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

Patrícia Mônica Campos Mayer Vicente

DECENTRALIZED SYNCHRONOUS DIAGNOSIS WITH COORDINATION

Florianópolis
2022

Patrícia Mônica Campos Mayer Vicente

DECENTRALIZED SYNCHRONOUS DIAGNOSIS WITH COORDINATION

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

Florianópolis
2022

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

Vicente, Patrícia Mônica Campos Mayer
Decentralized Synchronous Diagnosis with Coordination / Patrícia Mônica Campos Mayer Vicente ; orientador, Felipe Gomes de Oliveira Cabral, 2022.
81 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2022.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Diagnóstico síncrono. 3. Diagnóstico descentralizado. 4. Diagnóstico de falhas. I. Gomes de Oliveira Cabral, Felipe . II. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Engenharia de Automação e Sistemas. III. Título.

Patrícia Mônica Campos Mayer Vicente

DECENTRALIZED SYNCHRONOUS DIAGNOSIS WITH COORDINATION

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

Profa. Patrícia Nascimento Pena, Dra.
Universidade Federal de Minas Gerais

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

Prof. Max Hering de Queiroz, Dr.
Universidade Federal de Santa Catarina

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

ICP-Edu
Documento assinado digitalmente
Werner Kraus Junior
Data: 27/04/2022 21:08:22-0300
CPF: 531.085.239-53
Verifique as assinaturas em https://v.ufsc.br

Prof. Werner Kraus Junior, Dr.
Coordenação do Programa

ICP-Edu
Documento assinado digitalmente
Felipe Gomes de Oliveira Cabral
Data: 27/04/2022 18:19:27-0300
CPF: 124.807.927-22
Verifique as assinaturas em https://v.ufsc.br

Prof. Felipe Gomes de Oliveira Cabral, Dr.
Orientador

Florianópolis, 2022.

ACKNOWLEDGEMENTS

Firstly, I want to thank my biological family Antonio, Mabel and Patrick for all the caring during these years. They are my reason to wake up every day and be brave in my choices.

I am very thankful for my partner, Alejo, for the day-by-day conversations, love and support. Your kindness was essential to me, giving me strength to not give up at the first challenge. You were always there for me.

I want to thank my other family Inti and Paola. I learned a lot with you and I am grateful for your companionship and patience.

I thank my supervisor, Felipe, for all the orientation he gave me during this time. Your guidance is fundamental in my journey to grow as a researcher and human being.

I want to thank my master's friends Letícia and Matheus for all the conversations during hard times of this trajectory even with the physical distance between us.

I thank my friends, Priscila, Tatiana, Victória, Karináti, Alex, Eric, Catiane, Thalita, Bárbara, and Lucas for all the friendship support in my neediest time.

I also want to thank all the professors of the Automation and Systems Department at UFSC for the assistance provided.

Finally, I would like to thank FAPESC for allowing this work to be carried out through the master's scholarship.

"What is behind the visible?"
(PINKOLA, 1992)

RESUMO

O diagnóstico de falhas é uma tarefa fundamental em sistemas de engenharia com o intuito de evitar comportamentos indesejados que podem afetar o funcionamento de equipamentos ou a segurança humana. Neste trabalho, somente o diagnóstico de falhas em Sistemas a Eventos Discretos (DESs) modelados como autômatos são considerados. Recentemente, uma nova arquitetura para diagnóstico descentralizado, denominada Diagnóstico Síncrono Descentralizado, foi proposta. No Diagnóstico Síncrono Descentralizado, os diagnosticadores locais são calculados com base no comportamento livre de falha dos componentes do sistema, o que reduz o tamanho dos diagnosticadores locais ao ser implementado. Embora esse método tenha sido implementado com sucesso, sua principal desvantagem é o crescimento da linguagem livre de falha do sistema para o diagnóstico, reduzindo a sua eficiência. A fim de contornar este problema, é proposto um novo método de diagnóstico síncrono descentralizado (DSD) refinando o status do diagnóstico por meio de *cluster* autômatos dos componentes locais. Para tanto, um protocolo de comunicação entre os estimadores locais e um diagnosticador, descrito em um algoritmo de diagnóstico é proposto. O método elimina o crescimento da linguagem livre de falha para o diagnóstico, garantindo o mesmo desempenho de diagnóstico do método centralizado tradicional. Além disso, uma implementação prática do método em um sistema didático de manufatura é apresentada.

Palavras-chave: Sistemas a eventos discretos. Autômatos. Diagnóstico de falhas. Diagnóstico síncrono.

RESUMO EXPANDIDO

Introdução

O desenvolvimento da Indústria 4.0 aumenta a implementação de sistemas automatizados em inúmeras aplicações, como Internet das Coisas Industrial, Robôs Autônomos, Integração de Sistemas e Segurança Cibernética. O poder de processamento desses sistemas está aumentando, enquanto que o tamanho de seus componentes está diminuindo, e há mais capacidade de comunicação entre os diferentes sistemas e seus componentes. Além disso, esses sistemas podem estar fisicamente distribuídos ou até mesmo construídos de forma descentralizada. Esses recursos levam a sistemas integrados e cada vez mais complexos, conhecidos como Sistemas Ciber-Físicos (SCFs), que incluem ambientes virtuais e físicos.

A modelagem de SCF como um Sistema a Evento Discreto (SED) é útil para tais fins. Um SED é um sistema dinâmico cujo espaço de estados é um conjunto discreto, e cuja evolução é provocada pela ocorrência de eventos que representam mudanças instantâneas no sistema que podem modificar seu estado atual. Portanto, uma vez que um grande número de processos dos SCFs não depende diretamente da passagem do tempo, mas dessa abstração de eventos, um SED pode ser usado para modelar esses sistemas. A chegada ou partida de cargas em um armazém, uma mudança de estado do sensor, a conclusão de uma tarefa ou uma falha mecânica são exemplos de eventos. Podem ser classificados como observáveis quando sua ocorrência puder ser identificada por um sensor ou não-observável se sua ocorrência não estiver associada a um sensor.

Como a evolução de estados de um SED é dada pela ocorrência de eventos e não pela passagem do tempo, equações diferenciais ou diferenças não são apropriadas para representar este tipo de sistema. Os formalismos mais comuns utilizados para descrever e manipular SEDs são autômatos e redes de Petri (LAWSON, 2004; DAVID; ALLA, 2005; CASSANDRAS; LAFORTUNE, 2008). Neste trabalho, apenas sistemas modelados por autômatos são considerados. Autômatos são grafos direcionados, em que os vértices representam os estados, e os arcos são rotulados com eventos que provocam a mudança de um estado para outro (LAWSON, 2004). Quando modelado por autômatos, é possível construir um modelo de planta global complexo de um SED a partir de modelos mais simples de seus subsistemas.

Os sistemas ciber-físicos, como qualquer sistema de engenharia, são suscetíveis à ocorrência de falhas que podem afetar o comportamento esperado, podendo colocar em risco a segurança dos operadores ou agravar problemas nos equipamentos. Portanto, uma técnica de diagnóstico de falha que pode detectar com precisão a ocorrência de um evento de falha em SCFs mais complexos é uma tarefa fundamental que deve ser realizada.

Vários trabalhos na literatura abordam o problema de diagnóstico de falhas de SEDs modelados por autômatos (SAMPATH et al., 1995, 1996; DEBOUK et al., 2000; QIU; KUMAR, 2006; DAIGLE et al., 2007; LEFEBVRE; DELHERM, 2007; CARVALHO et al., 2012; CABASINO et al., 2012; BASILE, 2014; CABRAL et al., 2015; WHITE et al., 2019; CABRAL; MOREIRA, 2020; VERAS et al., 2021). Recentemente, uma nova arquitetura para diagnóstico, denominada Diagnóstico Síncrono Descentralizado, foi proposta. No

Diagnóstico Síncrono Descentralizado, diagnosticadores locais são calculados com base no comportamento livre de falha dos componentes do sistema, com o objetivo de reduzir o tamanho dos diagnosticadores para implementação. Embora esse método tenha sido implementado com sucesso, sua principal desvantagem é o crescimento da linguagem livre de falha do sistema para o diagnóstico, reduzindo a sua eficiência.

A fim de contornar este problema, um método de diagnóstico síncrono descentralizado com coordenador (DSDC) é proposto, refinando o status do diagnóstico por meio de *cluster* autômatos dos componentes locais. Para tanto, um protocolo de comunicação entre os estimadores locais e o coordenador é proposto. Este método impede o crescimento da linguagem livre de falha para o diagnóstico, garantindo o mesmo desempenho de diagnóstico do método centralizado tradicional. Além disso, uma implementação prática do método em um sistema didático de manufatura é apresentada.

Objetivos

O objetivo principal deste trabalho é desenvolver um método descentralizado de diagnóstico síncrono em que a linguagem livre de falha observada e aceita para o diagnóstico síncrono seja igual à linguagem do sistema livre de falha observada.

Primeiramente, exploram-se métodos de diagnóstico de falhas em SEDs modelados como autômatos. Em seguida, uma técnica que refina o status do diagnóstico em um módulo de sincronização é proposta. Finalmente, é apresentado que este método impede o crescimento da linguagem livre de falhas para o diagnóstico.

Objetivos específicos

1. Investigar métodos de diagnóstico de falhas em SEDs modelados como autômatos, especialmente o método de diagnóstico síncrono;
2. Desenvolver um método para eliminar o crescimento da linguagem livre de falhas aceita pelo diagnóstico síncrono;
3. Implementar o método em um sistema didático de manufatura;
4. Analisar o custo computacional do método;

Metodologia

Primeiramente, este trabalho consiste em realizar um levantamento bibliográfico sobre os métodos de diagnóstico de falhas abordados na literatura, com ênfase no diagnóstico síncrono. A partir disso, um estudo sobre o crescimento da linguagem livre de falha aceita pelo diagnóstico síncrono é apresentado para corroborar com a proposta desta dissertação.

Propõe-se o desenvolvimento de um método para eliminar esse crescimento da linguagem com o objetivo de obter a mesma linguagem observada pelo sistema. Para tanto, os componentes livre de falha do sistema e suas observações locais para o diagnóstico são utilizados. Considerando esses componentes, um protocolo de comunicação utilizando a estimativa local dos componentes e um diagnosticador para apresentar o *status* do diagnóstico são propostos. O protocolo de comunicação envia *clusters* correspondentes à observação de um evento para o diagnosticador, o qual realiza operações com esses *clusters* com o intuito de informar a ocorrência da falha. Além disso, o procedimento de diagnóstico considera as possíveis sincronizações de eventos não-observáveis em comum. Os algoritmos desenvolvidos foram implementados em um sistema didático de manufatura, considerando três estações que processam uma peça de trabalho.

Resultados e discussões

O método proposto considera características relevantes de componentes de um determinado sistema, tais como um evento ser observável para um componente e não-observável para outro ou possuir eventos não-observáveis em um comum entre os componentes. O protocolo de comunicação envia os *clusters* de acordo com o evento observado pelos estimadores locais, entretanto, também considera os eventos não-observáveis em sua composição. No procedimento de diagnóstico, obtém-se tanto o resultado sobre ocorrência da falha, quanto a real sincronização dos eventos não-observáveis. Por conta dessa sincronização, é possível obter a mesma linguagem observada do sistema sem o crescimento da linguagem que pode ocorrer no diagnóstico síncrono. Portanto, sistemas que eram não-sincronamente diagnosticáveis e monoliticamente diagnosticáveis podem ser diagnosticados com o método desenvolvido. Além disso, a implementação em um sistema real não-diagnosticável de forma síncrona, representa a viabilidade do método com um custo computacional menor do que a implementação do modelo com a planta completa na maioria dos casos.

Considerações finais

Neste trabalho, um método para o diagnóstico síncrono descentralizado com coordenador com o objetivo de eliminar o crescimento da linguagem aceita pelo diagnóstico é proposto. Uma revisão de literatura sobre arquiteturas de diagnóstico de falhas é apresentada a fim de contextualizar a problemática do crescimento da linguagem no diagnóstico síncrono. Um estudo de caso é apresentado para indicar que determinados sistemas podem ser monoliticamente diagnosticáveis e não-sincronamente diagnosticáveis. O método proposto visa contornar este problema com o desenvolvimento de dois algoritmos: um protocolo de comunicação e um procedimento de diagnóstico. Os algoritmos operam em conjunto e, com isso, conseguem identificar a ocorrência da falha mesmo em sistemas que ocorrem a não sincronização de eventos não-observáveis considerando o diagnóstico síncrono. A implementação do método em um sistema real com eventos não-observáveis em comum e não-sincronamente diagnosticável, uma planta didática de manufatura, mostra a eficácia do método. Além disso, em sistemas com muitas operações em paralelo, e que possuem em sua maior parte eventos observáveis, o método desenvolvido traz um custo computacional menor do que a implementação do modelo da planta completa com o mesmo poder de diagnóstico. Como trabalhos futuros, almeja-se implementar o método em CLPs e estudar possíveis atrasos de comunicação.

Palavras-chave: Sistemas a eventos discretos. Autômatos. Diagnóstico de falhas. Diagnóstico síncrono.

ABSTRACT

Fault diagnosis is a fundamental task that must be performed in engineering systems in order to avoid undesired behaviors that can affect equipment or human safety. In this work, we consider fault diagnosis of Discrete Event Systems (DESs) modeled as automata. Recently, a new architecture for diagnosis called Decentralized Synchronous Diagnosis (DSD) has been proposed. In the DSD, local diagnosers are computed based on the fault-free behavior of the system components with the view to reduce the size of the local diagnosers for implementation. Although this method has been successfully implemented, its main drawback is the growth of the fault-free language of the system for diagnosis, which reduces the diagnosis efficiency. In order to circumvent this problem, in this work, we propose a decentralized synchronous diagnosis method with coordination (DSDC) that refines the diagnosis status using cluster automata of the local components. To do so, we also propose a communication protocol between local state estimators and the coordinator. We show that this method prevents the growth of the fault-free language for diagnosis, which guarantees the same diagnosis performance as the traditional centralized diagnosis method. Furthermore, a practical implementation of the method to a didactic manufacturing system is also presented.

Keywords: Discrete-Event Systems. Automata. Fault diagnosis. Synchronous Diagnosis.

LIST OF FIGURES

LIST OF SYMBOLS

Σ	Set of events
ε	Empty trace
$\|s\|$	Length of a trace
$\star$	Kleene-closure operation
$\overline{L}$	Prefix-closure operation on language L
P_s^l	Projection operation defined as $P_s^l : \Sigma_l^\star \rightarrow \Sigma_s^\star$
$\setminus$	Set difference
$P_s^{l^{-1}}$	Inverse projection operation defined as $P_s^{l^{-1}} : \Sigma_s^\star \rightarrow 2^{\Sigma_l^\star}$
G	Automaton
Q	Set of states
f	Transition function
q_0	Initial state
Γ_G	Feasible event function of automaton G
$\mathcal{L}(G)$	Generated language of automaton G
L	Generated language of automaton G
$Ac(G)$	Accessible part of G
$\|$	Parallel composition
G_{prod}	Product composition automaton
G_{par}	Parallel composition automaton
Σ_o	Set of observable events
Σ_{uo}	Set of unobservable events
P_o	Projection operation defined as $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$
$UR(q)$	Unobservable reach of state q
Obs	Observer automaton of G
Σ_f	Set of fault events
σ_f	Fault event
L_N	Fault-free language of L
$\overline{L_F}$	Faulty language of L
G_N	Automaton that models the fault-free behavior of the system
G_F	Automaton that models the faulty behavior of the system
G_d	Diagnoser automaton
G_l	Labeling automaton
D_i	Local diagnosers
L_{N_a}	Fault-free language for synchronous diagnosis
G_N^a	Automaton resultant from the parallel composition of observer automata
$\mathcal{C}$	Coordinator of DSDC scheme
LM_i	Local measurement site

C	Cluster automaton
E_i	Current state estimate of D_i
C_i^{com}	Communicated local cluster automaton
R	Set of initial states
S	Automaton computed by the cluster synchronous composition
I	Set of indexes of local state estimators

CONTENTS

1 INTRODUCTION

The development of Industry 4.0 increases the implementation of automated systems in countless applications such as Industrial Internet of Things, Autonomous Robots, System Integration, and Cyber-security. In this context, automated systems are becoming more interconnected with more computation power between different systems and their components. Also, the system-human interaction is constantly changing to provide safer cooperation between them. In addition, these systems can be physically distributed or even built in a decentralized way. These features lead to integrated and more complex systems known as Cyber-Physical Systems (CPSs), which include virtual and physical environments.

A CPS is usually modeled as a Discrete Event System (DES). A DES is a dynamic system that has a discrete state space, and its evolution is driven by the occurrence of events that represent instantaneous changes in the system that can modify its current state. Therefore, since a large number of CPSs' processes do not depend directly on time passage but this event abstraction, a DES can be used to model these systems. The arrival or departure of loads on a warehouse, a sensor state change, the completion of a task, or a mechanical failure are examples of events. They can be classified as observable when their occurrence can be identified by a sensor or unobservable if their occurrence is not associated with a sensor.

Since the state evolution of a DES is driven by the occurrence of events, differential or difference equations are not appropriate to represent this type of system. The most common formalisms used to describe and analyze a DES behavior are automata and Petri nets (LAWSON, 2004; DAVID; ALLA, 2005; CASSANDRAS; LAFORTUNE, 2008). In this work, we only consider systems modeled as automata. Automata are directed graphs, where the vertices represent the states, and the arcs are labeled with events that provoke a change from a state to another (LAWSON, 2004). Usually, the automaton complete behavior model of a system can be obtained from the automata models of its subsystems.

Cyber-physical systems, like any engineering system, are subject to the occurrence of faults that can affect their expected behavior which can endanger the safety of operators or aggravate equipment problems. Therefore, a fault diagnosis technique that can accurately detect a fault event occurrence in more complex CPSs is a fundamental task that must be performed. Since the systems are more interconnected and interdependent, a fault event occurrence in one component can spread to its connected ones impacting the entire system. Thus, the fault diagnosis is even more relevant in Industry 4.0 applications analysis. In this work, we address the fault diagnosis problem for Discrete Event Systems modeled as automata to be applied in CPSs.

Several works in the literature address the problem of fault diagnosis of DESs modeled by automata (SAMPATH et al., 1995, 1996; DEBOUK et al., 2000; QIU; KUMAR, 2006; DAIGLE et al., 2007; LEFEBVRE; DELHERM, 2007; CARVALHO et al., 2012; CABASINO et al., 2012; BASILE, 2014; CABRAL et al., 2015; WHITE et al., 2019) and synchronous diagnosis (CABRAL; MOREIRA, 2020; VERAS et al., 2021). In the seminal work of Sampath et al. (1995, 1996), the authors proposed the notion of diagnosability of DESs and the monolithic diagnosis scheme. In order to diagnose the fault event occurrence, it is necessary to compare both fault-free and the post-fault behaviors of the global plant model. The system is diagnosable if the fault event can always be detected and isolated after a bounded number of events observations. The fault event is usually modeled as an unobservable event since its occurrence does not immediately cause a change in the sensors' readings (CASSANDRAS; LAFORTUNE, 2008).

In Sampath et al. (1995), a diagnoser that can verify the system diagnosability and provide a diagnosis status is proposed. In order to compute this diagnoser, the global system model is modified to build a twin-plant that corresponds to the global system model with labeled states. In this case, each state receives the label F if it is reached by a sequence of events that contains the fault, and N, otherwise. The diagnoser is then obtained by computing the observer automaton of the twin-plant. Although the diagnoser presented in Sampath et al. (1995) guarantees an accurate fault diagnosis status, its computation is, in general, avoided since, in the worst-case, the state-space of the diagnoser grows exponentially with the cardinality of the state-space of the plant model (SAMPATH et al., 1995, 1996; HASHTRUDI ZAD et al., 2003; QIU; KUMAR, 2006).

In Cabral et al. (2015), a Petri net diagnoser (PND) based on the fault-free model of the system is proposed. The PND provides the state estimate of the fault-free behavior model of the system after the observation of a sequence. If the state estimate is empty after an observation, the fault event is detected. Different from Sampath et al. (1995), the diagnoser grows polynomially according to the plant size. Furthermore, methods for implementation of the PND on Programmable Logic Controllers (PLC) are presented. Other works also address the fault diagnosis problem in a monolithic way for a robust diagnosis (CARVALHO et al., 2012), for computation of minimal diagnosis bases (SANTORO et al., 2017) and for systems modeled as Petri nets (CABASINO et al., 2012).

Although the monolithic diagnosis approach can be applied to several DESs, there are many applications where the diagnosis information is only available locally. For those systems, decentralized (DEBOUK et al., 2000; QIU; KUMAR, 2006; WANG et al., 2007) and distributed (QIU; KUMAR, 2005; KEROGLOU; HADJICOSTIS, 2014, 2018) architectures are more appropriated. In the following, these architectures will be presented.

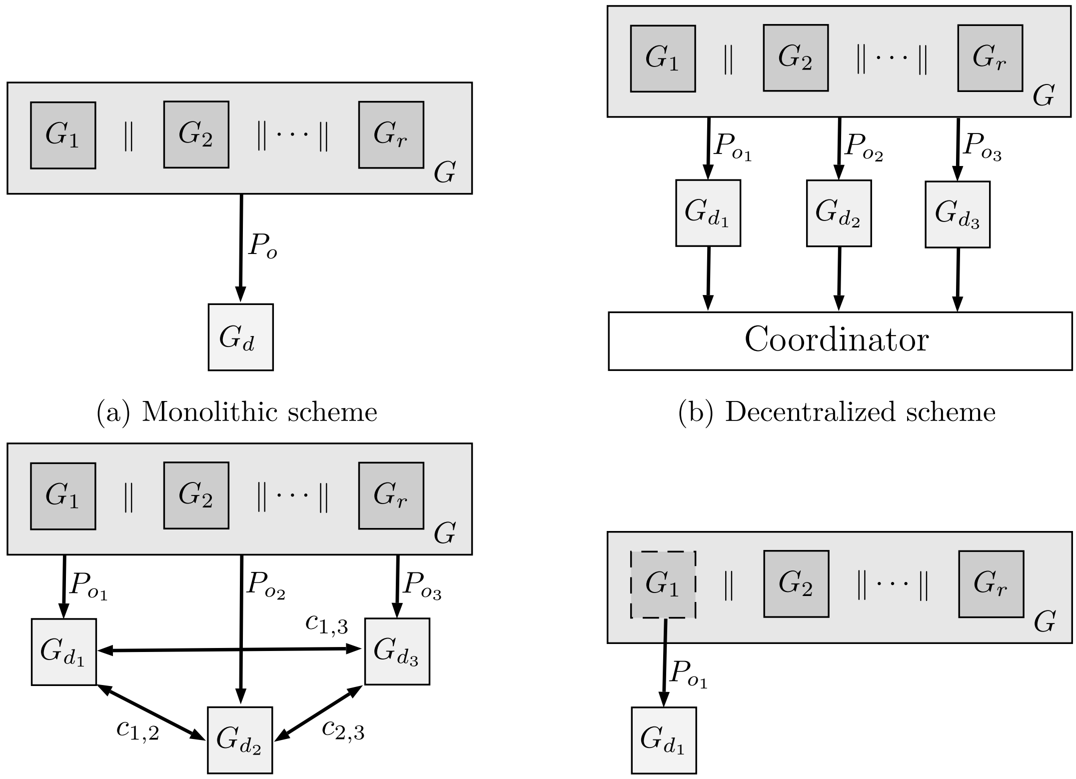
In Debouk et al. (2000), the fault diagnosis approach called decentralized diagnosis is presented. In Protocol 3 of Debouk et al. (2000), the authors extend the work presented in Sampath et al. (1995) for a decentralized architecture. In this context, local diagnosers are computed based on the global system model, considering local observation sites which lead to local observable event sets. The local diagnosers do not communicate with each other.

