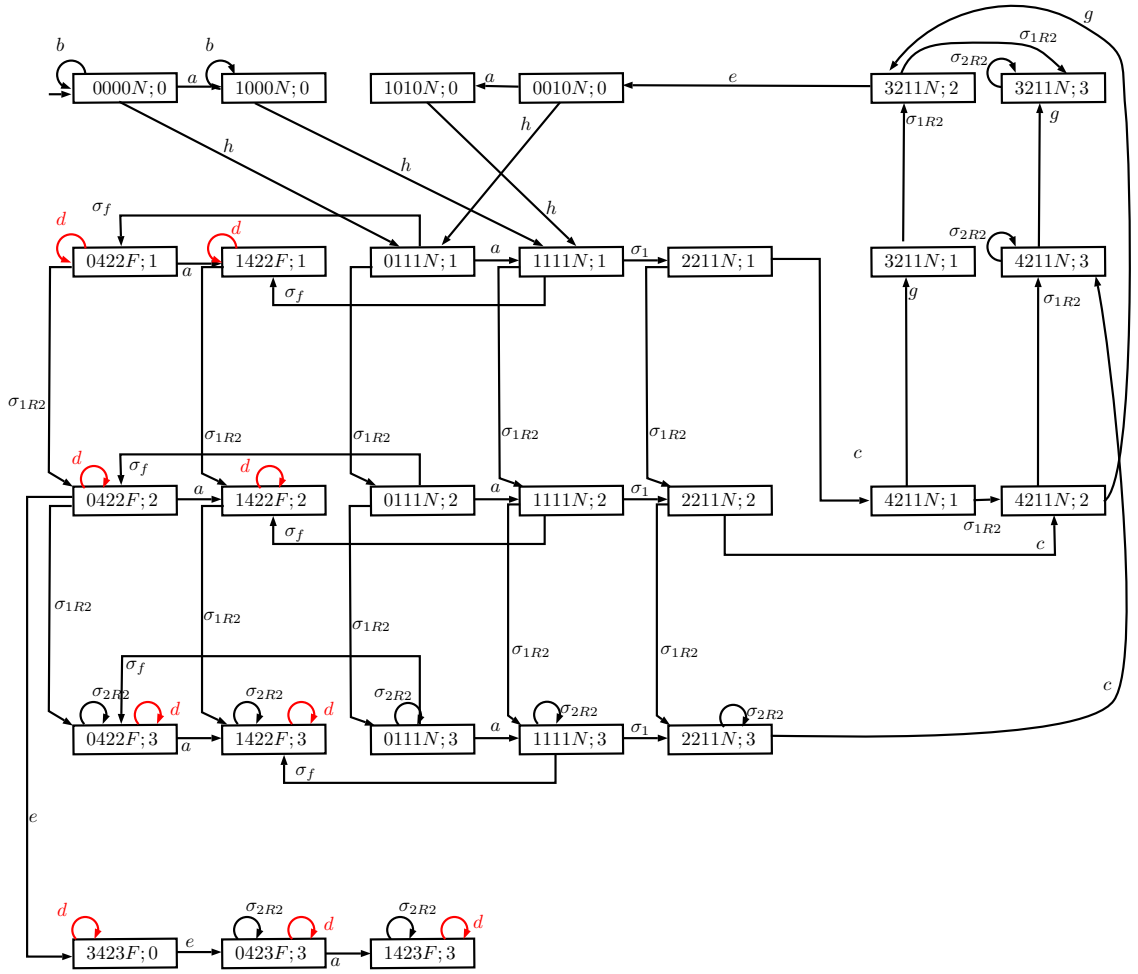


Figure 4.15: Verifier of module 2 of Example 4.3. $G_V^{\{2\}} = G_F \| G_{N_2}^R$.

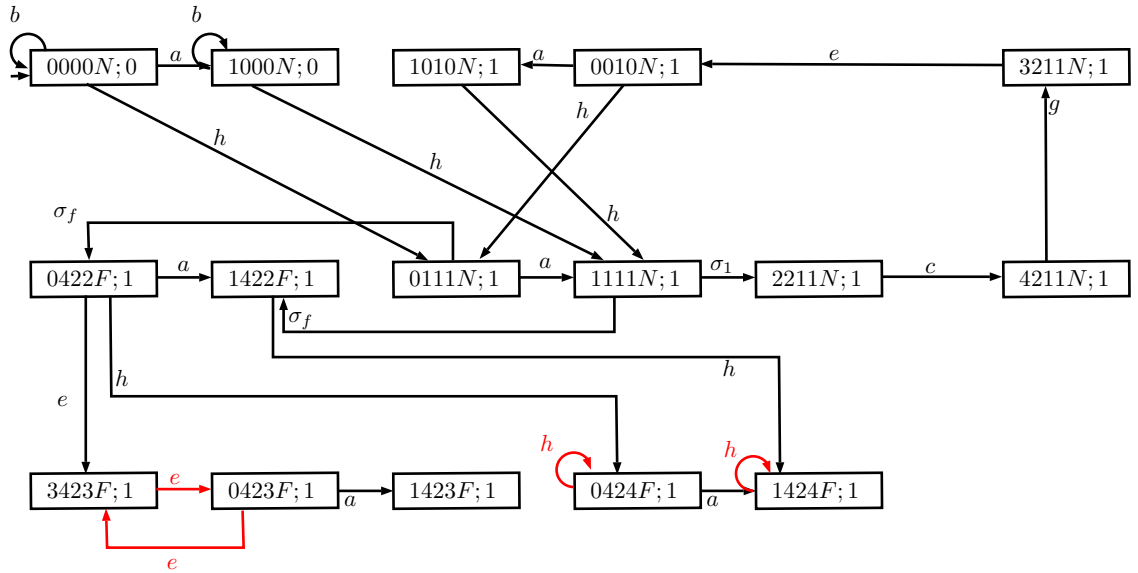


Figure 4.16: Verifier of module 3 of Example 4.3. $G_V^{\{3\}} = G_F \| G_{N_3}^R$.

at least one transition is performed by a non-renamed event. Thus, we conclude that L *is synchronously diagnosable with respect to* $L_{N_1}, L_{N_2}, L_{N_3}, P_{1,o}^o : \Sigma_o^* \to \Sigma_{1,o}^*, P_{2,o}^o :$

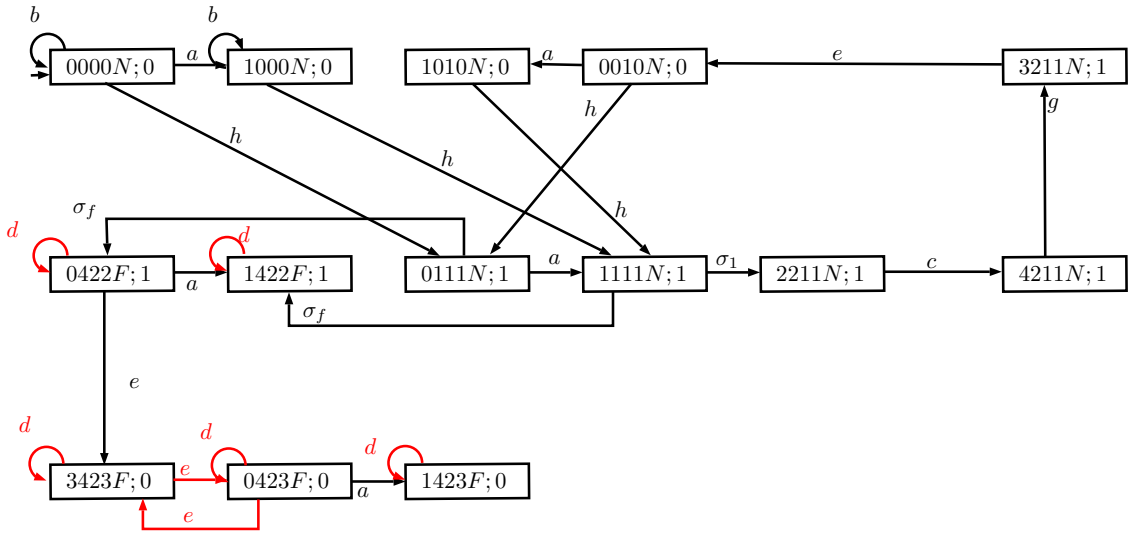


Figure 4.17: Verifier of module 4 of Example 4.3. $G_V^{\{4\}} = G_F \| G_{N_4}^R$.

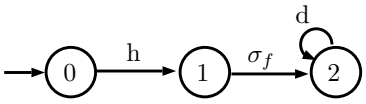


Figure 4.18: Automaton $G_{V_p}^{\{1\}}$, whose generated language is the prefix-closure of the sequence, that violates the diagnosability, associated with the selected path, from Example 4.3.

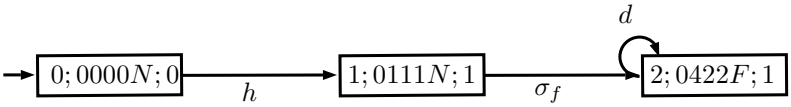


Figure 4.19: Automaton that represents $G_{V_p}^{2,\{1\}} = G_{V_p}^{\{1\}} \| G_V^{\{2\}}$ of Example 4.3.

$\Sigma_o^* \to \Sigma_{2,o}^*, P_{3,o}^o : \Sigma_o^* \to \Sigma_{3,o}^*, P_o : \Sigma^* \to \Sigma_o^*$, *and* Σ_f, *and the subset* $\{1, 2, 3\}$ *is added to* M. *As the method provides only minimal SDMB, the computation of verifier* $G_V^{\{1,2,3,4\}}$ *is avoided, since* $\{1, 2, 3\}$ *is already an SDMB.*

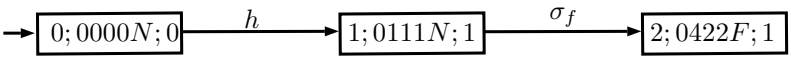


Figure 4.20: Automaton that represents $G_{V_p}^{3,\{1,2\}} = G_{V_p}^{\{1,2\}} \| G_V^{\{3\}}$ of Example 4.3.

In the sequel, Algorithm 4.2 attempts to add module 4 *to verifier* $G_V^{\{1,2\}}$. *The path* $\{(0000N; 00), h, (0111N; 01), \sigma_f, (0422F; 01), d, (0422F; 01)\}$ *is selected. The parallel composition of* $G_{V_p}^{\{1,2\}}$ *with* $G_V^{\{4\}}$ *is computed, resulting in automaton* $G_{V_p}^{4,\{1,2\}}$. *The sequence that violates the diagnosability leads to a violating cyclic path in* $G_{V_p}^{4,\{1,2\}}$, *and, consequently in* $G_V^{\{1,2,4\}}$. *Thus, the language of the system is not synchronously diagnosable using only modules* 1, 2 *and* 4.

In the following, Algorithm 4.2 attempts to add module 3 *to the verifier* $G_V^{\{1\}}$.

Consider again path $\{(0000N;0), h, (0111N;0), \sigma_f, (0422F;0), d, (0422F;0)\}$ *and automaton* $G_{V_p}^{\{1\}}$. *The parallel composition of* $G_{V_p}^{\{1\}}$ *with* $G_V^{\{3\}}$ *is computed, resulting in automaton* $G_{V_p}^{3,\{1\}}$. *In this case, the sequence that violates the diagnosability does not lead to a violating cyclic path in* $G_{V_p}^{3,\{1\}}$. *Then, the verifier* $G_V^{\{1,3\}}$ *is computed, but there exists cyclic paths labeled with* F *in which at least one transition is labeled with a non-renamed event. Thus, the language of the system is not synchronously diagnosable using only modules* 1 *and* 3.

In the sequel, Algorithm 4.2 attempts to add module 4 *to the verifier* $G_V^{\{1,3\}}$. *Consider now path* $\{(0000N;00), h, (0111N;01), \sigma_f, (0422F;01), h, (0424F;01), h, (0424F;01)\}$. *Automaton* $G_{V_p}^{\{1,3\}}$ *is obtained, and the parallel composition of* $G_{V_p}^{\{1,3\}}$ *with* $G_V^{\{4\}}$ *is computed, resulting in automaton* $G_{V_p}^{4,\{1,3\}}$. *In this automaton, the sequence that violates the diagnosability does not lead to a violating cyclic path in* $G_{V_p}^{4,\{1,3\}}$. *Considering this, the verifier* $G_V^{\{1,3,4\}}$ *is computed, but there exists cyclic paths labeled with* F *in which at least one transition is labeled with a non-renamed event. Thus, the language of the system is not synchronously diagnosable using only modules* 1, 3 *and* 4.

Algorithm 4.2, in the sequel, verify the possibility of adding module 4 *to verifier* $G_V^{\{1\}}$. *Consider again path* $\{(0000N;0), h, (0111N;0), \sigma_f, (0422F;0), d, (0422F;0)\}$ *and automaton* $G_{V_p}^{\{1\}}$. *The parallel composition of* $G_{V_p}^{\{1\}}$ *with* $G_V^{\{4\}}$ *is computed, resulting in automaton* $G_{V_p}^{4,\{1\}}$. *The sequence that violates the diagnosability leads to a violating cyclic path in* $G_{V_p}^{4,\{1\}}$, *and, consequently, in* $G_V^{\{1,4\}}$. *Thus, the language of the system is not synchronously diagnosable using only modules* 1 *and* 4.

Starting now with module 2 *and verifier* $G_V^{\{2\}}$, *Algorithm 4.2 attempts to add module* 3. *In order to do so, the path* $\{(0000N;0), h, (0111N;1), \sigma_f, (0422F;1), d, (0422F;1)\}$ *is selected. Automaton* $G_{V_p}^{\{2\}}$ *is computed. The parallel composition of* $G_{V_p}^{\{2\}}$ *with* $G_V^{\{3\}}$ *is obtained, resulting in automaton* $G_{V_p}^{3,\{2\}}$. *In this automaton, the sequence that violates the diagnosability does not lead to a violating cyclic path in* $G_{V_p}^{3,\{2\}}$. *Considering this, the verifier* $G_V^{\{2,3\}}$ *is computed, presented in 4.21, and since there does not exist any cyclic path labeled with* F *in which at least one transition is performed by a non-renamed event, we conclude that* L *is synchronously diagnosable with respect to* $L_{N_2}, L_{N_3}, P_{2,o}^o : \Sigma_o^* \to \Sigma_{2,o}^*, P_{3,o}^o : \Sigma_o^* \to \Sigma_{3,o}^*, P_o : \Sigma^* \to \Sigma_o^*$, *and* Σ_f, *and the subset* $\{2,3\}$ *is added to* M. *As the method searches only minimal SDMB, the computation of verifier* $G_V^{\{2,3,4\}}$ *is avoided, since* $\{2,3\}$ *is already an SDMB.*

Then, Algorithm 4.2 verify the possibility of adding module 4 *to verifier* $G_V^{\{2\}}$. *Consider again path* $\{(0000N;0), h, (0111N;1), \sigma_f, (0422F;1), d, (0422F;1)\}$ *and automaton* $G_{V_p}^{\{2\}}$. *The parallel composition of* $G_{V_p}^{\{2\}}$ *with* $G_V^{\{4\}}$ *is computed, resulting in automaton* $G_{V_p}^{4,\{2\}}$. *The sequence that violates the diagnosability leads to a violating cyclic path in* $G_{V_p}^{4,\{2\}}$, *and, consequently, in* $G_V^{\{2,4\}}$. *Thus, the language of the system is not synchronously diagnosable using only modules* 2 *and* 4.

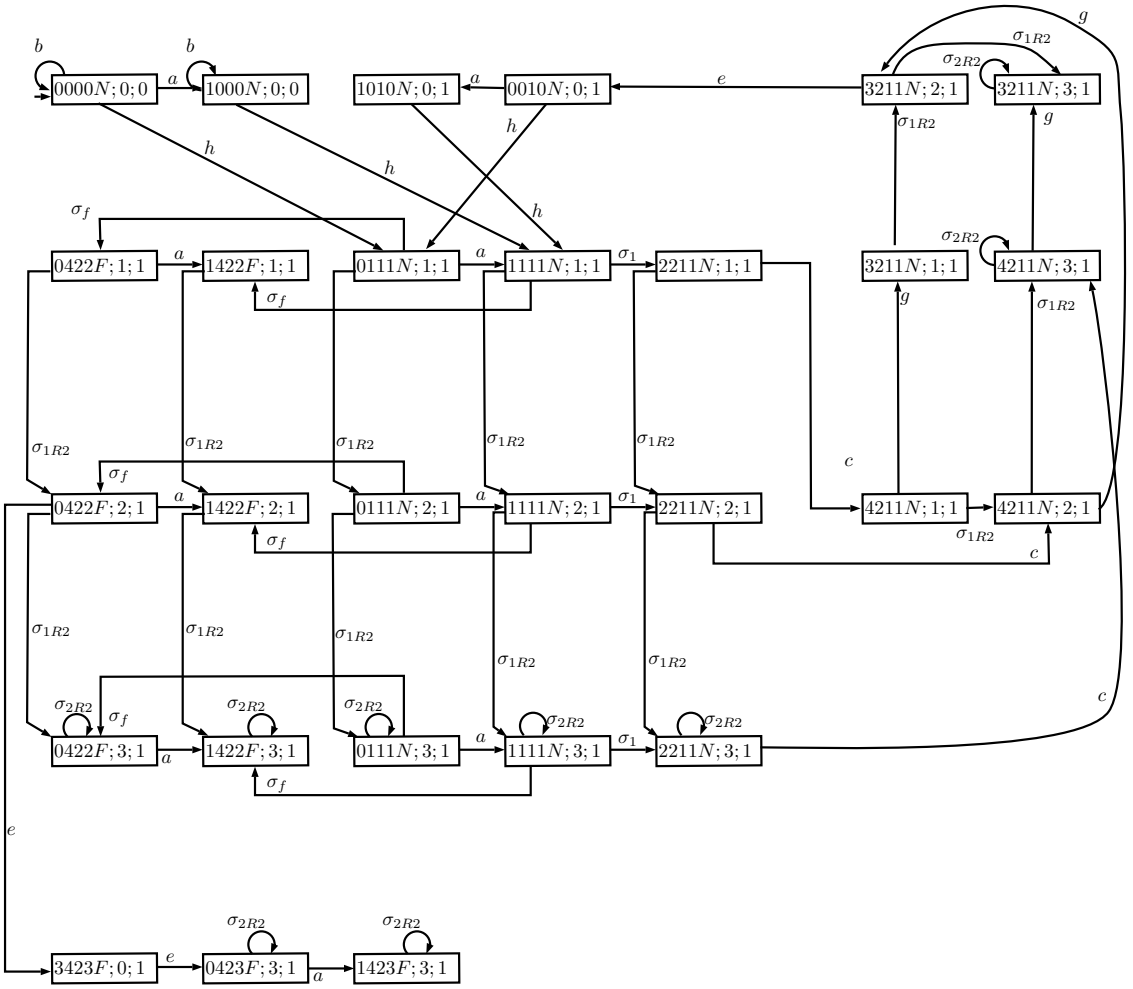


Figure 4.21: Verifier computed with modules $\{2, 3\}$ from Example 4.3 G_V^{23}.

The last subset to be tested is formed by modules $\{3, 4\}$*, adding module* 4 *to the verifier composed by module* 3*,* $G_V^{\{3\}}$*. Consider now the path* $\{(0000N; 0)$*,* h*,* $(0111N; 1)$*,* σ_f*,* $(0422F; 1)$*,* h*,* $(0424F; 1)$*,* h*,* $(0424F; 1)\}$*. Automaton* $G_{V_p}^{\{3\}}$ *is presented in Figure 4.22. The parallel composition of* $G_{V_p}^{\{3\}}$ *with* $G_V^{\{4\}}$ *is computed, resulting in automaton* $G_{V_p}^{4,\{3\}}$*. In this automaton, the sequence that violates the diagnosability does not lead to a violating cyclic path in* $G_{V_p}^{4,\{3\}}$*. Thus, the verifier* $G_V^{\{3,4\}}$ *is computed, depicted in 4.23, but it is possible to notice that there exists a cyclic path labeled with* F *in which at least one transition is labeled with a non-renamed event, in that case,* $\{(3423F; 10)$*,* e*,* $(0423F; 10)$*,* e*,* $(3423F; 10)\}$*. Thus, the language of the system is not synchronously diagnosable using only modules* 3 *and* 4*.*

Thus, the set of candidates of minimal SDMB, $M = \{\{1, 2, 3\}, \{2, 3\}\}$ *is formed in Step* 4*. In Step* 5*, the set* M *is refined leading to* $M = \{\{2, 3\}\}$*.*

Notice that in this case, the SDMB $\{2, 3\}$ *is the only minimal SDMB, and, consequently, it is the minimum SDMB.* □

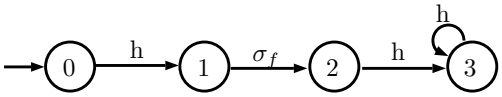


Figure 4.22: Automaton $G_{V_p}^{\{3\}}$, whose generated language is the prefix-closure of the sequence, that violates the diagnosability, associated with the selected path, from Example 4.3.

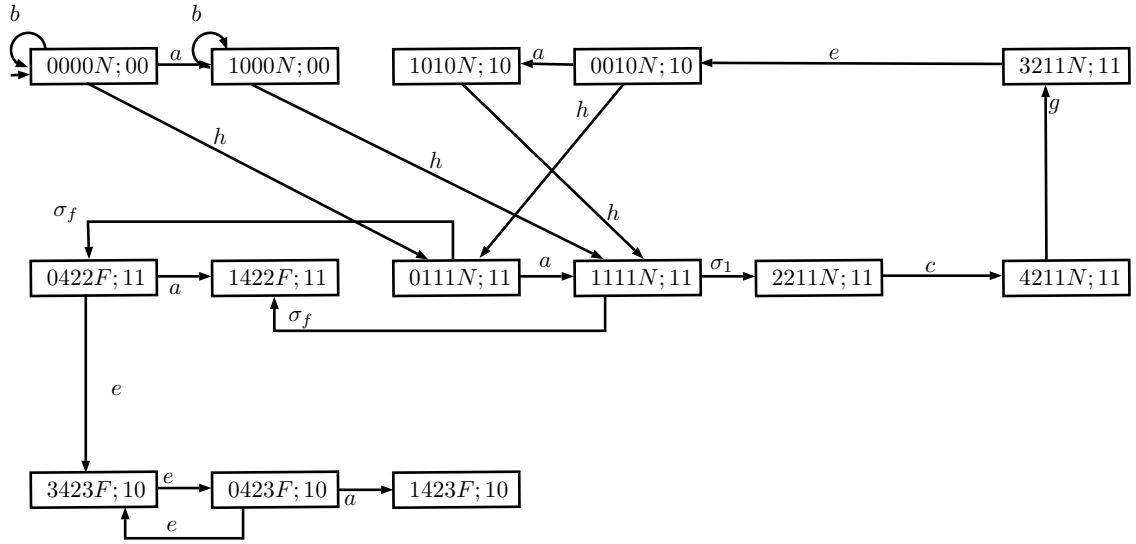


Figure 4.23: Verifier computed with modules $\{3, 4\}$ from Example 4.3 G_V^{34}.

In Example 4.3, we presented a system composed of four modules, which has one minimum SDMB given by $\{2, 3\}$. According to this example, a system may be synchronously diagnosable with a subset of modules. In the following, we present another approach to find all minimal SDMB, an approach based on the breadth-first search, implemented by Algorithm 4.3.

In Step 1 of Algorithm 4.3, M, T and R, representing the set of all minimal SDMB, the set of module indices k such that $\{k\}$ is not a minimal SDMB and the set of module subsets indexes i such that $\{i\}$ is not a minimal SDMB, respectively, are defined as the empty set. In Step 2, for each module k, the verifier $G_V^{\{k\}}$ is computed as the parallel composition of the fault automaton G_F and the renamed fault-free behavior model $G_{N_k}^R$. Then, the indexes of the verifiers that do not have cyclic paths that violate the synchronous diagnosability condition are added to M as minimal SDMB, and those that have this kind of cyclic path are added to T and R. In Step 3, the recursive inclusion of modules using Algorithm 4.4 is performed. Note that in this case, every subset with the same cardinality is testes previously to increment the cardinality, and in a case with 4 modules, the order that the subsets are formed is presented in Figure 4.24. This can be comprehended as the indexes of the subsets that are not SDMB are added to the the set R and the addition of a module is performed by the modules whose indexes are elements of T.

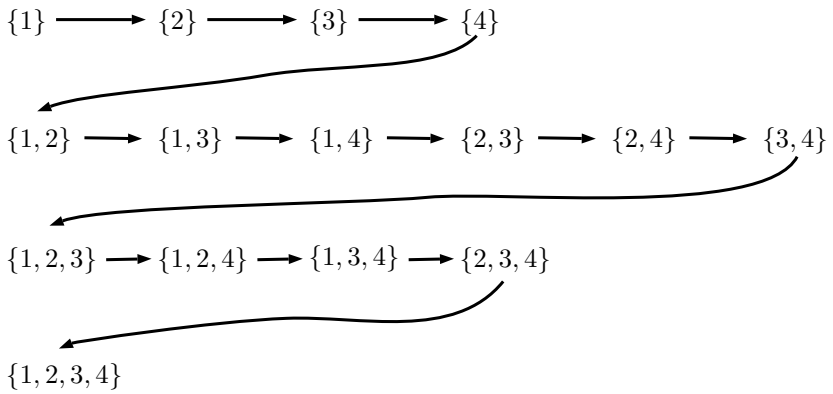


Figure 4.24: Architecture that defines the order of adding components.

Algorithm 4.4 is responsible for the module inclusions to form the SDMBs. In Step 1, a path p of verifier G_V^B, associated with a fault sequence that violates the synchronous diagnosability, is obtained, and in Step 2, the subautomaton $G_{V_p}^B$ formed from path p is computed. In Step 3 if there does not exist an element of M, E, such that $E \subseteq B \cup \{j\}$, the automaton $G_{V_p}^{j,B}$ is computed as the parallel composition of the partial verifier, $G_{V_p}^B$, and the verifier of the subset of modules $\{j\}$, that is being checked if it can be added to B, $G_V^{\{j\}}$. If there exists a cyclic path that violates the synchronous diagnosability in $G_{V_p}^{j,B}$, it means that module j is not capable of eliminating the violating cyclic path associated with p, and consequently, it will not be added to B. Otherwise, if no violating cyclic path remains in the test performed in Step 3.1, j is added to B in Step 3.2 and verifier $G_V^{B\cup\{j\}} = G_V^B \| G_V^{\{j\}}$ is computed. In step 3.2.1, it is verified if $B \cup \{j\}$ forms a SDMB. If $B \cup \{j\}$ forms a SDMB, then it is added to M, else, it is added to R.

Algorithm 4.3 *Computation of all minimal SDMB*

Input: *$G_{N_k}^R$, for $k \in I_r$, and G_F*
Output: *Set of all minimal SDMB, M.*

1: $M := \emptyset$, $T := \emptyset$, $R := \emptyset$

2: For $k \in I_r$

2.1: Compute $G_V^{\{k\}} = G_F \| G_{N_k}^R$

2.2: If L is synchronously diagnosable with respect to P_o, L_{N_k}, and Σ_f, then $M := M \cup \{\{k\}\}$, *else* $T := T \cup \{k\}$, $R := R \cup \{k\}$

3: For $i \in R$, where i is a subset with the indexes

3.1: For $k \in T$, such as index related to k is greater than the greater index in i

3.1.1: ADD_MODULE_MINIMUM($G_V^{\{i\}}$, $\{G_V^{\{k\}}$)

Algorithm 4.4 *ADD_MODULE_MINIMUM(G_V^B,G_V^j)*

1: Find a path p of G_V^B that departs from its initial state with an embedded cyclic path cl that violates the synchronous diagnosability

2: Compute the subautomaton of G_V^B from path p, denoted as $G_{V_p}^B$

3: If there does not exist $E \in M$ such that $E \subseteq B \cup \{j\}$ then

3.1: Compute $G_{V_p}^{j,B} = G_{V_p}^B \| G_V^{\{j\}}$

3.2: If there does not exist a cyclic path in $G_{V_p}^{j,B}$ associated with the cyclic path cl of G_V^B then compute $G_V^{B\cup\{j\}} = G_V^B \| G_V^{\{j\}}$

3.2.1: If there does not exist a cyclic path violating the synchronous diagnosability in $G_V^{B\cup\{j\}}$, then $M := M \cup \{B \cup \{j\}\}$, else $R := R \cup \{B \cup \{j\}\}$

It is important to remark that Algorithm 4.3 searches for the all minimal SDMB and forms set M. Thus, to find all minimum SDMB, it is necessary to search in M the sets with the smallest cardinality. However, if the objective is to find only the minimum SDMB the algorithm can be stop as soon as it finds a SDMB and finishes the search in its cardinality.