The fault event occurrence is diagnosed when at least one local diagnoser identifies its occurrence. When it does, the local diagnoser sends the diagnostic status to a coordinator that informs the system operator that the fault has occurred. This Protocol is explored in several works in the literature, such as Qiu and Kumar (2006) and Wang et al. (2007). In Debouk et al. (2000), the notion of decentralized diagnosability, known as codiagnosability, is also presented. The definition of diagnosability can be seen as a particular case of the notion of codiagnosability. Methods to verify the codiagnosability are presented in Qiu and Kumar (2005) and Moreira et al. (2011).

A different architecture for DES, called distributed diagnosis, is proposed in Qiu and Kumar (2008) and Keroglou and Hadjicostis (2018). In this context, the local diagnosers can communicate between them and exchange information regarding event observations and the state estimate of the system. This exchanged information is used to refine the diagnosis, $i.e.$, to perform a more accurate diagnosis. The main drawback of the decentralized and distributed approaches is that the local diagnosers are obtained from the global plant model, which can grow exponentially with the number of system components. Thus, the local diagnosers can also grow exponentially, leading to a high computational cost for diagnosis.

In order to avoid the use of the global plant model for fault diagnosis, the modular architecture is proposed in Debouk et al. (2002) and Contant et al. (2006). In these works, it is assumed that the fault event is modeled in a single component of the system, and notions of modular diagnosability are presented. Moreover, the cited works also consider two assumptions: (i) there are no common unobservable events between the components, and (ii) the faulty component model has *persistent excitation*, $i.e.$, the faulty component model always generates a new event. In practice, the same fault event can occur in more than one component model and in Contant et al. (2006) a method for the verification of the persistent excitation property is not presented. Thus, assumptions (i) and (ii) limit the application of this diagnosis approach.

A comparison between the main diagnosis architectures proposed in the literature is shown in Figure 1. Notice that, in Figure 1, G represents the global plant model and $G_1, G_2, \ldots, G_r$ are the component models of the global plant behavior G, where G is obtained composing the local system models. In Figure 1 (a), P_o represents the observation of observable events which are communicated to a single diagnoser G_d. In Figure 1 (b) and 1 (c), we present the decentralized and distributed scheme, respectively. In these schemes, local diagnosers represented by G_1, G_2, and G_3 computed based on the global system models with local observations represented by P_{o_1}, P_{o_2}, and P_{o_3}. Notice that, in Figure 1 (c), the local diagnosers can communicate between each other through channels $c_{1,2}, c_{2,3}$, and $c_{1,3}$. Finally, in Figure 1 (d), the modular scheme is presented where it is supposed that the faulty component model is G_1. Thus, a single diagnoser G_{d_1} is computed in G_1 with local event communication. In addition, the model behavior for



(a) Monolithic scheme

(b) Decentralized scheme

(c) Distributed scheme

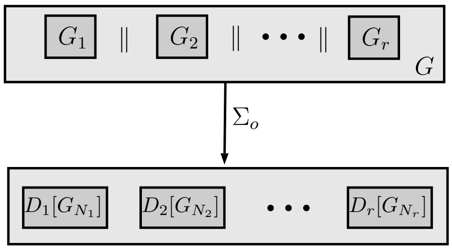
(d) Modular scheme

Figure 1 – Comparison between the main diagnosis architectures proposed in the literature:the monolithic scheme (a); the decentralized scheme (b); the distributed scheme (c); and the modular scheme (d).

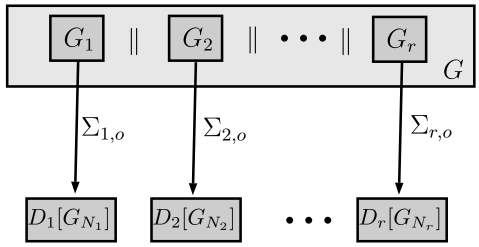
the diagnosis is based on the faulty component.

More recently, a new diagnosis strategy, called synchronous diagnosis, has been proposed in Cabral and Moreira (2020). In this approach, local state estimators computed from the fault-free behavior automata of the system components are implemented. Notions of synchronous diagnosability and codiagnosability and a method to verify these properties are presented in Cabral and Moreira (2020). Since in the synchronous diagnosis strategy, the local diagnosers are based on the local system models instead of the global system model, the local diagnosers do not grow exponentially with the number of system components. Moreover, the assumptions needed for modular diagnosis are not considered in the synchronous approach.

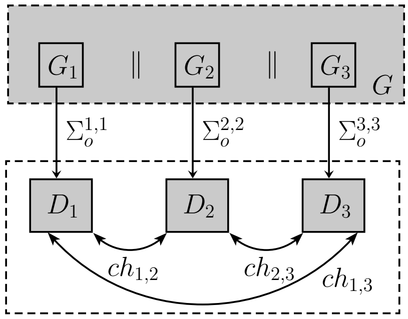
Although the method has been successfully used in real systems, the drawback of the synchronous diagnosis strategy is that the fault-free observable language for synchronous diagnosis can be a larger set than the fault-free observable language of the system. This shows that monolithically diagnosable systems cannot be synchronously codiagnosable. In order to decrease the exceeding language for synchronous diagnosis, the distributed synchronous diagnosis scheme is proposed in Cabral and Moreira (2020). In this scheme, the local diagnosers can exchange information regarding state estimate and event observations. The distributed scheme can reduce the exceeding language but not



(a) Centralized/conditional scheme



(b) Decentralized scheme



(c) Distributed scheme

Figure 2 – Synchronous diagnosis schemes

eliminate it, *i.e.*, the distributed synchronous approach still can present an exceeding language. The architectures proposed for the synchronous diagnosis are depicted in Figure 2, where $D_i[G_{N_i}]$ represents the local diagnosers computed from the local fault-free behavior models, Σ_o is the set of observed events, $\Sigma_{i,o}$ is the set of local observations and $ch_{i,j}$ represents the communication channel between the local diagnosers.

In this master thesis, we propose a decentralized synchronous diagnosis with coordination (DSDC) method, which eliminates the exceeding fault-free language accepted for the synchronous diagnosis approach to the fault-free language generated by the system. In the synchronous diagnosis scheme, the language grows due to the loss of unobservable events synchronization. In order to avoid this loss of synchronization, we propose in this work a coordinator that, based on the local state estimate, verifies the correct synchronization of unobservable events after the observation of an event. To do so, local state estimators send cluster automata of the fault-free behavior component models to the coordinator. We show that by using the DSDC method one can reconstruct the observed fault-free language of the system without using the global system model.

We show that the computational complexity is smaller than implementing the global system behavior model for diagnosis. Since we compute the unobservable reach of the fault-free behavior after each observable event occurrence, this state estimate can only grow if the local component models have a large number of transitions labeled with unobservable events. In practice, this feature has been modified with Industry 4.0 development, where more sensors are being used to communicate their signals reducing the number of unobservable events.

1.1 OBJECTIVES

The main goal of this work is to develop a decentralized synchronous diagnosis with coordination method where the observed fault-free language accepted for synchronous diagnosis is equal to the observed fault-free system language.

Firstly, we explore fault diagnosis schemes in the DES framework modeled as automata. Then, we propose a technique that refines the diagnosis status with a communication protocol and a coordinator. Finally, we show that this method prevents the growth of the fault-free language for diagnosis.

1.1.1 Specific objectives

1. Investigating fault diagnosis methods of DESs modeled as automata, especially the synchronous diagnosis method;

2. Develop a method to eliminate the exceeding fault-free language accepted by the synchronous diagnosis scheme;

3. Implement the method in a didactic manufacturing system;

4. Analyze the computational cost of the method.

1.2 WORK ORGANIZATION

This work is organized as follows. In Chapter 2, the fundamentals concepts about DESs modeled as automata are presented. In Chapter 3, the notion of diagnosability of DESs considering the classical approach and the definition of synchronous codiagnosability are introduced. Also, the growth of the language accepted for synchronous diagnosis is illustrated. The decentralized synchronous diagnosis with coordination method is proposed in Chapter 4. In Chapter 5, the decentralized synchronous diagnosis with coordination method applied to a real system is presented. Finally, in Chapter 6 the conclusions of this work along with its contributions and future works are presented.

2 FUNDAMENTALS OF DISCRETE EVENT SYSTEMS

A Discrete Event System (DES) is a dynamic system with a discrete state space, and its evolution depends on the occurrence of, usually, asynchronous events. Events can be seen as instantaneous actions that change the state reached by the system. In this work, only automata are considered as modeling formalism to describe DES.

In order to introduce the concepts of automata, we first consider the notion of languages and its characteristics. The formal definitions used in this work can also be found in Cassandras and Lafortune (2008).

2.1 LANGUAGES

In this work, Σ is denoted as the event set of a Discrete Event System (DES) and σ as a generic event. A sequence of events forms a trace. If a trace does not contain any event, it is called the empty trace and the symbol ε is used to represent it. $\|s\|$ represents the length of trace s. The empty trace ε has length equal to zero. Definitions of language and live language are stated in the sequel.

Definition 2.1 (Language)**.** *A language L defined over an event set Σ is a set of finite-length traces formed from events in Σ.*

Definition 2.2 (Live language)**.** *A language L is said to be live if for all $t \in L$, exists σ such that $t\sigma \in L$.*

For example, the language $L = \{\varepsilon, a, aa, ab, bc, abc\}$ is defined over the event set $\Sigma = \{a, b, c\}$ and it consists of six traces, including the empty trace ε. In the following, operations used to manipulate languages are presented.

2.1.1 Language operations

Since languages are sets, all set operations can be applied to languages. The concatenation is the main operation involved in the construction of traces, and consequently languages, from an event set Σ. Consider the example aforementioned, where the trace $aa \in L$ is an element of the language L. This trace is formed by the concatenation of the event a with another event a. The empty string ε is the identity element of the concatenation, *i.e.*, $t\varepsilon = \varepsilon t = t$, for any trace t.

Definition 2.3 (Concatenation)**.** *Let $L_1, L_2 \subseteq \Sigma^\star$, then:*

$$L_1L_2 = \{t \in \Sigma^\star : (t = t_1t_2) \text{ where } (t_1 \in L_1) \text{ and } (t_2 \in L_2)\}$$

A trace s is in L_1L_2 if it can be obtained by the concatenation of a trace $t_1 \in L_1$ with a trace $t_2 \in L_2$

The set of all finite traces that can be formed with the elements of Σ is denoted by $\Sigma^\star$ and it includes the empty trace ε. The notation $\star$ represents the Kleene-closure operation. Languages defined over Σ are subsets of $\Sigma^\star$. Sets $\emptyset$, Σ and $\Sigma^\star$ are also languages.

Definition 2.4 (Kleene-Closure). *Let $L \subseteq \Sigma^\star$, then:*

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

The concatenation of a finite number of elements of L is an element of $L^\star$. The empty string ε represents the concatenation of "zero" elements. Moreover, the operation $\star$ is idempotent, *i.e.*, $(L^\star)^\star = L^\star$.

Some important concepts regarding traces are the prefix, subtrace and suffix. Let $s = tuv$, where s is a trace and $t, u, v \in \Sigma^\star$, then it can be stated that: t is the *prefix* of s, u is the *subtrace* of s, and v is the *suffix* of s. Notice that, ε and s are also prefixes, subtraces, and suffixes of s.

The prefix-closure operation of a language L is defined in the sequel.

Definition 2.5 (Prefix-closure). *Let $L \subseteq \Sigma^\star$, then:*

$$\overline{L} = \{t \in \Sigma^\star : (\exists u \in \Sigma^\star)[tu \in L]\}$$

The prefix-closure of L is the language represented by $\overline{L}$ which contains all the prefixes of the traces of L. By definition, the language L is a subset of $\overline{L}$, *i.e.* $L \subseteq \overline{L}$. A language L is said to be *prefix-closed* if $L = \overline{L}$.

For a language $L = \emptyset$, $\overline{L} = \emptyset$ but if $L \neq \emptyset$, then $\varepsilon \in \overline{L}$. Furthermore, $\emptyset^\star = \{\varepsilon\}$ and $\{\varepsilon\}^\star = \{\varepsilon\}$. Also, the concatenation between the empty set and a language is equal to the empty set, *i.e.*, $\emptyset L = L\emptyset = \emptyset$.

Another operation that can be applied to traces is the *projection*, from a larger set of events, Σ_l, to a smaller set of events, Σ_s, where $\Sigma_s \subset \Sigma_l$. Cassandras and Lafortune (2008) present the formal definition as follows.

Definition 2.6 (Projection). *The projection $P_s^l \colon \Sigma_l^\star \to \Sigma_s^\star$ is defined as:*

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

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

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

where $\setminus$ denotes set difference.

According to definition 2.6, applying the projection to a trace s erases events from Σ_l that do not belong to Σ_s. The inverse projection can also be defined as follows.

Definition 2.7 (Inverse projection)**.** *The inverse projection* $P_s^{l^{-1}}: \Sigma_s^\star \to 2^{\Sigma_l^\star}$ *is defined as:*

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

The inverse projection operation applied to a trace t with events from Σ_s results in a set constituted by all traces that can be formed with the events of Σ_l which projection is equal to trace u.

The projection and inverse projection operations can also be applied to languages. In this case, these operations are applied to all traces of the language.

Example 1. *Let* $\Sigma_l = \{a, b, c\}$ *and consider the subsets* $\Sigma_1 = \{a, b\}$ *and* $\Sigma_2 = \{b, c\}$. *The language* $L = \{c, cca, cab, cbcab\} \subset \Sigma_l^\star$. *Consider the two projections* $P_i : \Sigma_l^\star \to \Sigma_i^\star, i = 1, 2$. *We have that:*

$$P_1(L) = \{\varepsilon, a, ab, bab\}$$

$$P_2(L) = \{c, cc, cb, cbcb\}$$

$$P_1^{-1}(\{\varepsilon\}) = \{c\}^\star$$

$$P_2^{-1}(\{b\}) = \{a\}^\star\{b\}\{a\}^\star$$

In the next section, the automata formalism that is used to represent languages is presented.

2.2 AUTOMATA

An automaton is a device that is capable of representing a language giving well-defined rules and it is defined in the following (CASSANDRAS; LAFORTUNE, 2008).

Definition 2.8. *An automaton, denoted by G, is a four-tuple*

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

where Q *is the set of states,* Σ *is the finite set of events,* $f\colon Q \times \Sigma \to Q$ *is the partial transition function, and* q_0 *is the initial state.*

In this work, the marked states symbolism is not used and it is omitted from the tuple definition.

The transition function $f(q_1, \sigma) = q_2$ represents that there is a transition from state q_1 to state q_2 labeled with the event σ, which can be extended to any trace of the generated language of the automaton. The feasible event function is defined as $\Gamma_G\colon Q \to 2^\Sigma$, where it is the set of all events σ for which $f(q, \sigma)!$, and "!" denotes that the function is defined.

An automaton is represented by a directed graph called state transition diagram. The vertices represent the states, and the arcs are labeled with events from one vertex to another (LAWSON, 2004). In order to represent the initial state of the automaton, an

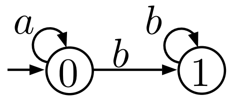


Figure 3 – State transition diagram of automaton G of Example 2.

arc that does not have a previous state is included. In the following, an example of an automaton and its state transition diagram is presented.

Example 2. *Let G be an automaton which state transition diagram is illustrated in Figure 3. The state and event sets of G are $Q = \{0, 1\}$ and $\Sigma = \{a, b\}$, respectively. The initial state of G is $q_0 = 0$. The feasible event function is defined as: $\Gamma_G(0) = \{a, b\}$ and $\Gamma_G(1) = \{b\}$. The transition function is defined as: $f(0, a) = 0$, $f(0, b) = 1$ and $f(1, b) = 1$.*

The notion of the language generated by an automaton is presented in the following.

Definition 2.9 (Generated language)**.** *The generated language of an automaton $G = (Q, \Sigma, f, q_0)$, $\mathcal{L}(G)$, is*

$$\mathcal{L}(G) = \{t \in \Sigma^\star : f(q_0, t)!\},$$

For the sake of simplicity, in this work, the generated language of G is represented as L. The language L represents all the traces that can be built by following the transitions of the state transition diagram starting at the initial state, *i.e.*, $f(q_0, t)!$. Thus, a trace $t \in L$ if, and only if, it corresponds to an admissible path in the state transition of G. It is important to notice that L is prefix-closed by definition since a trace in L is only possible if all its prefixes are also possible to be generated by G. Furthermore, if f is a total function over its domain, then $L = \Sigma^\star$. If $Q = \emptyset$, the language generated by G is also the empty set. If $\Gamma_G(q) \neq \emptyset$ for all $q \in Q$, the language generated by G is said to be live.

For two automata $S = (Q_S, \Sigma, f_S, q_{0,S})$ and $G = (Q, \Sigma, f, q_0)$, S is said to be a subautomaton of G if $f_S(q_{0,S}, t) = f(q_0, t)$ for all $t \in \mathcal{L}(S)$. Notice that this condition implies that $Q_S \subseteq Q$, $q_{0,S} = q_0$, and $\mathcal{L}(S) \subseteq L$. This definition also implies that the state transition diagram of S is a subgraph of that of G (CASSANDRAS; LAFORTUNE, 2008).

In the next section, automata operations are defined.

2.2.1 Operations on automata

There are some operations that can be used to modify the language in order to modify an automaton. In this context, there are unary operations for a single automaton and operations for more than one automaton that are usually used for compositions (CASSANDRAS; LAFORTUNE, 2008).

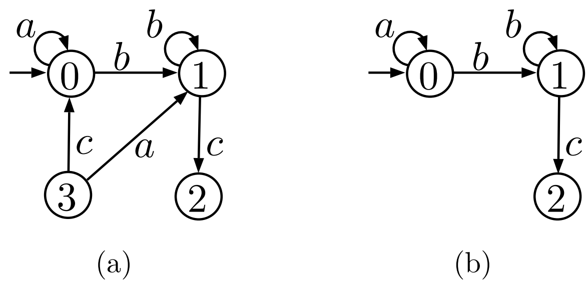


Figure 4 – Automaton G (a) and $Ac(G)$ (b).

The unary operation transforms the state transition diagram of an automaton while the event set Σ remains unchanged. In the following, the definition of accessible part of an automaton is presented.

Definition 2.10 (Accessible part)**.** *The accessible part of an automaton G, $Ac(G)$, is defined as:*

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

where $Q_{ac} = \{q \in Q : (\exists s \in \Sigma^\star)[f(q_0, s) = q]\}$*, and* $f_{ac} = f|_{Q_{ac} \times \Sigma \to Q_{ac}}$.

The notation $f|_{Q_{ac} \times \Sigma \to Q_{ac}}$ means that function f is restricted to a smaller domain of the accessible states Q_{ac}.

The accessible part of an automaton G produces an automaton $Ac(G)$, where all the states and its related transitions that are not reachable from the initial state q_0 are deleted. It is important to notice that this operation does not affect the generated language of G, $\mathcal{L}(G)$.

In the following, an example to show the accessible part of an automaton operation is presented.

Example 3. *Let G be the automaton depicted in Figure 4(a) with the set of states* $Q = \{0, 1, 2, 3\}$ *. The accessible part of G is represented in Figure 4(b), where* $Q_{ac} = \{0, 1, 2\}$.

The composition operations are used to obtain a single automaton from two or more automata. In this work, only one composition operation on automata is defined: the parallel compositions. This operation is used to compute an automaton model of a system from its subsystems models (CASSANDRAS; LAFORTUNE, 2008).

In general, systems are composed by smaller components or subsystems that interact between themselves. The component behavior can be internal or coupling, and are modeled by *private* and *common* events, respectively. The parallel composition, also named *synchronous composition*, is the operation which is capable of integrating individual systems components while considering their private behavior. This operation is formally defined as follows.

Definition 2.11 (Parallel composition)**.** *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. The parallel composition of* G_1 *and* G_2 *is the automaton:*

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

where

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

According to definition 2.11, a common event, *i.e.*, an event in $\Sigma_1 \cap \Sigma_2$, can only be executed if G_1 and G_2 executes it simultaneously. The private events, *i.e.*, those in $(\Sigma_1 \setminus \Sigma_2) \cup (\Sigma_2 \setminus \Sigma_1)$ can be executed whenever they are possible in G_1 or in G_2. Therefore, the parallel composition allows each component to execute their private behavior and only synchronizes the common behavior of the components.

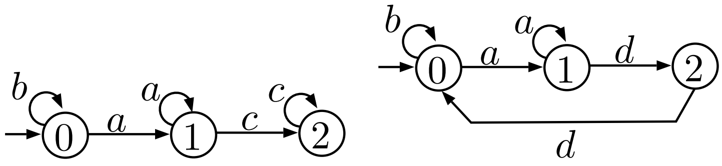
The parallel composition is equivalent to the product composition if $\Sigma_1 = \Sigma_2$ since all transitions will be synchronized. If $\Sigma_1 \cap \Sigma_2 = \emptyset$, then $G_1 \| G_2$ is the concurrent behavior of G_1 and G_2 because there are no synchronized transitions. Let $P_i = (\Sigma_1 \cup \Sigma_2)^\star \to \Sigma_i^\star$ be two projections for $i = 1, 2$. The language generated by $G_1 \| G_2$ is equal to $L(G_1 \| G_2) = P_1^{-1}(L(G_1)) \cap P_2^{-1}(L(G_2))$. Moreover, this operation has the associative property, *i.e.*, $(G_1 \| G_2) \| G_3 = G_1 \| (G_2 \| G_3)$.

In the following, an example of product and parallel composition operations is presented.