In the following, we present Example 4.4 that shows the implementation of Algorithm 4.3 to the same system of Example 4.3. Again, for the sake of comprehension, some verifiers automata are presented to illustrate Example 4.4, but all the verifiers automata computed are presented in Appendix 5.1.

Example 4.4 *Consider, again, the system G presented in Examples 4.1 and 4.3, and consider the problem of computing all the minimal and minimum SDMB using Algorithm 4.3.*

The automata that model the system components, their fault-free behavior, and faulty behavior are presented in Figures 4.1, 4.3, 4.4, and 4.2, respectively. In Step 1 *of Algorithm 4.3, the sets M, T and R are defined as $\emptyset$. In Step* 2 *the verifiers $G_V^{\{1\}}, G_V^{\{2\}}, G_V^{\{3\}}, G_V^{\{4\}}$ are computed as the parallel composition of the automaton that models the faulty behavior and the renamed fault-free automaton, presented in Figures 4.14, 4.15, 4.16 and 4.17, respectively. As noticed in Example 4.3, no module can be used alone to diagnose the system language. Thus, $M = \emptyset$, $T = \{1, 2, 3, 4\}$,*

$R = \{1, 2, 3, 4\}$, *and* $B_{G_V} = \{G_V^{\{1\}}, G_V^{\{2\}}, G_V^{\{3\}}, G_V^{\{4\}}\}$. *Then, Algorithm 4.3 starts a recursive search for the subsets with Algorithm 4.4.*

Differently from Algorithm 4.1, the recursive search tests the verifier with the same cardinality before increasing the cardinality of the verifiers, following Figure 4.24. Considering this, starting with the subset $\{1\}$*, attempting to add module* 2*, Algorithm 4.4 selected the path* $\{(0000N; 0)$, h, $(0111N; 0)$, σ_f, $(0422F; 0)$, d, $(0422F; 0)\}$*. Automaton* $G_{V_p}^{\{1\}}$*, whose generated language is the prefix-closure of the sequence associated with the selected path, is presented in Figure 4.18. To verify if the addition of module* 2 *eliminates the path* $\{(0000N; 0)$, h, $(0111N; 0)$, σ_f, $(0422F; 0)$, d, $(0422F; 0)\}$*, the parallel composition of* $G_{V_p}^{\{1\}}$ *with* $G_V^{\{2\}}$ *is computed, presented in Figure 4.19. As in Example 4.3, the sequence that violates the diagnosability leads to a violating cyclic path in* $G_{V_p}^{2,\{1\}}$*, and, consequently in* $G_V^{\{1,2\}}$*. Thus, the language of the system is not synchronously diagnosable using only modules* 1 *and* 2*, and the element* $\{1, 2\}$ *is added to the set* R*.*

In the following, Algorithm 4.4 attempts to add module 3 *to the verifier* $G_V^{\{1\}}$*. Consider again path* $\{(0000N; 0)$, h, $(0111N; 0)$, σ_f, $(0422F; 0)$, d, $(0422F; 0)\}$ *and automaton* $G_{V_p}^{\{1\}}$*. The parallel composition of* $G_{V_p}^{\{1\}}$ *with* $G_V^{\{3\}}$ *is computed, resulting in automaton* $G_{V_p}^{3,\{1\}}$*. In this case, the sequence that violates the diagnosability does not lead to a violating cyclic path in* $G_{V_p}^{3,\{1\}}$*. Then, the verifier* $G_V^{\{1,3\}}$ *is computed, but there exists cyclic paths labeled with* F *in which at least one transition is labeled with a non-renamed event. Thus, the language of the system is not synchronously diagnosable using only modules* 1 *and* 3*, and the element* $\{1, 3\}$ *is added to the set* R*.*

Algorithm 4.4, in the sequel, verify the possibility of adding module 4 *to verifier* $G_V^{\{1\}}$*. Consider again path* $\{(0000N; 0)$, h, $(0111N; 0)$, σ_f, $(0422F; 0)$, d, $(0422F; 0)\}$ *and automaton* $G_{V_p}^{\{1\}}$*. The parallel composition of* $G_{V_p}^{\{1\}}$ *with* $G_V^{\{4\}}$ *is computed, resulting in automaton* $G_{V_p}^{4,\{1\}}$*. The sequence that violates the diagnosability leads to a violating cyclic path in* $G_{V_p}^{4,\{1\}}$*, and, consequently, in* $G_V^{\{1,4\}}$*. Thus, the language of the system is not synchronously diagnosable using only modules* 1 *and* 4*, and the element* $\{1, 4\}$ *is added to the set* R*.*

In the following, Algorithm 4.4 attempts to add module 3 *to the verifier* $G_V^{\{2\}}$*. In order to do so, the path* $\{(0000N; 0)$, h, $(0111N; 1)$, σ_f, $(0422F; 1)$, d, $(0422F; 1)\}$ *is selected. Automaton* $G_{V_p}^{\{2\}}$ *is computed. The parallel composition of* $G_{V_p}^{\{2\}}$ *with* $G_V^{\{3\}}$ *is obtained, resulting in automaton* $G_{V_p}^{3,\{2\}}$*. In this automaton, the sequence that violates the diagnosability does not lead to a violating cyclic path in* $G_{V_p}^{3,\{2\}}$*. Considering this, the verifier* $G_V^{\{2,3\}}$ *is computed, presented in 4.21, and since there does not exist any cyclic path labeled with* F *in which at least one transition is performed by a non-renamed event, we conclude that* L *is synchronously diagnosable with respect to* $L_{N_2}, L_{N_3}, P_{2,o}^o : \Sigma_o^* \to \Sigma_{2,o}^*, P_{3,o}^o : \Sigma_o^* \to \Sigma_{3,o}^*, P_o : \Sigma^* \to \Sigma_o^*$*, and* Σ_f*,*

and the subset $\{2,3\}$ *is added to* M*. As the method searches only minimal SDMB, the computation of verifiers* $G_V^{\{1,2,3\}}$*,* $G_V^{\{1,2,3,4\}}$ *and* $G_V^{\{2,3,4\}}$ *are avoided, since* $\{2,3\}$ *is already an SDMB.*

Then, Algorithm 4.4 verify the possibility of adding module 4 *to verifier* $G_V^{\{2\}}$*. Consider again path* $\{(0000N;0)$, h, $(0111N;1)$, σ_f, $(0422F;1)$, d, $(0422F;1)\}$ *and automaton* $G_{V_p}^{\{2\}}$*. The parallel composition of* $G_{V_p}^{\{2\}}$ *with* $G_V^{\{4\}}$ *is computed, resulting in automaton* $G_{V_p}^{4,\{2\}}$*. The sequence that violates the diagnosability leads to a violating cyclic path in* $G_{V_p}^{4,\{2\}}$*, and, consequently, in* $G_V^{\{2,4\}}$*. Thus, the language of the system is not synchronously diagnosable using only modules* 2 *and* 4*, and the element* $\{2,4\}$ *is added to the set* R*.*

The last subset with cardinality 2 *to be tested is formed by modules* $\{3,4\}$*, adding module* 4 *to the verifier composed by module* 3*,* $G_V^{\{3\}}$*. Consider now the path* $\{(0000N;0)$, h, $(0111N;1)$, σ_f, $(0422F;1)$, h, $(0424F;1)$, h, $(0424F;1)\}$*. Automaton* $G_{V_p}^{\{3\}}$ *is presented in Figure 4.22. The parallel composition of* $G_{V_p}^{\{3\}}$ *with* $G_V^{\{4\}}$ *is computed, resulting in automaton* $G_{V_p}^{4,\{3\}}$*. In this automaton, the sequence that violates the diagnosability does not lead to a violating cyclic path in* $G_{V_p}^{4,\{3\}}$*. Thus, the verifier* $G_V^{\{3,4\}}$ *is computed, depicted in 4.23, but it is possible to notice that there exists a cyclic path labeled with* F *in which at least one transition is labeled with a non-renamed event, in that case,* $\{(3423F;10)$, e, $(0423F;10)$, e, $(3423F;10)\}$*. Thus, the language of the system is not synchronously diagnosable using only modules* 3 *and* 4*, and the element* $\{3,4\}$ *is added to the set* R*.*

Starting the verifiers of cardinality 3*, the verifier composed of modules* $\{1,2,3\}$ *does not need to be computed due to the fact that* $\{2,3\}$ *is a SDMB. Thus, in the sequel, Algorithm 4.4 attempts to add module* 4 *to verifier* $G_V^{\{1,2\}}$*, and as that verifier was not computed, now it is necessary to compute it. Then, the path* $\{(0000N;00)$, h, $(0111N;01)$, σ_f, $(0422F;01)$, d, $(0422F;01)\}$ *is selected. The parallel composition of* $G_{V_p}^{\{1,2\}}$ *with* $G_V^{\{4\}}$ *is computed, resulting in automaton* $G_{V_p}^{4,\{1,2\}}$*. The sequence that violates the diagnosability leads to a violating cyclic path in* $G_{V_p}^{4,\{1,2\}}$*, and, consequently in* $G_V^{\{1,2,4\}}$*. Thus, the language of the system is not synchronously diagnosable using only modules* 1*,* 2 *and* 4*, and the element* $\{1,2,4\}$ *is added to the set* R*.*

In the sequel, Algorithm 4.4 attempts to add module 4 *to the verifier* $G_V^{\{1,3\}}$*. Consider now path* $\{(0000N;00)$, h, $(0111N;01)$, σ_f, $(0422F;01)$, h, $(0424F;01)$, h, $(0424F;01)\}$*. Automaton* $G_{V_p}^{\{1,3\}}$ *is obtained, and the parallel composition of* $G_{V_p}^{\{1,3\}}$ *with* $G_V^{\{4\}}$ *is computed, resulting in automaton* $G_{V_p}^{4,\{1,3\}}$*. In this automaton, the sequence that violates the diagnosability does not lead to a violating cyclic path in* $G_{V_p}^{4,\{1,3\}}$*. Considering this, the verifier* $G_V^{\{1,3,4\}}$ *is computed, but there exists cyclic paths labeled with* F *in which at least one transition is labeled with a non-renamed event. Thus, the language of the system is not synchronously diagnosable using only modules* 1*,* 3 *and* 4*, and the element* $\{1,3,4\}$ *is added to the set* R*.*

The verifiers composed of modules $\{2,3,4\}$ *and* $\{1,2,3,4\}$ *do not need to be computed due to the fact that* $\{2,3\}$ *is a SDMB. Thus, the set of minimal SDMB,* $M = \{\{2,3\}\}$ *is formed in Step* 4. *Notice that, as expected, the SDMB* $\{2,3\}$ *is the only minimal SDMB, and, consequently, it is the minimum SDMB.* □

Example 4.4 presents the implementation of Algorithm 4.3, which is based on the breadth-first search. In order to find all minimal SDMB, the difference between Algorithms 4.1 and 4.3 are not very significant. However, searching for all minimum SDMB with Algorithm 4.3 makes possible to stop the search as soon as the cardinality of the first SDMB is found. In the following, in Example 4.5, we present a system with eight modules that the minimum SDMB is found with three modules.

Example 4.5 *Consider a system* G, *composed of eight modules,* G_1, G_2, G_3, G_4, G_5, G_6, G_7 *and* G_8 *showed in Figure 4.25, where their event sets are* $\Sigma_1 = \{a,c,e,g,\sigma_1\}, \Sigma_2 = \{e,h,\sigma_1,\sigma_2,\sigma_f\}, \Sigma_3 = \{d,h,\sigma_f\}, \Sigma_4 = \{e,h,\sigma_f\}$, $\Sigma_5 = \{b,h,\sigma_f\}, \Sigma_6 = \{a,c,g,\sigma_1\}, \Sigma_7 = \{e,h\}, \Sigma_8 = \{c,e,g,\sigma_1\}$, *with observable event set* $\Sigma_o = \{a,b,c,d,e,h,g\}$, *unobservable event set* $\Sigma_{uo} = \{\sigma_1,\sigma_2,\sigma_f\}$, *and fault event set* $\Sigma_f = \{\sigma_f\}$. *In order to compute the verifiers of each module, it is necessary to compute automaton* G_F, *that is not represented due to the size and complexity of the figure.*

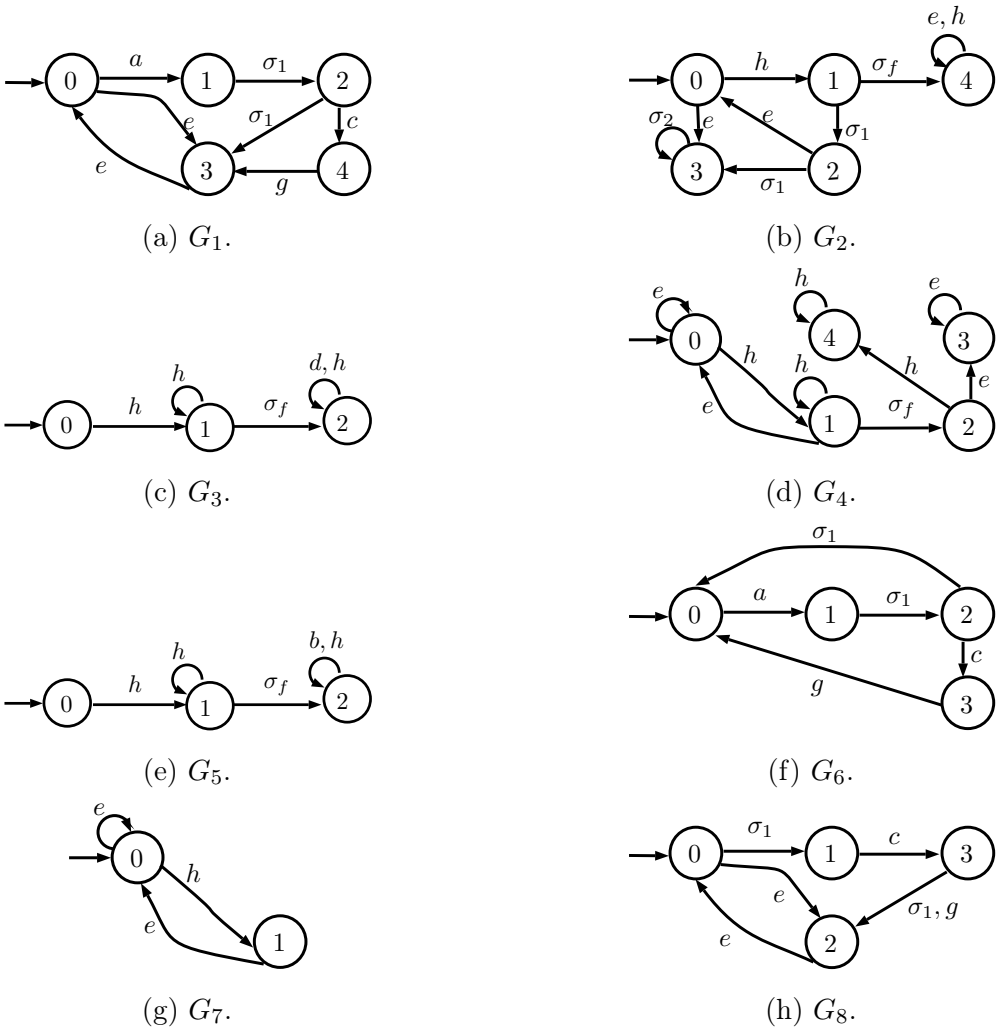
In Figure 4.26 automata $G^R_{N_1}$, $G^R_{N_2}$, $G^R_{N_3}$, $G^R_{N_4}$, $G^R_{N_5}$, $G^R_{N_6}$, $G^R_{N_7}$, *and* $G^R_{N_8}$, *that represent the fault-free behavior of the system modules after applying the renaming function are depicted.*

In the following, Algorithm 4.3 computes the verifier for each module and starts the search for all minimum SDMB. In that case, there is only one minimum SDMB, which is $\{2,3,5\}$

In Examples 4.3 and 4.4 were presented the implementation of Algorithms 4.1, 4.2, 4.3 and 4.4 in order to obtain all the minimal and minimum SDMB. In Example 4.5 Algorithms 4.3 and 4.4 were implemented in order to find only the minimum SDMB. The results of the implementation of the proposed method and its discussions are presented in next Section.

4.2 Results and Discussions

In this section we compare the computational cost of finding all minimal SDMB using the exhaustive search method with the method proposed in this work, considering the system worked in Examples 4.1, 4.3 and 4.4, and Algorithms 4.1 and 4.3, since the order that the modules are computed is different.



(a) G_1. (b) G_2.

(c) G_3. (d) G_4.

(e) G_5. (f) G_6.

(g) G_7. (h) G_8.

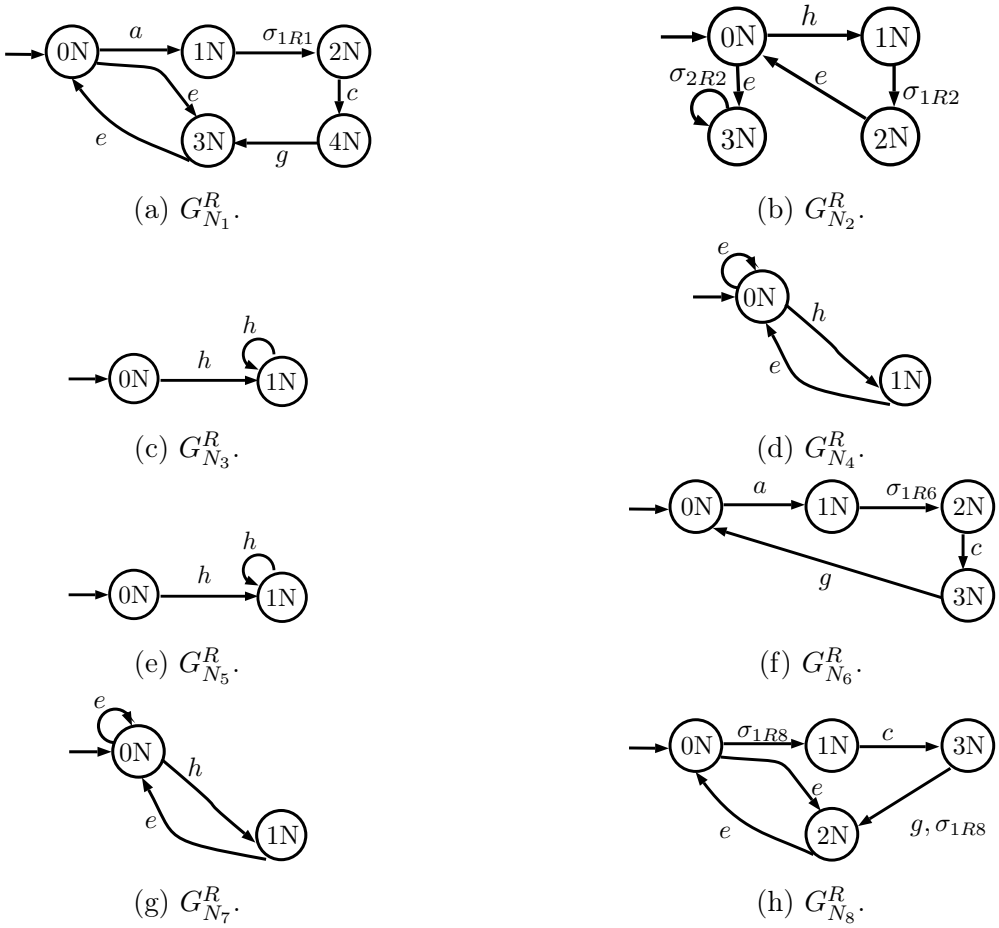
Figure 4.25: Automata that models the system components G_1, G_2, G_3, G_4, G_5, G_6, G_7, and G_8 from Example 4.5

Another comparison we present is the comparison between the computational cost of finding all minimum SDMB using Algorithm 4.3 and the exhaustive search, and the computation cost of finding all mininal SDMB with Algorithms 4.1 and 4.3 and the exhaustive search related to each one. For that comparison, the system worked was the one presented in Example 4.5, a system that is composed of eight modules.

4.2.1 Searching for minimal and minimum SDMB in a system with four modules.

The order to compute the subsets of modules presented in Algorithm 4.1 follows the trees architecture, presented in Figure 4.27. In Examples 4.3 and 4.4, there are four components and the minimal SDMB found was $\{2, 3\}$.

Considering the order to compute subsets of modules in Algorithm 4.1, per-



(a) $G^R_{N_1}$. (b) $G^R_{N_2}$. (c) $G^R_{N_3}$. (d) $G^R_{N_4}$. (e) $G^R_{N_5}$. (f) $G^R_{N_6}$. (g) $G^R_{N_7}$. (h) $G^R_{N_8}$.

Figure 4.26: Automata that represents the fault-free modules with unobservable events renamed from Example 4.5

forming the exhaustive search to find all minimal SDMB, 13 automata must be computed, which results in the computation of 368 states and 794 transitions. The time spent in the process was 1,129.57ms, and the complete information about the verifiers that are computed using the exhaustive search is presented in Table 4.2 and the automata that are computed are presented in table 4.1.

Using the method proposed in this work, following the steps of Algorithm 4.1, instead of computing several parallel compositions with verifiers with a large number of states, partial verifiers $G^B_{V_p}$, presented in Table 4.3 are calculated, as those depicted in Figures 4.18 and 4.22. Since all verifiers G^B_V, such that B is a singleton, have a cyclic path that violates the synchronous diagnosability, then Algorithm 4.2 is recursively repeated until all minimal SDMB are computed. In this procedure, the verifiers presented in Table 4.1 are computed. Note that verifiers $G_V^{\{1,4\}}$, $G_V^{\{2,4\}}$, and $G_V^{\{1,3,4\}}$, that are computed in the exhaustive search method respecting the order proposed in Algorithm 4.1, are not computed using this proposed method.

The total number of automata that are computed using Algorithm 4.1, presented in Tables 4.2, 4.3, and 4.4, was 24, with total number of states equal to 338 and

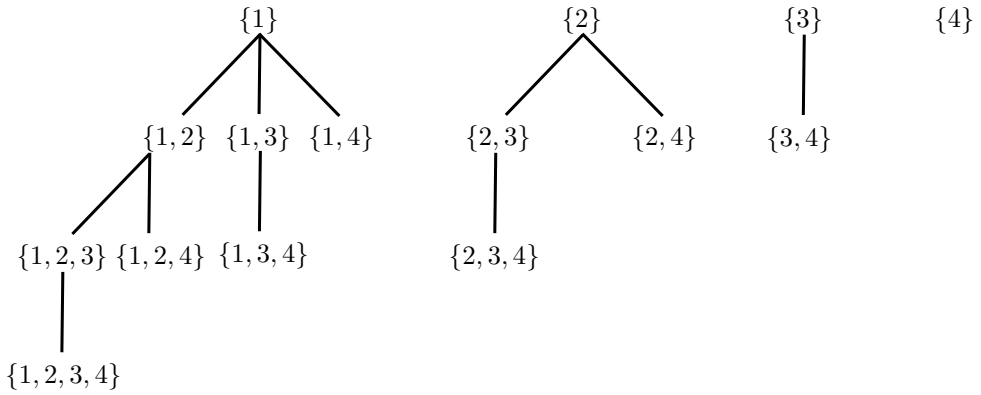


Figure 4.27: Trees architecture that defines the order of adding components.

Table 4.1: Verifiers G_V^B computed using the exhaustive search and the proposed method.

B	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{1,2\}$	$\{1,3\}$	$\{1,4\}$	$\{2,3\}$
Exhaus. tree	X	X	X	X	X	X	X	X
Algorithm 4.1	X	X	X	X	X	X		X
Exhaus. cardin.	X	X	X	X	X	X	X	X
Algorithm 4.3	X	X	X	X	X	X		X
B	$\{2,4\}$	$\{3,4\}$	$\{1,2,3\}$	$\{1,2,4\}$	$\{1,3,4\}$			
Exhaus. tree	X	X	X	X	X			
Algorithm 4.1		X	X		X			
Exhaus. cardin.	X	X		X	X			
Algorithm 4.3		X			X			

with total number of transitions 654. The time spent in the process of obtaining the minimal SDMB, $\{2,3\}$, using Algorithm 4.1, was 804.57ms. This shows a reduction of 29% in time, 17% in the number of states and 30% in the number of transitions, as shown in Table 4.5, in comparison with the exhaustive search method respecting the order proposed in Algorithm 4.1.

On the other hand, the order to compute the subsets of modules presented in Algorithm 4.3 respects the cardinality of the subsets and is presented in Figure 4.28. Considering that order, performing the exhaustive search to find all minimal SDMB, 12 automata must be computed, resulting in the computation of 344 states and 738 transitions. The time spent in the process was 845.26ms, and the complete information about the verifiers that are computed using the exhaustive search considering this order to compute the subsets of modules is presented in Table 4.2 and the automata that are computed are presented in table 4.1. Notice that subset $\{1,2,3\}$ was not computed due to the fact that the subset $\{2,3\}$ is a SDMB and was computed previously.

Using the method proposed in this work, following the steps of Algorithm 4.3, again, partial verifiers $G_{V_p}^B$, presented in Table 4.3 are calculated. Since all verifiers

Table 4.2: Number of states and transitions of the verifiers G_V^B from Example 4.3.

B	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{1,2\}$	$\{1,3\}$	$\{1,4\}$	$\{2,3\}$
States	30	28	16	14	52	30	26	28
Transitions	66	65	25	25	135	53	51	56
B	$\{2,4\}$	$\{3,4\}$	$\{1,2,3\}$	$\{1,2,4\}$	$\{1,3,4\}$			
States	28	14	52	52	26			
Transitions	65	20	118	135	42			

Table 4.3: Number of states and transitions of the partial verifiers $G_{V_p}^B$ that are computed in the Example 4.3 using Algorithms 4.1 and 4.2.

$G_{V_p}^B$	$G_{V_p}^{\{1\}}$	$G_{V_p}^{\{2\}}$	$G_{V_p}^{\{3\}}$	$G_{V_p}^{\{1,2\}}$	$G_{V_p}^{\{1,3\}}$
States	3	3	4	3	4
Transitions	3	3	4	3	4

G_V^B, such that B is a singleton, have a cyclic path that violates the synchronous diagnosability, then Algorithm 4.4 is recursively repeated until all minimal SDMB are computed. In this procedure, the verifiers presented in Table 4.1 are computed. Note that verifiers $G_V^{\{1,4\}}$, $G_V^{\{2,4\}}$, and $G_V^{\{1,2,4\}}$, that are computed in the exhaustive search method respecting the order proposed in Algorithm 4.3, are not computed using this proposed method. Note that comparing with Algorithm 4.1, the subset $\{1,2,3\}$ was not computed due to the fact that the subset $\{2,3\}$ is a SDMB and was computed previously.

The total number of automata that are computed using Algorithm 4.3, presented in Tables 4.2, 4.3, and 4.4, was 22, with total number of states equal to 283 and with total number of transitions 534. The time spent in the process of obtaining the minimal SDMB, $\{2,3\}$, using Algorithm 4.3, was 507.86ms. This shows a reduction of 40% in time, 17% in the number of states and 28% in the number of transitions, as shown in Table 4.6, in comparison with the exhaustive search method respecting the order proposed in Algorithm 4.3.