Example 4. *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, where* $\Sigma_1 = \{a, b, c\}$ *and* $\Sigma_2 = \{a, b, d\}$*. The state transition diagrams of* G_1 *and* G_2 *are shown in Figure 5 (a) and 5 (b), respectively. The automata* G_{prod}= $G_1 \times G_2$ *and* G_{par}= $G_1 \| G2$ *that correspond to the product and parallel composition are presented in Figures 6 (a) and 6 (b), respectively. Notice that in automaton* G_{prod} *all transitions are labeled with events from* $\Sigma_1 \cap \Sigma_2 = \{a, b\}$*, whereas* G_{par} *models the synchronization of* G_1 *and* G_2 *and their concurrent behavior through events* $\Sigma_1 \cup \Sigma_2 = \{a, b, c, d\}$*.*

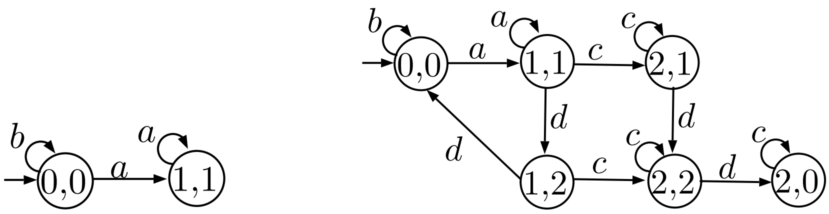
2.2.2 Partially-observed automata

The event set of an automaton G can be partitioned as $\Sigma = \Sigma_o \dot\cup \Sigma_{uo}$, where Σ_o and Σ_{uo} denote the set of observable and unobservable events, respectively. An event is observable when its occurrence can be detected by a sensor, for example. Fault events are usually modeled as unobservable events since their occurrence does not immediately provoke a change in sensor readings (CASSANDRAS; LAFORTUNE, 2008). In order



(a) G_1 (b) G_2

Figure 5 – Automata G_1 and G_2 of Example 4.



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

Figure 6 – Automata G_{prod} and G_{par} of Example 4.

to improve the readability of this work, the unobservable events are represented inside brackets in the automata transition state diagrams.

The observed language of a system G can be obtained from its generated language L by applying the projection operation $P_o(L)$, where $P_o : \Sigma^\star \to \Sigma_o^\star$. Given a system with observable and unobservable events, it is necessary to know the set of possible states reachable from a state $q \in Q$ after the occurrence of an observable event. The unobservable reach represents this set of states and its definition is presented in the following.

Definition 2.12 (Unobservable reach)**.** *The unobservable reach of a state $q \in Q$, denoted as $UR(q)$, is defined as:*

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

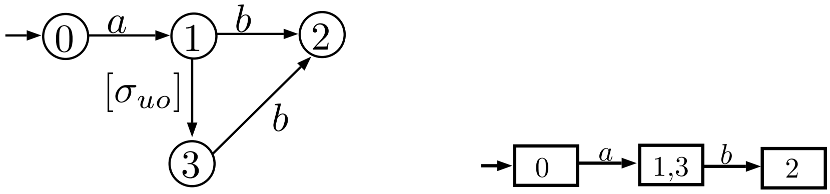
The unobservable reach is extended to sets of states $A \subseteq Q$ as:

$$UR(A) = \bigcup_{q \in A} UR(q).$$

Definition 2.12 shows that the unobservable reach of a state $q \in Q$ is a set formed by all states reached from q by sequences of transitions labeled with unobservable events. The unobservable reach can be used to build an observer automaton from G, $Obs(G, \Sigma_o)$, that generates the observed language of G, $P_o(L)$. This automaton is defined as follows.

Definition 2.13 (Observer automaton)**.** *The observer automaton of G with respect to a set of observable events Σ_o, denoted as $Obs(G, \Sigma_o)$, is given by:*

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



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

Figure 7 – State transition diagram of automaton G of Example 5 (a), and observer automaton of G, $Obs(G, \Sigma_o)$ (b).

where $Q_{obs} \subseteq 2^Q$. f_{obs} and $q_{0,obs}$ are obtained from the Algorithm 1 (CASSANDRAS; LAFORTUNE, 2008; BASILIO et al., 2010).

Algorithm 1 Observer automaton

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

1: Define $q_{obs} \leftarrow UR(q_0)$, $Q_{obs} \leftarrow \{q_{0,obs}\}$ and $\tilde{Q}_{obs} \leftarrow Q_{obs}$
2: $\bar{Q}_{obs} \leftarrow \tilde{Q}_{obs}$ and $\tilde{Q}_{obs} \leftarrow \emptyset$
3: **for** each $B \in \bar{Q}_{obs}$ **do**
4: $\quad \Gamma_{obs}(B) \leftarrow (\bigcup_{q \in B} UR(q)) \cap \Sigma_o$
5: $\quad$ **for** each $\sigma \in \Gamma_{obs}(B)$ **do**
6: $\quad\quad f_{obs}(B, \sigma) \leftarrow UR(\{q \in Q : (\exists y \in B)[q = f(y, \sigma)]\})$
7: $\quad$ **end for**
8: $\quad \tilde{Q}_{obs} \leftarrow \tilde{Q}_{obs} \cup f_{obs}(B, \sigma)$
9: **end for**
10: $Q_{obs} \leftarrow Q_{obs} \cup \tilde{Q}_{obs}$
11: Repeat lines 2 to 10 until all accessible part of $Obs(G, \Sigma_o)$ is constructed

In the sequel, an example of an observer automaton of a system G is presented.

Example 5. *Let G be an automaton which state transition diagram is shown in Figure 7 (a). The state set of G is $Q = \{0, 1, 2, 3\}$ and the event set of G is $\Sigma = \Sigma_o \dot\cup \Sigma_{uo} = \{a, b, \sigma_{uo}\}$, where $\Sigma_o = \{a, b\}$ and $\Sigma_{uo} = \{\sigma_{uo}\}$. The observer automaton of G, $Obs(G, \Sigma_o)$, is illustrated in Figure 7 (b). If we consider that the system has executed the trace $t = a\sigma_{uo}b$, the observed trace is $P_o(t) = ab$, where $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$. It is important to notice that the state reached after the observation of the trace $P_o(t) = ab$ in $Obs(G, \Sigma_o)$ is $q = \{2\}$. The states of $Obs(G, \Sigma_o)$ correspond to state estimates of G after the observation of a trace.*

2.3 FINAL REMARKS

In this chapter, the formal definition of a DES language and the automata formalism used to represent DESs behavior are presented. In this work, only DESs modeled as automata are considered and it is important to analyze their behavior. The unary and composition operations have also been presented in this chapter.

In the next chapter, the problem of fault diagnosis of DES for the monolithic (SAMPATH et al., 1995) and decentralized synchronous schemes (CABRAL; MOREIRA, 2020) are presented.

3 DIAGNOSIS OF DES

In this chapter, the notion of diagnosability of Discrete Events Systems considering the monolithic approach (SAMPATH et al., 1995) is presented. This method is explored in this chapter because it is the first one presented in the literature and it can be used to explain the diagnosis problem in the context of DESs. The monolithic approach uses the global system model in order to diagnose the fault event occurrence. However, the global system model can grow exponentially with the number of system components in the worst-case scenario.

Thus, in Cabral and Moreira (2020) the synchronous diagnosis approach (CABRAL; MOREIRA, 2020) is presented in order to reduce this computational complexity by using only the fault-free behavior component models for diagnosis. The synchronous diagnosis has been successfully implemented in manufacturing systems and it has a smaller computational complexity than the monolithic approach. However, the fault-free accepted language for synchronous diagnosis can be a larger set than the observed fault-free language of the global system.

The idea of this Master thesis is to refine the synchronous diagnosis strategy in order to restore the diagnosis power of the monolithic approach, $i.e$, diagnose systems that are diagnosable and may be not synchronously diagnosable, without the exponential computational cost with respect to the number of system components. Therefore, in this chapter, both methods are presented.

3.1 MONOLITHIC DIAGNOSIS OF DES

The notion of diagnosability of a language L is to identify an occurrence of determined unobservable event from the observation of the language generated by the system. Since fault events are modeled as unobservable events, it is said that the system is diagnosable with respect to the projection $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$ and the fault event, if the fault event occurrence can be identified. Let G be the automaton that models a system and L be the language generated by G. The fault event set is denoted as Σ_f, where $\Sigma_f \subseteq \Sigma_{uo}$. For the sake of simplicity, it is assumed that there is only one fault event σ_f, $i.e.$, $\Sigma_f = \{\sigma_f\}$. If the system has more than one fault event type, each fault type can be considered separately (WANG et al., 2007). In the following, the definition of faulty and fault-free traces of a system is presented (CASSANDRAS; LAFORTUNE, 2008).

Definition 3.1 (Faulty and fault-free traces)**.** *A faulty trace is a sequence s which contains the fault event σ_f. On the other hand, a fault-free trace does not contain it.*

The fault-free language $L_N \subset L$ denotes the set of all fault-free traces of L. Notice that, $L_N = \overline{L_N}$. In addition, the set of all fault traces of L is given by $L_F = L \setminus L_N$. The subautomaton of G that generates the language L_N and $\overline{L_F}$ are represented as G_N and G_F, respectively.

In Sampath et al. (1995), the definition of language diagnosability is presented considering two assumptions:

A1. The language generated by the system is live;

A2. There is no cycle of unobservable events in the system.

Then, the following definition of language diagnosability can be stated (SAMPATH et al., 1995).

Definition 3.2 (Language diagnosability)**.** *The prefix-closed and live language L is diagnosable with respect to the projection $P_o : \Sigma^\star \rightarrow \Sigma_o^\star$ and Σ_f if*

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

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

It is also important to remark that there are works for fault diagnosis that do not consider assumption A2. In this work, assumption A2 is not necessary as well.

The definition 3.2 indicates that L is diagnosable, if and only if, all fault traces with arbitrarily long length do not have the same projection as any fault-free trace of L_N. Thus, if L is diagnosable, it is always possible to detect and isolate the occurrence of fault events within a bounded number of event occurrences.

In Sampath et al. (1995), a diagnoser automaton, denoted as G_d, that can be used to verify the diagnosability of L and to diagnose a system, is presented. This diagnoser is built based on a labeling automaton, denoted as G_l, computed from the plant model G. The automaton G_l is obtained by labeling the states of G according to the traces that reach them. If a trace that contains the fault event σ_f reaches a state of G, it is labeled with F, otherwise, it is labeled with N. The diagnoser automaton G_d is the observer of G_l with respect to its observable events, *i.e.*, $G_d = Obs(G_l, \Sigma_o)$.

Definition 3.3 (Diagnoser automaton)**.** *The diagnoser automaton G_d with respect to the faulty set Σ_f and the observable events set Σ_o is given by:*

$$G_d = (Q_d, \Sigma_o, f_d, q_{0,d}),$$

where $Q_d \subseteq 2^{Q \times \{N,F\}}$. The transition function f_d, and the initial state $q_{0,d}$ are defined according to Algorithm 2.

The state transition diagram of automaton A_l is illustrated in Figure 8. It is important to notice that the language generated by G_l and G are the same. Furthermore, there are two types of states $q \in Q$ according to the trace that reaches q, $q_l = (q, N)$ if

Algorithm 2 Diagnoser automaton G_d

Input: $G = (Q, \Sigma, f, q_0)$.
Output: $G_d = (Q_d, \Sigma_o, f_d, q_{0,d})$

Define automaton $A_l = (Q_l, \Sigma_f, f_l, q_{0,l})$, where $Q_l = N, F$, $q_{0,l} = N$, $f_l(N, \sigma_f) = F$, and $f_l(F, \sigma_f) = F$

Compute the labeling automaton $G_l = G \| A_l$

Compute the diagnoser automaton $G_d = Obs(G_l, \Sigma_o)$

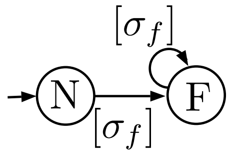


Figure 8 – State transition diagram of automaton A_l.

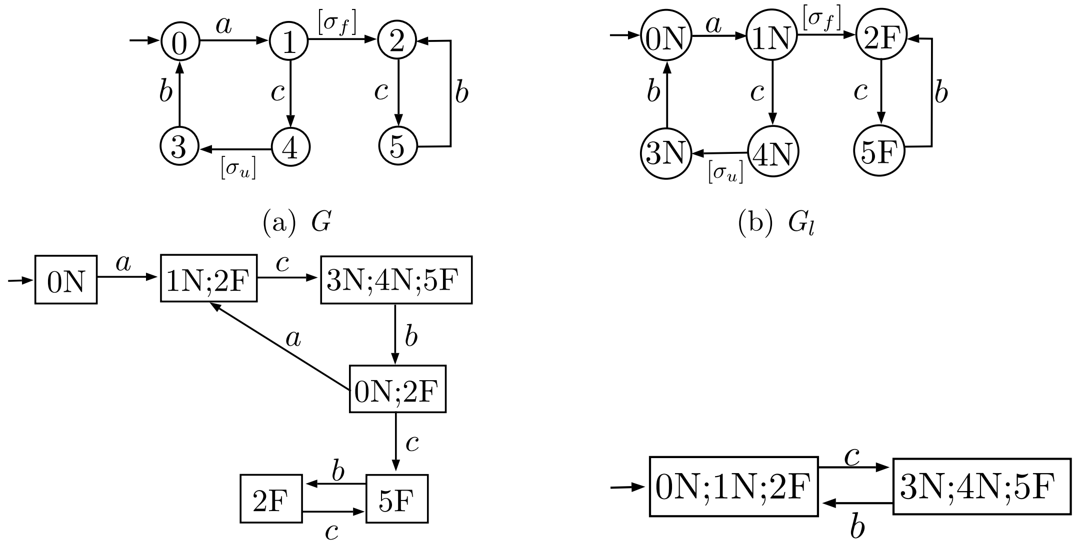
q is reached by a fault-free trace and $q_l = (q, F)$ if q is reached by a faulty trace. The generated language of G_d is the natural projection of the generated language of G, L, *i.e.*, $L(G_d) = P_o(L)$.

If the automaton G_d reaches a state labeled only with F, it indicates that the fault event has certainly occurred and it is diagnosed. On the other hand, if a state is labeled only with N, it represents that the fault event has not been executed by the system. States that have both labels are called uncertain states, indicating that the occurrence of the fault event is not certain. This occurs when an observed generated trace can be mapped both in a fault-free trace and in a faulty trace. In order to verify the diagnosability of a language L using G_d, it is necessary to search for indeterminate cycles in G_d. Indeterminate cycles are cycles of uncertain states that are associated to both faulty and fault-free cycles in the plant G. If there is an indeterminate cycle in G_d, the language L generated by G is not diagnosable.

In the sequel, an example that illustrates the construction of the diagnoser automaton G_d for a given plant G is presented.

Example 6. *Let G be the system depicted in Figure 9(a), such that $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo} = \{a, b, c, \sigma_u, \sigma_f\}$, where $\Sigma_o = \{a, b, c\}$ and $\Sigma_{uo} = \{\sigma_u, \sigma_f\}$. Automaton $G_l = G \| A_l$ is illustrated in Figure 9(b). Finally, in the Figure 9(c) the diagnoser automaton G_d computed from the observer of G_l with respect to its observable event set Σ_o, $G_d = Obs(G_l, \Sigma_o)$ is presented.*

Notice that the initial state of G_d is $\{0N\}$, which corresponds to the unobservable reach of the initial state of G_l. After the occurrence of event a, G_d reaches state $\{1N; 2F\}$. The observations of traces ac and acb leads to states $\{3N; 4N; 5F\}$ and $\{0N; 2F\}$, respectively. It is important to remark that all these states are labeled with N and F, corresponding to uncertain states, which means that the occurrence of the fault event is uncertain. There is an uncertain cycle formed by the states $\{1N; 2F\}$, $\{3N; 4N; 5F\}$, $\{0N; 2F\}$ and it is necessary to verify if this cycle is also indeterminate. In this case, all the states of the



(a) G (b) G_l

(c) G_d (d) G'_d

Figure 9 – Automaton G (a), G_l (b), G_d (c), and G'_d (d). Adapted from (CABRAL, 2017).

unique cycle of G_l *associated to this uncertain cycle have the label* N*, i.e., only with states reached by fault-free traces. Thus, this cycle is not indeterminate. If the system executes the fault trace* $a\sigma_f(cb)^n$*, the fault event is diagnosed when* G_d *reaches state* $\{5F\}$*. Since there are no indeterminate cycles in* G_d*, the language of* G *is diagnosable with respect to the projection* $P_o : \Sigma^\star \to \Sigma_o^\star$ *and* Σ_f*.*

Let us now consider that the observable event set of G *is* $\Sigma'_o = \{b, c\}$ *and the unobservable event set is* $\Sigma_{uo} = \{a, \sigma_u, \sigma_f\}$*. The diagnoser* G'_d *considering* Σ'_o *as the set of observable events is shown in Figure 9 (d). Notice that the states* $\{0N; 1N; 2F\}$ *and* $\{3N; 4N; 5F\}$ *forms an uncertain cycle. Differently from the uncertain cycle of* G_d*, this is an indeterminate cycle since it is associated with cycles in* G_l *labeled with* N *and* F*, for example, cycle formed by the states* $\{0N\}$*,* $\{1N\}$*,* $\{3N\}$*, and* $\{4N\}$*, and* $\{2F\}$ *and* $\{5F\}$ *in* G_l*. Thus, the language generated by* G*,* L*, is not diagnosable with respect to the projection* $P'_o : \Sigma^\star \to \Sigma_o^\star$ *and* Σ_f*.*

Remark 3.1. *The diagnoser presented in this work is used to illustrate the diagnosability definition (Definition 3.2). However, this work does not compute the observers for a diagnoser computation (SAMPATH et al., 1995).*

It is important noticing that the diagnoser automaton G_d is computed based on an observer, and its construction is in general avoided since, in the worst-case, the state-space of the diagnoser grows exponentially with the cardinality of the state-space of the plant model.

3.2 SYNCHRONOUS DIAGNOSIS

In the decentralized diagnosis scheme proposed in (CABRAL; MOREIRA, 2020) the system G is supposed to be formed by r local components, such that $G = \|_{i=1}^{r} G_i$. Local diagnosers D_i are implemented for each local component G_i and they are computed from the fault-free behavior models of $G_i = (Q_i, \Sigma_i, f_i, q_{0,i})$, $G_{N_i} = (Q_{N_i}, \Sigma_i \setminus \Sigma_f, f_{N_i}, q_{0,i})$, $i = 1, \ldots, r$. In this setting, the set of local events Σ_i is partitioned into observable, $\Sigma_{i,o}$, and unobservable, $\Sigma_{i,uo}$, event sets, such that $\Sigma_i = \Sigma_{i,o} \dot{\cup} \Sigma_{i,uo}$. It is important to remark that in the decentralized setting, an event can be observable to local diagnoser D_i and unobservable to local diagnoser D_j, *i.e.*, $\Sigma_{i,o} \cap \Sigma_{j,uo}$ is not necessarily equal to the empty set for $i, j \in \{1, \ldots, r\}$ and $i \neq j$.

Since the decentralized synchronous diagnosis (DSD) scheme proposed in Cabral and Moreira (2020) is performed based on the fault-free local component models G_{N_i} and with local observations, the fault-free language for synchronous diagnosis, L_{N_a}, can be larger than the fault-free system language L_N due to the loss of synchronizations (CABRAL; MOREIRA, 2020). Language L_{N_a} can be written as:

$$L_{N_a} = \cap_{i=1}^{r} P_{i,o}^{o^{-1}}(P_{i,o}(L_{N_i})),$$

where $P_{i,o}^{o} : \Sigma_o^{\star} \to \Sigma_{i,o}^{\star}$ and $P_{i,o} : \Sigma^{\star} \to \Sigma_{i,o}^{\star}$ are projections. Then, the following definition of synchronous codiagnosability can be stated (CABRAL; MOREIRA, 2020).

Definition 3.4 (Synchronous codiagnosability). *Let $G_N = \|_{i=1}^{r} G_{N_i}$, where G_{N_i} is the automaton that models the fault-free behavior of G_i. Assume that L_{N_i} denotes the language generated by G_{N_i}, for $i = 1, \ldots, r$. Let $P_o : \Sigma^{\star} \to \Sigma_o^{\star}$, with $\Sigma_o = \cup_{i=1}^{r} \Sigma_{i,o}$. Then, L is said to be synchronously codiagnosable with respect to P_o, L_{N_i}, $i = 1, \ldots, r$, and Σ_f if*

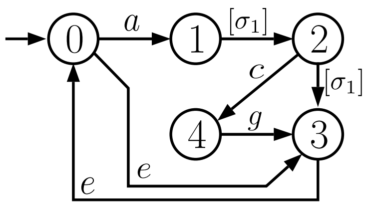
$$(\exists z \in \mathbb{N})(\forall s \in L_F)(\forall st \in L_F, \|t\| \geq z \Rightarrow P_o(st) \notin L_{N_a}).$$

□

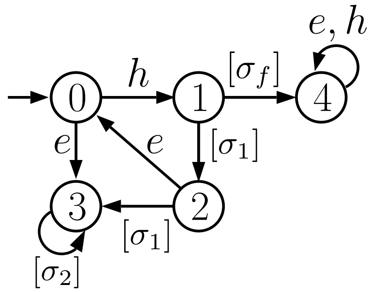
Definition 3.4 of synchronous codiagnosability of a language L is equivalent to definition 3.2 of diagnosability for a system where the fault-free language is L_{N_a} and the faulty language is L_F.

Since $P_o(L_N) \subseteq L_{N_a} = \|P_o(L_{N_i})$ (CABRAL; MOREIRA, 2020), then a system can be diagnosable with respect to P_o, L_N and Σ_f according to Definition 3.2 and not synchronously codiagnosable according to Definition 3.4. The growth of the fault-free observable language L_{N_a} for DSD scheme is associated with the lost of synchronization of common unobservable events between two or more components of the system.

In Cabral and Moreira (2020) two architectures for synchronous diagnosis are presented: (i) the centralized and (ii) the decentralized. In the centralized scheme, all local diagnosers D_i are implemented in one unique site and they receive all information regarding event observations trough a unique communication channel. Thus, the centralized



(a) G_1



(b) G_2

Figure 10 – Automata G_1 and G_2 of case study.

synchronous diagnosis is a particular case of the decentralized synchronous diagnosis, since in the centralized setting, an observable event is observable to all components where it is defined.

Remark 3.2. *In Cabral and Moreira (2020), a method for the verification of synchronous codiagnosability is presented. Although the verification is exponential according to the components number, the synchronous diagnosis method considers only the fault-free behavior model of the system components. Therefore, this growth is avoided for the diagnosis.*

In the sequel, a case study is presented to show the growth of the fault-free language L_{N_a} for the synchronous diagnosis.

3.3 CASE STUDY

Consider a system G composed of two modules G_1 and G_2 depicted in Figure 10. The event sets of the two modules are $\Sigma_1 = \{a, c, e, g, \sigma_1\}$, $\Sigma_2 = \{e, h, \sigma_1, \sigma_2, \sigma_f\}$. The observable event sets of G_1 and G_2 are $\Sigma_{1,o} = \{a, c, e, g\}$, $\Sigma_{2,o} = \{e, h\}$, respectively. The sets of unobservable events of G_1 and G_2 are $\Sigma_{1,uo} = \{\sigma_1\}$, $\Sigma_{2,uo} = \{\sigma_1, \sigma_2, \sigma_f\}$, respectively. The fault event set is $\Sigma_f = \{\sigma_f\}$. Automaton $G = G_1 \| G_2$, and the composed fault-free behavior automaton G_N are shown in Figures 11 and 12, respectively. The fault-free behavior of automata G_1 and G_2, denoted by G_{N_1} and G_{N_2}, respectively, are shown in Figure 13. All automata considered in this case study are taken from Cabral (2017).