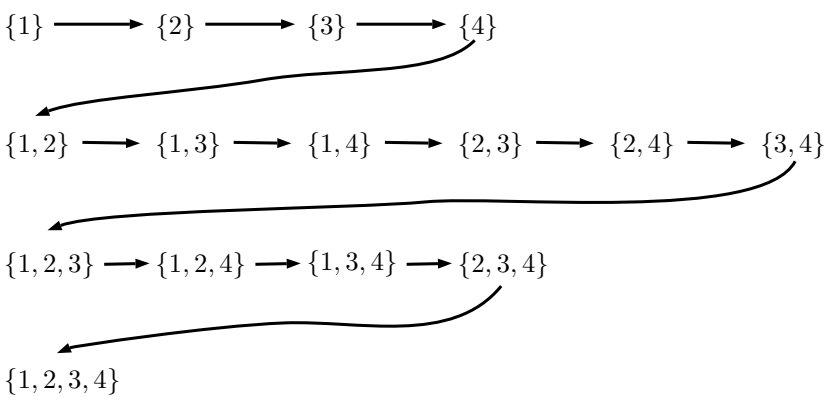


Figure 4.28: Architecture that defines the order of adding components.

Table 4.4: Number of states and transitions of the testing automata $G_{V_p}^{j,B}$ computed in Example 4.3 using Algorithms 4.1 and 4.2.

$G_{V_p}^{j,B}$	$G_{V_p}^{2,\{1\}}$	$G_{V_p}^{3,\{1\}}$	$G_{V_p}^{4,\{1\}}$	$G_{V_p}^{3,\{2\}}$	$G_{V_p}^{4,\{2\}}$	$G_{V_p}^{4,\{3\}}$	$G_{V_p}^{3,\{1,2\}}$	$G_{V_p}^{4,\{1,2\}}$	$G_{V_p}^{4,\{1,3\}}$
States	7	3	3	3	3	3	3	3	3
Trans.	13	2	3	2	3	2	2	3	2

Table 4.5: Total number of states and transitions that are computed using the exhaustive search and the proposed method with Algorithm 4.1, and reduction in the computational cost and execution time in Example 4.3.

	States	Transitions	Execution Time (ms)
Exhaustive tree	368	794	1,129.57ms
Algorithm 4.1	318	613	804.57ms
Reduction	17%	30%	29%

After computing all minimal SDMB, it is possible to obtain all minimum SDMB, and in the system worked in Examples 4.3 and 4.4, it is equal to $\{2, 3\}$. Even with the increase in the number of automata calculated, the number of states and transitions were reduced, and the time spent had a relevant reduction due to the fact that the auxiliary verifiers are simple automata, and the test to verify if a component should be added to the verifier is performed with this simpler automata.

It is important to remark that with Algorithm 4.3 it is possible to perform a direct search for the minimum SDMB, finishing the search in the cardinality that the first SDMB is found. In Example 4.4, Algorithm 4.3 stops in cardinality 2, computing only 7 verifiers. The total automata computed was 16, with total number of states equal to 192 and with total number of transitions 345. The time spent in the process of obtaining the minimum SDMB, $\{2, 3\}$, using Algorithm 4.3, was 171ms. The economy in time lies in the fact that the verifiers with greater cardinality demands more time to be computed.

4.2.2 Searching for minimal and minimum SDMB in a system with eight modules.

Considering more complex systems, with a greater number of components the expected reduction in number of event transitions, states and execution time is even greater. In order to illustrate it, Example 4.5 was presented. In that Example, a system composed of eight modules is synchronously diagnosable with three modules, $\{2, 3, 5\}$. In order to compute the minimal SDMB for this system, Algorithms 4.1 and 4.3 were implemented in a python program [33]. The summary of the implementation of the search for the minimal SDMB is presented in Table 4.7. It is possible to notice that the number of automata using Algorithms 4.1 and 4.3 is greater than the

Table 4.6: Total number of states and transitions that are computed using the exhaustive search and the proposed method with Algorithm 4.3, and reduction in the computational cost and execution time in Example 4.4.

	States	Transitions	Execution Time (ms)
Exhaustive cardinality	344	738	845.29ms
Algorithm 4.3	283	534	507.86ms
Reduction	17%	28%	40%

respective exhaustive search. On the other hand, the number of states, transitions and execution time are smaller.

Table 4.7: Total number of states and transitions that are computed using the exhaustive search and the proposed method with Algorithms 4.1 and 4.3, and reduction in the computational cost and execution time in Example 4.5

	Automata	States	Transitions	Exec. Time (ms)	Economy
Exhaus. tree	227	8428	19834	72549.00ms	-
Algorithm 4.1	462	5210	9266	14036.00ms	80%
Exhaus. cardinality	224	8347	19697	78854.00ms	-
Algorithm 4.3	462	5120	9123	14296.00ms	82%

The advantage of Algorithm 4.3 is that it is possible to search directly for all the minimum SDMB. In that case, we performed an exhaustive search stopping in cardinality 3 in order to compare the results with Algorithm 4.3. The summary of that comparison is presented in Table 4.8.

Table 4.8: Total number of states and transitions that are computed using the exhaustive search and the proposed method with Algorithm 4.3, and reduction in the computational cost and execution time in Example 4.5 searching directly for the minimum SDMB.

	States	Transitions	Execution Time (ms)
Exhaustive cardinality	2376	5177	9274.00ms
Algorithm 4.3	1187	1946	1128.00ms
Reduction	50%	62%	88%

It is possible to notice that Algorithm 4.3 shows a significant reduction in execution time in comparison with the exhaustive search for all minimum SDMB. It is important to remark that the search for all minimum SDMB with Algorithm 4.3 is faster than the search for all minimal SDMB with the proposed method in this work with Algorithms 4.1 or 4.3. The advantage is that it is possible to find all minimum SDMB with a lower computational cost. On the other hand, the disadvantage is that other minimal SDMB are not computed.

4.3 Final Remarks

In this chapter, we present in Example 4.3, a system that can be synchronously diagnosable using a subset of modules to compute a verifier. After that, in order to show that the proposed method can be more effective with more complex systems, we present a Example 4.5, with eight modules, that is synchronously diagnosable with three module. Finally, we present the results considering the computational costs and evaluate its reduction.

Chapter 5

Conclusions

In this work, we defined the synchronous diagnosis modular basis and proposed a method to discover all the minimal and consequently all minimum SDMB. This method is based on an algorithm and a test. Two algorithms were proposed, one based on the depth-first search, Algorithm 4.1, and another based on the breadth-first search, Algorithm 4.3. Both algorithms can be used to find all minimal SDMB, and then find all minimum SDMB. However, with Algorithm 4.3, it is possible to search directly for all minimum SDMB, since the algorithms searches by cardinality, and as soon as a SDMB is found, the algorithm stops the search at the end of that cardinality.

The algorithms define the order the modules will be tested to compose the verifier. The test consists of building an automaton $G^B_{V_p}$, whose generated language is the prefix-closure of the sequence associated with the path that violate the synchronous diagnosability of subset $\{B\}$, and compute the parallel composition with the verifier of the module we want to add $G^{\{j\}}_V$. Thus, it is possible to verify if the sequence that violates the synchronous diagnosability does lead to a violating cyclic path in this parallel composition, $G^{j,B}_{V_p} = G^B_{V_p} \| G^{\{j\}}_V$. This test avoids the computation of verifiers, reducing the computational cost.

The computational cost of the proposed algorithms is compared with the exhaustive search method respecting the order the modules are added, and we show that, with the proposed method, there is a significant reduction in the execution time and in the number of states and transitions of the automata needed to compute the minimal SDMB as presented in examples 4.3 and 4.4. The examples shows that the test with auxiliary verifiers avoid the computation of some verifiers, and that results in the economy in computational cost.

As the complexity of the systems grows in number of components, the complexity of the verifier grows polynomially. Thus, with a complex system, even with a synchronous diagnosis approach, the diagnoser will demand an elevated amount of memory to be stored. With the method presented in this work, finding a subset of

modules will need a lower computational cost to be obtained and demand less memory to store the diagnoser. This can be observed in Example 4.5, where it is possible to notice as 80% economy in execution time searching for all minimal SDMB. In this example, using Algorithm 4.3, it is possible to notice a good performance searching directly for all minimum SDMB, with almost 90% of economy in execution time.

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

- In order to reduce the computational cost, the first approach is to define the order the modules are supposed to be added. The proposed orders, Algorithm 4.1 and 4.3, guarantee that every combination of subsets will be considered. Once a SDMB is found, adding modules to this SDMB creates a new SDMB with no necessary calculation due to the monotonicity property.

- Adding a component to the verifier may cause considerable calculations. In order to avoid this, a test with a sequence that violates the synchronous diagnosability in the current verifier with the verifier that is supposed to be added is carried out. This test consist in the parallel composition of automaton whose generated language is the prefix-closure of the sequence associated with the selected path, and the verifier of the module that is supposed to be added. If the sequence that violates the diagnosability does not lead to a violating cyclic path in the parallel composition, the new verifier is computed, otherwise, the algorithm chooses another module.

5.1 Future works

It is important to remark that this work is focused in reducing the computational cost to obtain the SDMB, and, consequently, have a reduced synchronous diagnoser to be stored and used. Considering this, we are currently investigating strategies to reduce even more the computational cost of the method in order to mitigate the exponential complexity of computing all minimal SDMB. Initially, we will search for characteristics of the modules, since that the method is sensitive to the module that is used in the beginning of Algorithms 4.1 and 4.3. Depending on the first module, the reduction in computational cost may be higher. In Examples 4.3 and 4.4, the minimal SDMB is $\{2, 3\}$. If Algorithm 4.1 started with module 3 and then modules 2, 4 and 1 are chosen, the computational cost would be smaller. Furthermore, other characteristics of the automata that model the system components will be studied in order to find any characteristic, such as the number of states, transitions, observed events and fault events, that help to define which module must be chosen to start Algorithms 4.1 and 4.3, and the modules that are chosen to continue with Algorithms 4.1 and 4.2.

Another possibility is to consider the delay of diagnosis. It is a important factor to consider and, once the minimal SDMB are defined, this is important to consider together with the computational memory required to store the synchronous diagnoser.

Another possible future work is to extend the method proposed in this work to the decentralized synchronous diagnosis.

References

[1] CASSANDRAS, C., LAFORTUNE, S. *Introduction to Discrete Event Systems*. 2 ed. Secaucus, NJ, Springer-Verlag New York Inc., 2008.

[2] HOPCROFT, J. E., MOTWANI, R., ULLMAN, J. D. *Introduction to Automata Theory Languages and Computation*. 3 ed. Boston, Addison Wesley, 2006.

[3] MIYAGI, P. E. *Controle Programável: Fundamentos do Controle de Sistemas a Eventos Discretos*. Edgard Blucher, 2001.

[4] LAWSON, M. V. *Finite Automata*. Florida, CRC Press, 2003.

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

[6] SAMPATH, D., SENGUPTA, R., LAFORTUNE, S., et al. "Diagnosability of Discrete-Event Systems", *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, v. 40, n. 9, pp. 1555–1575, set. 1995.

[7] SAMPATH, D., 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, mar. 1996.

[8] ZAD, S. H., KWONG, R., WONHAM, W. "Fault Diagnosis in Discrete-Event Systems: Framework and Model Reduction", *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, v. 48, n. 7, pp. 1199–1212, jul. 2003.

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

[10] FANTI, M. P., MANGINI, A. M., UKOVICH, W. "Fault detection by labeled Petri nets in centralized and distributed approaches", *IEEE Transactions on Automation Science and Engineering*, v. 10, n. 2, pp. 392–404, abr. 2013.

[11] GASCARD, E., SIMEU-ABAZI, Z. "Modular Modeling for the Diagnostic of Complex Discrete-Event Systems", *IEEE Transactions on Automation Science and Engineering*, v. 10, n. 4, pp. 1101–1123, out. 2013.

[12] CABASINO, M. P., GIUA, A., SEATZU, C. "Diagnosability of Discrete-Event Systems Using Labeled Petri Nets", *IEEE Transactions on Automation Science and Engineering*, v. 11, n. 1, pp. 144–153, 2014.

[13] BASILE, F., CABASINO, M. P., SEATZU, C. "Diagnosability Analysis of Labeled Time Petri Net Systems", *IEEE Transactions on Automatic Control*, v. 62, n. 3, pp. 1384–1396, 2017.

[14] 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.

[15] 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.

[16] CARVALHO, L. K., BASILIO, J. C., MOREIRA, M. V. "Robust diagnosis of discrete-event systems against intermittent loss of observations", *Automatica*, v. 48, n. 9, pp. 2068–2078, 2012.

[17] CARVALHO, L. K., BASILIO, J. C., MOREIRA, M. V. "Diagnosability of intermittent sensor faults in discrete event systems", *Automatica*, v. 79, pp. 315–325, 2017.

[18] WATANABE, A., SEBEM, R., LEAL, A. B., et al. "Fault prognosis of discrete event systems: An overview", *Annual Reviews in Control*, v. 51, pp. 100–110, 2021.

[19] WATANABE, A. T. Y., , LEAL, A. B., et al. "Combining Online Diagnosis and Prognosis for Safe Controllability", *IEEE Transactions on Automatic Control*, pp. 1–1, 2021.

[20] CABRAL, F. G., MOREIRA, M. V. "Synchronous Diagnosis of Discrete-Event Systems", *IEEE Transactions on Automation Science and Engineering*, v. 17, n. 2, pp. 921–932, 2020.

[21] CABRAL, F. G., VERAS, M. Z. M., MOREIRA, M. V. "Conditional Synchronized Diagnoser for Modular Discrete-Event Systems". In: *14th Interna-*

tional Conference on Informatics in Control, Automation and Robotics (ICINCO), v. 2, pp. 88–97, Madrid, Spain, 2017.

[22] VERAS, M. Z. M., CABRAL, F. G., MOREIRA, M. V. "Distributed Synchronous Diagnosability of Discrete-Event Systems", *IFAC Papers Online*, v. 51, n. 7, pp. 88–93, 2018.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] "Christiani sharpline - automation technology". [online]. Available `http://www.cstt.in/AutomationTechnology.html`.

[32] REIS, L. N. R., MOREIRA, M. V. "Optimal Selection of Subsystems for Synchronous Diagnosis". In: *15th Simpósio Brasileiro de Automação Inteligente*, pp. 1466–1471, Rio Grande, Brazil, 2021.

[33] "Python Code - developed during this work". [online]. Available `https://github.com/LCA-UFRJ/OtimalSelectionSD`.

Appendices

Automata for Examples 4.3 and 4.4

In this appendix, we present all computed verifier automata for Examples 4.3 and 4.4. In chapter 4 some of those automata were presented in order to illustrated the examples, but for the sake of comprehension, they were presented here.

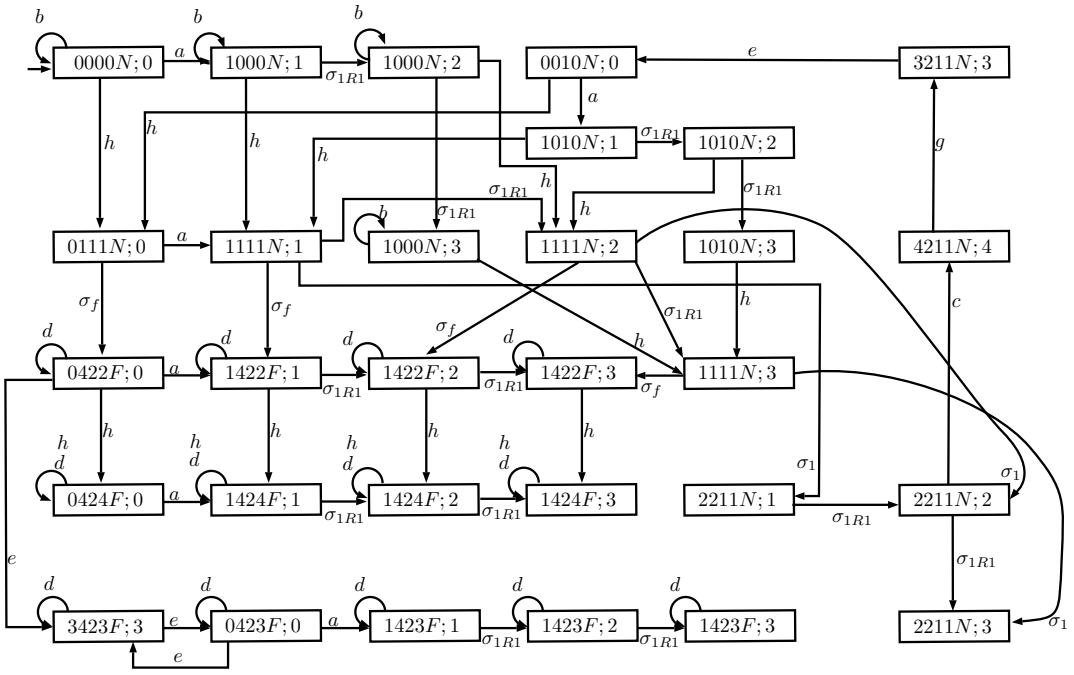


Figure 1: Verifier computed with module $\{1\}$ from Examples 4.3 and 4.4, G_V^1.

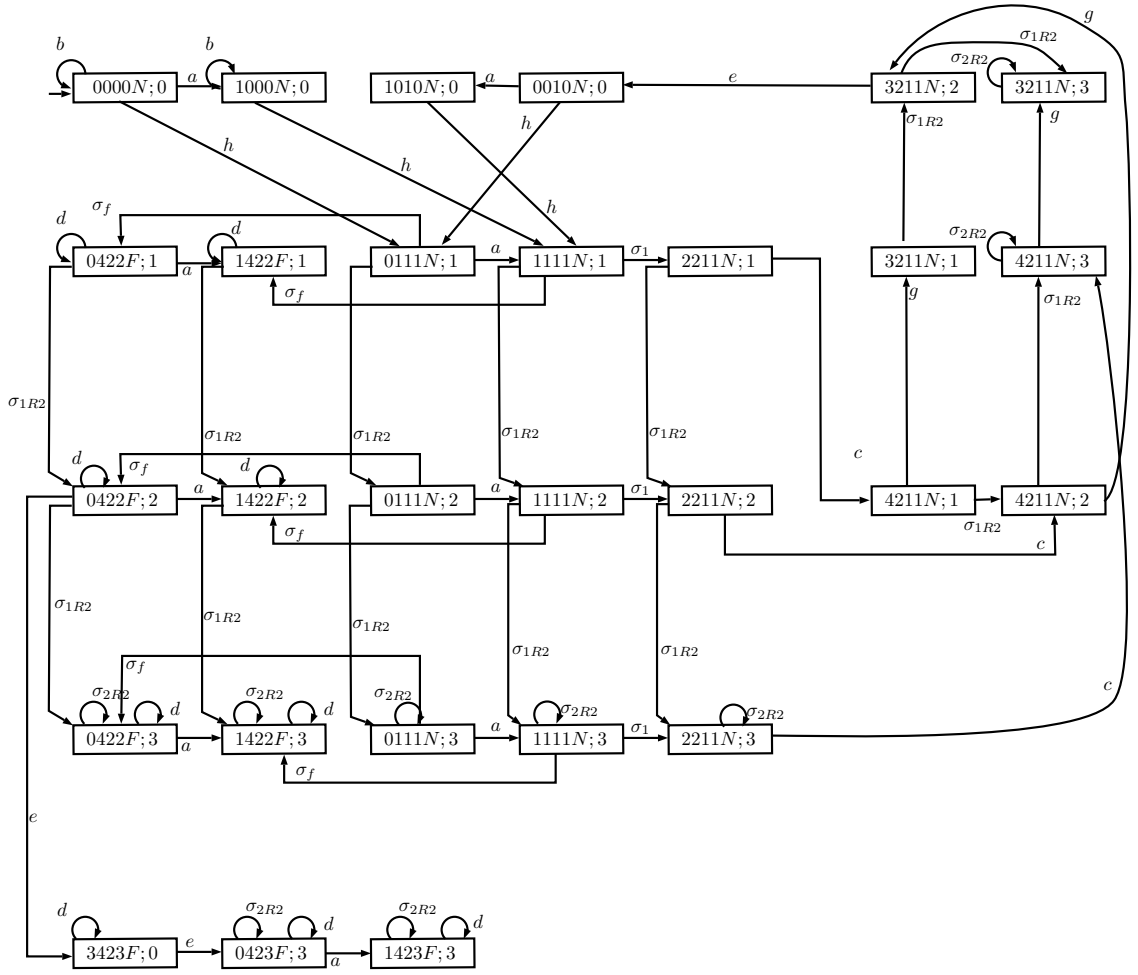


Figure 2: Verifier computed with module {2} from Examples 4.3 and 4.4, G_V^2.

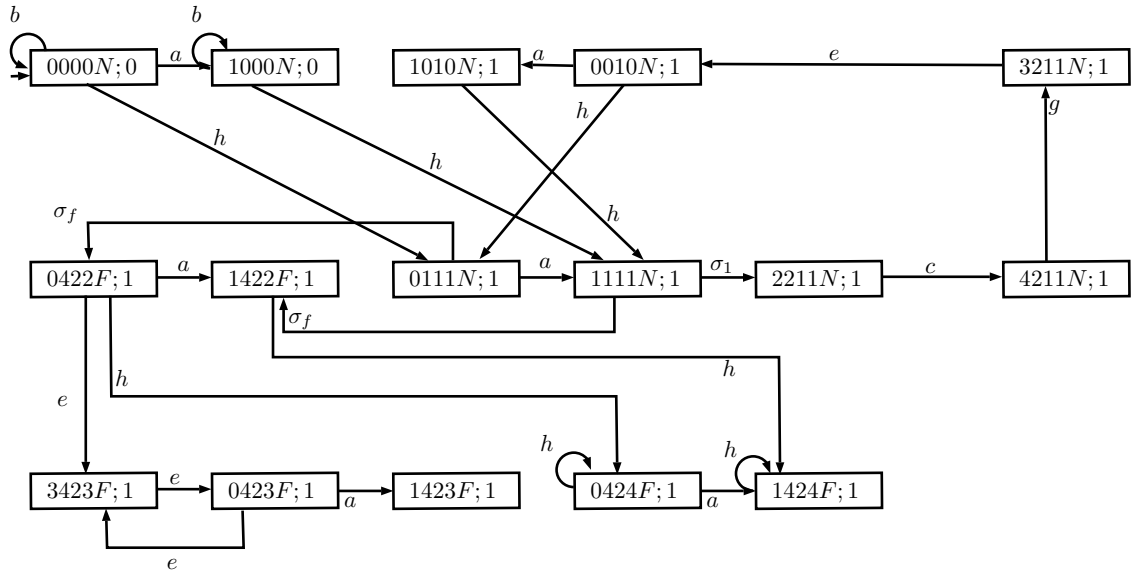


Figure 3: Verifier computed with module {3} from Examples 4.3 and 4.4, G_V^3.

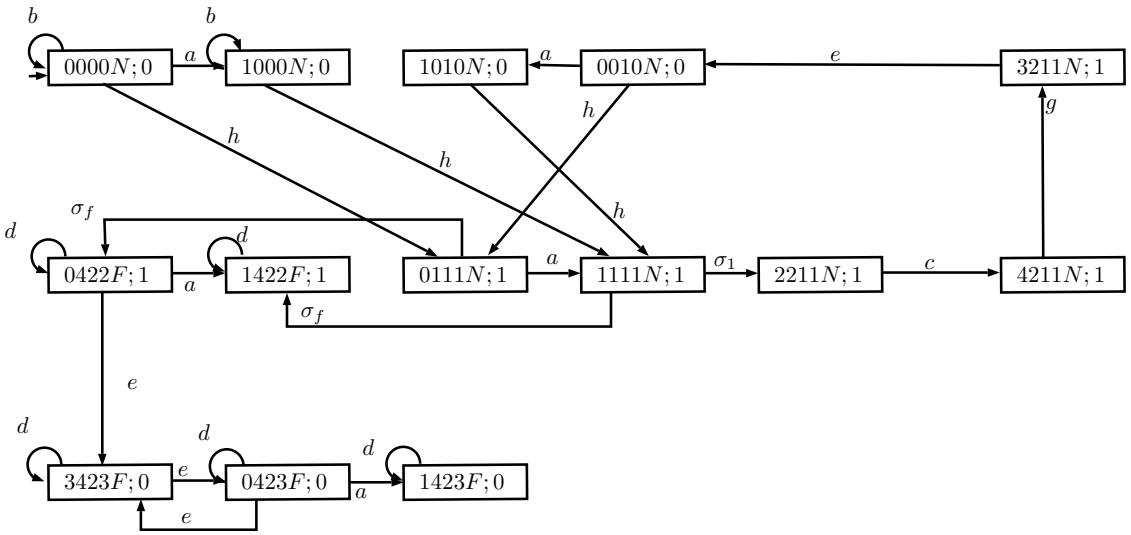


Figure 4: Verifier computed with module $\{4\}$ from Examples 4.3 and 4.4, G_V^4.

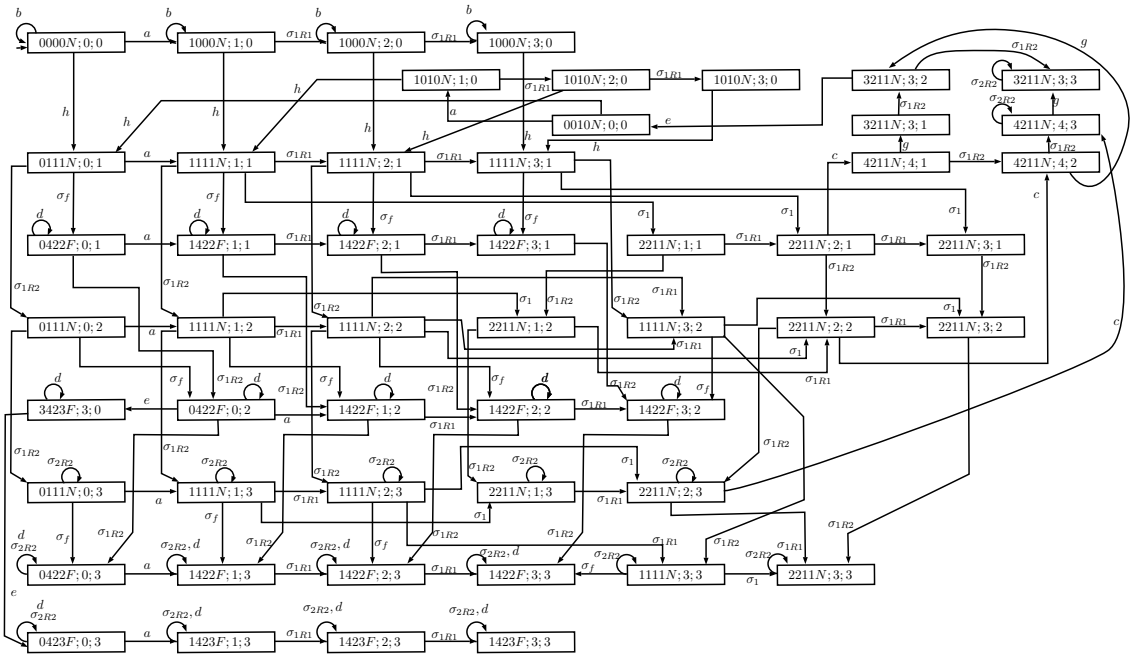


Figure 5: Verifier computed with modules $\{1, 2\}$ from Examples 4.3 and 4.4, G_V^{12}.