Since all synchronous diagnosis schemes are based on the observation of the fault-free behavior of the system components, we can compare the accepted fault-free languages for the different diagnosis methods using observers[1]. According to definition 2.13, the language generated by an observer is the projection of the generated language of the original automaton (CASSANDRAS; LAFORTUNE, 2008). The computation of an observer is presented in Algorithm 1. In the sequence, we present the observer automata of G_{N_1}, G_{N_2},

[1] It is importante to remark that the use of observers in this document is only to simplify the case study. In Cabral et al. (2017) and Cabral and Moreira (2020) observers are also avoided in order to escape from their computational exponential growth in the worst case scenario.

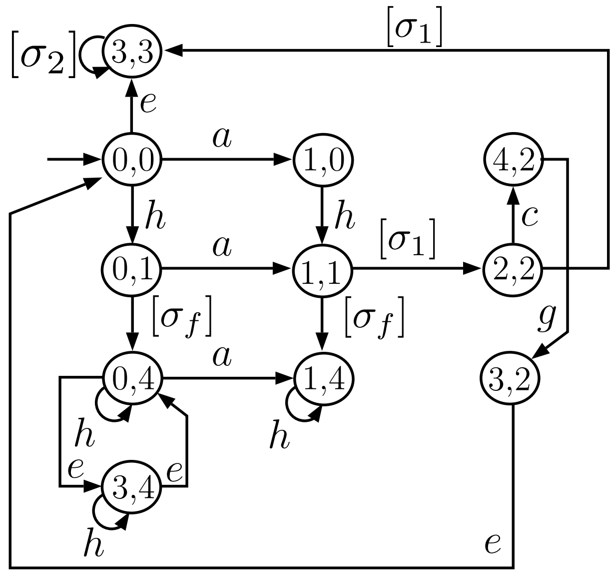


Figure 11 – Automaton G of case study.

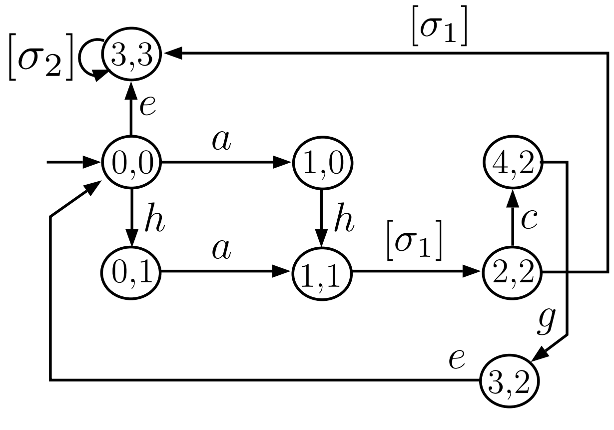
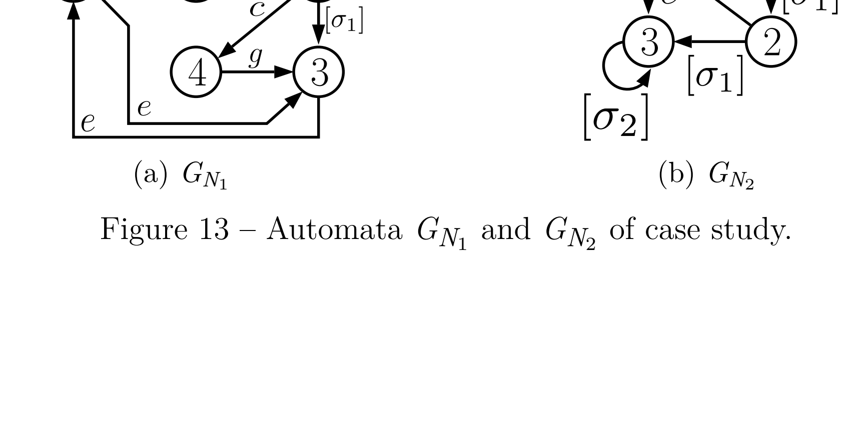
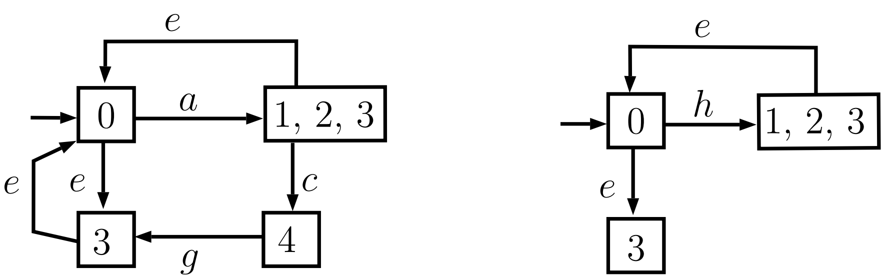


Figure 12 – Automaton G_N of case study.



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

Figure 13 – Automata G_{N_1} and G_{N_2} of case study.



(a) $Obs(G_{N_1}, \Sigma_{1,o})$ (b) $Obs(G_{N_2}, \Sigma_{2,o})$

Figure 14 – Automata $Obs(G_{N_1}, \Sigma_{1,o})$ and $Obs(G_{N_2}, \Sigma_{2,o})$ of case study.

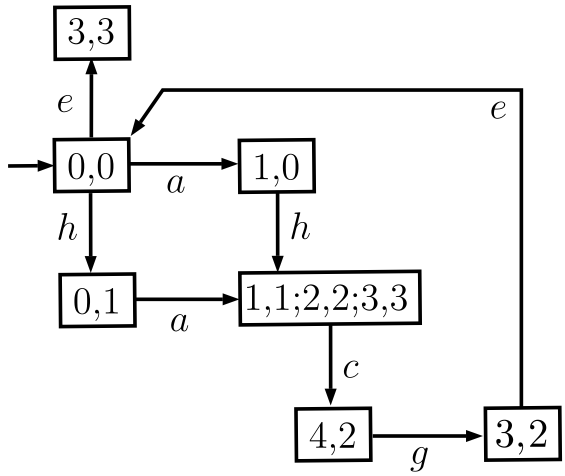


Figure 15 – Automaton $Obs(G_N, \Sigma_o)$ of case study.

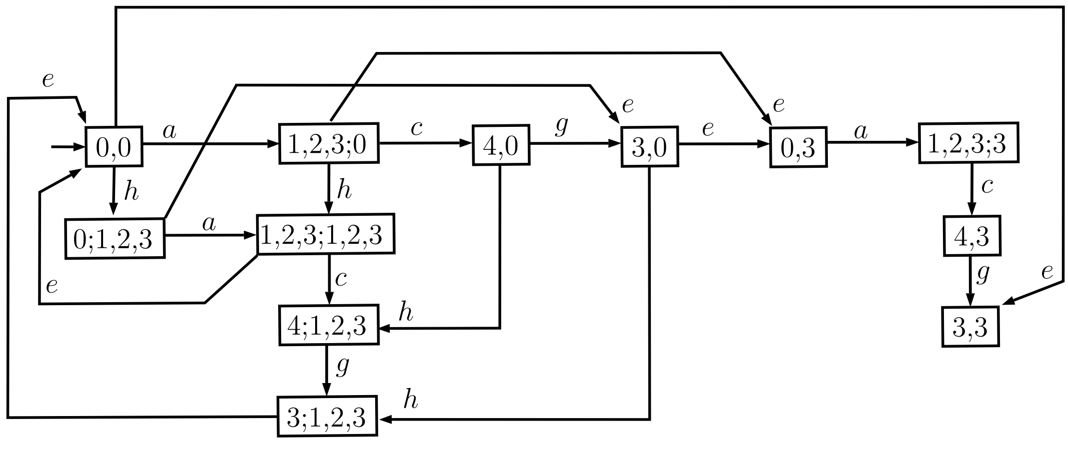


Figure 16 – Automaton G_N^a of case study, $G_N^a = Obs(G_{N_1}, \Sigma_{1,o}) \| Obs(G_{N_2}, \Sigma_{2,o})$.

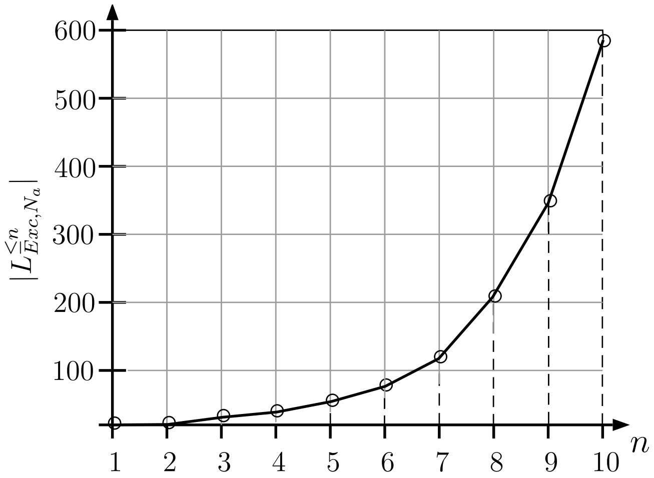


Figure 17 – Cardinality of the exceeding language generated by the decentralized synchronous diagnosis scheme $L^{\leq n}_{Exc,N_a}$ ($\circ$) for different values of n for case study.

and G_N, $Obs(G_{N_1}, \Sigma_{1,o})$, $Obs(G_{N_2}, \Sigma_{2,o})$, and $Obs(G_N, \Sigma_o)$, respectively, in Figures 14 and 15.

Since, in the decentralized synchronous diagnosis scheme, state estimators of the fault-free system component models are implemented in parallel, we can model the fault-free language L_{N_a} by making the parallel composition of the observers $G^a_N = Obs(G_{N_1}, \Sigma_{1,o}) \| Obs(G_{N_2}, \Sigma_{2,o})$, depicted in Figure 16 (CABRAL; MOREIRA, 2020). In the sequel, we compare the generated language by G^a_N with the generated language by $Obs(G_N, \Sigma_o)$. In order to do so, we define languages $L^{\leq n}_{N_o}$ and $L^{\leq n}_{N_a}$ formed of all possible observed fault-free traces of length less than or equal to a given number $n \in \mathbb{N}$ of $Obs(G_N, \Sigma_o)$ and G^a_N, respectively as follows.

$$L^{\leq n}_{N_o} := \{s \in \mathcal{L}(Obs(G_N, \Sigma_o)) : \|s\| \leq n\}, \tag{1}$$

$$L^{\leq n}_{N_a} := \{s \in \mathcal{L}(G^a_N)) : \|s\| \leq n\}, \tag{2}$$

where $\|s\|$ denotes the length of trace s.

Thus, we can define the exceeding language $L^{\leq n}_{Exc,N_a} = L^{\leq n}_{N_a} \setminus L^{\leq n}_{N_o}$. Language $L^{\leq n}_{Exc,N_a}$ corresponds to the traces with length less than or equal to $n \in \mathbb{N}$ that are accepted as fault-free in the decentralized synchronous diagnosis method and do not belong to the fault-free language of the system model. The cardinality of the exceeding language $L^{\leq n}_{Exc,N_a}$ for the decentralized synchronous diagnosis scheme is presented in Figure 17. Note that the cardinality of the exceeding language of this scheme significantly increases as n grows. For example, for $n = 10$, there are 586 more traces in $L^{\leq n}_{Exc,N_a}$ than in $L^{\leq n}_{Exc,N_o}$.

Since each exceeding trace in the decentralized synchronous diagnosis scheme can lead to an increase in the delay bound for diagnosis, the reduction of the accepted fault-free language is of great importance. Moreover, when we consider arbitrary long values of n, this accepted language growth can lead a diagnosable system to be not synchronously codiagnosable. Particularly, in this case study, the language generated by G is diagnosable according to Definition 3.2 and it is not synchronously diagnosable according to Definition 3.4.

3.4 FINAL REMARKS

In this chapter, the definitions of language diagnosability and synchronous codiagnosability are presented. In order to diagnose a fault using the monolithic scheme it is necessary to construct a diagnoser automaton based on an observer and search for indeterminate cycles in the diagnoser. If there is at least one indeterminate cycle, the language generated by the system is not diagnosable. Methods that avoid the use of observers for verification of diagnosability and online diagnosis have also been proposed in the literature.

Furthermore, a case study to verify the fault-free language growth in the decentralized synchronous diagnosis is presented. The concept of observers presented in Sampath et al. (1995) are used to illustrate it. Since the local component models are implemented in parallel, the fault-free language accepted can be seen through the parallel composition of the local observers. The case study aforementioned shows that a diagnosable system can be not synchronously codiagnosable because of this language growth.

In the next chapter, we propose the decentralized synchronous diagnosis with coordination scheme in order to reduce the fault-free language accepted for synchronous diagnosis, L_{N_a}, into the observed system language, $P_o(L_N)$. Thus, the synchronous codiagnosability verification is not needed for the diagnosis method proposed in this work. Since the fault-free observed language of the system according to the DSDC method is equal to $P_o(L_N)$.

4 DECENTRALIZED SYNCHRONOUS DIAGNOSIS WITH COORDINATION

In this chapter, the decentralized synchronous diagnosis with coordination scheme is presented. The goal of the method is to provide a diagnosis scheme with the same diagnosis power as the monolithic approach proposed in Sampath et al. (1995) without using the global plant model for diagnosis. The use of the global automaton model of the system is avoided in order to provide a diagnosis method that has better computational complexity than other approaches that need to implement the whole system model. Since, in this work, only an online diagnosis scheme is proposed, unless stated otherwise, it is assumed that all systems are monolithic diagnosable according to Definition 3.2.

As presented in the previous chapter, the fault-free observed language for the synchronous diagnosis scheme, L_{N_a}, can be a larger set than the observed fault-free system language, $P_o(L_N)$, when there are unobservable events in common between two or more components of the system. This is due to the loss of synchronization of unobservable events, and in order to maintain the same diagnosis power, one needs to reduce L_{N_a} into $P_o(L_N)$. Thus, in this work, a decentralized synchronous diagnosis with coordination scheme is proposed with the view to preserve the synchronization of unobservable events without the use of the fault-free global system model. To do so, the method is based on local state estimators of the fault-free component models of the system that can send cluster automata to a coordinator after the observation of an event. These clusters are composed of the unobservable reach of the local fault-free models after an observed event. The coordinator uses these clusters to build the correct state estimate of the fault-free global model of the system and verifies if an observed event is feasible in the current state estimate. If the event is not feasible, the fault event occurrence is detected.

In the sequel, the diagnosis scheme proposed in this work is presented in details.

4.1 DIAGNOSIS SCHEME

In this work, we introduce a decentralized synchronous diagnosis with coordination (DSDC) scheme based on local state estimators of the fault-free component models of the system D_i that are implemented in parallel and a coordinator $\mathcal{C}$ that receives state estimate information from D_i, $i = 1, \ldots, r$. Coordinator $\mathcal{C}$ uses the local state estimation of D_i to refine the diagnosis and provide the fault event occurrence status. In this regard, suppose that the plant model consists of r components, *i.e.*, $G = \|_{i=1}^{r} G_i$ and, associated with each component G_i, for $i \in \{1, \ldots, r\}$ there is a D_i based on the fault-free component model, G_{N_i}. Each module G_i has a local measurement site LM_i that provides observation of events directly to its state estimator D_i. The set of events of G_i is defined as $\Sigma_i = \Sigma_{i,o} \dot{\cup} \Sigma_{i,uo}$, where $\Sigma_{i,o}$ and $\Sigma_{i,uo}$ denote the sets of observable and unobservable events of G_i, respectively. The set of events of G is denoted as $\Sigma = \cup_{i=1}^{r} \Sigma_i$. The observable and

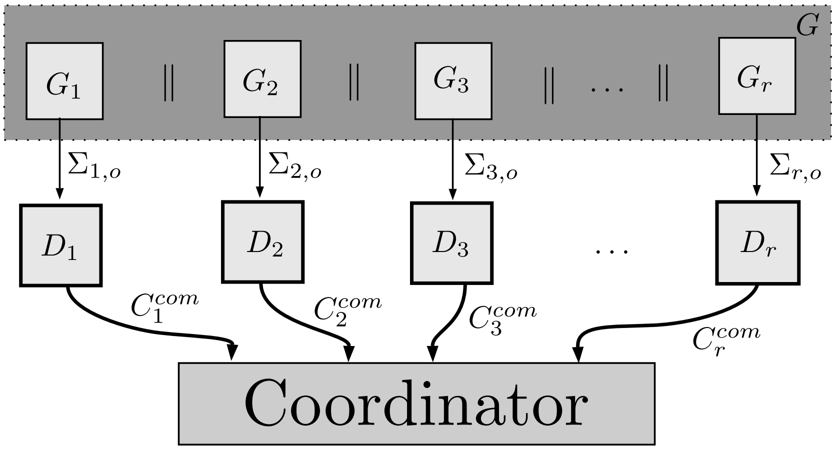


Figure 18 – Decentralized Synchronous Diagnosis scheme.

unobservable events of G are denoted as $\Sigma_o = \cup_{i=1}^{r} \Sigma_{i,o}$ and $\Sigma_{uo} = \Sigma \setminus \Sigma_o$, respectively. $\Sigma_f \subseteq \Sigma_{uo}$ denotes the fault event set.

In the DSDC framework, when an event $\sigma_o \in \Sigma_{i,o}$ is observed by the local measurement site LM_i, this information is sent to the local state estimator D_i, which updates the current state estimate of G_{N_i}. Then, D_i sends to the coordinator a subautomaton of G_{N_i}, referred in this work as a cluster automaton C_i^{com}, composed of the states of the last state estimate for which σ_o is feasible, the states of the state estimate reached after the occurrence of σ_o, and all of its unobservable transitions.

After a cluster C_i^{com} is communicated to the coordinator, a composition between all local clusters is built by the coordinator in order to keep tracking the possible synchronizations of unobservable events. This is necessary to prevent the diagnosis scheme to consider fault-free traces that cannot be generated by the global plant model. These exceeding fault-free traces, caused by the lost of synchronization of unobservable events, cause the growth of the fault-free language for the DSD scheme, L_{N_a}, presented in (CABRAL; MOREIRA, 2020).

In this work, the communication between the local measurement sites LM_i and local state estimators D_i, and between local state estimators D_i and the coordinator are assumed to be ideal, *i.e.*, there are no communication delays and/or package losses. The DSDC scheme is depicted in Figure 18.

4.2 PROBLEM FORMULATION

In order to correctly synchronize common unobservable events, we propose the DSDC scheme, where the local state estimators D_i send clusters C_i^{com} of G_{N_i} to the coordinator $\mathcal{C}$ that can select the correct traces of G_{N_i} according to the observed trace generated by the system. To do so, a communication protocol between the local state estimators D_i and coordinator $\mathcal{C}$, and a fault diagnosis procedure that indicates if a fault has occurred after the observation of a trace are proposed. Before these algorithms are presented, it is first necessary to introduce the concepts of cluster automaton and natural cluster of an automaton.

Definition 4.1 (Cluster automaton). *A cluster $C = (Q_C, \Sigma_C, f_C, \emptyset)$ is an automaton that has no initial states.* □

Given a subset of states of an automaton $G = (Q, \Sigma, f, q_0)$, $E \subseteq Q$, and an observable event $\sigma_o \in \Sigma_o$, we define set $B \subseteq E$ composed of the states of E such that σ_o is feasible:

$$B = \{q \in E : f(q, \sigma_o)!\}. \tag{3}$$

Then, the following definition of natural cluster of an automaton can be stated.

Definition 4.2 (Natural cluster automaton). *Let $G = (Q, \Sigma, f, q_0)$. Let $E \subseteq Q$ be a subset of states of G and consider an observable event $\sigma_o \in \Sigma_o$. The natural cluster automaton $NC(E, \sigma_o) = (Q_C, \Sigma_C, f_C, \emptyset)$ is a subautomaton of $G = (Q, \Sigma, f, q_0)$ and is defined as:*

- $Q_C = B \cup (\cup_{q \in E} UR(f(q, \sigma_o)))$;
- $\Sigma_C = \{\sigma_o\} \cup \Sigma_u$;
- $f_C : Q_C \times \Sigma_C \to Q_C$, *where*

$$f_C(q, \sigma) = \begin{cases} f(q, \sigma_o) & \textit{if } (q \in B) \wedge (\sigma = \sigma_o); \\ f(q, \sigma_u) & \textit{if } (q \in Q_C \setminus B) \wedge (\sigma_u \in \Sigma_u); \\ \textit{undefined, otherwise.} & \end{cases}$$

□

The natural cluster $NC(E, \sigma_o)$ of G defined in Definition 4.2 corresponds to the states $q \in E$ such that σ_o is feasible, the transitions (q, σ_o, q'), for $q \in E$, the unobservable reach of states q', and the unobservable transitions between the elements of $UR(q')$. The following example illustrates the natural cluster introduced in Definition 4.2.

Example 7. *Consider automaton $G = (Q, \Sigma, f, q_0)$ of Figure 19, where its set of observable and unobservable events are $\Sigma_o = \{a, b\}$ and $\Sigma_{uo} = \{\sigma_1\}$, respectively. Let $E = \{1, 3\}$, where $E \subseteq Q$. The natural cluster $NC(\{1, 3\}, b)$ is shown in Figure 20. Notice that since event b is feasible in state 1, i.e., $b \in \Gamma_G(1)$, state 1 belong to $NC(\{1, 3\}, b)$.* □

The communication protocol of the DSDC method is given by Algorithm 3. Algorithm 3 is initialized in lines 1-5, where each local state estimator D_i, for $i = 1, \ldots, r$, sends a subautomaton of G_{N_i}, $S_{0,i}$, corresponding to the states in the unobservable reach of its initial state and their related unobservable transitions. Then, in lines 6-16 the communication procedure is detailed when an event is observed. If an observed event $\sigma_o \in \Sigma_{i,o}$

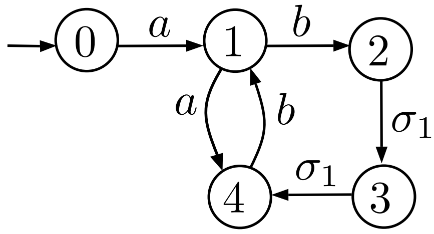


Figure 19 – Automaton G of Example 7.

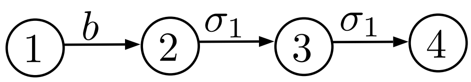


Figure 20 – Cluster $C(\{1, 3\}, b)$ of Example 7.

Algorithm 3 DSDC communication protocol

Input: $G_{N_i} = (Q_{N_i}, \Sigma_i \setminus \Sigma_f, f_{N_i}, q_{0,i})$, for $i \in \{1, \dots, r\}$.