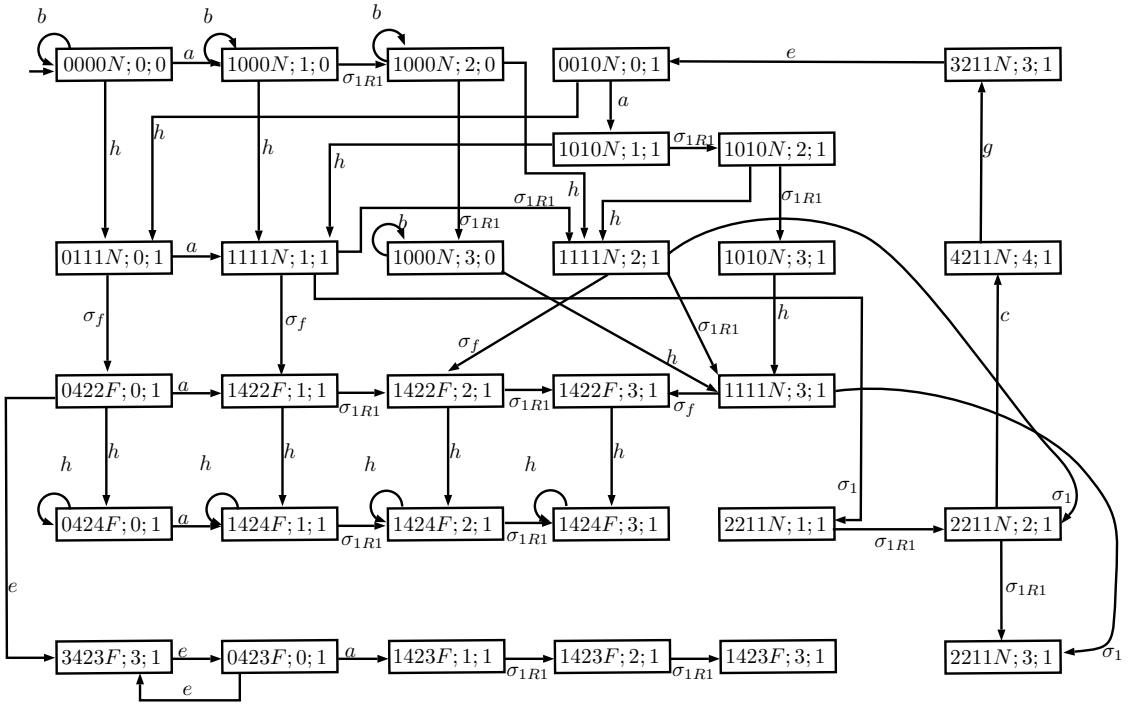


Figure 6: Verifier computed with modules $\{1, 3\}$ from Examples 4.3 and 4.4, G_V^{13}.

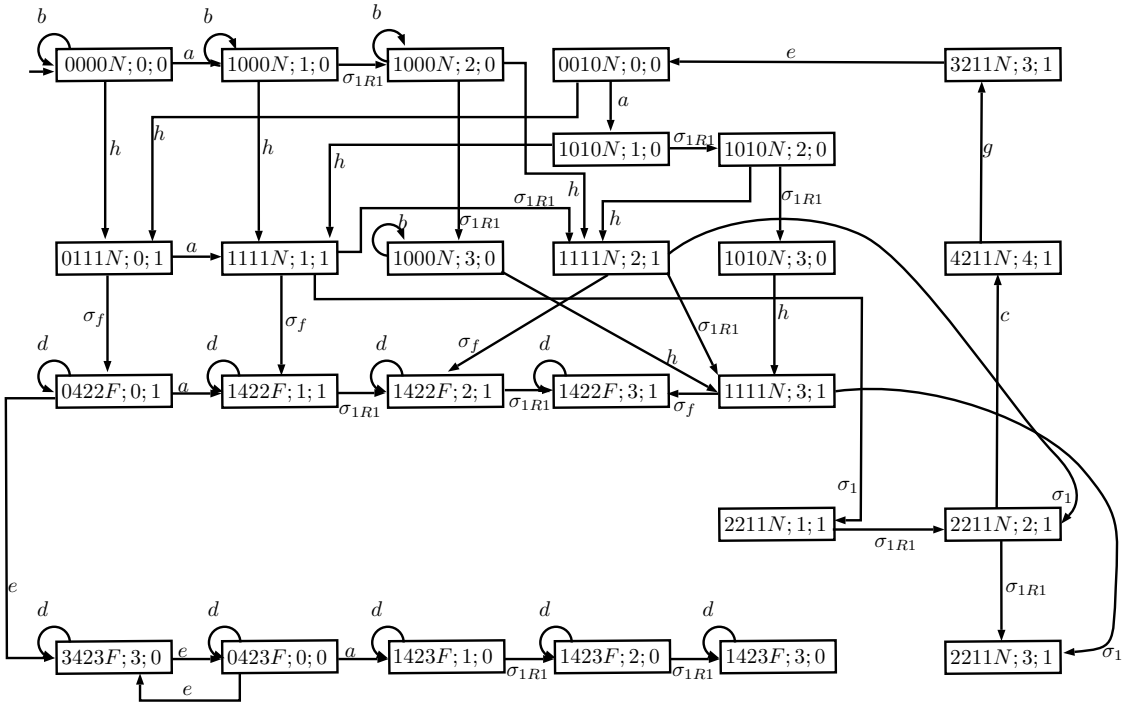


Figure 7: Verifier computed with modules $\{1, 4\}$ from Examples 4.3 and 4.4, G_V^{14}.

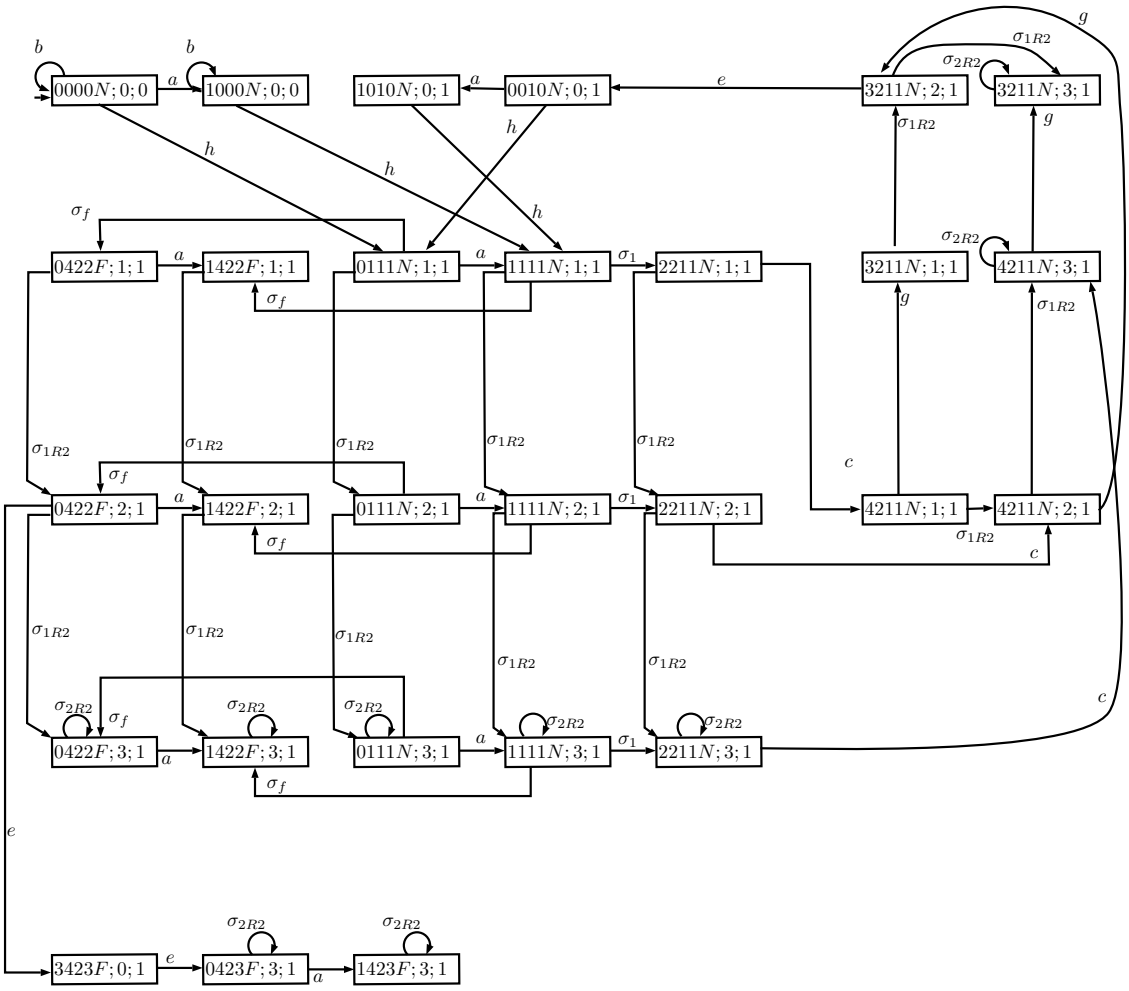


Figure 8: Verifier computed with modules $\{2, 3\}$ from Examples 4.3 and 4.4, G_V^{23}.

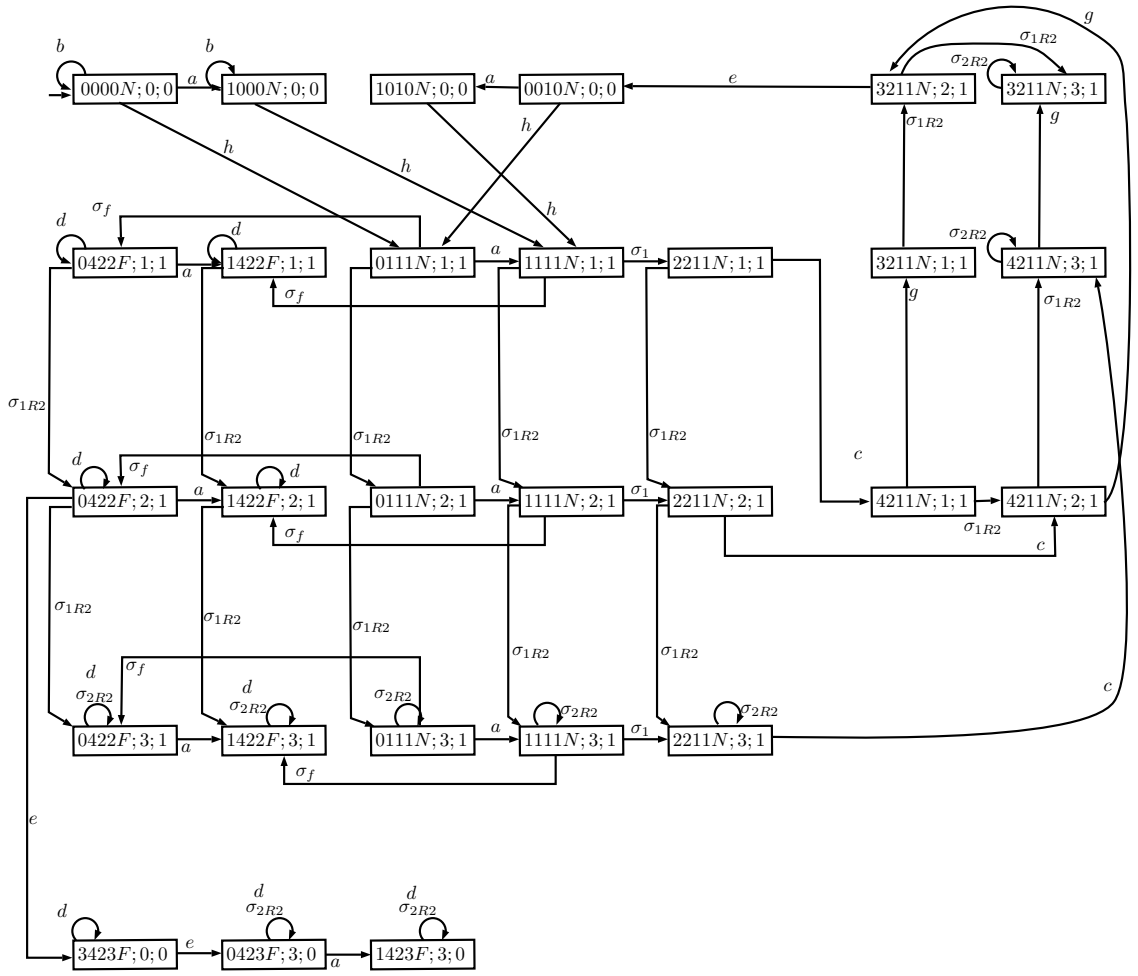


Figure 9: Verifier computed with modules $\{2, 4\}$ from Examples 4.3 and 4.4, G_V^{24}.

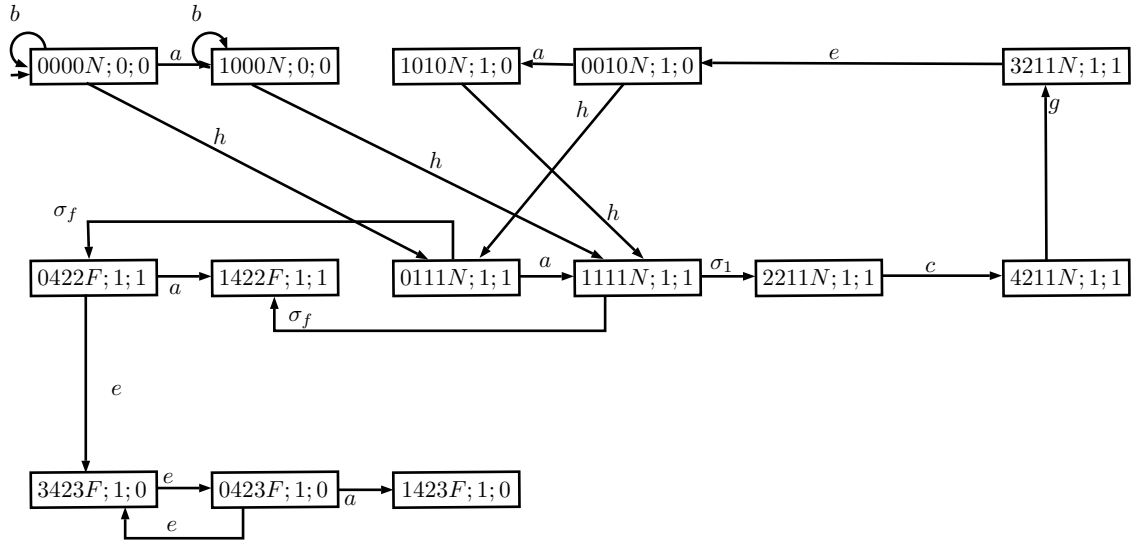


Figure 10: Verifier computed with modules $\{3, 4\}$ from Examples 4.3 and 4.4, G_V^{34}.

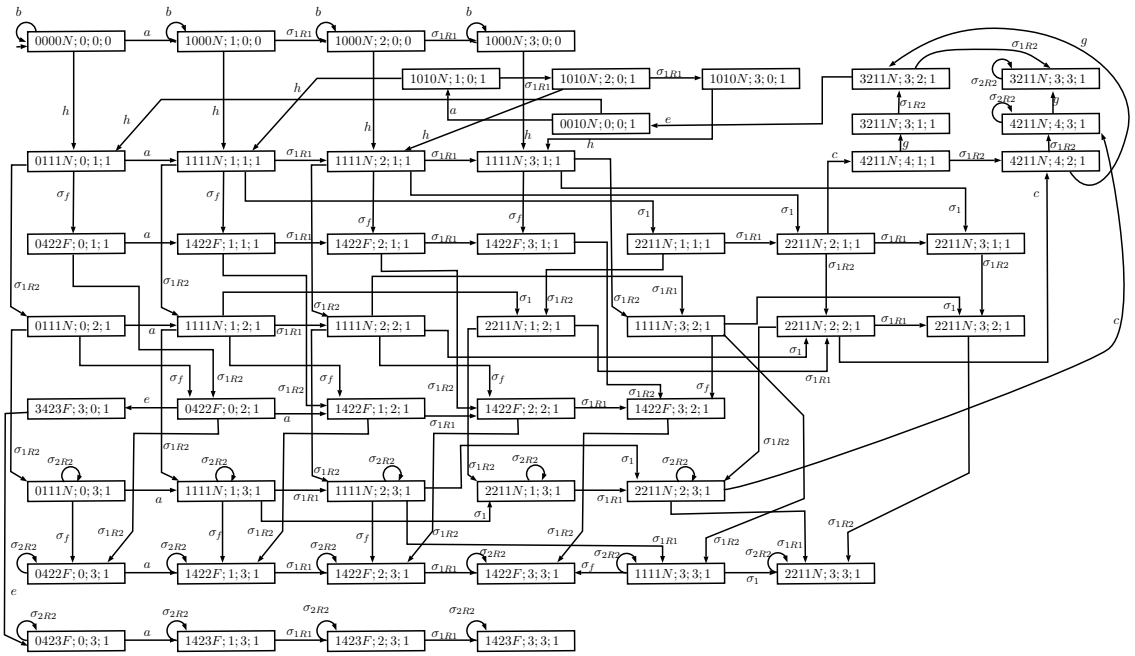


Figure 11: Verifier computed with modules $\{1, 2, 3\}$ from Examples 4.3 and 4.4, G_V^{123}.

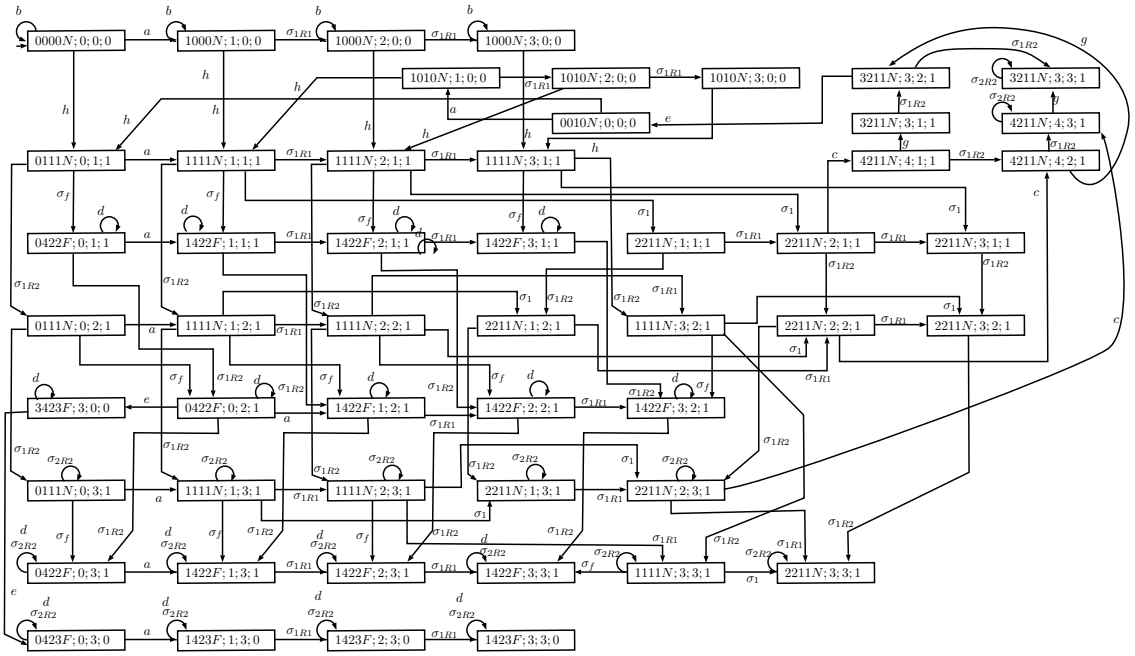


Figure 12: Verifier computed with modules $\{1, 2, 4\}$ from Examples 4.3 and 4.4, G_V^{124}.

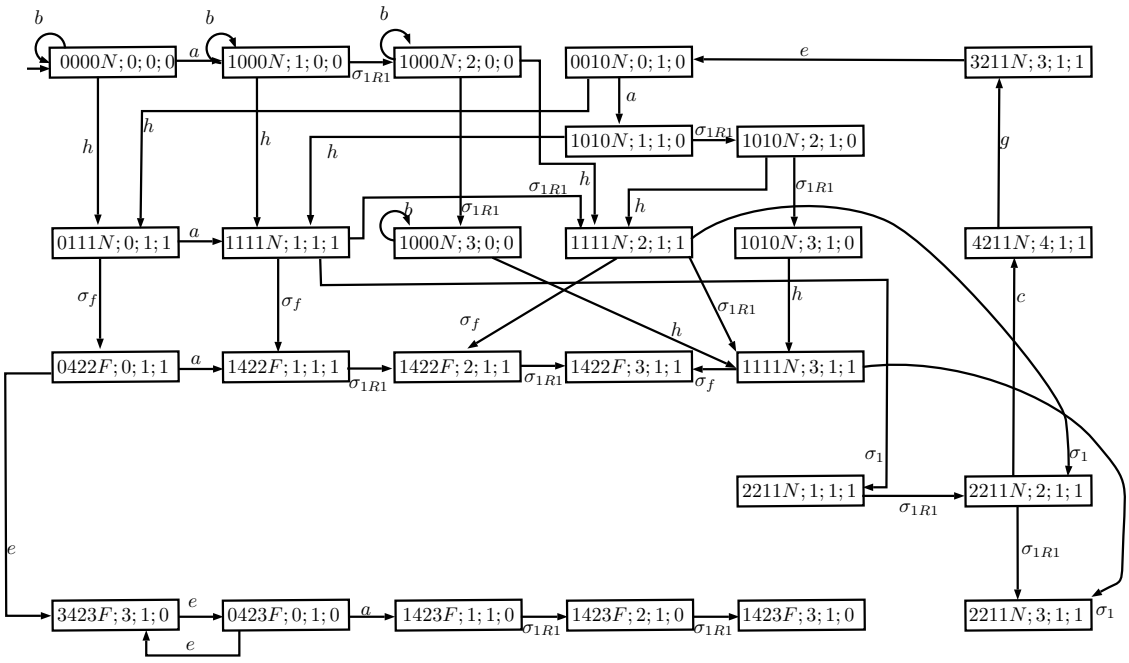


Figure 13: Verifier computed with modules $\{1, 3, 4\}$ from Examples 4.3 and 4.4, G_V^{134}.

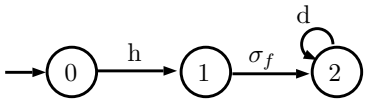


Figure 14: Automaton $G_{V_p}^{\{1\}}$, whose generated language is the prefix-closure of the sequence, that violates the diagnosability, associated with the selected path, from Examples 4.3 and 4.4.

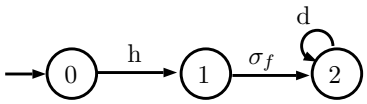


Figure 15: Automaton $G_{V_p}^{\{2\}}$, whose generated language is the prefix-closure of the sequence, that violates the diagnosability, associated with the selected path, from Examples 4.3 and 4.4.

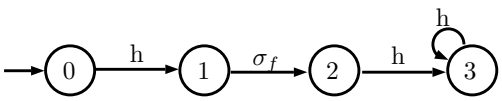


Figure 16: Automaton $G_{V_p}^{\{3\}}$, whose generated language is the prefix-closure of the sequence, that violates the diagnosability, associated with the selected path, from Examples 4.3 and 4.4.

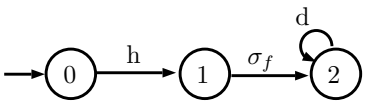


Figure 17: Automaton $G_{V_p}^{\{1,2\}}$, whose generated language is the prefix-closure of the sequence, that violates the diagnosability, associated with the selected path, from Examples 4.3 and 4.4.

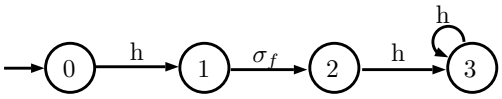


Figure 18: Automaton $G_{V_p}^{\{1,3\}}$, whose generated language is the prefix-closure of the sequence, that violates the diagnosability, associated with the selected path, from Examples 4.3 and 4.4.

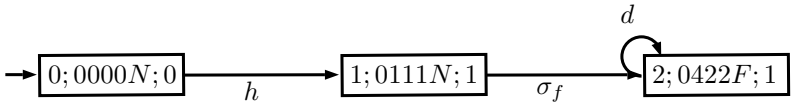


Figure 19: Automaton that represents $G_{V_p}^{2,\{1\}} = G_{V_p}^{\{1\}} \| G_V^{\{2\}}$ of Example 4.3 and 4.4.

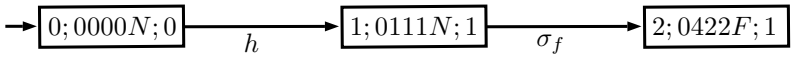


Figure 20: Automaton that represents $G_{V_p}^{3,\{1\}} = G_{V_p}^{\{1\}} \| G_V^{\{3\}}$ of Example 4.3 and 4.4.

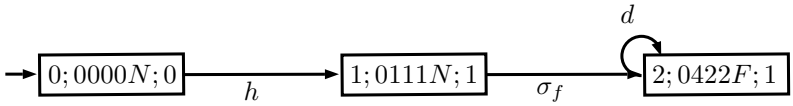


Figure 21: Automaton that represents $G_{V_p}^{4,\{1\}} = G_{V_p}^{\{1\}} \| G_V^{\{4\}}$ of Example 4.3 and 4.4.

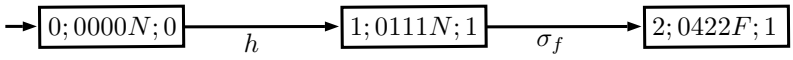


Figure 22: Automaton that represents $G_{V_p}^{3,\{2\}} = G_{V_p}^{\{2\}} \| G_V^{\{3\}}$ of Example 4.3 and 4.4.

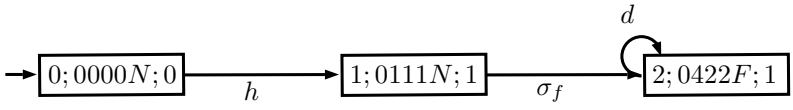


Figure 23: Automaton that represents $G_{V_p}^{4,\{2\}} = G_{V_p}^{\{2\}} \| G_V^{\{4\}}$ of Example 4.3 and 4.4.

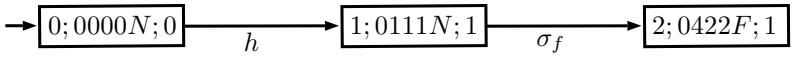


Figure 24: Automaton that represents $G_{V_p}^{4,\{3\}} = G_{V_p}^{\{3\}} \| G_V^{\{4\}}$ of Example 4.3 and 4.4.

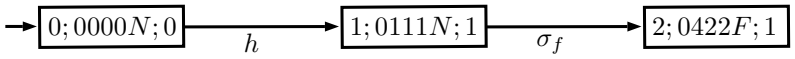


Figure 25: Automaton that represents $G_{V_p}^{3,\{1,2\}} = G_{V_p}^{\{1,2\}} \| G_V^{\{3\}}$ of Example 4.3 and 4.4.

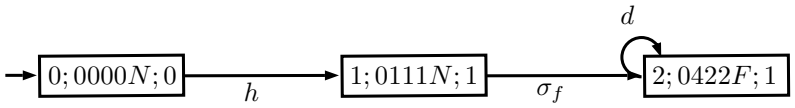


Figure 26: Automaton that represents $G_{V_p}^{4,\{1,2\}} = G_{V_p}^{\{1,2\}} \| G_V^{\{4\}}$ of Example 4.3 and 4.4.

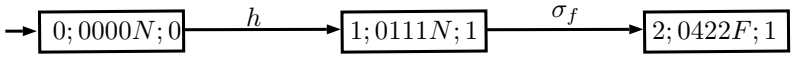


Figure 27: Automaton that represents $G_{V_p}^{4,\{1,3\}} = G_{V_p}^{\{1,3\}} \| G_V^{\{4\}}$ of Example 4.3 and 4.4.