1: **for** $i = 1, \dots, r$ **do**
2:     Compute $E_i \leftarrow UR(q_{0,i})$
3:     Compute the subautomaton $S_{0,i} = (E_i, \Sigma_{i,u}, f_{S_i}, q_{0,i})$ of G_{N_i}, where $f_{S_i}(q, \sigma) = f_{N_i}(q, \sigma)$ for $q \in E_i$ and $\sigma \in \Sigma_{i,uo}$
4:     Send $S_{0,i}$ to the Coordinator
5: **end for**
6: Wait for the observation of an event $\sigma_o \in \Sigma_{i,o}$ generated by the system
7: **for each** G_{N_i} such that $\sigma_o \in \Sigma_{i,o}$ **do**
8:     **if** $\sigma_o \in \Gamma_{G_{N_i}}(E_i)$ **then**
9:         Compute cluster $C_i^{com} \leftarrow NC_i(E_i, \sigma_o)$ and send it to the Coordinator
10:         Compute $E_i' \leftarrow \cup_{q_i \in E_i} UR(f(q_i, \sigma_o))$
11:         Update $E_i \leftarrow E_i'$
12:     **else**
13:         Send $C_i^{com} \leftarrow \emptyset$ to the Coordinator
14:     **end if**
15: **end for**
16: Return to line 6

---

is feasible in the current state estimate of D_i, E_i, the natural cluster $NC_i(E_i, \sigma_o)$ is computed and it is communicated to the coordinator $\mathcal{C}$. Otherwise, $C_i^{com} = \emptyset$ is communicated to the coordinator $\mathcal{C}$, which uses this information to directly diagnose the fault event occurrence.

It is important to remark that the computation of $NC_i(E_i, \sigma_o)$ that is required in Line 9 of Algorithm 3 according to Definition 4.2 can be done in linear time by using any graph search algorithm (CORMEN et al., 2009).

4.3 DIAGNOSIS PROCEDURE

In the DSDC scheme, a coordinator $\mathcal{C}$ needs to compute a composition between r clusters each time a new event σ_o is observed in order to verify if σ_o is feasible according to the fault-free behavior of the system. According to Algorithm 3, after an initialization, the coordinator $\mathcal{C}$ receives a new cluster $C_i^{com} = NC_i(E_i, \sigma_o)$ from all local state estimators D_i, such that $\sigma_o \in \Sigma_{i,o}$, each time an event σ_o is observed by LM_i. Each new cluster is incorporated in a previous stored one using a cluster union operation, defined in the following[1]. The definition of the cluster synchronous composition is also presented in the sequel.

Definition 4.3 (Cluster union). *Let $C_1 = (Q_1, \Sigma_1, f_1, \emptyset)$ and $C_2 = (Q_2, \Sigma_2, f_2, \emptyset)$ be two clusters. The union operation of C_1 and C_2, is defined as $C = C_1 \sqcup C_2 = (Q, \Sigma, f, \emptyset)$, where*

- $Q = Q_1 \cup Q_2$;
- $\Sigma = \Sigma_1 \cup \Sigma_2$;
- $f : Q \times \Sigma \to Q$, *where*

$$f(q, \sigma) = \begin{cases} f_1(q_1, \sigma_1) = q_1' & \text{if } q_1, q_1' \in Q_1 \text{ and } \sigma_1 \in \Sigma_1; \\ f_2(q_2, \sigma_2) = q_2' & \text{if } q_2, q_2' \in Q_2 \text{ and } \sigma_2 \in \Sigma_2; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Definition 4.4 (Cluster synchronous composition). *Let $C_1 = (Q_1, \Sigma_1, f_1, \emptyset)$ and $C_2 = (Q_2, \Sigma_2, f_2, \emptyset)$ be two cluster automata. The synchronous composition between C_1 and C_2 is defined as $sync((C_1, C_2), R) = Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, f, R)$, where $R \subseteq Q_1 \times Q_2$ is a set of initial states and*

$$f((q_1, q_2), \sigma) = \begin{cases} (f_1(q_1, \sigma), f_2(q_2, \sigma)) & \text{if } \sigma \in \Gamma_{C_1}(q_1) \cap \Gamma_{C_2}(q_2); \\ (f_1(q_1, \sigma), q_2) & \text{if } \sigma \in \Gamma_{C_1}(q_1) \setminus \Sigma_2; \\ (q_1, f_2(q_2, \sigma)) & \text{if } \sigma \in \Gamma_{C_2}(q_2) \setminus \Sigma_1; \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Note that Definition 4.4 is equivalent to the parallel composition between automata using R as the set of the initial states. The coordinator $\mathcal{C}$ uses the unobservable part of the communicated clusters to correctly follow the fault-free global behavior of the system by syncing the clusters of all system components after the observation of an event. In the following, the coordinator $\mathcal{C}$, that computes the fault diagnostic status, for the DSDC scheme proposed in this work is formally presented in Algorithm 4. Algorithm 4 runs together with Algorithm 3.

The idea of Algorithm 4 is to verify if the observation of an event is indeed feasible in the fault-free behavior of the system, considering the synchronization of unobservable events. If the occurrence of an observable event is not possible in the fault-free system behavior, the algorithm informs the occurrence of the fault event and stops. To do so,

[1] The cluster union operation is inspired in the graph union operation presented in Harary (1969).

Algorithm 4 Diagnosis procedure

Input: $\Sigma_i = \Sigma_{i,o} \cup \Sigma_{i,uo}$ for $i = 1, \ldots, r$, $\Sigma_o = \cup_{i=1}^{r} \Sigma_{i,o}$, subautomata $S_{0,i}$, and communicated clusters C_i^{com}, for $i = 1, \ldots, r$.
Output: Diagnosis decision.

1: Define $R \leftarrow \emptyset$ and $I \leftarrow \emptyset$
2: Define $C_i' \leftarrow \emptyset$, for $i = 1, \ldots, r$
3: After receiving subautomata $S_{0,i}$, $i = 1, \ldots, r$, compute $S = \|_{i=1}^{r} S_{0,i}$
4: Assign to R the initial state of S
5: For $S_{0,i} = (E_i, \Sigma_{i,u}, f_{S_i}, q_{0,i})$, define $C_i \leftarrow (E_i, \Sigma_{i,u}, f_{S_i}, \emptyset)$, $i = 1, \ldots, r$
6: Wait for the observation of an event $\sigma_o \in \Sigma_{i,o}$
7: **for each** C_i^{com} communicated from D_i **do**
8:     **if** $C_i^{com} = \emptyset$ **then**
9:         Inform the occurrence of the fault event and Stop
10:     **else**
11:         Define $C_i \leftarrow C_i \sqcup C_i^{com} = (Q_{C_i}, \Sigma_i, f_{C_i}, \emptyset)$
12:         Define $I \leftarrow I \cup \{i\}$
13:     **end if**
14: **end for**
15: Compute $S \leftarrow sync((C_1, C_2, \ldots, C_r), R) = (Q_S, \Sigma_N, f_S, R)$
16: **if** $f_S(q, \sigma_o)$ is undefined for all $q \in Q_S$ **then**
17:     Inform the occurrence of the fault event and Stop
18: **else**
19:     Set $R \leftarrow \{q' \in Q_S : f_S(q, \sigma_o) = q'\}$, for all $q, q' \in Q_S$
20:     **for each** $i \in I$ **do**
21:         Compute $C_i' = (Q_i', \Sigma_i, f_i', \emptyset)$ by removing from C_i all states and transitions that are not reachable after σ_o
22:         Define $C_i \leftarrow C_i'$
23:     **end for**
24: **end if**
25: Define $I \leftarrow \emptyset$
26: Return to line 6

Algorithm 4 is initialized in lines 1-5 by assigning set $\emptyset$ to R, I, and C_i'. In line 3, $S_{0,i}$, communicated in the initialization of Algorithm 3, are used to compute automaton S. The initial state of S is assigned to variable R in line 4. Clusters C_i are then initialized using automata $S_{0,i}$ in line 5.

From lines 6 to 26, the main cycle of Algorithm 4 is carried out when a new event is observed by at least one local state estimator D_i that communicates a new cluster to the coordinator. If an observable event σ_o is common to more than one system component and it occurs, each corresponding local state estimator D_i will communicate a new natural cluster C_i^{com}. For each C_i^{com} communicated, clusters C_i are updated in lines 7-14. If the empty automaton is communicated, then the fault is diagnosed, and the algorithm stops, as indicated in lines 8 and 9 of Algorithm 4. Otherwise, C_i is updated in line 11 to consider the behavior of the communicated cluster from D_i, C_i^{com}, using the cluster union operation. Set I is then updated to record the indexes of local state estimators that have communicated in this run due to the observation of σ_o.

In line 15 of Algorithm 4, automaton S is computed by the cluster synchronous composition considering R as the set of initial states. If there is no observable transitions labeled with σ_o in S, then the occurrence of σ_o is not feasible in the fault-free system behavior. Therefore, the fault is diagnosed and the coordinator informs the occurrence of the fault event and stops. Otherwise, in line 19, a new set of initial states R is computed, formed by all states of S that are immediately reached by an observable transition labeled with σ_o, in order to be considered after a new event observation. In lines 21 and 22, the clusters C_i are updated to consider only the possible unobservable behavior after the occurrence of σ_o. Finally, the set of indexes I is defined as the empty set in line 25, and Algorithm 4 waits for a new event observation by D_i.

The following theorem guarantees that the fault-free language considered in the method presented in this work is equal to the observable fault-free language $P_o(L_N)$ of the system.

Theorem 4.1. *Consider a system G whose generated language is L and the fault-free language is L_N. The observable fault-free language considered in the DSDC scheme is equal to the observable fault-free language of the system $P_o(L_N)$, $P_o : \Sigma^\star \to \Sigma_o^\star$.*

Proof. The proof is done by induction with respect to the size of an observed trace s for the state estimate provided by the DSDC scheme and the state estimate of the fault-free automaton model of the system G_N.

- $\|s\| = 0$:
  For $\|s\| = 0$, algorithms 3 and 4 are initialized. This process is done by the communication of the subautomata $S_{0,i}$, $i = 1, \dots, r$, computed in Algorithm 3 to the coordinator $\mathcal{C}$, which is carried out in lines 1-4 of Algorithm 3. Note that the state set of $S_{0,i}$ is E_i which is equal to $UR(q_{0,i})$, where $q_{0,i}$ is the initial state of G_{N_i}. In addition, the transitions of $S_{0,i}$ correspond to the unobservable transitions of G_{N_i} related to the states belonging to E_i. After Algorithm 4 receives subautomata $S_{0,i}$, automaton $S = \|_{i=1}^{r} S_{0,i}$ is computed in line 3. Since $S = \|_{i=1}^{r} S_{0,i}$, the states of S correspond to the initial state estimate of automaton $G_N = \|_{i=1}^{r} G_{N_i}$ for the observed sequence ε.

- $\|s\| = n$, $n \in \mathbb{N}$:
  Suppose now that a trace $s = s'\sigma_o$, such that $\|s\| = n$, has been observed by the DSDC scheme. Consider automaton $S = (Q_S, \Sigma_N, f_S, R)$, computed in line 15 of Algorithm 4. For all $q \in Q_S$, such that $q' = f_S(q, \sigma_o)$, $UR(q')$ correspond to the state estimate of automaton G_N after the observation of trace s.

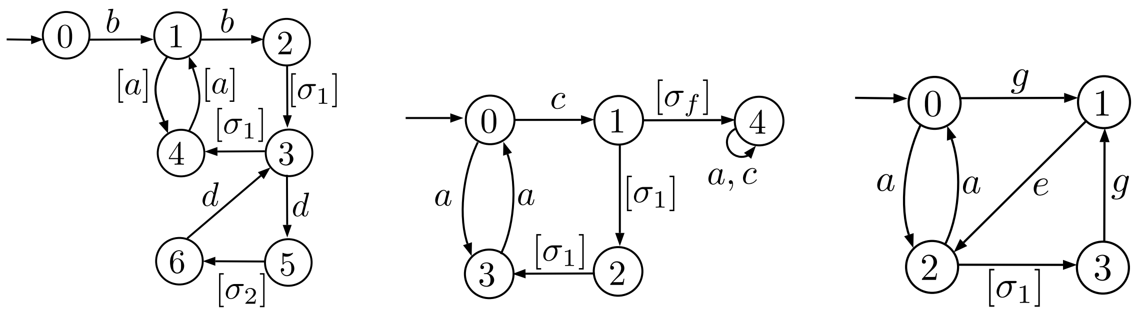
- $\|s\| = n + 1$, $n \in \mathbb{N}$:
  Let us suppose that $s = s'\sigma_o\sigma_o'$, such that $\|s\| = n + 1$. Automaton S computed in

the previous step, *i.e.*, after the observation of trace $s'\sigma_o$, contains the correct state estimate of G_N after the observation of σ_o. Notice that, after obtaining automaton S, set R is computed in line 19 of Algorithm 4 and it stores all states of S reached after event σ_o. In the sequence, for each local component i such that $\sigma_o \in \Sigma_{i,o}$, clusters C_i are updated in lines 21 and 22 in order to consider only the unobservable behavior after σ_o. It is important noticing that, according to the previous step, if the operation $sync((C_1, C_2, \dots, C_r), R)$ is performed, the result is equal to an automaton that corresponds to the state estimate of G_N after the observed trace $s'\sigma_o$.

After the observation of σ'_o, clusters C_i^{com} are communicated to the coordinator $\mathcal{C}$, where $\sigma'_o \in \Sigma_{i,o}$. These clusters correspond to all possible occurrences of σ'_o at the current state estimate of G_{N_i} and all their unobservable continuations. In line 11 of Algorithm 4, the cluster union operation is performed, which guarantees that clusters C_i are updated to also consider the information available in C_i^{com}. Then, in line 15, a new automaton S is built using clusters C_i and the set of initial states R, computed after the observation of $s'\sigma_o$. Since the *sync* operation is equivalent to the parallel composition considering all elements of R as the initial state of the resulting automaton, the unobservable reach of the states of automaton S reached after σ'_o correspond to the state estimate of G_N after the observed trace s. This result is achieved since the communicated clusters correspond to the unobservable reach of automata G_{N_i} after the observation of s, which concludes the proof. ■

In the following, an example is presented to illustrate the use of Algorithm 4 for the Decentralized Synchronous Diagnosis with coordination process.

Example 8. *Consider the system $G = \|_{i=1}^{3} G_i$, where G_1, G_2 and G_3 are depicted in Figure 21, automata G is shown in Figure 22 and the fault-free model G_N is illustrated in Figure 23. The sets of events of G_1, G_2 and G_3 are, respectively, $\Sigma_1 = \Sigma_{1,o} \cup \Sigma_{1,uo} = \{a, b, d, \sigma_1, \sigma_2\}$, $\Sigma_2 = \Sigma_{2,o} \cup \Sigma_{2,uo} = \{a, c, \sigma_1, \sigma_f\}$ and $\Sigma_3 = \Sigma_{3,o} \cup \Sigma_{3,uo} = \{a, e, g, \sigma_1\}$, where $\Sigma_{1,o} = \{b, d\}$, $\Sigma_{2,o} = \{a, c\}$ and $\Sigma_{3,o} = \{a, e, g\}$ are the sets of observable events of G_1, G_2 and G_3, and $\Sigma_{1,uo} = \{a, \sigma_1, \sigma_2\}$, $\Sigma_{2,uo} = \{\sigma_1, \sigma_f\}$ and $\Sigma_{3,uo} = \{\sigma_1\}$ are the sets of unobservable events of G_1, G_2 and G_3. The set of observable events of G is $\Sigma_o = \Sigma_{1,o} \cup \Sigma_{2,o} \cup \Sigma_{3,o} = \{a, b, c, d, e, g\}$ and the set of fault events is $\Sigma_f = \{\sigma_f\}$.*



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

Figure 21 – Automata G_1, G_2 and G_3 of Example 8.

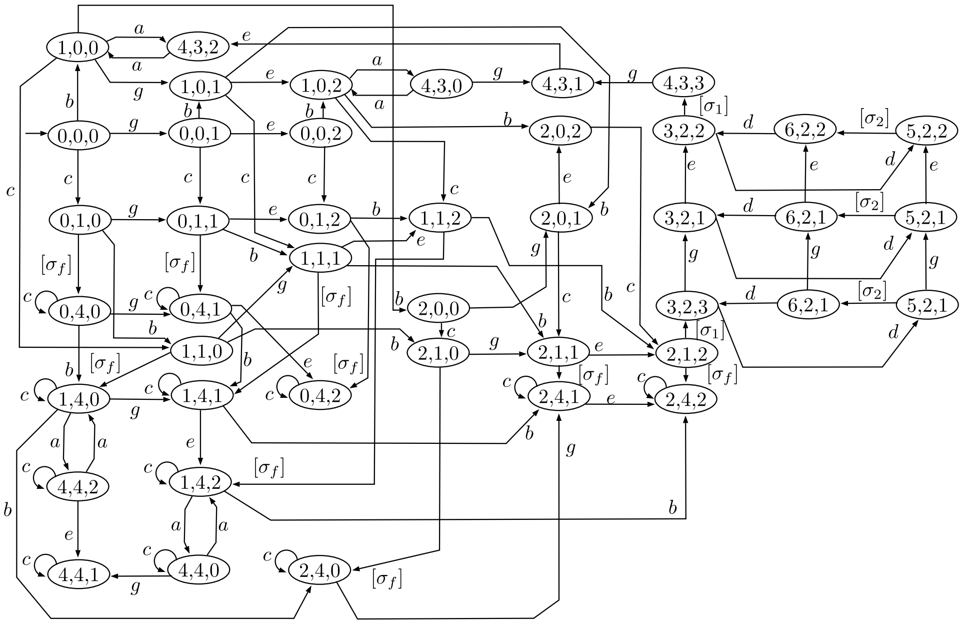


Figure 22 – Automaton G of Example 8.

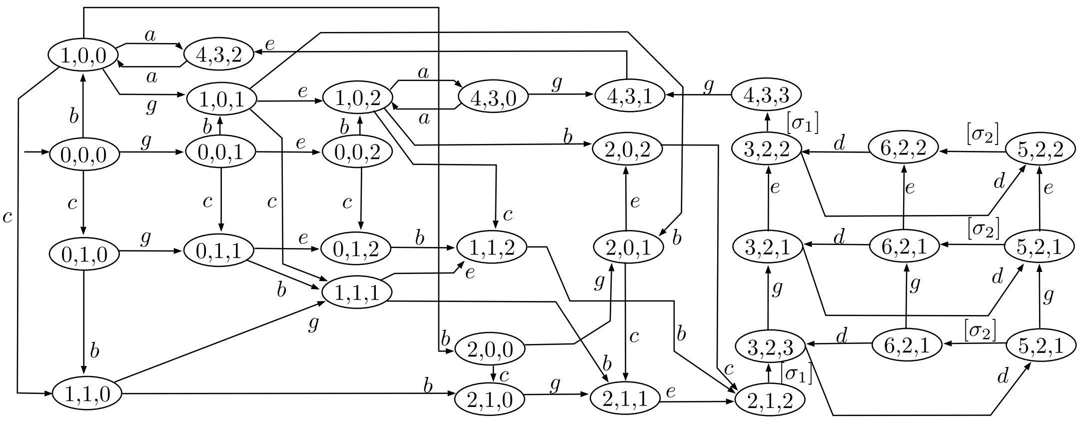
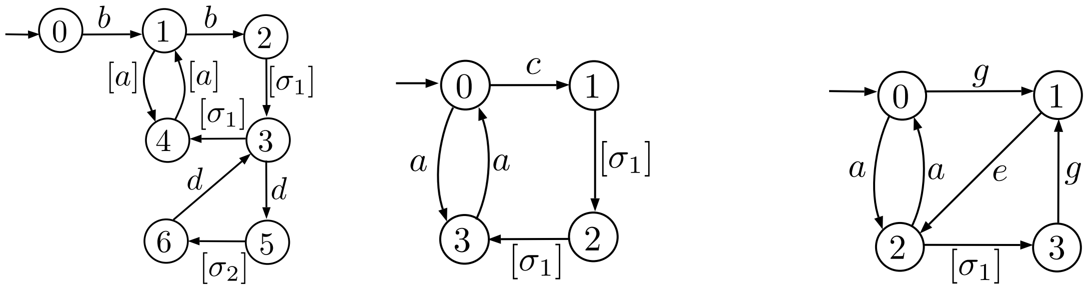


Figure 23 – Automaton G_N of Example 8.

Suppose that the system G has generated the fault trace $s_f = bc\sigma_f a^n$, $n \in \mathbb{N}$, *which its local projections are* $P_{1,o}(s_f) = b$, $P_{2,o}(s_f) = ca^n$ *and* $P_{3,o}(s_f) = a^n$, *where*

$P_{i,o} : \Sigma^\star \to \Sigma_{i,o}^\star$, $i \in \{1, 2, 3\}$, *are projections. It is important to notice that this system is not synchronously codiagnosable since the local projections of the fault trace can be observed in the fault-free component models* G_{N_1}, G_{N_2}, *and* G_{N_3}, *presented in Figure 24. The DSDC scheme is illustrated by using Algorithms 3 and 4 after each observed event.*

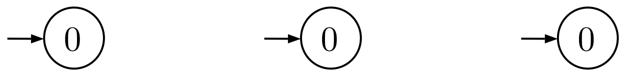


(a) G_{N_1} (b) G_{N_2} (c) G_{N_3}

Figure 24 – Automata G_{N_1}, G_{N_2} and G_{N_3} of Example 8.

- ***Observed trace*** ε

  Before any event is observed, Algorithm 3 sends $S_{0,1}$, $S_{0,2}$, *and* $S_{0,3}$, *shown in Figure 25, to the Coordinator. Then* $S = S_{0,1} \| S_{0,2} \| S_{0,3}$, *shown in Figure 26, is computed by Algorithm 4 that stores the initial state of* S *in* $R = \{(0, 0, 0)\}$.



(a) $S_{0,1}$ (b) $S_{0,2}$ (c) $S_{0,3}$

Figure 25 – Subautomata $S_{0,1}$ (a), $S_{0,2}$ (b), and $S_{0,3}$ (c) of Example 8.

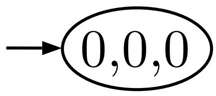


Figure 26 – Initial composition $S = S_{0,1} \| S_{0,2} \| S_{0,3}$ of Example 8.

At this stage, Algorithms 3 and 4 wait for a new observed event by D_1, D_2 *or* D_3.

- ***Observed trace b***

  When event b *occurs, it is observed by* D_1, *and* C_1^{com}, *depicted in Figure 27, is communicated.*

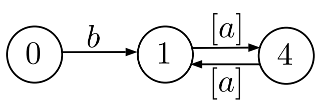
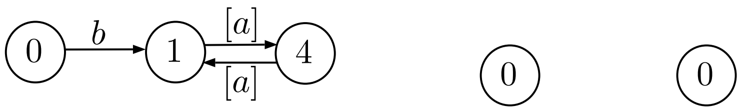


Figure 27 – Cluster automaton C_1^{com} after observation of event b.

In Algorithm 4, the union operation of C_1 *and* C_1^{com} *is computed in line 11, and the result is assigned to* C_1, *depicted in Figure 28 (a). Notice that, in this case,* C_1 *is* *equal to C_1^{com}. Since only local state estimator D_1 observes event b, set I is updated to $I = \{1\}$. Clusters C_2 and C_3 are not modified as shown in Figure 28 (b) and 28 (c), respectively.*



(a) C_1 (b) C_2 (c) C_3

Figure 28 – Cluster automata C_1, C_2, and C_3 after observation of event b.

A new S is computed, according to line 15 of Algorithm 4 and it is depicted in Figure 29. Since a transition labeled with b, namely $((0,0,0), b, (1,0,0))$, exists in S of Figure 29, the fault event is not detected and set R is updated to the states of S reached by transitions labeled with b, i.e., $R = \{(1,0,0)\}$ in line 19 of Algorithm 4.

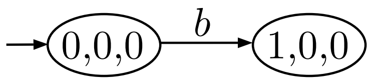


Figure 29 – S after observation of event b.

After that, C_1', shown in Figure 30, is computed and it is used to update C_1, according to lines 21 and 22 of Algorithm 4 by removing the states and transitions that are not reachable after event b. Algorithm 4 then updates set $I = \emptyset$ and waits for a new observed event.

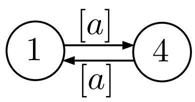


Figure 30 – Cluster automaton $C_1 = C_1'$ after observation of event b.

- ***Observed trace bc***

  The next generated event is c, observed by state estimator D_2 that sends C_2^{com}, depicted in Figure 31. Then, the cluster automaton C_2 shown in Figure 32 (b) is updated and, in this case, it is equal to C_2^{com}. Notice that cluster automata C_1 and C_3, illustrated in Figure 32 (a) and 32 (c), respectively, do not change.

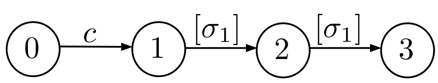
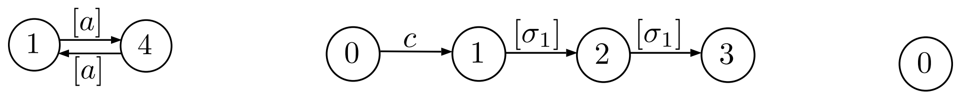


Figure 31 – Cluster automaton C_2^{com} after observation of trace bc.



(a) C_1 (b) C_2 (c) C_3

Figure 32 – Cluster automata C_1, C_2, and C_3 after observation of trace bc.

Set I is updated to $I = \{2\}$ and a new S is computed using $R = \{(1,0,0)\}$ as the initial state, as shown in Figure 33.

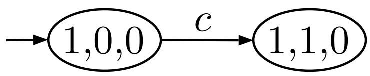


Figure 33 – S after observation of trace bc.

After that, set R is updated to $R = \{(1,1,0)\}$ and a new C_2 is computed, as presented in Figure 34.

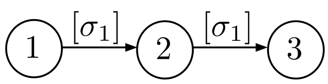


Figure 34 – Cluster automaton $C_2 = C_2'$ after observation of trace bc.

- ***Observed trace bca***

When event a occurs, state estimators D_2 and D_3 send clusters C_2^{com} and C_3^{com}, depicted in Figures 35 and 36, respectively.

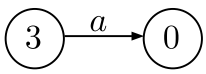


Figure 35 – Cluster automaton C_2^{com} after observation of trace bca.

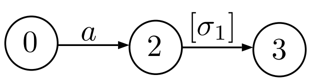
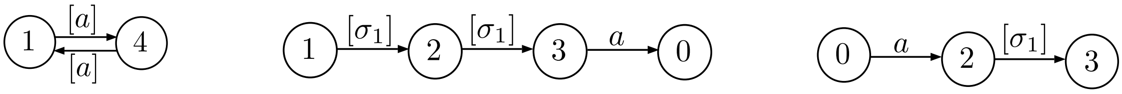


Figure 36 – Cluster automaton C_3^{com} after observation of trace bca.

Clusters C_2 and C_3, illustrated in Figure 37(b) and 37(c), are updated, and C_3, in this case, is equal to C_3^{com}, shown in Figure 36. Notice that cluster C_1, shown in Figure 37(a), does not change.



(a) C_1 (b) C_2 (c) C_3

Figure 37 – Cluster automata C_1, C_2, and C_3 after observation of trace bca.

Set I is then computed, and it is equal to $I = \{2, 3\}$. Notice that since $R = \{(1, 1, 0)\}$, which corresponds to the initial state of S, and events a and σ_1 are common to all three components but are not feasible in all three states, (1,1,0), no transitions leave state $(1, 1, 0)$ of S. Thus, S is equal to a graph composed of only state $(1, 1, 0)$, shown in Figure 38, and since there are no transitions in S labeled with event a, the fault event occurrence is detected in line 17 of Algorithm 4.

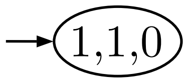


Figure 38 – S after observation of trace *bca*.

Remark 4.1. *It is important to remark that the method presented in this work can be implemented even in the case where there are events that are observable to one component and unobservable to other components, as it is the case in Example 8.*

4.4 COMPUTATIONAL COMPLEXITY ANALYSIS

Since mostly operations of Algorithms 3 and 4 have linear computational complexity and are performed on automata G_{N_i}, $i = 1, \ldots, r$, the computational complexity of the DSDC scheme can be analyzed according to the construction of S in line 15 of Algorithm 4. To do so, let Q_i denote the set of states of G_i, $i = 1, \ldots, r$, and suppose that all G_i have the same number of states $|Q_i|$. Due to the cluster synchronous composition performed in line 15 of Algorithm 4, automaton S can have, in the worst case scenario, $|Q_i|^r$ states and $|Q_i|^r \times (|\Sigma_{uo}| + 1)$ transitions. Thus, automaton S can grow exponentially with the number of system components.

However, this complexity order is achieved only if the majority of transitions of the component models are labeled with unobservable events since the communicated clusters C_i^{com} are composed of states and unobservable transitions reached after an observed event. Thus, the DSDC method performs better than fault diagnosis techniques based on the global system model, as it is the case in Example 8, where G_N has 31 states, while the sum of states of G_{N_1}, G_{N_2} and G_{N_3} is equal to 15 and automaton S has, in the worst case, two states and one transition.

It is important to remark that one of the tendencies of Industry 4.0 is the increasing number of sensors in order to achieve more information of the system. Therefore, in practice, the number of unobservable events tend to be much smaller than the number of observable events, which also contributes to a better performance of the method presented in this work. This result can be seen in the next chapter, where a practical application of the DSDC method is presented.

4.5 FINAL REMARKS

In this chapter, the decentralized synchronous diagnosis with coordination (DSDC) scheme has been presented. The DSDC method is based on the implementation of state estimators computed from the fault-free component models of the system components. As in Cabral and Moreira (2020), if an event is observed and it is not feasible in the current state estimate of all state estimators, the fault event occurrence is diagnosed. Otherwise, the local state estimators that observed the event send clusters composed of the state estimates and their related unobservable transitions to a coordinator. This coordinator verifies if the observed event is indeed possible to occur in the synchronization of all local clusters. If the answer is negative, the fault event is diagnosed.

The method is formalized using two algorithms: (i) a communication protocol, that establishes when the cluster automata must be communicated from the state estimators to the coordinator; and (ii) the diagnosis procedure, that provides the diagnosis status after the observation of an event. Although, in the worst case scenario, the method is exponential with the number of the system components, it performs better than other methods based on the global system model, mainly for systems with a low number of unobservable events/transitions.

The DSDC scheme can be seen as a method to compute the state estimate of the global fault-free behavior model online, without the need to store the whole automaton model. Thus, it preserves the diagnosis power of the monolithic approach presented in Sampath et al. (1995) without the use of observers.

In the next chapter, the DSDC method is applied to a didactic manufacturing system.

5 DSDC METHOD APPLIED TO A MANUFACTURING SYSTEM

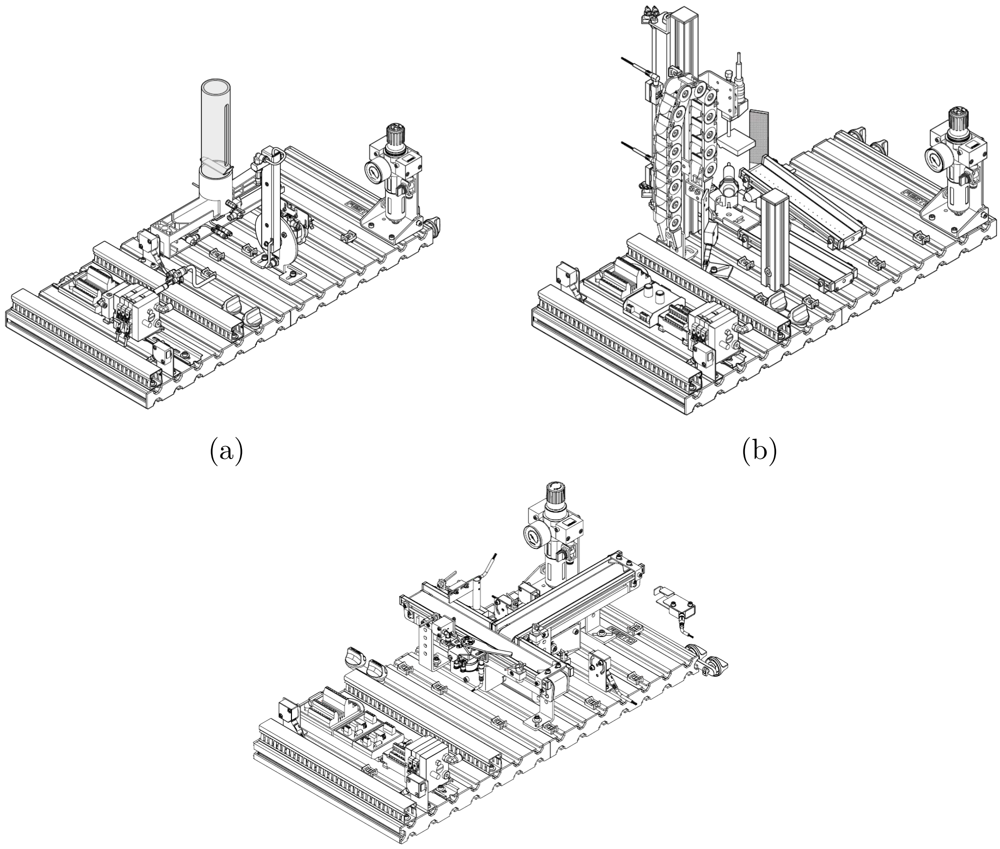
In this chapter, the DSDC method is applied to a didactic manufacturing system. The system has unobservable events in common to two components and it is previously known to be diagnosable according to Definition 3.2 and not synchronously codiagnosable according to Definition 3.4. Therefore, one can use the monolithic diagnosis approach to diagnosis the fault event, however, it leads to a higher computation cost.

This chapter is organized as follows. First, the controlled plant and the system model are presented and then the fault-free behavior system components necessary to run the DSDC method are shown. A fault trace that cannot be diagnosed using the synchronous decentralized diagnosis scheme is used to illustrate the DSDC method. This fault trace is successfully diagnosed using the DSDC approach. Finally, final remarks about this implementation are drawn at the end of the chapter.

5.1 CASE STUDY SYSTEM

The controlled plant is a workpiece assembly manufacturing system of the manufacturer FESTO (FESTO, 2006) installed at the Industrial Informatics and Automation Laboratory of the Federal University of Santa Catarina. This didactic manufacturing system consists of six stations: (*i*) the Distributing station that removes workpieces from the magazine and transfer them to the next station through a robotic arm with a suction cup; (*ii*) the Testing station that either reject a workpiece or make it available to the subsequent station according to its height; (*iii*) the Separating station that separates the workpiece according to its positioning; (*iv*) the Pick and Place station which positions a lid over the workpiece; (*v*) the Fluidic Muscle Press station presses the lid to lock it in the workpiece; and (*vi*) the Sorting Station sorts the workpieces according to its material and color. In this work, only the first three stations functioning on their operation cycle are considered. The system schematics is presented in Figure 39 and a top view picture of the real system is shown in Figure 40.

The first three stations are designed to select workpieces with the appropriated height and in the right position. The detailed behavior is presented as follows: In the Distributing station, a pneumatic cylinder pushes a workpiece out of the magazine. A robotic arm with a suction cup turns on the vacuum and delivers it to the Testing station. Then, the workpiece is allocated in an elevator that moves it to be tested according to its height. If the workpiece is not higher than the standard model, a pneumatic cylinder pushes it to a conveyor belt with air pockets and the workpiece is delivered to the Separating station. Otherwise, it is discarded. Then, in the Separating station, the workpiece is allocated under a sensor to evaluate if it is rightly positioned. If the answer is yes, the workpiece is transferred to the following station. Otherwise, it is discarded.



(a) (b)

(c)

Figure 39 – Stations considered for this case study: (a) Distributing station; (b) Testing station; and (c) Separating station.

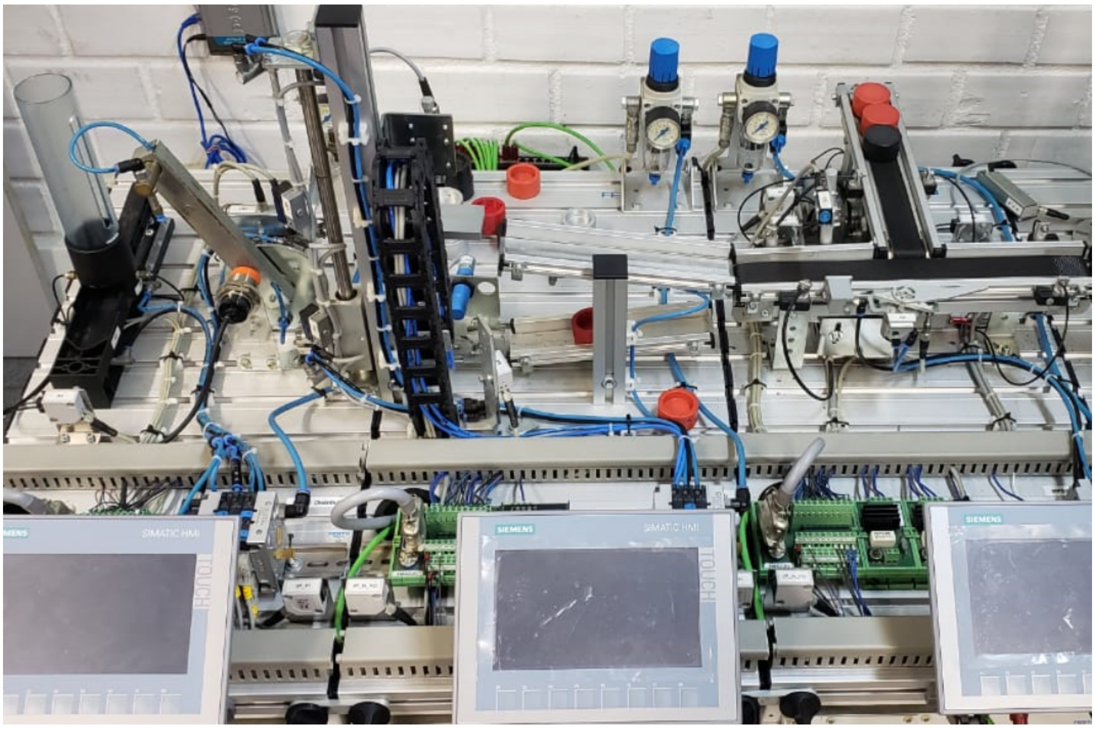
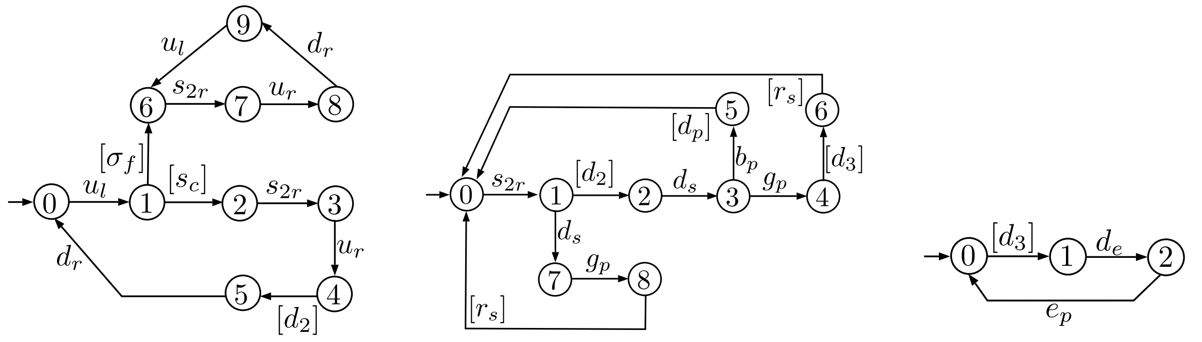
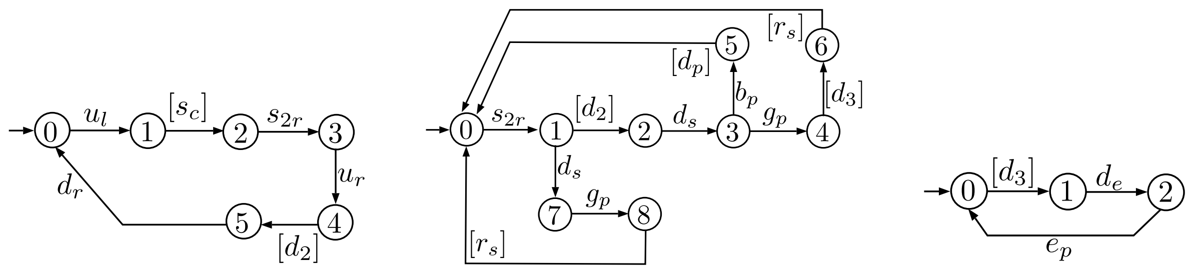


Figure 40 – Real didactic manufacturing system of Industrial Computing and Automation Laboratory.



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

Figure 41 – Automata models: (a) Distributing station - G_1, (b) Testing station - G_2 and (c) Separating station - G_3.



(a) G_{N_1} (b) G_{N_2} (c) G_{N_3}

Figure 42 – Fault-free behavior of the stations: (a) Distributing station - G_{N_1}, (b) Testing station - G_{N_2} and (c) Separating station - G_{N_3}.

5.2 SYSTEM MODEL

In order to implement the DSDC method, it is first necessary to model the plant according to its controlled behavior. It is considered that the global system is composed of three stations: (i) the Distributing station; (ii) the Testing station; and (iii) the Separating station. The automata models of the stations and their fault-free behaviors are illustrated in Figures 41 and 42, respectively. The global plant model, G, has 159 states and 369 transitions and its fault-free behavior model, G_N, has 96 states and 218 transitions, and are omitted due to their large number of states and transitions.

The initial state of automaton G_1 represents that the workpiece is ready to be transported. The robotic arm leaves the neutral position in order to grab the workpiece, modeled as event u_l, and G_1 reaches state 1. This means that the robotic arm is ready to turn on the suction cup. When the workpiece is grabbed by the suction cup, modeled as event s_c, the system evolves to state 2. If the Testing station is ready to receive the workpiece, the robotic arm moves towards the right position to deliver the workpiece, and event s_{2r} occurs and the system evolves to state 3. The occurrence of event u_r indicates the rising edge of the right position sensor of the robotic arm. This leads the system to reach state 4, where the robotic arm can deliver the workpiece to the Testing station.

After that, the suction cup is turned off, event d_2 occurs, and the system evolves to state 5. In this state, the workpiece is successfully delivered and the robotic arm can return to its neutral position when event d_r occurs, leading G_1 back to this initial state.

In this system, the fault event, represented by σ_f, models the suction cup malfunctioning. Therefore, the workpieces cannot be grabbed by the robotic arm and transported to the Testing station. It is important to notice that the robotic arm trajectory is not affected by the fault event and the system cannot recognize if the workpiece was successfully delivered. The fault behavior is modeled in the cycle formed by (6,s_{2r},7), (7,u_r,8), (8,d_r,9), and (9,u_l,6) transitions in G_1.

The initial state of automaton G_2 represents that the station has no workpiece in the elevator and it is waiting to receive a new workpiece. When event s_{2r} occurs, the station can receive a new workpiece and reaches state 1 meaning that it is ready to test it. The occurrence of event d_2 illustrates that the workpiece was delivered to the elevator and the system reaches state 2. When event d_s occurs, it means that the robotic arm has returned to a safe position and the elevator can rise to test the workpiece height. Then, the system evolves to state 3, which represents that the workpiece has been tested and can be either accepted or discarded. If the workpiece has an acceptable height, then the pneumatic cylinder pushes the workpiece to a conveyor belt with air pockets, modeled as event g_p. After that, the workpiece is in the conveyor belt and the elevator is in an upper position, illustrated by state 4. The successful deliver to the separating station is represented by the occurrence of event d_3. Then, the system reaches state 6, where there are no workpieces in the workstation and event r_s can occur returning the system to its initial state. If the workpiece has greater height than the accepted, it is discarded, which is modeled as event b_p. The system now evolves from state 3 to state 5, where the pneumatic cylinder pushes the workpiece to a discard ramp. When event d_p occurs, the workpiece was discarded and the system can return to its initial state. The cycle (0,s_{2r},1),(1,d_s,7),(7,g_p,8), and (8,r_s,0) represents the behavior of the station when the fault has occurred and the robotic arm of the previous station only attempts to deliver a workpiece. Since the system only tests if a piece has a greater height, when the first station fails to deliver a real workpiece, the Testing station considers that a good height workpiece was tested.

The automaton G_3 represents the Separating station and its essential functioning. The initial state illustrates that the station is ready to receive a workpiece. When event d_3 occurs, the system evolves to state 1, where the workpiece has been placed. If the station detects a workpiece, modeled as event d_e, the process starts leading the system to state 2. The occurrence of event e_p means that the workpiece was successfully verified and the system is available to receive another workpiece. The behavior model of the Separating station was simplified, since it does not affect the diagnosability status of the whole system.

The sets of events of G_1, G_2, and G_3 are $\Sigma_1 = \{u_l, s_c, s_{2r}, u_r, d_2, d_r, \sigma_f\}$, $\Sigma_2 =$

$\{s_{2r}, d_2, d_s, g_p, r_s, b_p, d_3, d_p\}$, and $\Sigma_3 = \{d_3, d_e, e_p\}$, respectively, where the sets of observable events of each component are $\Sigma_{1,o} = \{u_l, s_{2r}, u_r, d_r\}$, $\Sigma_{2,o} = \{s_{2r}, d_s, g_p, b_p\}$, and $\Sigma_{3,o} = \{d_e, e_p\}$. Thus, the set of events of $G = G_1 \| G_2 \| G_3$ is $\Sigma = \{u_l, s_c, s_{2r}, u_r, d_2, d_r, \sigma_f, d_s, g_p, r_s, b_p, d_3, d_p, d_e, e_p\}$, where the set of observable events of G is $\Sigma_o = \Sigma_{1,o} \cup \Sigma_{2,o} \cup \Sigma_{3,o} = \{u_l, s_{2r}, u_r, d_r, d_s, g_p, b_p, d_e, e_p\}$ and the set of unobservable events of G is $\Sigma_{uo} = \Sigma \setminus \Sigma_o$.

In the sequel, we present the list of states and events of automata G_1, G_2, and G_3.

State	Meaning
0	Workpiece is in the right position to be transported
1	Robotic arm is in the right position to turn on the suction cup
2	Workpiece has been grabbed by the robotic arm with the suction cup on
3	Robotic arm is moving towards the position to deliver the workpiece
4	Robotic arm is ready to deliver the workpiece to the next station
5	Workpiece was successfully delivered
6	Workpiece has not been grabbed by the robotic arm with the suction cup off
7	Robotic arm moving towards the delivering position without a workpiece
8	Robotic arm is at the delivering position without a workpiece
9	Robotic arm is returning to its neutral position

Table 1 – States of G_1.

State	Meaning
0	The station is waiting for a new workpiece
1	Station is ready to test another workpiece
2	Workpiece delivered to the elevator
3	Workpiece height tested by the sensor
4	Workpiece in the conveyor belt and elevator in upper position
5	Pneumatic cylinder pushed the workpiece off
6	Station with no workpiece waiting to return to its initial state
7	Phantom workpiece height tested by the sensor
8	Phantom workpiece in the conveyor belt and elevator in upper position

Table 2 – States of G_2.

State	Meaning
0	The station is waiting for a new workpiece
1	Workpiece is in the right position to be verified
2	Workpiece is detected by the sensor

Table 3 – States of G_3.

5.3 DSDC SCHEME APPLIED TO THE CASE STUDY

In this section, Algorithms 3 and 4 are applied to this manufacturing system in order to diagnose the fault event occurrence. Suppose that the system has generated

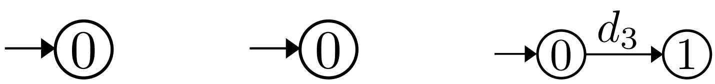
Event	Meaning
u_l	Robotic arm leaves the neutral position
s_c	Suction cup is turned on
s_{2r}	Testing station is ready to receive the workpiece
d_2	Workpiece is delivered to the elevator
u_r	Rising edge of the right position of the robotic arm
d_r	Falling edge of the right position of the robotic arm
σ_f	The suction cup fails
d_s	Robotic arm returns to a safe position
g_p	Pneumatic cylinder pushes the workpiece to a conveyor belt
d_3	Workpiece is delivered to separating station
r_s	Returns the system to its initial state
d_p	Workpiece is discarded
b_p	Pneumatic cylinder pushes the workpiece to a ramp
d_e	Workpiece is detected
e_p	Workpiece is verified

Table 4 – Events of G_1, G_2, and G_3.

the fault trace $s_f = u_l\sigma_f(s_{2r}u_r d_r d_s g_p u_l r_s)^n$, $n \in \mathbb{N}$, where its observation is $P_o(s_f) = u_l(s_{2r}u_r d_r d_s g_p u_l)^n$, for $P_o : \Sigma^\star \to \Sigma_o^\star$. It is important to notice that this system is not synchronously codiagnosable since the local projections of the fault trace can be observed in the fault-free component models presented in Figure 42.

- **Observed trace** ε

  At first, Algorithm 3 sends subautomata $S_{0,1}$, $S_{0,2}$, and $S_{0,3}$ depicted in Figure 43 to the Coordinator. After that, Algorithm 4 computes $S = S_{0,1}\|S_{0,2}\|S_{0,3}$ shown in Figure 44 and stores the initial state of S in $R = \{(0, 0, 0)\}$.



(a) $S_{0,1}$ (b) $S_{0,2}$ (c) $S_{0,3}$

Figure 43 – Subautomata $S_{0,1}$ (a), $S_{0,2}$ (b), and $S_{0,3}$ (c).

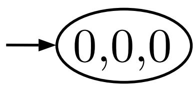


Figure 44 – Composition $S = S_{0,1}\|S_{0,2}\|S_{0,3}$.

- **Observed trace** u_l

  Algorithms 3 and 4 wait for a new event observation by the local state estimators. When event u_l occurs, it is observed by D_1 that communicates cluster C_1^{com} depicted in Figure 45. Since D_2 and D_3 do not observe any event, their clusters are equal to the ones sent before as Figure 46 illustrates.

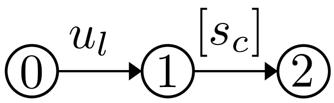
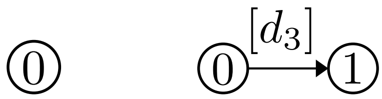


Figure 45 – C_1^{com} after observation of event u_l.



(a) C_2 (b) C_3

Figure 46 – C_2 and C_3 after observation of event u_l.

In Algorithm 4, the union operation of C_1 and C_1^{com} is computed in line 11, and the result is assigned to C_1 shown in Figure 47. Notice that, in this case, C_1 and C_1^{com} are the same clusters. Since only local state estimator D_1 observes event u_l, set I is updated to $I = \{1\}$.

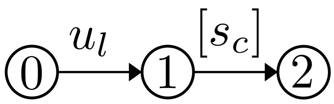


Figure 47 – C_1 after observation of event u_l.

Then, a new S is computed, according to line 15 of Algorithm 4 and it is depicted in Figure 48. Since a transition labeled with u_l exists in S of Figure 48, the fault is not detected and set R is updated to $R = \{(1, 0, 0)\}$.

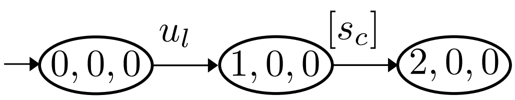


Figure 48 – S after observation of event u_l.

After that, only C_1 is updated from C_1' as shown in Figure 49, according to lines 21 and 22 of Algorithm 4. The states and transitions that are not reachable after event u_l are removed and the set I is updated to $I = \emptyset$. Then, the Algorithms 3 and 4 wait for a new event occurrence.

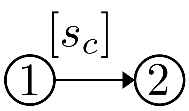


Figure 49 – C_1' after observation of event u_l.

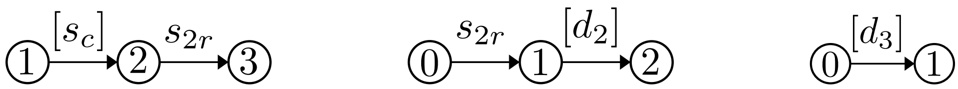
- **Observed trace** $u_l s_{2r}$

The next event, s_{2r}, is observed for local state estimators D_1 and is D_2. The cluster automata C_1^{com} and C_2^{com} illustrated in Figure 50 are sent to the diagnoser. Then,

C_1 and C_2 depicted in Figure 51 (a) and 51 (b), respectively, are updated after the union operation. Notice that C_1 and C_2 are equal to C_1^{com} and C_2^{com}, respectively. It is important to notice that the cluster automaton C_3 - shown in Figure 51 (c) - has not changed.

(a) C_1^{com} (b) C_2^{com}

Figure 50 – C_1^{com} and C_2^{com} after observation of trace $u_l s_{2r}$.



(a) C_1 (b) C_2 (c) C_3

Figure 51 – Cluster automata C_1, C_2, and C_3 after observation of trace $u_l s_{2r}$.

According to line 12 of Algorithm 4, set I is updated to $I = \{1, 2\}$ and a new S is computed and it is shown in Figure 52.

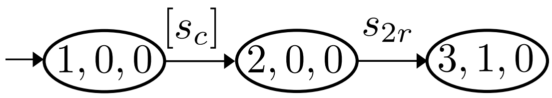


Figure 52 – S after observation of trace $u_l s_{2r}$.

After that, set R is updated to $R = \{(3, 1, 0)\}$ and new C_1 and C_2 are computed from C_1' and C_2', as presented in Figure 53 (a) and 53 (b). Then, Algorithm 4 updates set $I = \emptyset$.

(a) C_1' (b) C_2'

Figure 53 – C_1' and C_2' after observation of trace $u_l s_{2r}$.

- **Observed trace** $u_l s_{2r} u_r$

  The next generated event is u_r, and it is observed by D_1 which sends C_1^{com} to the diagnoser as illustrated in Figure 54. Then, C_1 is updated as shown in Figure 55 (a) and cluster automata C_2 and C_3 remain the same as depicted in Figure 55 (b) and 55 (c). In this case, C_1 is equal to C_1^{com}.

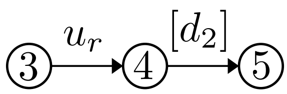
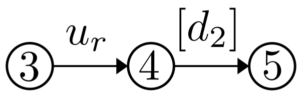


Figure 54 – C_1^{com} after observation of trace $u_l s_{2r} u_r$.



(a) C_1 (b) C_2 (c) C_3

Figure 55 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r$.

After that, set I is equal to $I = \{1\}$ and a new S is computed, as presented in Figure 56. Then, set R is updated to $R = \{(4,1,0)\}$.

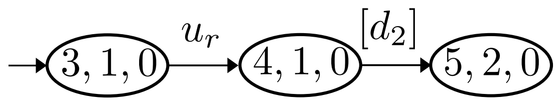
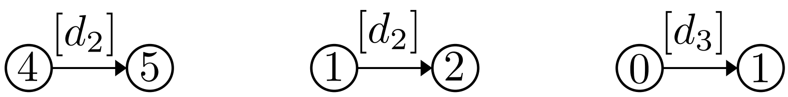


Figure 56 – S after observation of trace $u_l s_{2r} u_r$.

Cluster C_1 is updated as shown in Figure 57 (a) and cluster automata C_2 and C_3 are not modified as depicted in Figure 57 (b) and 57 (c). Then, set I is updated to $I = \emptyset$.



(a) C_1' (b) C_2 (c) C_3

Figure 57 – C_1', C_2, and C_3 after observation of trace $u_l s_{2r} u_r$.

- **Observed trace** $u_l s_{2r} u_r d_r$

  The occurrence of event d_r is observed by D_1 and C_1^{com} is sent to the Diagnoser as illustrated in Figure 58. After that, C_1 is updated according to line 11 of Algorithm 4 as presented in Figure 59. Clusters C_2 and C_3 are the same as shown in Figure 57 (b) and 57 (c).

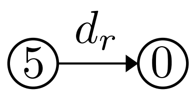


Figure 58 – C_1^{com} after observation of trace $u_l s_{2r} u_r d_r$.

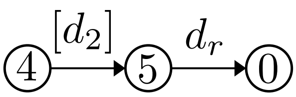


Figure 59 – C_1 after observation of trace $u_l s_{2r} u_r d_r$.

Then, set I is updated to $I = \{1\}$ and a new S is computed, according to line 15 of Algorithm 4 and it is presented in Figure 60. Since there is a transition labeled as d_r in S, set R is updated to $R = \{(0,2,0)\}$.

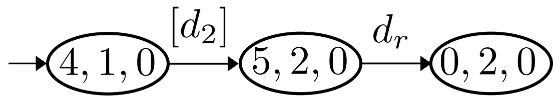
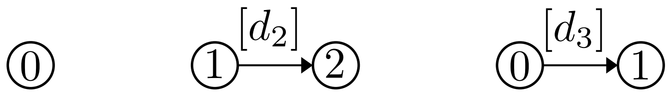


Figure 60 – S after observation of trace $u_l s_{2r} u_r d_r$.

After that, a new cluster automaton C_1 is computed from C_1' as shown in Figure 61 (a). C_2 and C_3 illustrated in Figure 61 (b) and 61 (c) do not change since there was not an event observation in D_2 or D_3. Also, Algorithm 4 updates set $I = \emptyset$.



(a) C_1' (b) C_2 (c) C_3

Figure 61 – C_1', C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r$.

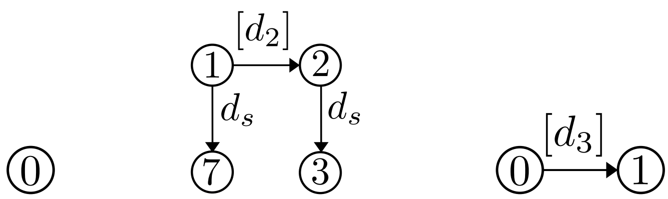
- **Observed trace** $u_l s_{2r} u_r d_r d_s$

  When event d_s occurs, local state estimator D_2 communicates two clusters C_2^{com} to the diagnoser according to line 8 of Algorithm 3 and it is presented in Figure 62 (a) and 62 (b).

(a) C_2^{com} (b) C_2^{com}

Figure 62 – C_2^{com} after observation of trace $u_l s_{2r} u_r d_r d_s$.

Then, C_2 is updated after the union operation and it is depicted in Figure 63 (a). C_1 and C_3 are not modified as illustrated in Figure 63 (b) and 63 (c).



(a) C_1 (b) C_2 (c) C_3

Figure 63 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s$.

Set I is updated to $I = \{2\}$ and a new S is computed from these clusters as shown in Figure 64.

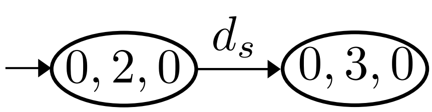
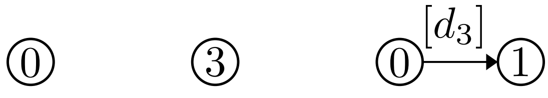


Figure 64 – S after observation of trace $u_l s_{2r} u_r d_r d_s$.

After that, set R is updated to $R = \{(0,3,0)\}$ and the cluster C_2 is computed from C'_2 presented in Figure 65 (b). In this case, C_1 and C_3 - depicted in Figure 65 (a) and 65 (c) - are the same as before the event observation. At last, set I is updated to $I = \emptyset$.



(a) C_1 (b) C'_2 (c) C_3

Figure 65 – C_1, C'_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s$.

- **Observed trace** $u_l s_{2r} u_r d_r d_s g_p$

  The next generated event is g_p and it is observed by D_2. Then, the local state estimator communicates C_2^{com} as illustrated in Figure 66.

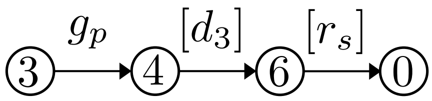


Figure 66 – C_2^{com} after observation of trace $u_l s_{2r} u_r d_r d_s g_p$.

After that, C_2 is updated as shown in Figure 67. In this case, C_2 is equal to C_2^{com}. Since event g_p was not observed by D_1 or D_3, their clusters were not modified and they are depicted in Figure 65 (a) and 65 (c).

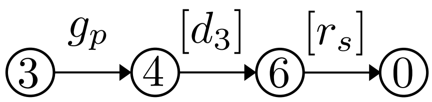


Figure 67 – C_2 after observation of trace $u_l s_{2r} u_r d_r d_s g_p$.

Set I is updated to $I = \{2\}$ and a new S - depicted in Figure 68 - is computed.

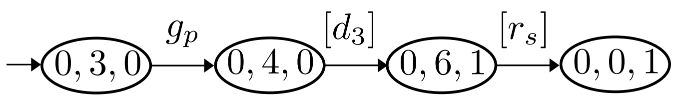
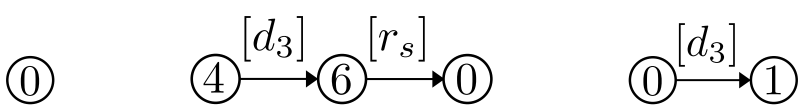


Figure 68 – S after observation of trace $u_l s_{2r} u_r d_r d_s g_p$.

After that, set R is updated to $R = \{(0,4,0)\}$, a new C'_2 is computed and C_2 is updated as shown in Figure 69 (b). C_1 and C_3 are not modified and they are illustrated in Figure 69 (a) and 69 (c), respectively. Then, set I is updated to $I = \emptyset$.



(a) C_1 (b) C'_2 (c) C_3

Figure 69 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p$.

- **Observed trace** $u_l s_{2r} u_r d_r d_s g_p u_l$

  After the second occurrence of event u_l, Algorithm 3 sends C_1^{com} to the diagnoser and it is depicted in Figure 70.

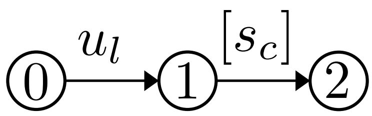
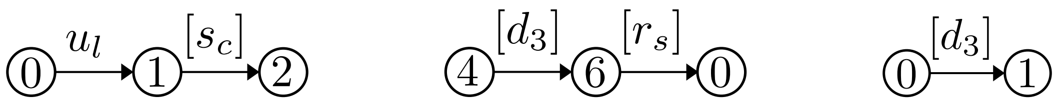


Figure 70 – C_1^{com} after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l$.

Then, C_1 is updated and equal to C_1^{com} as shown in Figure 71 (a). C_2 and C_3 are the same as before the event observation as depicted in Figure 71 (b) and 71 (c), respectively. A new S is computed from C_1, C_2, and C_3 as presented in Figure 72. At this point, set I is updated to $I = \{1\}$. Notice that events d_3 and r_s are now possible in S.



(a) C_1 (b) C_2 (c) C_3

Figure 71 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l$.

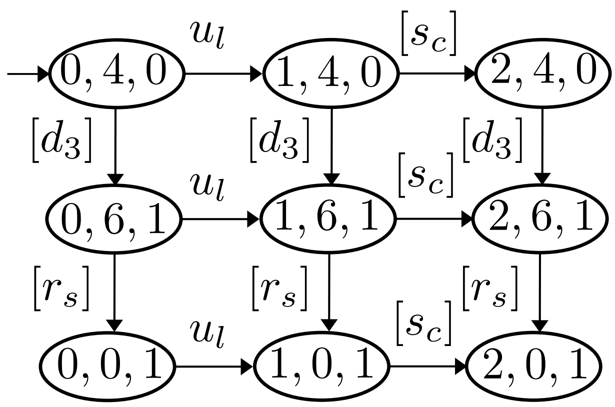
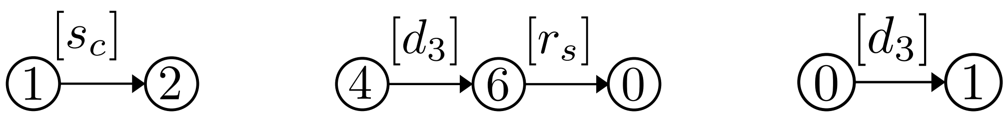


Figure 72 – S after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l$.

After that, Algorithm 4 updates $R = \{(1,4,0),(1,6,1),(1,0,1)\}$. Cluster automaton C_1, illustrated in Figure 73 (a) is computed according to lines 21 and 22 of Algorithm 4. Also, set I is updated to $I = \emptyset$. C_2 and C_3 are not modified and can be seen in Figure 73 (b) and 73 (c).

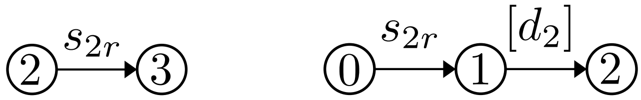


(a) C_1 (b) C_2 (c) C_3

Figure 73 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l$.

- **Observed trace** $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r}$

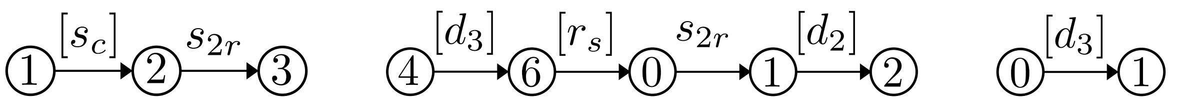
  When event s_{2r} occurs for the second time, it is observed by D_1 and D_2. These local state estimators communicate C_1^{com} and C_2^{com} to the diagnoser as shown in Figure 74 (a) and 74 (b), respectively.



(a) C_1^{com} (b) C_2^{com}

Figure 74 – C_1^{com} and C_2^{com} after observation of trace $u_l s_{2r} u_r d_r d_s\ g_p u_l s_{2r}$.

Then, C_1 and C_2 are computed according to the operation union of each cluster as depicted in Figure 75 (a) and 75 (b), respectively. C_3 remains the same cluster automaton as seen in Figure 75 (c). Algorithm 4 updates $I = \{1, 2\}$ according to line 12.



(a) C_1 (b) C_2 (c) C_3

Figure 75 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r}$.

In this point, a new S is computed and it is depicted in Figure 76. Notice that S has three initial states since $R = \{(1, 4, 0), (1, 6, 1), (1, 0, 1)\}$.

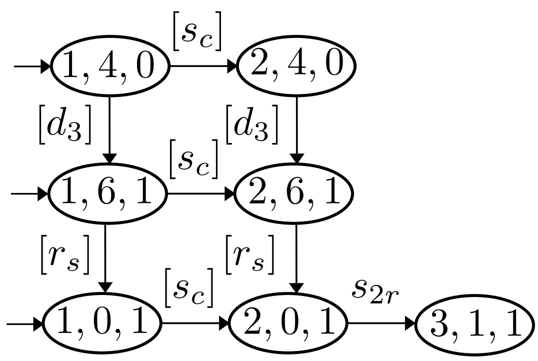
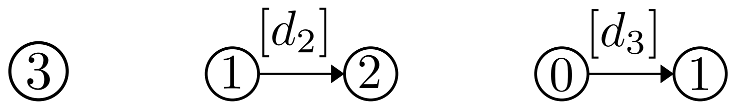


Figure 76 – S after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r}$.

Then, set R is updated to $R = \{(3, 1, 1)\}$. Cluster automata C_1 and C_2 are updated, as presented in Figure 77 (a) and 77 (b), respectively. C_3 does not have any modification as can be seen in Figure 77 (c). Algorithm 4 then updates set $I = \emptyset$ and waits a new event occurrence.



(a) C_1 (b) C_2 (c) C_3

Figure 77 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l\ s_{2r}$.

- **Observed trace** $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r$

  The occurrence of event u_r is observed by D_1 which sends C_1^{com} to the diagnoser as illustrated in Figure 78.

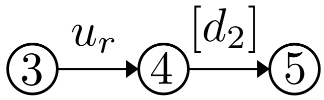


Figure 78 – C_1^{com} after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r$.

After that, C_1 is computed as shown in Figure 79 and set I is updated to $I = \{1\}$. In this case, C_1 is equal to C_1^{com}.

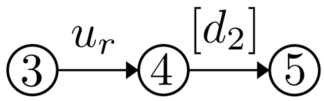


Figure 79 – C_1 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r$.

A new S is computed by Algorithm 4 and it is shown in Figure 80. Set R is now updated to $R = \{(4, 1, 1)\}$.

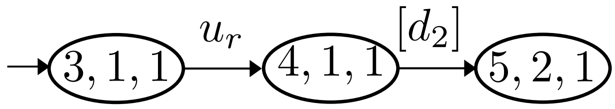
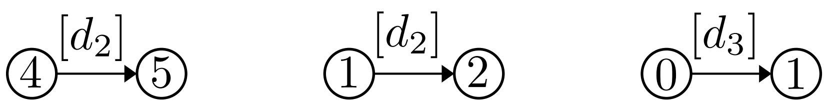


Figure 80 – S after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r$.

A new C_1' - presented in Figure 81 (a) - is computed and C_1 is updated. Notice that C_2 and C_3 do not change as depicted in Figure 81 (b) and 81 (c), respectively. Set I is updated to $I = \emptyset$.



(a) C_1' (b) C_2 (c) C_3

Figure 81 – C_1', C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l\ s_{2r} u_r$.

- **Observed trace** $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r$

The next generated event is d_r and it is observed by D_1 which communicates the cluster automaton C_1^{com} as shown in Figure 82. Then, C_1 is updated as presented in Figure 83 (a). Clusters C_2 and C_3 are not modified since D_2 or D_3 observe the event d_r and can be seen in Figure 83 (b) and 83 (c), respectively. Algorithm 4 then updates $I = \{1\}$.

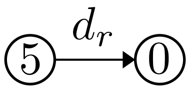
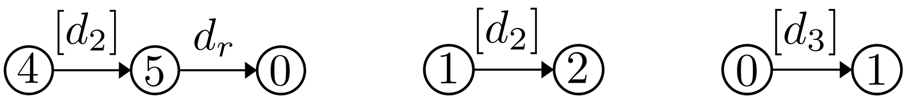


Figure 82 – C_1^{com} after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r$.



(a) C_1 (b) C_2 (c) C_3

Figure 83 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l\ s_{2r} u_r d_r$.

After that, a new S is computed from C_1, C_2, and C_3 as illustrated in Figure 84. Algorithm 4 now updates set R to $R = \{(0, 2, 1)\}$.

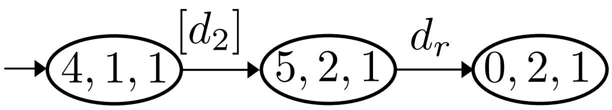
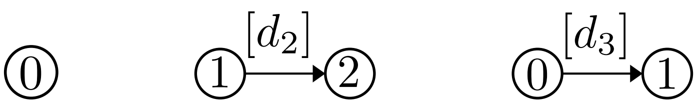


Figure 84 – S after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r$.

Then, C_1 is updated according to lines 21 and 22 from Algorithm 4 as depicted in Figure 85 (a). C_2 and C_3 are the same as before the observed event as shown in Figure 85 (b) and 85 (c), respectively. Set I is now updated to $I = \emptyset$.



(a) C_1 (b) C_2 (c) C_3

Figure 85 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l\ s_{2r} u_r d_r$.

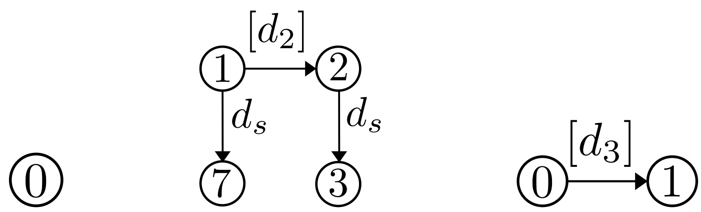
- **Observed trace** $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r d_s$

  The second occurrence of event d_s is observed by D_2 which communicates two clusters C_2^{com} - depicted in Figure 86 (a) and 86 (b) - to the diagnoser.

(a) C_2^{com} (b) C_2^{com}

Figure 86 – C_2^{com} after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r d_s$.

Then, C_2 is updated after the union operation according to line 11 of Algorithm 4 and it is depicted in Figure 87 (a). C_1 and C_3 are not modified as shown in Figure 87 (b) and 87 (c). Algorithm 4 updates set $I = \{2\}$.



(a) C_1 (b) C_2 (c) C_3

Figure 87 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l\ s_{2r} u_r d_r d_s$.

A new S is computed as presented in Figure 88. Then R is updated to $R = \{(0, 3, 1)\}$.

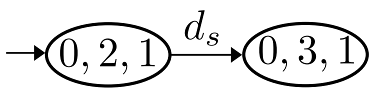
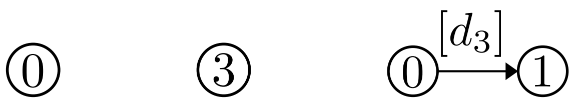


Figure 88 – S after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r d_s$.

After that, C_2 is updated, presented in Figure 89 (b), according to lines 21 and 22 of Algorithm 4 by removing the states and transitions that are not reachable after event d_r. $C1$ and C_3 are not modified as illustrated in Figure 89 (a) and 89 (c), respectively. Then, set I is updated to $I = \emptyset$.



(a) C_1 (b) C_2 (c) C_3

Figure 89 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l\ s_{2r} u_r d_r d_s$.

- **Observed trace** $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r d_s g_p$

When event g_p occurs for the second time, it is observed by D_2 which communicates C_2^{com} as shown in Figure 90.

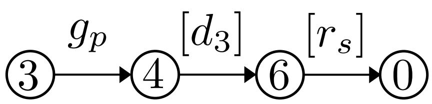


Figure 90 – C_2^{com} after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r\ d_r d_s g_p$.

After that, C_2 is updated as depicted in Figure 91. In this case, C_2 is equal to C_2^{com}. Clusters C_1 and C_3 - depicted in Figure 89 (a) and 89 (c) - are not modified at this point since local state estimators D_1 and D_3 do not observe event g_p. Then, set I is update to $I = \{2\}$.

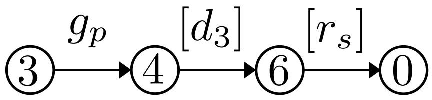


Figure 91 – C_2 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r\ d_r d_s g_p$.

A new S is now computed by Algorithm 4 and it is shown in Figure 92. Then, set R is updated to $R = \{(0,4,1)\}$.

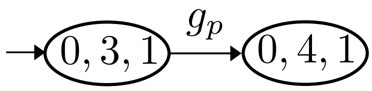
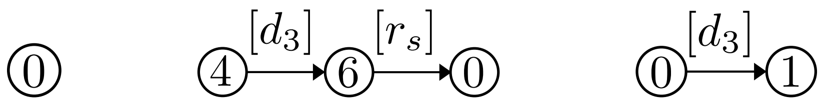


Figure 92 – S after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r d_s g_p$.

After that, cluster C_2 is updated, illustrated in Figure 93 (b). C_1 and C_3 are not updated as presented in Figure 93 (a) and 93 (c), respectively. Then, Algorithm 4 updates set $I = \emptyset$.



(a) C_1 (b) C_2 (c) C_3

Figure 93 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l\ s_{2r} u_r d_r d_s g_p$.

- **Observed trace** $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r d_s g_p u_l$

  After the third occurrence of event u_l, D_1 sends C_1^{com} to diagnoser as depicted in Figure 94.

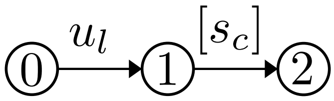


Figure 94 – C_1^{com} after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r\ d_r d_s g_p u_l$.

Then, C_1 is updated and, in this case, it is equal to C_1^{com} as shown in Figure 95. After that, set I is updated to $I = \{1\}$.

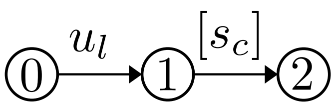


Figure 95 – C_1 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r\ d_r d_s g_p u_l$.

A new S is computed from C_1, C_2, and C_3 as presented in Figure 96. Then, set R is updated to $R = \{(1,4,1)\}$.

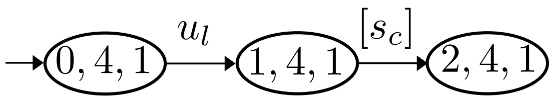
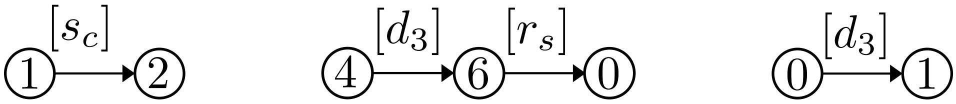


Figure 96 – S after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r d_s g_p u_l$.

After that, C_1 is updated as shown in Figure 97 (a). Clusters C_2 and C_3 are not modified and can be seen in Figure 97 (b) and 97 (c), respectively. Then, set I is updated to $I = \emptyset$.



(a) C_1 (b) C_2 (c) C_3

Figure 97 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l$ $s_{2r} u_r d_r d_s g_p u_l$.

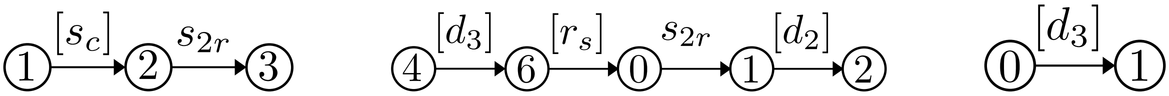
- **Observed trace** $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r d_s g_p u_l s_{2r}$

  Finally, the third occurrence of event s_{2r} is observed by D_1 and D_2 which send C_1^{com} and C_2^{com} as presented in Figure 98 (a) and 98 (b), respectively.

(a) C_1^{com} (b) C_2^{com}

Figure 98 – C_1^{com} and C_2^{com} after observation of trace $u_l s_{2r} u_r d_r d_s$ $g_p u_l s_{2r} u_r d_r d_s g_p u_l s_{2r}$.

Then, clusters C_1 and C_2 are computed according to the operation union of each cluster as illustrated in Figure 99 (a) and 99 (b), respectively. C_3 is not modified at this stage as shown in Figure 99 (c). After that, set I is updated to $I = \{1, 2\}$.



(a) C_1 (b) C_2 (c) C_3

Figure 99 – C_1, C_2, and C_3 after observation of trace $u_l s_{2r} u_r d_r d_s$ $g_p u_l s_{2r} u_r d_r d_s g_p u_l s_{2r}$.

Then, a new S is computed as shown in Figure 100. Notice that, since $R = \{(1, 4, 1)\}$, event s_{2r} is common to two components but not feasible in both states, no transition leave state $(2, 4, 1)$ of S. Therefore, S is equal to a graph composed of two states $(1, 4, 1)$ and $(2, 4, 1)$, presented in Figure 100. Since there are no transitions in S labeled with event s_{2r}, the fault occurrence is detected in line 17 of Algorithm 4.

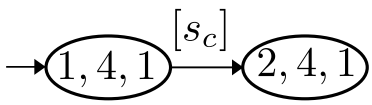


Figure 100 – S after observation of trace $u_l s_{2r} u_r d_r d_s g_p u_l s_{2r} u_r d_r d_s g_p u_l s_{2r}$.

5.3.1 Concluding remarks of the case study

It is important to notice that the fault occurrence is detected after the third observation of event s_{2r}. Moreover, in the worst-case scenario for this example, S has 9 states and the sum of states of G_{N_1}, G_{N_2}, and G_{N_3} is equal to 26. On the other hand, the fault-free behavior automaton, G_N, has 96 states.

Notice also the growth of the cardinality of the exceeding language $L_{Exc,N_a}^{\leq n}$ for the DSD scheme presented in section 3.3 applied to this practical implementation in Figure 101. For example, for $n = 9$, there are 9,093 more traces in $L_{N_a}^{\leq n}$. Notice that the cardinality of the exceeding language considerably increases as n grows, which is avoided in the DSDC scheme proposed in this work.

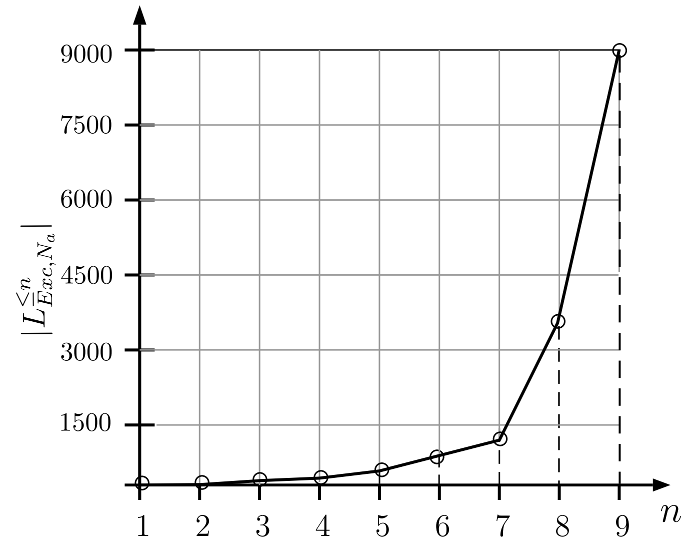


Figure 101 – Cardinality of the exceeding language generated by the decentralized synchronous diagnosis scheme $L_{Exc,N_a}^{\leq n}$ ($\circ$) for different values of n.

5.4 FINAL REMARKS

In this chapter, the DSDC method is applied to a real system. A didactic manufacturing system installed at Industrial Computing and Automation Laboratory of Federal University of Santa Catarina was considered. In order to implement the DSDC method, the controlled plant is first modeled from three component models where a brief explanation of their operations is presented. The fault-free component models are used as input to the communication protocol algorithm, which runs together to the diagnosis procedure.

In this case, the system is not synchronously codiagnosable since the local projections of the fault trace can be observed in the fault-free models of the local components. In order to diagnose the fault event occurrence, a fault trace generated by the system was considered and the DSDC method is applied to each event observation of the fault trace. The exceeding language accepted for the decentralized synchronous diagnosis for this implementation is also presented.

6 CONCLUSION

Recently, the synchronous diagnosis strategy has been presented in the literature. In this scheme, state estimators of the fault-free system component models are implemented in parallel, and the fault event is diagnosed when an observable event that is not feasible in the current state estimate of at least one fault-free component model occurs. Although, by using this method, an exponential complexity order, with the number of the system components, is avoided for diagnosis, the fault-free language accepted by the synchronous diagnosis can be larger than the observable fault-free language generated by the system. Thus, a system may be diagnosable and not synchronously codiagnosable due to this property, which reduces the diagnosis efficiency. Thus, in this work, a decentralized synchronous diagnosis with coordination (DSDC) for discrete event systems modeled as automata is proposed with the view to eliminate this fault-free language growth.

In the DSDC method, two algorithms run together: the communication protocol and the diagnosis procedure, also called coordinator. Local state estimators send the information regarding event observations, and the coordinator provides the diagnosis status. The method uses cluster automata of the fault-free component models to online computed the current fault-free state estimate of the global system model in order to verify if an observed event is feasible. If an observed event is not feasible in the current fault-free state estimate, the fault event occurrence is diagnosed. The main advantage of the proposed method is that the fault-free language accepted by the DSDC is equal to the observable fault-free language of the global system. In general, the DSDC method has a smaller computational complexity than traditional diagnosis approaches. Moreover, the same diagnosis power is achieved as the monolithic technique using only the fault-free component models for diagnosis.

The method was implemented to a real system that is not synchronously codiagnosable. It is shown that the DSDC method successfully diagnose the fault event occurrence after a bounded number of event observations. The implementation of the method illustrates that if the system is monolithically diagnosable, the DSDC method can be applied.

The main contributions of this work are highlighted in the following:

- An algorithm for the communication protocol between local state estimators and a coordinator is proposed;
- A cluster automata synchronous composition is proposed;
- An algorithm for the coordinator that detects if the fault event has been occurred and synchronizes the common unobservable events is proposed;
- The DSDC method is guaranteed to have the same diagnosis power as the monolithic diagnosis approach;

- An application of the DSDC method in a real system is presented.

In the sequel, future research themes that can be carried out from this work are presented.

- A full implementation of the DSDC method using PLCs and a Supervisory Control and Data Acquisition (SCADA) approach is currently being investigated.
- In this work, the communication between local state estimators and the coordinator is supposed to be ideal, which cannot always be guaranteed. A DSDC method that is robust to communication delays and/or package losses is an open research theme.

The results presented in this thesis have been submitted for presentation in the next International Conference on Automatic Control and Soft Computing (CONTROLO 2022) as Mayer et al. (2022).

REFERENCES

BASILE, Francesco. Overview of Fault Diagnosis Methods based on Petri Net Models. In: EUROPEAN Control Conference - ECC 2014. [S.l.: s.n.], 2014. P. 2636–2642.

BASILIO, J. C.; CARVALHO, Lilian K.; MOREIRA, Marcos V. Diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos. **Sba: Controle & Automação Sociedade Brasileira de Automatica**, v. 21, n. 5, p. 510–533, 2010.

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

CABRAL, Felipe G. **Synchronous Failure Diagnosis of Discrete-Event Systems**. 2017. S. 143. PhD thesis – Universidade Federal do Rio de Janeiro.

CABRAL, Felipe G.; MOREIRA, Marcos V. Synchronous Diagnosis of Discrete-Event Systems. **IEEE Transactions on Automation Science and Engineering**, IEEE, v. 17, n. 2, p. 921–932, 2020.

CABRAL, Felipe G.; MOREIRA, Marcos V.; DIENE, Oumar; BASILIO, João Carlos. A Petri Net Diagnoser for Discrete Event Systems Modeled by Finite State Automata. **IEEE Transactions on Automatic Control**, v. 60, n. 1, p. 59–71, 2015.

CABRAL, Felipe G.; VERAS, Maria Z.M.; MOREIRA, Marcos V. Conditional synchronized diagnoser for modular discrete-event systems. In: PROCEEDINGS of the 14th International Conference on Informatics in Control, Automation and Robotics - ICINCO 2017. [S.l.: s.n.], 2017. P. 88–97.

CARVALHO, Lilian K.; BASILIO, João C.; MOREIRA, Marcos V. Robust diagnosis of discrete event systems against intermittent loss of observations. **Automatica**, Elsevier Ltd, v. 48, n. 9, p. 2068–2078, 2012.

CASSANDRAS, Christos G.; LAFORTUNE, Stéphane. **Introduction to Discrete Event Systems**. 2. ed. [S.l.]: Spring, 2008. P. 757.

CONTANT, Olivier; LAFORTUNE, Stéphane; TENEKETZIS, Demosthenis. Diagnosability of discrete event systems with modular structure. **Discrete Event Dynamic Systems: Theory and Applications**, v. 16, n. 1, p. 9–37, 2006.

CORMEN, Thomas H; LEISERSON, Charles E; RIVEST, Ronald L; STEIN, Clifford. **Introduction to algorithms**. 3. ed. [S.l.]: MIT Press, 2009.

DAIGLE, Matthew; KOUTSOUKOS, Xenofon; BISWAS, Gautam. Fault diagnosis of continuous systems using discrete-event methods. In: IEEE Conference on Decision and Control. [S.l.: s.n.], 2007. P. 2626–2632.

DAVID, René; ALLA, Rassane. **Discrete, Continuous and Hybrid Petri Nets**. [S.l.]: Springer, 2005. P. 541.

DEBOUK, Rami; LAFORTUNE, Stéphane; TENEKETZIS, Demosthenis. Coordinated decentralized protocols for failure diagnosis of discrete event systems. **Discrete Event Dynamic Systems: Theory and Applications**, v. 10, n. 1, p. 33–86, 2000.

DEBOUK, Rami; MALIK, Robi; BRANDIN, Bertil. A modular architecture for diagnosis of discrete event systems. In: IEEE Conference on Decision and Control. [S.l.: s.n.], 2002. P. 417–422.

FESTO. **Festo Didactic Learning System for Automation and Technology**. Denkendorf: [s.n.], 2006. P. 104. Available from: `www.festo-didactic.com`.

HARARY, Frank. **Graph Theory**. [S.l.]: Addison-Wesley, 1969. P. 281.

HASHTRUDI ZAD, Shahin; KWONG, Raymond H.; WONHAM, W. M. Fault diagnosis in discrete-event systems: Framework and model reduction. **IEEE Transactions on Automatic Control**, v. 48, n. 7, p. 1199–1212, 2003.

KEROGLOU, Christoforos; HADJICOSTIS, Christoforos N. Distributed diagnosis using predetermined synchronization strategies. In: IEEE Conference on Decision and Control. [S.l.: s.n.], 2014. P. 5955–5960.

KEROGLOU, Christoforos; HADJICOSTIS, Christoforos N. Distributed Fault Diagnosis in Discrete Event Systems via Set Intersection Refinements. **IEEE Transactions on Automatic Control**, v. 63, n. 10, p. 3601–3607, 2018.

LAWSON, Mark. **Finite Automata**. Florida: CRC Press company, 2004. ISBN 1-58488-255-7.

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

MAYER, Patrícia C.; CABRAL, Felipe G.; MOREIRA, Marcos V. Decentralized Synchronous Diagnosis with Coordination. In: CONTROLO. [S.l.: s.n.], 2022. Submitted for presentation.

MOREIRA, Marcos V.; JESUS, Thiago C; BASILIO, João C. Polynomial Time Verification of Decentralized Diagnosability of Discrete Event Systems. n. 7, 2011.

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

QIU, Wenbin; KUMAR, Ratnesh. Distributed diagnosis under bounded-delay communication of immediately forwarded local observations. In: PROCEEDINGS of the American Control Conference. [S.l.: s.n.], 2005. P. 1027–1032.

QIU, Wenbin; KUMAR, Ratnesh. Distributed diagnosis under bounded-delay communication of immediately forwarded local observations. **IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans**, v. 38, n. 3, p. 628–643, 2008.

SAMPATH, Meera; SENGUPTA, Raja; LAFORTUNE, Stéphane; SINNAMOHIDEEN, Kasim; TENEKETZIS, Demosthenis C. Failure diagnosis using discrete-event models. **IEEE Transactions on Control Systems Technology**, v. 4, n. 2, p. 105–124, 1996.

SAMPATH, Meera; SINNAMOHIDEEN, Kasim; LAFORTUNE, Stephane; TENEKETZIS, Demosthenis. Diagnosability of Discrete-Event Systems. **IEEE Transactions on Automatic Control**, v. 40, n. 9, p. 1555–1575, 1995.

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

VERAS, Maria Z.M.; CABRAL, Felipe G.; MOREIRA, Marcos V. Distributed synchronous diagnosis of discrete event systems modeled as automata. **Control Engineering Practice**, v. 115, p. 104892, 2021.

WANG, Yin; YOO, Tae Sic; LAFORTUNE, Stéphane. Diagnosis of discrete event systems using decentralized architectures. **Discrete Event Dynamic Systems: Theory and Applications**, v. 17, n. 2, p. 233–263, 2007.

WHITE, A.; KARIMODDINI, A.; SU, R. Fault Diagnosis of Discrete Event Systems under Unknown Initial Conditions. **IEEE Transactions on Automatic Control**, v. 64, n. 12, p. 5246–5252, 2019.